Thèse

présentée à l'Université d'Orléans

pour obtenir le grade de

Docteur de l'Université d'Orléans

par : Nicolas Markey

Discipline : Informatique

Logiques temporelles pour la vérification : expressivité, complexité, algorithmes

Soutenue le 03 avril 2003

Membres du Jury :

– Jean-Philippe Anker	examinateur
– Ahmed Bouajjani	rapporteur
– François Le Berre	directeur de thèse
– Philippe Schnoebelen	co-directeur de thèse
– Colin Stirling	rapporteur
– Pierre Wolper	président du jury

Merci !

C'est difficile d'écrire une page de remerciements : d'une part, la liste des personnes que je veux remercier est longue, et je risque d'en oublier certaines par inadvertance (je les prie de m'en excuser) ; et d'autre part, parce que c'est généralement la page la plus lue de la thèse, et qu'il faudrait, si je respectais la proportion de 178 pages en trois ans, que j'y consacre six jours[1]. Mais bon, une thèse sans sa page de remerciements, c'est comme un aquarium sans poisson rouge : c'est intriguant... Je me plie donc à la tradition.

Je remercie donc en tout premier lieu les membres de mon jury : Pierre WOLPER qui, sans même avoir candidaté, a été élu président du jury, et a accepté cette tâche ; Ahmed BOUAJJANI et Colin STIRLING, qui ont accepté d'être rapporteur de ma thèse ; et Jean-Philippe ANKER, qui a accepté d'être le représentant de l'École Doctorale dans ce jury.

Je remercie très chaleureusement François et Philippe, qui m'ont tous les trois[2] encadré durant cette thèse :

- François LE BERRE a été mon directeur de thèse « administratif » à Orléans, me permettant ainsi de rester « en banlieue parisienne ». Il a toujours été disponible pour répondre aux questions que je lui posais, autant pour ma thèse que pour mon monitorat ;
- Philippe SCHNOEBELEN, qui a été mon co-directeur de thèse, à Cachan. Son enthousiasme, sa patience et sa disponibilité —que je sois à Gênes, à Brno, ou à Canberra, il était là...— ont beaucoup facilité et agrémenté mon travail de thèse ;
- François LAROUSSINIE, qui a été mon coco-directeur de thèse au cours de ces trois ans, et qui m'a initié à l'informatique quand j'étais petit (avec Antoine PETIT (aussi), que je remercie également au passage). Au cours de toutes ces années, j'ai beaucoup apprécié ses conseils, sa bonne humeur, sa générosité...

Merci aussi à mes parents, mes frères et sœurs, pour moultes raisons, mais plus particulièrement ici, pour s'être occupés du pot de thèse. Plus généralement, grand merci à toute ma famille, en particulier Anita, Jean-François et Pauline, qui m'ont hébergé lors de mes excursions orléanaises.

Le LIFO et le LSV sont deux endroits où il fait bon travailler. Je remercie Gaëtan HAINS et Michel BIDOIT, leurs directeurs respectifs, pour m'avoir accueillis dans leurs laboratoires. J'adresse également mes remerciements aux chercheurs, enseignants, ingénieurs-systèmes,

[1] Tous comptes faits, c'est quand même à peu près le temps que ça m'a pris.
[2] Il y a deux François...

secrétaires et thésards que j'y ai côtoyés. Merci toute particulièrement à ceux qui ont été mes « co-bureauliers » au cours de ces trois ans, et avec qui j'ai passé d'excellents moments.

J'ai également une pensée particulière pour le groupe des « vieux cachanais », ~~que je~~ qui me supporte depuis sept ans maintenant. Je n'oublierai jamais les cours, pauses, discussions, bouffes, vacances, coinches, rappels du B, petits foots nocturnes, etc. en leur compagnie. Merci aussi à mes ami(e)s qui ne sont pas cachanais, ou qui ne sont pas vieux, et qui m'ont soutenu pendant ma thèse, en particulier pendant que je rédigeais. Je remercie en particulier Luigi lavAzza et les abbayes belges pour leurs excellents breuvages.

Je remercie enfin tous ceux qui liront ces quelques lignes (autant celles qui précèdent que celles qui suivent), puisque, après tout, c'est quand même dans ce but que je les ai écrites.

Table des matières

Introduction

« Ça ne se produit pas immédiatement, mais sans prévenir vous vous apercevrez que les choses ne sont pas telles que vous pensiez qu'elles étaient. »

Mark Z. DANIELEWSKI, *La Maison des feuilles.*

Informatique et fiabilité...

L'informatique est aujourd'hui omniprésente dans la vie quotidienne, dans des appareils plus ou moins « critiques », et de plus ou moins grande taille. Des systèmes automatisés de contrôle et de traitement de l'information sont présents dans les machines à laver, dans les téléphones, ou dans les voitures. À plus grande échelle, dans les distributeurs automatiques de billets, les usines ou les centres de télécommunications. On les trouve également dans des applications qui mettent en jeu des vies humaines, comme par exemple des appareils de traitements médicaux, les systèmes de gestion du trafic des trains et des avions, les centrales nucléaires, *etc.* Il n'est pas possible d'en dresser une liste exhaustive.

Le développement de ces appareils au cours des trente dernières années a été fulgurant, et il a très rapidement été confronté au problème de la sûreté des applications gérées par ces systèmes. Ce problème reste relativement limité quand il s'agit de la machine à laver ou du téléphone, mais devient crucial dès lors qu'il touche à des systèmes vitaux ou de très grande taille. Du fait de la croissance rapide des capacités des systèmes informatiques, et de la mise en place de « parallélisme » entre ces systèmes, les difficultés se sont multipliées dans la conception de ces machines.

On peut ainsi citer de nombreux problèmes liés à des failles dans les systèmes informatiques [Der00] :

- [SG98] raconte qu'un processus lancé en 1967 sur une machine du M.I.T. n'avait toujours pas eu accès au processeur en 1973. Ce genre de problème (accès à une ressource partagée par plusieurs systèmes concurrents) intervient très fréquemment dans les systèmes de communication, par exemple ;
- l'explosion de la fusée Ariane, le 4 juin 1996, est due à un problème informatique. Le rapport de la commission d'enquête [EC96] affirme en effet les choses suivantes :

> *« C'est la perte totale des informations de guidage et d'attitude 37 secondes après le démarrage de la séquence d'allumage du moteur principal (30 secondes après le décollage) qui est à l'origine de l'échec d'Ariane 501. Cette perte d'informations est due à des erreurs de spécification et de conception du logiciel du système de référence inertielle.* [...]
>
> – *l'exception logiciel interne du SRI s'est produite pendant une conversion de données de représentation flottante à 64 bits en valeurs entières à 16 bits. Le nombre en représentation flottante qui a été converti avait une valeur qui était supérieure à ce que pouvait exprimer un nombre entier à 16 bits. Il en est résulté une erreur d'opérande. Les instructions de conversion de données (en code Ada) n'étaient pas protégées contre le déclenchement d'une erreur d'opérande bien que d'autres conversions de variables comparables présentes à la même place dans le code aient été protégées ;*
> – *l'erreur s'est produite dans une partie du logiciel qui n'assure que l'alignement de la plate-forme inertielle à composants liés. Ce module de logiciel calcule des résultats significatifs avant le décollage seulement. Dès que le lanceur décolle, cette fonction n'est plus d'aucune utilité ».*

La commission suggère de

> *« mettre sur pied une équipe qui sera chargée d'élaborer la procédure de qualification des logiciels, de proposer des règles strictes de confirmation de la qualification, et de s'assurer que la spécification, la vérification et les essais des logiciels seront d'un niveau de qualité systématiquement élevé dans le programme Ariane 5. Envisager de faire appel à des spécialistes externes en matière de RAMS (fiabilité, disponibilité, facilité de maintenance, sécurité). »*

– la perte de Mars Climate Observer, le 23 septembre 1999 : le satellite a été perdu lors de sa manœuvre d'insertion dans l'orbite de Mars. Le problème est le suivant [MCO99] :

> *« thruster performance data in English units instead of metric units was used in the software application code* [...] *Subsequent processing of the data from AMD file by the navigation software algorithm therefore underestimated the effect of the spacecraft trajectory by a factor of 4.45, which is the required conversion factor from force in pounds to newtons. »*

– citons encore le problème de l'opérateur de division flottante du pentium d'Intel, apparu en 1995 ; la panne des centraux téléphoniques de Washington, Pittsburgh, Los Angeles et San Francisco, en juillet 1991 ; un missile antimissile américain ratant sa cible, en février 1991, à cause d'erreurs d'arrondis ; des malades gravement irradiés par un appareil médical contre le cancer, entre 1985 et 1987 ; ou encore le célèbre « bug de l'an 2000 »...

Cette liste est encore longue, et il apparaît primordial de développer des méthodes formelles de vérification des systèmes informatiques, et d'appliquer ces méthodes dès la conception du système.

Vérification formelle des systèmes informatiques

Comme nous l'avons vu au travers des exemples ci-dessus, les causes des problèmes sont diverses, mais elles sont souvent liées à un problème de conception, et rarement à un problème d'implémentation. Des techniques de vérification formelles sont donc tout indiquées pour « lutter » contre ces problèmes.

La vérification formelle se déroule généralement en trois phases :
– une phase qui consiste à modéliser le système étudié, et souvent à l'« abstraire », c'est-à-dire à le simplifier, en gardant au minimum les comportements nécessaires à la vérification des propriétés qui nous intéressent ;
– une phase de spécification, qui exprime, sous différentes formes possibles que nous verrons plus loin, les propriétés que le modèle devra vérifier ;
– une phase de vérification à proprement parler, qui s'assure que le modèle vérifie la propriété.

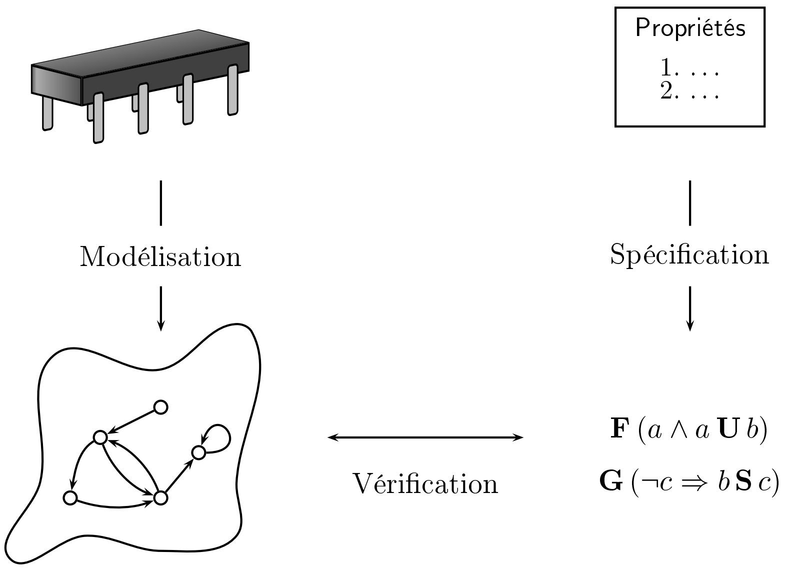
Il existe plusieurs approches pour la vérification formelle, notamment :
– le *test*, dans lequel la deuxième phase du processus ci-dessus consiste à générer un *scénario* visant à « mettre en échec » une propriété que devrait vérifier le modèle. La troisième phase consiste alors à « exécuter » ce scénario, afin de voir s'il crée une erreur dans le modèle ou pas. Si aucun scénario ne parvient à trouver une faille dans le modèle, alors le modèle est validé pour cette propriété. L'inconvénient est que, pour une propriété donnée, il peut exister une multitude de scénarios à tester, et cette méthode n'est donc en général pas exhaustive ;
– le *model checking* , qui est l'approche qui nous intéresse ici. Dans cette méthode, la propriété est exprimée dans un certain langage, choisi en fonction de son expressivité, sa maniabilité, ou l'efficacité de ses algorithmes de vérification. La troisième phase consiste alors à vérifier, par des algorithmes *ad hoc*, que le modèle construit satisfait la propriété ainsi exprimée. Cette méthode a l'avantage d'être complètement automatique et exhaustive, mais en contrepartie, elle est plus complexe à mettre en œuvre.

Dans cette thèse, nous nous intéresserons plus particulièrement aux techniques de model checking.

Le model checking

Comme nous l'avons vu précédemment, le model checking peut être « découpé » en trois phases, représentées sur la figure 1. Cette figure indique les étapes à suivre dans le processus de vérification de certaines propriétés sur un système informatique. Les trois phases de ce processus correspondent à trois domaines de recherche différents :
– au niveau du modèle : de nombreuses structures ont été proposées par le passé afin de modéliser des systèmes réactifs : structures de Kripke, automates, automates temporisés, réseaux de Petri, . . .
– au niveau du langage de spécification : plusieurs langages ont été proposés et utilisés : logiques temporelles du temps linéaire ou du temps arborescent, μ-calcul, . . . Ces différents langages sont plus ou moins expressifs, plus ou moins faciles à utiliser, plus ou moins complexes à vérifier. . .

Figure 1 Le model checking



– au niveau de la vérification : en fonction du modèle et du langage de spécification, il existe différents algorithmes permettant de vérifier qu'un modèle satisfait sa spécification. Des techniques de *model checking symbolique* ont également été développées : elles consistent à représenter le système de façon « symbolique » (par opposition à « explicite »).

Mes travaux de thèse s'intéressent plus particulièrement à la partie « langage de spécification », et, parmi ces langages, à différentes logiques temporelles.

La logique modale, la logique temporelle

La *logique modale* [Gar01, BdRV01] est l'étude des déductions possibles à partir d'expressions comme « *il est nécessaire que...* » ou « *il est possible que...* ». Elle regroupe en fait également d'autres systèmes de déduction, comme la *logique temporelle* (« *un jour dans le futur...* », « *toujours dans le futur...* », ...) ou la *logique déontique* (« *il est obligatoire que...* », « *il est autorisé que...* », ...).

Les formules de ces logiques sont bâties à partir de « modalités ». Par exemple, en logique modale, on utilise généralement les symboles $\Box$ et $\Diamond$, qui signifient respectivement « *il est nécessaire que...* » et « *il est possible que...* ». Ainsi, la formule $\Box(\Diamond A \Rightarrow B)$ signifie que *il est nécessaire que, si A est possible, alors B est vraie.*

La *logique temporelle* [Pri57, Pri67, Pri69, Gal99, GRF00] a été introduite par Arthur Prior en 1957 afin de traiter des informations temporelles dans un cadre logique proche de la logique modale. La logique temporelle est construite de manière similaire, mais à partir des deux modalités suivantes :

– $\mathbf{G}$, qui signifie « *il sera toujours vrai, dans le futur, que...* » ;
– $\mathbf{H}$, qui signifie « *il a toujours été vrai, dans le passé, que...* ».

À partir de ces modalités, on peut en définir deux autres :

– **F**, qui signifie « *il sera vrai, au moins une fois dans le futur, que...* ». Cette modalité peut être définie par $\mathbf{F}\,A \stackrel{\text{def}}{=} \neg\mathbf{G}\,\neg A$ (c'est à dire que A sera vraie au moins une fois dans le futur si, et seulement si, $\neg A$ n'est pas toujours vraie dans le futur) ;
– **P**, qui signifie « *il a été vrai, au moins une fois dans le passé, que...* ». Nous noterons cette modalité $\mathbf{F}^{-1}$ dans la suite, pour bien marquer sa symétrie avec **F**. Comme pour la modalité **F**, on peut définir cette modalité à partir de **H**, par $\mathbf{P}\,A \stackrel{\text{def}}{=} \neg\mathbf{H}\,\neg A$.

Ces modalités sont interprétées sur des *structures de Kripke* [Kri63]. Une structure de Kripke est un ensemble d'états, appelés *mondes*. Ces états sont reliés entre eux par une relation binaire qui indique les transitions possibles entre les mondes. Chaque monde est étiqueté par un ensemble de propositions, qui sont les propositions vraies dans ce monde.

Plusieurs extensions de la logique temporelle ont été proposées : Hans W. Kamp a ainsi défini les modalités binaires **U** et **S** [Kam68]. Ces modalités se lisent respectivement *until* et *since*, et prennent cette fois deux arguments. Par exemple, $\mathbf{U}\,(A, B)$ signifie que A est vraie jusqu'à ce que B le soit. Arthur Prior a étendu les modalités classiques de la logique temporelle afin qu'elle puissent mesurer « combien de temps » s'écoule entre deux évènements. Il a ainsi défini les modalités « quantitatives » $\mathbf{F}\,(n, p)$, $\mathbf{G}\,(n, p)$, qui signifient respectivement que p sera vraie au plus tard dans n mondes (*i.e.* au plus tard dans n transitions), et que p sera vraie dans tous les n prochains états. Les modalités du passé équivalentes peuvent être définies, par exemple, par $\mathbf{P}\,(n, p) \stackrel{\text{def}}{=} \mathbf{F}\,(-n, p)$.

La logique temporelle dans le cadre du model checking

En 1977, Amir Pnueli a proposé d'utiliser la logique temporelle dans le domaine de la vérification formelle de systèmes réactifs [Pnu77]. En effet, les structures de Kripke sont des graphes d'états étiquetés qui permettent facilement de modéliser le comportement de systèmes informatiques, et la logique temporelle permet d'exprimer des propriétés que ces systèmes doivent vérifier. Les principales propriétés à vérifier sont des propriétés de « vivacité », qui indiquent qu'une propriété « désirable » sera atteinte et qui s'expriment simplement avec la modalité **F**, et des propriétés de « sûreté », indiquant qu'un état « indésirable » ne sera jamais atteint, ce qui s'exprime aisément avec la modalité **G** [Lam83].

Dans cette optique, [Pnu77] a défini une sémantique précise pour les modalités de la logique temporelle, et a montré qu'il est possible de vérifier, de manière automatique, qu'une propriété exprimée en logique temporelle est satisfaite dans un modèle fini.

Dès lors, un nouveau domaine de l'informatique était né [CGP99, SBB^{+}99, CS01], et de nombreuses questions se posaient :

– du point de vue de la sémantique tout d'abord, Leslie Lamport a soulevé le problème de la définition de « futur » [Lam80]. Il met ainsi en opposition la sémantique de $\mathbf{F}\,A$, qui signifie que A sera vraie dans le futur, et $\neg\mathbf{G}\,\neg A$, qui signifie qu'il n'est pas vrai que $\neg A$ est toujours vraie. L'adverbe *toujours* peut être interprété de deux manières différentes ici : soit « toujours dans une exécution du système », soit « toujours, quelle que soit l'exécution du système ». De cette remarque se sont dégagées deux classes de logiques temporelles : les *logiques temporelles du temps linéaire*, qui se rattachent à la première sémantique, et les *logiques temporelles du temps arborescent* qui correspondent à la seconde approche ;

- à partir de cette remarque, de nombreuses logiques temporelles ont été définies. En effet, outre les deux sémantiques possibles concernant le temps, il est possible d'utiliser les modalités classiques, d'utiliser les modalités quantitatives définies par Prior, ou de définir d'autres modalités. De nombreuses logiques temporelles ont ainsi vu le jour au cours des années 1980, dont les deux plus connues sont LTL (Linear-time Temporal Logic) [GPSS80, Pnu81] et CTL (Computation Tree Logic) [CE81, QS82]. La « sémantique du temps » est différente dans les deux cadres, et ces logiques temporelles ne sont pas comparables du point de vue de l'expressivité. Le choix de la logique temporelle utilisée dépendra donc en partie du pouvoir d'expression nécessaire pour exprimer les propriétés à vérifier ;
- le fait que le model checking soit décidable (*i.e. réalisable de manière automatique par une machine*) soulève inévitablement les questions d'« algorithmes » et de « complexité » de ces problèmes. Du point de vue des algorithmes, plusieurs méthodes ont été proposées, que ce soit pour les logiques du temps arborescent [CE81, QS82] ou pour les logiques du temps linéaire [WVS83, LP85, VW86]. Dans le même temps, des études ont été menées afin de déterminer la complexité effective de ces algorithmes ainsi que la complexité théorique des problèmes correspondants. Ainsi, il a été constaté que les propriétés de la logique CTL peuvent être vérifiées de manière efficace, alors que la vérification de propriétés exprimées dans LTL est beaucoup plus complexe.

En parallèle à ces questions théoriques, les algorithmes proposés ont été mis en pratique. Ainsi, des model checkers comme SMV (pour le temps arborescent), SPIN (pour le temps linéaire), ou UPPAAL, KRONOS et HYTECH pour des logiques *temporisées*, ont été développés, et leur efficacité n'est plus à démontrer, notamment dans les protocoles de communication ou les composants électroniques. Les méthodes de model checking sont maintenant appliquées dès la conception des systèmes, afin de débusquer les erreurs de conception au plus tôt.

Ainsi, les techniques de model checking ont permis de déceler, comprendre et résoudre de nombreux problèmes. Citons par exemple la découverte et la correction d'une erreur dans un protocole de contrôle audio/vidéo de Bang & Olufsen, grâce à l'outil UPPAAL [HSLL97] : cette erreur avait été constatée auparavant, mais il avait été impossible jusque là de trouver son origine et de la corriger. Mais ces techniques permettent surtout de nos jours de valider, dès leur conception, des protocoles de communication (par exemple, le protocole IEEE 1394 FireWire [SB01]), des protocoles cryptographiques [MS02] ou des circuits électroniques [Dre00].

Contenu de cette thèse

Ce que nous venons de voir concernant les différentes logiques temporelles pour la vérification peut se résumer en une question : quelle est la *meilleure* logique temporelle ? Pour tenter de répondre à cette question, il convient d'identifier les différents critères permettant de comparer les lgoiques entre elles. Parmi ces critères, citons :

- l'expressivité : ce critère assez général peut être vu sous différents aspects : par exemple, comme nous l'avons évoqué ci-dessus, les logiques LTL et CTL n'ont pas la même expressivité. Certaines propriétés peuvent être exprimées dans l'une et pas

dans l'autre. Le *pouvoir d'expression* de ces logiques est alors différent. Inversement, [GPSS80] a montré, en se basant sur un résultat de [Kam68], que les modalités du passé n'apportaient pas d'expressivité dans la logique LTL, et Dov M. Gabbay fournit un algorithme pour traduire effectivement une formule de LTL utilisant des modalités du passé en une formule de LTL n'utilisant que des modalités du futur [Gab89]. Les logiques LTL et LTL+Passé ont donc le même *pouvoir d'expression*. Un deuxième critère important est la « facilité » d'utilisation : si la vérification d'une formule sur un système peut être faite de manière exacte et exhaustive, la traduction d'une propriété (écrite dans le langage naturel) en une formule d'une logique temporelle nécessite une certaine habitude. Ce critère est difficilement quantifiable, mais il est possible cependant de quantifier la *concision* d'une logique : une propriété peut s'exprimer par une formule succincte dans une logique et par une formule beaucoup plus longue dans une autre logique. Par exemple, la logique CTL^+ [EH85], une extension de la logique CTL, a le même *pouvoir d'expression* que la logique CTL, mais permet d'exprimer certaines propriétés de manière plus *concise* que la logique CTL [Wil99b] ;
- la complexité des problèmes de vérification : le model checking est un processus coûteux, et l'efficacité des algorithmes est importante. Au début des années 1980, il a été prouvé que le model checking de la logique CTL pouvait être traité en temps polynomial, alors que le problème est PSPACE-complet pour LTL. Cela peut sembler être un gros avantage pour CTL. Il convient cependant de nuancer cette analyse, et affinant les critères : en effet, la complexité du model checking de LTL est en fait linéaire dans la taille du modèle (*i.e.* lorsque la formule est fixée et que le modèle varie), et PSPACE-complet dans la taille de la formule (*i.e.* lorsque le modèle est fixé). Ainsi, le fait que le model checking de LTL soit PSPACE-complet (ce qui est souvent considéré comme étant « dissuasif ») doit être nuancé par le fait qu'il est linéaire dans la taille (souvent grande) du modèle, et PSPACE dans la taille (souvent assez petite) de la formule.

La question posée au début de cette partie n'a probablement pas de réponse définitive. Mais il est cependant important de connaître les avantages et inconvénients des différentes logiques, afin de trouver le compromis entre expressivité et complexité.

Par ailleurs, cette question a déjà été beaucoup étudiée par le passé, et le sujet a assez peu évolué au cours des dix dernières années. Il reste cependant de nombreuses questions ouvertes. Cette thèse résout deux problèmes importants dans le domaine :

- la complexité du model checking de l'extension de CTL avec des conditions d'équité (FCTL) n'était pas connue : seules deux bornes inférieures (NP et co-NP) étaient connues. Le problème appartient en fait à une classe intermédiaire entre NP et PSPACE. Nous développons dans cette thèse une technique permettant de caractériser les problèmes complets pour cette classe. Nous appliquons par la suite cette technique à d'autres logiques ;
- le coût de la traduction de la logique temporelle du temps linéaire avec passé dans la logique sans passé n'est pas connu. La meilleure traduction connue semble être non élémentaire. Par ailleurs, aucune borne inférieure n'était connue. Nous allons montrer qu'il existe une borne inférieure exponentielle, et une borne supérieure triplement exponentielle.

Plan de la thèse

Nous allons étudier dans la suite les questions d'expressivité et de complexité de quatre points de vue différents, c'est-à-dire pour quatre « familles » de logiques temporelles différentes.

Le premier chapitre traitera de la logique temporelle du temps linéaire avec passé. Comme mentionné ci-dessus, les modalités du passé n'apportent pas d'expressivité dans la logique temporelle du temps linéaire, et de nombreuses études n'ont porté que sur le fragment « pur futur » de la logique temporelle du temps linéaire, considérant que les modalités du passé ne pouvaient que compliquer la phase de vérification. Nous allons prendre le contre-pied de ce raisonnement dans ce chapitre, puisque, l'expressivité étant la même, la vérification n'est pas plus complexe, et la spécification est plus proche de la langue naturelle, et surtout plus concise.

Dans le deuxième chapitre, nous étudierons toujours le cas de la logique temporelle du temps linéaire avec passé, à laquelle nous adjoignons un nouvel opérateur permettant, à tout moment, d'oublier le passé, c'est à dire de ne plus pouvoir faire référence à ce qui s'est produit avant un certain instant. Nous nous intéresserons à l'expressivité, à la concision et à la complexité de la logique ainsi obtenue.

Le troisième chapitre s'intéressera à différentes logiques temporelles du temps arborescent. Alors que la complexité de la logique CTL a été précisément établie depuis son introduction, au début des années 1980, la complexité de la vérification pour de nombreuses extensions naturelles de cette logique n'a pas été précisément établie. Nous verrons que cela tient au fait que ces problèmes sont complets pour une classe de complexité assez peu connue, et pour laquelle peu de problèmes « naturels » sont complets.

Le quatrième chapitre s'intéressera enfin à une version « quantitative » de la logique temporelle. De nombreuses approches ont été proposées et étudiées pour faire de la vérification de propriétés « quantitatives ». L'approche que nous étudirons ici est une extension du modèle des structures de Kripke permettant d'assigner un « coût », une « longueur », aux transitions de la structure de Kripke. La logique temporelle que nous utiliserons permet alors de quantifier sur la longueur des exécutions menant d'un état du système à un autre.

CHAPITRE 1

La logique temporelle du temps linéaire avec passé

Introduction

La logique temporelle du temps linéaire permet d'exprimer des propriétés sur l'ordre d'apparition de certains évènements au cours d'une exécution d'un système. Elle utilise pour ce faire des modalités, afin de parler d'un ou plusieurs moments de l'exécution. Par exemple, la modalité $\mathbf{F}$ permet de spécifier qu'une propriété sera vérifiée à (au moins) un moment dans le futur pour l'exécution considérée. Symétriquement, la modalité $\mathbf{F}^{-1}$ permet d'exprimer qu'une propriété a été satisfaite (au moins) une fois dans le passé.

Cette logique temporelle a été introduite par Amir Pnueli dans le domaine de la vérification [Pnu77], parce qu'elle permet de spécifier des propriétés sur des modèles finis de systèmes réactifs. Afin de vérifier formellement ces propriétés, plusieurs algorithmes ont été développés (*cf.* [WVS83, SC85, LPZ85, VW94, Var95]). Les algorithmes proposés nécessitent au minimum un espace de taille polynomial dans la taille des données (le modèle et la formule à vérifier), et [SC85] a prouvé que le problème de la vérification d'une formule de la logique temporelle du temps linéaire est PSPACE-complet.

Parmi les algorithmes dont il est question ci-dessus, certains ne prennent pas en compte les modalités relatives au passé de l'état courant. En effet, de deux résultats de [Kam68] et [GPSS80] découle le fait que la logique du temps linéaire sans les modalités du passé a le même pouvoir d'expression que la logique du temps linéaire « avec passé ». [GPSS80] et [Gab89] fournissent des algorithmes permettant de transformer une formule utilisant des modalités du passé en une formule équivalente n'utilisant que des modalités du futur. Pour cette raison, et dans un souci de « minimalité », les outils de vérification qui ont été développés pour la logique du temps linéaire (SPIN [VW86, Hol97, Hol91], SMV [McM93b, McM93a], nuSMV [CCGR99, CCG+02] ...) n'intègrent pas les modalités du passé.

Dans ce chapitre, nous allons voir que le passé a néanmoins de nombreux avantages.

Dans un premier temps, nous allons formaliser et définir toutes les notions dont nous parlons ici de manière précise. Nous verrons ensuite que l'utilisation des modalités du passé facilite grandement la spécification de certaines propriétés. Cette notion de « facilité » est difficilement quantifiable de manière formelle, mais elle se formalise en partie par la notion de « concision » : nous verrons que certaines propriétés peuvent s'exprimer de manière plus concise en utilisant les modalités du passé, et que toute formule équivalente écrite sans utiliser les modalités du passé est nécessairement « beaucoup » plus longue. Nous verrons enfin, dans une troisième partie, que l'ajout du passé n'augmente pas la complexité de la vérification, en étudiant selon plusieurs angles d'attaque la complexité de la vérification de propriétés de la logique du temps linéaire.

1.1 Présentation de la logique du temps linéaire avec passé

1.1.1 Quelques définitions

Ces premières définitions introduisent la logique PLTL (logique du temps linéaire avec passé). Nous en donnerons dans un premier temps la syntaxe et la sémantique. Puis, en guise d'exemples et de premières manipulations, nous définirons de nouvelles modalités et en rechercherons la sémantique. À partir de ces modalités, nous verrons comment définir des logiques « plus petites » que PLTL.

Dans tout ce qui suit, nous adoptons le point de vue de la spécification et de la vérification. D'autres présentations générales, avec parfois des points de vue différents, peuvent être trouvées dans [Gab89, Eme90, Sti92, MP95, SBB+99, CGP99] par exemple.

1.1.1.1 Syntaxe de PLTL

Dans toute la suite, Prop désigne un ensemble fixé, non vide et dénombrable, de propositions atomiques. Ces propositions atomiques représentent les « évènements », les « états », qui se rapportent au système que nous étudions.

Définition 1.1
La logique du temps linéaire avec passé *est définie par la syntaxe suivante :*

$$\mathsf{PLTL} \ni \varphi, \psi \ ::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\psi \mid \mathbf{X}^{-1}\,\varphi \mid \varphi\,\mathbf{S}\,\psi \mid p \mid q \mid \ \ldots$$

où p, q, ... sont des éléments de Prop.

Les opérateurs $\mathbf{X}$, $\mathbf{U}$, $\mathbf{X}^{-1}$, $\mathbf{S}$ s'appellent des *modalités*. Sans entrer dans les détails de la sémantique pour le moment, précisons que $\varphi\,\mathbf{U}\,\psi$ se lit « φ *until* ψ » et indique que φ sera vrai jusqu'à ce que ψ le soit. Symétriquement, $\varphi\,\mathbf{S}\,\psi$ se lit « φ *since* ψ » et indique que φ est vraie dans le passé depuis un moment où ψ était satisfaite. La même symétrie se retrouve entre $\mathbf{X}\,\varphi$ et $\mathbf{X}^{-1}\,\varphi$, le premier se lisant « *next* φ » et signifiant que φ sera vraie dans l'état suivant, le second se lisant « *previous* φ » et signifiant que φ était satisfaite dans l'état précédent.

Les modalités $\mathbf{X}$ et $\mathbf{U}$ sont des *modalités du futur*, alors que $\mathbf{X}^{-1}$ et $\mathbf{S}$ sont des *modalités du passé*. La logique du temps linéaire (sans passé), notée LTL, est définie par la même syntaxe, mais sans les modalités du passé.

La syntaxe proposée dans la définition 1.1 est la définition syntaxique habituelle de PLTL, mais dans la pratique, plusieurs autres opérateurs et modalités, définissables à partir des opérateurs et modalités de PLTL, sont utilisés : par exemple, cette syntaxe autorise l'utilisation de l'opérateur booléen $\vee$ mais pas de l'opérateur $\wedge$. Cependant, dans la suite, nous utiliserons l'opérateur $\wedge$, car il peut se définir comme suit :

$$\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\psi \vee \neg\varphi)$$

De même, nous aurons besoin des « propositions » $\top$ et $\bot$, qui sont respectivement toujours vraie et toujours fausse. Nous pouvons les définir de la façon suivante :

$$\top \stackrel{\text{def}}{=} p \vee \neg p \qquad\qquad \bot \stackrel{\text{def}}{=} \neg\top$$

Nous pouvons également définir de « nouvelles »[1] modalités, comme par exemple :

$$\mathbf{F}\,\varphi \stackrel{\text{def}}{=} \top\,\mathbf{U}\,\varphi \qquad\qquad \mathbf{G}\,\varphi \stackrel{\text{def}}{=} \neg\mathbf{F}\,\neg\varphi.$$

Intuitivement, $\mathbf{F}\,\varphi$ signifie que φ sera satisfaite un jour dans le futur, et n'impose pas de condition intermédiaire (puisque $\top$ est toujours vraie). La formule $\mathbf{F}\,\varphi$ se lit « *eventually* φ ». Alors $\neg\mathbf{F}\,\neg\varphi$ signifie qu'il n'est pas vrai que la négation de φ sera satisfaite dans le futur, *i.e.* que φ sera toujours vraie dans le futur. La formule $\mathbf{G}\,\varphi$ se lit « *always* φ ».

Définition 1.2
La taille d'une formule *de logique temporelle est le nombre total de modalités, d'opérateurs et de propositions atomiques nécessaires à écrire cette formule.*

Il est évident que la taille d'une formule dépend des modalités que la logique autorise. Par exemple, la formule $\mathbf{F}\,a$ est de taille 2, mais à proprement parler, cette formule n'est pas dans PLTL. Une formule équivalente dans PLTL est $\top\,\mathbf{U}\,a$, qui est de taille 3. Cependant, ces différences de taille sont « faibles », *i.e.* sont polynomiales. Dans la suite, la taille des formules interviendra dans deux situtions :

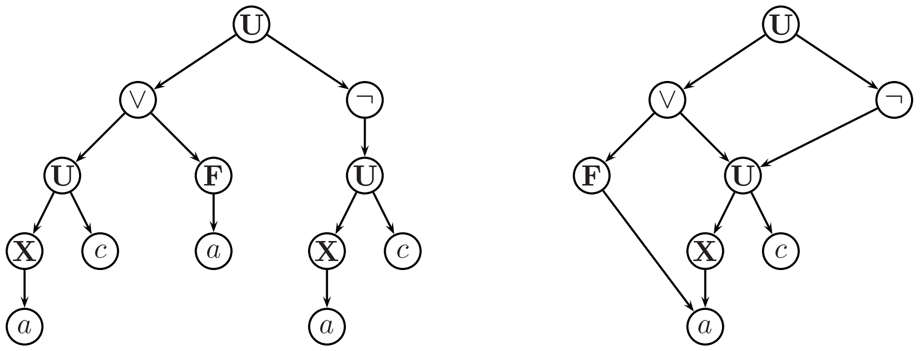
- lors des différentes équivalences que nous établirons entre les logiques : en effet, pour certaines logiques, il existe des méthodes de traduction dans d'autres logiques, mais la taille de la formule qui en résulte est parfois beaucoup plus grande ;
- dans la complexité des algorithmes de vérification que nous écrirons : le temps (ou l'espace) de calcul nécessaire aux algorithmes de vérification dépendent directement de la taille de la formule de départ (et également de la taille de modèle que nous vérifions).

Dans les deux cas, les « mesures » sont faites « à un polynôme près », et il n'est donc pas gênant de considérer que $\mathbf{F}\,a$ et $\top\,\mathbf{U}\,a$ sont « de même taille ».

Par ailleurs, il est possible de représenter les formules par des « arbres », comme sur la partie gauche de la figure 1.1. Il est également possible de les représenter sous forme de « graphe orienté acyclique », comme sur la partie droite de la figure 1.1. Le graphe orienté acyclique s'obtient à partir de l'arbre en « factorisant » les sous-formules (*i.e.* les

[1] Ces modalités ne sont pas vraiment « nouvelles » : historiquement, dans la définition de Prior, la logique du temps linéaire n'autorisait que ces modalités $\mathbf{F}$, $\mathbf{G}$, $\mathbf{F}^{-1}$ et $\mathbf{G}^{-1}$. Les modalités $\mathbf{U}$ et $\mathbf{S}$ ont été introduites dans [Kam68].

Figure 1.1 Représentations de la formule $(((\mathbf{X}\,a)\,\mathbf{U}\,b) \vee \mathbf{F}\,a)\,\mathbf{U}\,(\neg((\mathbf{X}\,a)\,\mathbf{U}\,b))$



sous-arbres) identiques. Le nombre de sommets de ce graphe peut alors être très inférieure au nombre de sommets de l'arbre, qui est égal à la taille de la formule.

Une autre notion qui nous servira beaucoup par la suite est la *hauteur temporelle* d'une formule, qui représente le nombre maximal d'emboîtements des modalités dans une formule.

Définition 1.3
Soit φ une formule de PLTL, *et H un ensemble de modalités. La* hauteur temporelle de φ par rapport aux modalités de H, *notée $h_H(\varphi)$, est définie récursivement par les formules suivantes :*

$$
\begin{aligned}
h_H(p) &= 0 && \textit{pour toute proposition } p \in \mathrm{Prop} \\
h_H(\varphi_1 \vee \varphi_2) &= \max(h_H(\varphi_1), h_H(\varphi_2)) && \\
h_H(\neg\psi) &= h_H(\psi) && \\
h_H(\mathbf{M}(\varphi_1, \ldots, \varphi_a)) &= 1 + \max(h_H(\psi_1), \ldots, h_H(\psi_a)) && \textit{si } \mathbf{M} \in H \\
h_H(\mathbf{M}'(\varphi_1, \ldots, \varphi_b)) &= \max(h_H(\psi_1), \ldots, h_H(\psi_b)) && \textit{si } \mathbf{M}' \notin H
\end{aligned}
$$

Généralement, H sera soit l'ensemble des modalités du passé ou du futur, soit l'ensemble des modalités de la logique considérée. Nous donnons donc des noms à ces cas particuliers :

Définition 1.4
La hauteur temporelle du futur *d'une formule φ est la hauteur temporelle de φ par rapport à l'ensemble des modalités du futur de cette formule. Nous noterons $F = \{\,\mathbf{U}\,, \mathbf{X}\,\}$, et $h_F(\varphi)$ la hauteur temporelle du futur d'une formule φ de* PLTL *écrite en utilisant uniquement les modalités $\mathbf{U}$, $\mathbf{X}$, $\mathbf{S}$ et $\mathbf{X}^{-1}$.*

La hauteur temporelle du passé *d'une formule φ est la hauteur temporelle de φ par rapport à l'ensemble des modalités du passé de cette formule. Nous noterons, par la suite, $P = \{\,\mathbf{S}\,, \mathbf{X}^{-1}\,\}$ l'ensemble des modalités du passé. La hauteur temporelle du passé d'une formule φ est donc notée $h_P(\varphi)$.*

La hauteur temporelle *d'une formule φ est la hauteur temporelle de φ par rapport à l'ensemble des modalités de la logique considérée. Nous noterons $h(\varphi)$ la hauteur temporelle d'une formule φ.*

On définit en particulier les notions de formules « pur passé » et « pur futur » :

Définition 1.5
Une formule de PLTL *est dite* « pur futur » *si sa hauteur temporelle du passé est nulle, et* « pur passé » *si sa hauteur temporelle du futur est nulle.*

1.1.1.2 Sémantique de PLTL

Les formules de la logique du temps linéaires expriment des propriétés sur *une exécution* d'un (modèle d'un) système réactif. Une telle exécution est représentée par la suite des états dans lesquels se trouve ce système au cours du temps. Une telle suite est appelée « chemin ».

Définition 1.6
Un chemin *π est une suite d'éléments de $\{\top, \bot\}^{\mathrm{Prop}}$, indexée par $\mathbb{N}$ si le chemin est infini, et par $[\![0, n]\!]$ si le chemin est fini. L'ensemble des indices est appelé le* domaine *du chemin π, et noté* $\mathrm{dom}(\pi)$*. La longueur d'un chemin π, notée* $\mathrm{long}(\pi)$*, est le cardinal de son domaine.*

Un état *de π est un élément de cette suite. C'est donc un élément de $\{\top, \bot\}^{\mathrm{Prop}}$,* i.e. *une valuation des propositions atomiques. L'*état initial *d'un chemin est le premier état de la suite.*

Par ailleurs, étant donné un chemin π et un entier i strictement inférieur à la longueur du chemin, le i-ème suffixe de π, noté π^i, est le chemin obtenu à partir de π en supprimant les i premiers états. Plus précisément, pour tout entier j, le j-ème état de π^i est le $(j+i)$-ème état de π : $\pi^i_j = \pi_{i+j}$.

Sauf mention explicite du contraire, un chemin sera supposé infini dans toute la suite.

Nous pouvons maintenant définir la sémantique de PLTL. Cette définition est valable aussi bien pour les chemins finis que pour les chemins infinis.

Définition 1.7
Soit π un chemin, fini ou infini, i un entier naturel strictement inférieur à la longueur du chemin, φ et ψ deux formules de PLTL*, et p une proposition atomique de* Prop*. On définit la relation $\models$ de manière inductive, à partir des règles suivantes :*

$$\begin{array}{lll}
\pi, i \models \neg\varphi & \textit{si, et seulement si,} & \pi, i \not\models \varphi \\
\pi, i \models \varphi \vee \psi & \textit{si, et seulement si,} & \pi, i \models \varphi \textit{ ou } \pi, i \models \psi \\
\pi, i \models \mathbf{X}\,\varphi & \textit{si, et seulement si,} & i+1 < \mathrm{long}(\pi) \ \wedge\ \pi, i+1 \models \varphi \\
\pi, i \models \varphi\,\mathbf{U}\,\psi & \textit{si, et seulement si,} & \exists k.\ ((i \leqslant k < \mathrm{long}(\pi)) \ \wedge\ (\pi, k \models \psi) \ \wedge \\
 & & \qquad\qquad (\forall i \leqslant j < k.\ \pi, j \models \varphi)) \\
\pi, i \models \mathbf{X}^{-1}\,\varphi & \textit{si, et seulement si,} & i \geqslant 1 \textit{ et } \pi, i-1 \models \varphi
\end{array}$$

$$\begin{array}{lll} \pi, i \models \varphi \,\mathbf{S}\, \psi & \textit{si, et seulement si,} & \exists k \leqslant i.\ ((\pi, k \models \psi)\ \wedge\ (\forall i \geqslant j > k.\ \pi, j \models \varphi)) \\ \pi, i \models p & \textit{si, et seulement si,} & \pi_i(p) = \top \qquad\qquad\qquad\qquad (\forall p \in \mathrm{Prop}) \end{array}$$

Lorsque $\pi, i \models \varphi$ est vraie, nous dirons que la formule φ est vraie, ou est satisfaite, *à la position i le long du chemin π.*

Bien entendu, ces définitions donnent également la sémantique de LTL. On peut voir dans cette sémantique en quoi les modalités $\mathbf{U}$ et $\mathbf{X}$ sont des modalités « du futur » : elles expriment des propriétés sur des positions postérieures à l'état courant. Le résultat suivant, dont nous nous servirons à plusieurs reprises par la suite, exprime le fait que LTL est une logique « du futur », contrairement à PLTL.

Proposition 1.8
Soient π un chemin, i un entier naturel, et φ une formule de LTL*. On a alors l'équivalence suivante :*

$$\pi, i \models \varphi \textit{ si, et seulement si, } \pi^i, 0 \models \varphi$$

Démonstration :

Ce résultat se montre par induction sur la structure de la formule φ :

- si φ est une proposition atomique p de Prop, on a

$$\pi^i, 0 \models p \ \Leftrightarrow\ \pi^i_0(p) = \top \ \Leftrightarrow\ \pi_{i+0}(p) = \top \ \Leftrightarrow\ \pi, i \models p\,;$$

- si $\varphi = \varphi_1 \vee \varphi_2$ ou si $\varphi = \varphi_1 \vee \varphi_2$, le résultat est immédiat puisqu'il est vrai, par hypothèse d'induction, pour φ_1 et φ_2 ;
- si $\varphi = \mathbf{X}\,\varphi_1$, on a les équivalence suivantes :

$$\begin{aligned} \pi^i, 0 \models \mathbf{X}\,\varphi_1 &\Leftrightarrow \pi^i_1 \text{ existe et } \pi^i, 1 \models \varphi_1 \\ &\Leftrightarrow \pi_{i+1} \text{ existe et } \pi^{i+1}, 0 \models \varphi_1 \quad \text{(d'après l'hypothèse d'induction)} \\ &\Leftrightarrow \pi_{i+1} \text{ existe et } \pi, i+1 \models \varphi_1 \\ &\Leftrightarrow \pi, i \models \mathbf{X}\,\varphi_1\,; \end{aligned}$$

- si $\varphi = \varphi_1 \,\mathbf{U}\, \varphi_2$, alors

$$\pi^i, 0 \models \varphi_1 \,\mathbf{U}\, \varphi_2 \ \Leftrightarrow\ \exists k \geqslant 0.\ ((\pi^i, k \models \varphi_2) \wedge \forall 0 \leqslant j < k.\ (\pi^i, j \models \varphi_1)).$$

  L'hypothèse d'induction, appliquée aux deux sous-formules φ_1 et φ_2, nous donne alors

$$\begin{aligned} \pi^i, 0 \models \varphi_1 \,\mathbf{U}\, \varphi_2 &\Leftrightarrow \exists k \geqslant 0.\ ((\pi, i+k \models \varphi_2) \wedge \forall 0 \leqslant j < k.\ (\pi, i+j \models \varphi_1)) \\ &\Leftrightarrow \exists k' \geqslant i.\ ((\pi, k' \models \varphi_2) \wedge \forall i \leqslant j' < k'.\ (\pi, j' \models \varphi_1)) \\ &\Leftrightarrow \pi, i \models \varphi_1 \,\mathbf{U}\, \varphi_2 \end{aligned}$$

□

Par la suite, on notera $\pi \models \varphi$ au lieu de $\pi, 0 \models \varphi$. Cette notion de satisfaisabilité *initiale* d'une formule est primordiale dans le domaine de la vérification : c'est cette propriété que nous voudrons vérifier pour les exécutions du système sur lequel nous travaillons. La notion de satisfaisabilité *globale* d'une formule est rendue nécessaire par l'utilisation de modalités du passé.

1.1.1.3 Définition de nouveaux opérateurs

La syntaxe de PLTL, telle qu'elle est définie à la définition 1.1, ne contient pas certains opérateurs qui nous seront cependant bien utiles par la suite. Nous allons définir ici, à partir des modalités de PLTL, de nouvelles modalités, et expliquer leur signification.

Soit p une proposition atomique de Prop, φ et ψ deux formules quelconques de PLTL. Nous définissons les opérateurs et modalités suivantes :

– les formules $\top \stackrel{\text{def}}{=} p \vee \neg p$ et $\bot \stackrel{\text{def}}{=} \neg\top$, et les opérateurs booléens $\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)$, $\psi \Rightarrow \varphi \stackrel{\text{def}}{=} (\neg\psi) \vee \varphi$ et $\varphi \Leftrightarrow \psi \stackrel{\text{def}}{=} (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$ ont leur signification « classique » ;

– $\mathbf{F}\,\varphi \stackrel{\text{def}}{=} \top\,\mathbf{U}\,\varphi$: on a donc

$$\begin{aligned} \pi, i \models \mathbf{F}\,\varphi \quad &\Leftrightarrow \quad \exists k \geqslant i.\ ((\pi, k \models \varphi)\ \wedge\ (\forall i \leqslant j < k.\ \pi, j \models \top)) \\ &\Leftrightarrow \quad \exists k \geqslant i.\ (\pi, k \models \varphi) \end{aligned}$$

puisque pour tout $0 \leqslant j \leqslant k$, la condition $\pi, j \models \top$ est toujours satisfaite. $\mathbf{F}\,\varphi$ signifie donc que la formule φ sera satisfaite un jour dans le futur. La modalité $\mathbf{F}$ se lit « *eventually* » ;

– $\mathbf{G}\,\varphi \stackrel{\text{def}}{=} \neg\mathbf{F}\,\neg\varphi$:

$$\begin{aligned} \pi, i \models \mathbf{G}\,\varphi \quad &\Leftrightarrow \quad \neg(\exists k \geqslant i.\ (\pi, k \models \neg\varphi) \\ &\Leftrightarrow \quad \forall k \geqslant i.\ (\pi, k \models \varphi) \end{aligned}$$

Donc, $\mathbf{G}\,\varphi$ signifie que φ est satisfaite dans tous les états postérieurs à l'état courant. La modalité $\mathbf{G}$ se lit « *always* » ;

– $\mathbf{F}^{-1}\,\varphi \stackrel{\text{def}}{=} \top\,\mathbf{S}\,\varphi$: de la même manière que pour $\mathbf{F}$, on a :

$$\pi, i \models \mathbf{F}^{-1}\,\varphi \quad \Leftrightarrow \quad \exists k \leqslant i.\ (\pi, k \models \varphi).$$

$\mathbf{F}^{-1}\,\varphi$ signifie donc qu'il existe une position, entre le début du chemin et l'état courant, où φ est vérifiée. $\mathbf{F}^{-1}$ se lit « *eventually in the past* » ;

– $\mathbf{G}^{-1}\,\varphi \stackrel{\text{def}}{=} \neg\mathbf{F}^{-1}\,\neg\varphi$:

$$\pi, i \models \mathbf{G}^{-1}\,\varphi \quad \Leftrightarrow \quad \forall k \leqslant i.\ (\pi, k \models \varphi)\,;$$

$\mathbf{G}^{-1}\,\varphi$ exprime donc que φ est satisfaite dans tous les états précédant l'état courant. Cette modalité se lit « *always in the past* » ;

– $\overset{\infty}{\mathbf{F}}\,\varphi \stackrel{\text{def}}{=} \mathbf{G}\,\mathbf{F}\,\varphi$: la sémantique est la suivante :

$$\begin{aligned} \pi, i \models \overset{\infty}{\mathbf{F}}\,\varphi \quad &\Leftrightarrow \quad \forall j \geqslant i.\ (\pi, j \models \mathbf{F}\,\varphi) \\ &\Leftrightarrow \quad \forall j \geqslant i.\ (\exists k \geqslant j.\ (\pi, k \models \varphi)) \end{aligned}$$

Le long d'un chemin infini, cela signifie que φ doit être satisfaite une infinité de fois. Cette modalité se lit alors « *infinitely often* ». Le long d'un chemin fini, cette modalité signifie que le dernier état du chemin doit satisfaire φ, et on la lit « *finally* » ;

– $\overset{\infty}{\mathbf{G}}\,\varphi \stackrel{\text{def}}{=} \neg\overset{\infty}{\mathbf{F}}\,\neg\varphi$: cette modalité a la signification suivante :

$$\begin{aligned} \pi, i \models \overset{\infty}{\mathbf{G}}\,\varphi \quad &\Leftrightarrow \quad \neg(\forall j \geqslant i.\ (\exists k \geqslant j.\ (\pi, k \models \neg\varphi))) \\ &\Leftrightarrow \quad \exists j \geqslant i.\ (\forall k \geqslant j.\ (\pi, k \models \varphi)) \end{aligned}$$

Cela signifie donc que φ doit être toujours vraie à partir d'un état dans le futur, si le chemin est infini, ou que φ doit être vraie dans le dernier état, si le chemin est fini. Dans le premier cas, on la lit « *almost always* ». Dans le deuxième cas, elle est équivalente à la modalité $\overset{\infty}{\mathbf{F}}$, et on la lira également « *finally* » ;

– $\mathbf{I}\,\varphi \stackrel{\text{def}}{=} \mathbf{G}^{-1}\,\mathbf{F}^{-1}\,\varphi$: cette modalité a la sémantique suivante :

$$\pi, i \models \mathbf{I}\,\varphi \quad \Leftrightarrow \quad \forall j \leqslant i.\ (\exists k \leqslant j.\ (\pi, k \models \varphi))$$

Puisque le passé est fini, on en déduit l'équivalence suivante :

$$\pi, i \models \mathbf{I}\,\varphi \quad \Leftrightarrow \quad \pi, 0 \models \varphi$$

On en déduit immédiatement que $\mathbf{F}^{-1}\,\mathbf{G}^{-1}\,\varphi \stackrel{\text{def}}{=} \neg\mathbf{G}^{-1}\,\mathbf{F}^{-1}\,\neg\varphi$ a exactement la même sémantique. Ces deux modalités se lisent donc « *initially* » ;

– $\varphi\,\mathbf{W}\,\psi \stackrel{\text{def}}{=} \neg(\neg\psi\,\mathbf{U}\,(\neg\varphi \wedge \neg\psi))$: on a :

$$\begin{aligned} \pi, i \models \varphi\,\mathbf{W}\,\psi \quad &\Leftrightarrow \quad \neg(\exists k \geqslant i.\ ((\pi, k \models \neg\varphi \wedge \neg\psi)\ \wedge\ (\forall i \leqslant j < k.\ \pi, j \models \neg\psi)) \\ &\Leftrightarrow \quad \forall k \geqslant i.\ ((\pi, k \models \varphi \vee \psi)\ \vee\ (\exists i \leqslant j < k.\ \pi, j \models \psi)) \\ &\Leftrightarrow \quad (\forall k \geqslant i.\ \pi, k \models \varphi) \vee \\ &\qquad\qquad (\exists k \geqslant i.\ ((\pi, k \models \psi)\ \wedge\ (\forall i \leqslant j < k.\ (\pi, j \models \varphi)) \\ &\Leftrightarrow \quad \pi, i \models (\mathbf{G}\,\varphi)\ \vee\ (\varphi\,\mathbf{U}\,\psi) \end{aligned}$$

Cet opérateur est une version faible du « *until* », et est appelé, comme il se doit, « *weak until* », ou parfois « *waiting for* » ;

– $\varphi\,\mathbf{B}\,\psi \stackrel{\text{def}}{=} \neg(\neg\psi\,\mathbf{S}\,(\neg\varphi \wedge \neg\psi))$: les mêmes manipulations que pour la modalité $\mathbf{W}$ donnent la sémantique suivante :

$$\pi, i \models \varphi\,\mathbf{B}\,\psi \quad \Leftrightarrow \quad \pi, i \models (\mathbf{G}^{-1}\,\varphi)\ \vee\ (\varphi\,\mathbf{S}\,\psi)$$

Cette modalité est souvent appelée « *back to* », ou parfois « *weak since* ».

– $\varphi \, \mathbf{U}_s \, \psi \stackrel{\text{def}}{=} \mathbf{X} \, (\varphi \, \mathbf{U} \, \psi)$: c'est une version *stricte* de la modalité « *until* » :

$$\pi, i \models \varphi \, \mathbf{U}_s \, \psi \quad \Leftrightarrow \quad \exists k > i. \ ((\pi, k \models \psi) \ \wedge \ (\forall i < j < k. \ \pi, j \models \varphi))$$

Cette modalité s'appelle « *strict until* ». Elle permet, à elle seule, de définir le « *next* » ($\mathbf{X} \, \varphi \equiv \bot \, \mathbf{U}_s \, \varphi$) et le « *until* » ($\varphi \, \mathbf{U} \, \psi \equiv \psi \vee (\varphi \wedge \varphi \, \mathbf{U}_s \, \psi)$) ;

– $\varphi \, \mathbf{S}_s \, \psi \stackrel{\text{def}}{=} \mathbf{X}^{-1} \, (\varphi \, \mathbf{S} \, \psi)$: tout comme on a définit le « *until strict* », on peut définir le « *since strict* » :

$$\pi, i \models \varphi \, \mathbf{S}_s \, \psi \quad \Leftrightarrow \quad \exists k < i. \ ((\pi, k \models \psi) \ \wedge \ (\forall k < j < i. \ \pi, j \models \varphi))$$

Cette modalité est le « *strict since* ».

Toutes les modalités que nous venons de définir peuvent être classées en deux catégories, comme les modalités utilisées dans la définition syntaxique de PLTL : les modalités du passé ($\mathbf{S}$, $\mathbf{X}^{-1}$ et les modalités définies à partir de ces deux modalités, comme $\mathbf{I}$) et les modalités du futur ($\mathbf{U}$, $\mathbf{X}$ et les modalités définies à partir de ces modalités, comme $\overset{\infty}{\mathbf{F}}$, $\overset{\infty}{\mathbf{G}}$, ...) Il est bien entendu possible de définir de nouvelles modalités combinant passé et futur, mais ce type de modalités est très peu utile en pratique.

1.1.1.4 Définition de nouveaux fragments

Ces « abréviations » que nous venons de définir peuvent être utilisées « directement » dans l'écriture de formules de PLTL. Par ailleurs, on peut définir des sous-ensembles de PLTL dans lesquels seules quelques modalités sont autorisées. Nous allons définir ici trois restrictions (syntaxiques) de PLTL. Dans la suite du chapitre, nous étudierons l'influence de l'ajout du passé dans ces fragments, du point de vue de l'expressivité et de la vérification.

Fragments simples de PLTL. La première restriction porte simplement sur l'ensemble des modalités autorisées :

Définition 1.9
Soient $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ *des modalités d'arités respectives* $a_1, a_2, \ldots, a_n$. *Nous définissons le fragment* $\mathrm{L}(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)$ *par la syntaxe suivante :*

$$\mathrm{L}(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n) \ni \varphi_1, \ldots, \varphi_r \ ::= \ \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{M}_i(\varphi_1, \ldots, \varphi_{a_i}) \mid p \mid q \mid \ \ldots$$

Ainsi, par exemple, PLTL est égal à $\mathrm{L}(\mathbf{U}, \mathbf{X}, \mathbf{S}, \mathbf{X}^{-1})$, et LTL est le fragment $\mathrm{L}(\mathbf{U}, \mathbf{X})$. Nous pouvons également définir, par exemple, le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$, dans lequel seules les modalités $\mathbf{F}$ et $\mathbf{F}^{-1}$ peuvent être utilisées (et bien entendu, en les combinant avec la négation, $\mathbf{G}$, $\mathbf{G}^{-1}$, $\overset{\infty}{\mathbf{F}}$, $\overset{\infty}{\mathbf{G}}$, $\mathbf{I}$, ...)

Fragments positifs. Sur la base de ce dernier exemple, nous pourrions vouloir définir un fragment dans lequel $\mathbf{F}$ serait autorisée, mais dans lequel $\mathbf{G}$ serait interdite.

Définition 1.10
Soient $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m$ *des modalités d'arités respectives* $a_1, a_2, \ldots, a_m$. *Nous définissons la logique* $\mathrm{L}^+(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m)$ *par la syntaxe suivante :*

$$\mathrm{L}^+(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m) \ni \varphi_1, \ldots, \varphi_r \ ::= \ \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{M}_i(\varphi_1, \ldots, \varphi_{a_i}) \mid p \mid q \mid \ \ldots$$

Par exemple, le fragment $\mathrm{L}^+(\mathbf{F}, \mathbf{F}^{-1})$ est le fragment dans lequel les modalités $\mathbf{F}$ et $\mathbf{F}^{-1}$ sont autorisées, mais les modalités $\mathbf{G}$ et $\mathbf{G}^{-1}$ ne le sont pas.

Fragment stratifié [MP91]. La troisième restriction que nous proposons se rapporte à l'emboîtement des modalités. Dans un fragment stratifié, nous interdisons l'utilisation de modalités du futur (les modalités $\mathbf{U}$ et $\mathbf{X}$, ainsi que celles qui sont définies à partir de ces deux-là) dans la portée de modalités du passé ($\mathbf{S}$, $\mathbf{X}^{-1}$, et celles qui en découlent). Plus précisément :

Définition 1.11
Soient $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m$ *des modalités du futur, d'arités respectives* $a_1, a_2, \ldots, a_m$*, et soient* $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_n$ *des modalités du passé d'arités respectives* $b_1, b_2, \ldots, b_n$*. La logique* $\mathrm{L}_s(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m, \mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_n)$ *est composée des formules définies par la syntaxe suivante :*

$$\begin{aligned} \psi_1, \ldots, \psi_r &::= \neg\psi_1 \mid \psi_1 \vee \psi_2 \mid \mathbf{N}_i(\psi_1, \ldots, \psi_{b_i}) \mid p \mid q \mid \ldots \\ \varphi_1, \ldots, \varphi_r &::= \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{M}_i(\varphi_1, \ldots, \varphi_{a_i}) \mid \psi_1 \end{aligned}$$

Intuitivement, cela signifie qu'on ne peut pas mettre une modalité du futur dans la portée d'une modalité du passé. Par exemple, dans $\mathrm{L}_s(\mathbf{F}, \mathbf{F}^{-1})$, on ne peut pas écrire $\mathbf{G}^{-1}(a \vee \mathbf{F}\, b)$. La stratification n'est pas une « grosse » restriction, puisqu'il est prouvé dans [MP90b] que toute formule de PLTL peut être exprimée dans $\mathrm{L}_s(\mathbf{F}, \mathbf{X}^{-1}, \mathbf{S})$.

La sémantique de ces fragments découle bien entendu de celle de PLTL. Les fragments $\mathrm{L}_s^+(\cdot)$ sont définis de manière évidente, en combinant les restrictions de « positivité » et de « stratification ».

1.1.2 La notion d'expressivité

Parmi toutes les modalités que nous avons définies à la section 1.1.1.3, nous avons vu que certaines pouvaient se définir de différentes manières à partir d'autres modalités. Le but de cette section est de définir formellement cette notion d'*équivalence* entre formules.

1.1.2.1 L'équivalence globale

Intuitivement, le fait que deux formules de PLTL soient équivalentes indique qu'elles ont la même « signification ». En d'autres termes, qu'elles sont satisfaites simultanément, pour tout chemin et à toute position.

Définition 1.12
Soient φ *et* ψ *deux formules de* PLTL*. Soit* Π *un ensemble de chemins. Les formules* φ *et* ψ *sont dites* globalement équivalentes *(ou, plus simplement,* équivalentes*) pour* Π *lorsque la propriété suivante est vérifiée :*

$$\forall \pi \in \Pi.\ \forall i \in \mathbb{N}.\ (\pi, i \models \varphi \ \Leftrightarrow\ \pi, i \models \psi). \tag{1.1}$$

Nous écrirons $\varphi \equiv_g^\Pi \psi$*, ou plus simplement* $\varphi \equiv^\Pi \psi$*, dans ce cas.*

On omet la mention de l'ensemble Π*, et on écrit donc simplement* $\varphi \equiv \psi$*, lorsque l'équivalence* (1.1) *doit être satisfaite pour tous les chemins (infinis) engendrés par* Prop.

Comme il est précisé à la page 21, nous considérons ici des chemins infinis uniquement. Nous pourrions envisager l'équivalence de deux formules sur des chemins finis, ou sur tous les chemins, finis ou infinis, ce qui donnerait des équivalences différentes. Par exemple, sur les chemins infinis, $\overset{\infty}{\mathbf{G}}\, a$ et $\overset{\infty}{\mathbf{F}}\, a$ ne sont pas équivalentes, alors qu'elles le sont sur les chemins finis.

Pour illustrer la notion d'équivalence globale, nous avons vu, à la page 24, l'équivalence suivante :

$$\neg(\neg\psi\,\mathbf{U}\,(\neg\varphi \wedge \neg\psi)) \equiv (\mathbf{G}\,\varphi) \vee (\varphi\,\mathbf{U}\,\psi).$$

Lorsque deux formules sont équivalentes, nous aimerions qu'elles soient complètement « interchangeables ». C'est à dire que, lorsqu'on substitue à une sous-formule une sous-formule équivalente, la formule obtenue doit être équivalente à la formule de départ. Cette propriété s'appelle « substitutivité ». Elle est effectivement satisfaite pour PLTL :

Proposition 1.13
Soit Π un ensemble de chemins. Soit $\varphi[\psi]$ une formule de PLTL *dont ψ est une sous-formule. Soit ψ' une formule globalement équivalente à ψ pour Π. Alors la formule $\varphi[\psi']$, obtenue en remplaçant une ou plusieurs occurrences de ψ par ψ', est globalement équivalente, pour Π, à $\varphi[\psi]$.*

Démonstration :
Soit ξ une formule de PLTL. Si ψ et ψ' sont globalement équivalentes, on se convainc alors facilement que les équivalences suivantes sont vérifiées :

$$\begin{array}{cc}
\neg\psi \equiv^{\Pi} \neg\psi' & \psi \vee \xi \equiv^{\Pi} \psi' \vee \xi \\
\psi\,\mathbf{U}\,\xi \equiv^{\Pi} \psi'\,\mathbf{U}\,\xi & \xi\,\mathbf{U}\,\psi \equiv^{\Pi} \xi\,\mathbf{U}\,\psi' \\
\psi\,\mathbf{S}\,\xi \equiv^{\Pi} \psi'\,\mathbf{S}\,\xi & \xi\,\mathbf{S}\,\psi \equiv^{\Pi} \xi\,\mathbf{S}\,\psi' \\
\mathbf{X}\,\psi \equiv^{\Pi} \mathbf{X}\,\psi' & \mathbf{X}^{-1}\,\psi \equiv^{\Pi} \mathbf{X}^{-1}\,\psi'
\end{array}$$

Par récurrence sur la « profondeur » d'apparition de ψ, on peut alors montrer le résultat recherché : nous allons montrer que, si une sous-formule à la profondeur k est remplacée par une sous-formule équivalente pour Π, la formule qui en résulte est équivalente à la formule de départ.

– lorsque $k = 0$, le résultat est évident ;
– si une sous-formule ψ apparaît à la profondeur k, alors à la profondeur $k-1$ apparaît une sous-formule de la forme $\neg\psi$, $\psi \vee \xi$, $\psi\,\mathbf{U}\,\xi$, $\xi\,\mathbf{U}\,\psi$, $\mathbf{X}\,\psi$, $\psi\,\mathbf{S}\,\xi$, $\xi\,\mathbf{S}\,\psi$ ou $\mathbf{X}^{-1}\,\psi$. On sait d'après ce qui précède que cette sous-formule est équivalente, pour Π, à $\neg\psi'$, $\psi' \vee \xi$, $\psi'\,\mathbf{U}\,\xi$, $\xi\,\mathbf{U}\,\psi'$, $\mathbf{X}\,\psi'$, $\psi'\,\mathbf{S}\,\xi$, $\xi\,\mathbf{S}\,\psi'$ ou $\mathbf{X}^{-1}\,\psi'$, respectivement. En appliquant l'hypothèse de récurrence pour cette sous-formule, qui apparaît à la profondeur $k-1$, on obtient le résultat. $\square$

1.1.2.2 L'équivalence initiale

De la même façon que nous avons défini une version « initiale » de la satisfaction d'une formule, nous définissons une version « initiale » de l'équivalence.

Définition 1.14
Soient φ et ψ deux formules de PLTL*. Soit Π un ensemble de chemins. Les formules φ et ψ sont dites* initialement équivalentes *lorsque la propriété suivante est vérifiée :*

$$\forall \pi \in \Pi.\ (\pi, 0 \models \varphi \ \Leftrightarrow\ \pi, 0 \models \psi). \tag{1.2}$$

Nous écrirons $\varphi \equiv_i^{\Pi} \psi$.

De la même façon que dans la définition de l'équivalence globale, on omet la mention de Π, et on écrit donc simplement $\varphi \equiv_i \psi$, lorsque l'équivalence (1.2) *doit être satisfaite pour tous les chemins « engendrés » par* Prop.

Le résultat suivant est évident :

Proposition 1.15
Soient Π un ensemble de chemins, et φ et ψ deux formules de LTL*. Si $\varphi \equiv^{\Pi} \psi$, alors $\varphi \equiv_i^{\Pi} \psi$.*

L'implication inverse n'est cependant pas vraie : si φ est ψ sont deux formules de PLTL, on a alors les équivalences suivantes :

$$\varphi \,\mathbf{S}\, \psi \equiv_i \psi \qquad\qquad \mathbf{X}^{-1}\, \varphi \equiv_i \bot$$

Par ailleurs, il est clair que si φ et ψ sont initialement équivalentes, et si ξ est une troisième formule, on a bien $\neg\varphi \equiv_i \neg\psi$, et $\varphi \vee \xi \equiv_i \psi \vee \xi$. En utilisant ces équivalences, on peut montrer de façon récursive que le résultat suivant :

Proposition 1.16
*Toute formule « pur passé » (*i.e. *n'utilisant que des modalités du passé) est initialement équivalente à une combinaison booléenne de propositions atomiques.*

Il en découle que l'équivalence initiale est bien distincte de l'équivalence globale, et qu'elle n'est pas substitutive. En se restreignant à LTL, cependant, les deux équivalences coïncident :

Proposition 1.17
Soient Π un ensemble de chemins, et φ et ψ deux formules de LTL*. Alors $\varphi \equiv^{\Pi} \psi$ si, et seulement si, $\varphi \equiv_i^{\Pi} \psi$.*

Ce résultat découle immédiatement de la proposition 1.8.

1.1.2.3 Relations entre ces deux notions d'équivalence

Nous avons vu précédemment que, pour PLTL, l'équivalence initiale et l'équivalence globales sont deux relations différentes. Nous pouvons cependant les relier par le résultat suivant :

Proposition 1.18
Soient φ et ψ deux formules de PLTL, *et Π un ensemble de chemins.*

$$\begin{aligned}\varphi \equiv_i^\Pi \psi \text{ si, et seulement si, } & \mathbf{I}\,\varphi \equiv_g^\Pi \mathbf{I}\,\psi \\ \text{si, et seulement si, } & \mathbf{I}\,(\varphi \Leftrightarrow \psi) \equiv_g^\Pi \top\,;\end{aligned}$$

$$\varphi \equiv_g^\Pi \psi \text{ si, et seulement si, } \mathbf{G}\,(\varphi \Leftrightarrow \psi) \equiv_i^\Pi \top$$

On notera que l'équivalence « $\varphi \equiv_g^\Pi \psi$ *si, et seulement si,* $\mathbf{G}\,\varphi \equiv_i^\Pi \mathbf{G}\,\psi$ » n'est pas vraie. Par exemple, $\mathbf{G}\,a \not\equiv_g a$, alors que $\mathbf{G}\,\mathbf{G}\,a \equiv_i \mathbf{G}\,a$.

Démonstration :
Commençons par les deux premières équivalences :

- Supposons que $\varphi \equiv_i^\Pi \psi$. Soit π un chemin et i un entier naturel. Alors

$$\pi, i \models \mathbf{I}\,\varphi \;\Leftrightarrow\; \pi, 0 \models \varphi \;\Leftrightarrow\; \pi, 0 \models \psi \;\Leftrightarrow\; \pi, i \models \mathbf{I}\,\psi;$$

- supposons maintenant que $\mathbf{I}\,\varphi \equiv_g^\Pi \mathbf{I}\,\psi$, et considérons un chemin $\pi \in \Pi$ et une position i. L'hypothèse se traduit par $\pi, i \models \mathbf{I}\,\varphi \quad\Leftrightarrow\quad \pi, i \models \mathbf{I}\,\psi$, et on a donc $\pi, 0 \models \varphi \quad\Leftrightarrow\quad \pi, 0 \models \psi$. Cela est équivalent à $\pi, 0 \models (\varphi \Leftrightarrow \psi)$. Il est donc toujours vrai que $\pi, i \models \mathbf{I}\,(\varphi \Leftrightarrow \psi)$, ce qui est le résultat cherché.
- supposons enfin que $\mathbf{I}\,(\varphi \Leftrightarrow \psi) \equiv_g^\Pi \top$. Soit π un chemin tel que $\pi, 0 \models \varphi$. On sait que $\pi, 0 \models \mathbf{I}\,(\varphi \Leftrightarrow \psi)$, par hypothèse, donc que $\pi, 0 \models \varphi \Leftrightarrow \psi$. Donc $\pi, 0 \models \psi$. Par symétrie du rôle de φ et ψ, on a l'équivalence inverse.

Passons maintenant à la dernière équivalence :

- supposons que $\varphi \equiv_g^\Pi \psi$. Soit π un chemin de Π, et i une position de ce chemin. L'hypothèse se traduit par $\pi, i \models \varphi \;\Leftrightarrow\; \pi, i \models \psi$. Donc $\pi, i \models (\varphi \Leftrightarrow \psi)$. Ce résultat est vrai quelle que soit la position i. Donc $\pi, 0 \models \mathbf{G}\,(\varphi \Leftrightarrow \psi)$. Ce résultat étant valable pour tous les chemins de Π, on a bien le résultat voulu ;
- réciproquement, si $\mathbf{G}\,(\varphi \Leftrightarrow \psi)$ est vérifiée initialement le long de tout chemin de Π, considérons un chemin $\pi \in \Pi$ et une position i tels que $\pi, i \models \varphi$. Par hypothèse, $\pi, 0 \models \mathbf{G}\,(\varphi \Leftrightarrow \psi)$. Donc $\pi, i \models \varphi \Leftrightarrow \psi$. On en déduit que $\pi, i \models \psi$. Par symétrie des rôles de φ et ψ, on a l'implication inverse. $\square$

Cette proposition nous permet, *a posteriori*, de montrer que l'équivalence globale est « la bonne » relation d'équivalence lorsqu'on travaille sur PLTL : l'équivalence globale est la plus petite équivalence qui contienne l'équivalence initiale et qui soit substitutive.

Soient en effet deux formules φ et ψ équivalentes pour une équivalence $\simeq$ substitutive et contenant l'équivalence initiale. Considérons la formule $\Phi[X] \stackrel{\text{def}}{=} \mathbf{G}\,(X \Leftrightarrow \varphi)$. Par substitutivité, on a nécessairement $\Phi[\varphi] \simeq \Phi[\psi]$. En particulier, puisque cette équivalence contient l'équivalence initiale, $\Phi[\varphi] \equiv_i \Phi[\psi]$. Donc $\mathbf{G}\,(\varphi \Leftrightarrow \varphi) \equiv_i \mathbf{G}\,(\psi \Leftrightarrow \varphi)$. La partie gauche de cette équivalence est globalement équivalente à $\top$. D'après la proposition précédente, on en déduit que φ et ψ sont globalement équivalentes.

1.2 Expressivité de PLTL

1.2.1 La logique du premier ordre

La logique du premier ordre est la logique que nous avons utilisée pour définir la sémantique de PLTL. Nous allons voir ici que, inversement, une formule de la logique du premier ordre peut être traduite dans PLTL.

1.2.1.1 Définitions

Soit Prop un ensemble de propositions atomiques, π un chemin sur Prop, et $X = \{x, y, \ldots\}$ un ensemble de variables. Pour chaque proposition atomique $a \in \text{Prop}$, on définit le prédicat correspondant, P_a, de la façon suivante : $P_a \colon n \mapsto \pi_n(a)$. Nous utiliserons également un prédicat binaire $<$ (qui est l'ordre naturel sur les entiers) et la fonction successeur, notée Succ. Une propriété de FOMLO est une formule de logique construite à partir des variables de X, des quantificateurs existentiel et universel sur ces variables, des opérateurs booléens et des prédicats et fonction définis ci-dessus. Ces formules sont interprétées sur $\mathbb{N}$ dans le cas présent, *i.e.* sur des chemins.

On peut définir plusieurs restrictions de la logique FOMLO : nous noterons FO^n le fragment de FOMLO dans lequel on ne peut utiliser que n variables. À l'intérieur de ce fragment, on pourra considérer le fragment dans lequel la fonction Succ est interdite. Nous noterons ce fragment $\text{FO}^n[<]$. Noter que cette deuxième restriction n'a pas d'effet si on ne limite pas le nombre de variables, puisqu'on a l'équivalence suivante :

$$y = \text{Succ}(x) \;\equiv\; x < y \,\wedge\, \forall z.\, \neg(x < z \,\wedge\, z < y)$$

On peut donc, au prix d'une variable supplémentaire, définir Succ à partir de $<$.

1.2.1.2 Équivalence d'expressivité avec PLTL

Il est clair qu'il existe une traduction, en temps linéaire, de PLTL vers FOMLO, puisque la sémantique de PLTL est définie avec des formules de FOMLO.

Ces deux logiques sont en fait équivalentes :

Théorème 1.19 ([Kam68, GPSS80])
PLTL *et* LTL *ont la même expressivité que* FOMLO*.*

Un corollaire immédiat, et qui est la partie qui nous intéresse ici, est que PLTL et LTL ont la même expressivité. La traduction de FOMLO vers PLTL ou LTL est effective ; cependant, la taille de la formule obtenue est non-élémentaire [Sto74, EVW97]. On en déduit un algorithme de traduction d'une formule de PLTL en une formule initialement équivalente de LTL, en passant par FOMLO.

Le fait que PLTL et LTL soient aussi expressives que FOMLO nous fournit également le résultat suivant : LTL et PLTL définissent exactement les langages « sans étoile » (*cf.* [McN66, McN71, Tho96]). À chaque formule de PLTL peut donc être associé un automate de Büchi sans compteur. Nous reviendrons sur ce fait plusieurs fois dans la suite de ce chapitre.

On peut, par ailleurs, montrer que trois variables suffisent dans la traduction de PLTL vers FOMLO, c'est à dire que PLTL et LTL ont la même expressivité que FO^3 [Kam68, IK89]. En restreignant encore, on obtient des fragments de PLTL :

Théorème 1.20 ([EVW97])
– $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$ *a la même expressivité que* FO^2.
– $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ *a la même expressivité que* $\mathrm{FO}^2[<]$.

1.2.2 Traduction de PLTL vers LTL

La traduction d'une formule de PLTL en une formule équivalente de LTL n'est pas chose facile. Nous avons vu dans la section précédente une méthode consistant à traduire, dans un premier temps, la formule de PLTL en une formule de FOMLO, puis de cette formule de FOMLO en une formule de LTL. La formule qui en résulte est de taille non-élémentaire dans le cas général, et cette méthode est donc inutilisable en pratique.

Nous allons voir ici deux autre méthodes : la première consiste à « séparer » le passé et le futur dans la formule de PLTL. La seconde utilise la traduction vers les automates de Büchi sans compteurs.

1.2.2.1 La méthode de séparation

L'algorithme de séparation, proposé dans [Gab89], fournit une preuve constructive du résultat suivant :

Théorème 1.21 ([Gab89, theorème 2.4])
Toute formule $\varphi \in$ PLTL peut être effectivement transformée, par des règles de réécriture, en une formule initialement équivalente de la forme

$$\bigvee_{i=1}^{n} (\varphi_i^- \wedge \varphi_i^+)$$

où $\varphi_i^- \in \mathrm{L}(\mathbf{S}, \mathbf{X}^{-1})$ et $\varphi_i^+ \in \mathrm{L}(\mathbf{U}, \mathbf{X})$.

En utilisant ce résultat et celui de la proposition 1.16, étant donnée une formule φ de PLTL, on peut donc construire une formule de LTL qui lui soit initialement équivalente.

Cette méthode de traduction présente cependant un inconvénient majeur : la taille de la formule d'arrivée est encore non-élémentaire par rapport à la taille de la formule de départ [KPV01]. La séparation est donc, en pratique, inutilisable.

1.2.2.2 En passant par les automates de mots infinis

En passant par les automates de mots infinis, nous allons obtenir une traduction un peu plus efficace : cette méthode se fait en trois étapes :
– toute formule φ de PLTL peut être « traduite » en un automate de Büchi $\mathcal{B}$ « équivalent » dans le sens suivant : un chemin π satisfait initialement la formule φ si, et seulement si, le mot π est accepté par l'automate de Büchi. La construction est effective, et l'automate $\mathcal{B}$ est de taille exponentielle dans la taille de la formule φ de départ. Une telle construction est donnée, par exemple, dans [LPZ85] ou [VW94] ;

- cet automate de Büchi $\mathcal{B}$, non déterministe, peut être transformé en un automate de Muller déterministe $\mathcal{M}$ équivalent. L'automate ainsi construit est de taille exponentielle en la taille de l'automate $\mathcal{B}$ de départ. Le langage reconnu par $\mathcal{M}$ est dont égal à l'ensemble des chemins satisfaisant initialement la propriété φ. On peut donc supposer que l'automate $\mathcal{M}$ est sans compteur, c'est à dire qu'il n'existe pas une suite de n états distincts $q_0, q_1, \ldots, q_{n-1}$ de $\mathcal{M}$, et un mot u, tels que la lecture du mot u à partir d'un état q_i arrive dans l'état q_{i+1} (avec comme convention que $q_n = q_0$). Dans le cas contraire, le langage reconnu par $\mathcal{M}$ ne serait pas « sans étoile », alors que c'est nécessairement le cas du « langage » défini par la formule φ ;
- enfin, il existe une traduction exponentielle d'un automate de Muller sans compteur en une formule de LTL équivalente dans le sens défini ci-dessus (*cf.* [MP90a, MP94]).

Ces trois étapes fournissent une traduction d'une formule de PLTL en une formule de LTL initialement équivalente le long de tout chemin infini. La taille de la formule d'arrivée est, dans le pire des cas, triplement exponentielle en la taille de la formule initiale. Une traduction similaire existe pour les chemins finis (*cf.* [Wil99a]).

Cette traduction est donc plus efficace que les méthodes précédentes. Elle reste cependant extrêmement coûteuse (par exemple, $2^{2^{2^3}}$ est un nombre à 77 chiffres, et $2^{2^{2^4}}$ est un nombre à près de 20 000 chiffres). Cependant, nous verrons plus tard que la traduction de PLTL vers LTL conduit à une formule de taille au moins exponentiellement plus longue dans le pire des cas.

1.2.3 Expressivité des fragments de PLTL

Nous allons voir dans cette section que les modalités du passé peuvent augmenter strictement l'expressivité dans le cas de certains fragments de PLTL.

1.2.3.1 Fragments sans U et S

Proposition 1.22
Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X})$ *est strictement moins expressif que* LTL*.*

Ce résultat est énoncé dans [Kam68] ; c'est d'ailleurs ce qui a motivé la définition de la modalité $\mathbf{U}$. Bien entendu, des résultats similaires peuvent être énoncés avec les modalités « symétriques » (*cf.* [Sti92]).

Cela signifie donc qu'on ne peut pas exprimer $\mathbf{U}$ en utilisant $\mathbf{F}$ et $\mathbf{X}$. La preuve du résultat ci-dessus peut être étendue comme suit :

Proposition 1.23
Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$ *est strictement moins expressif (en ce qui concerne l'équivalence initiale) que le fragment* $\mathrm{L}(\mathbf{U}, \mathbf{X})$ *(et donc que* PLTL*).*

Ce résultat est un cas particulier de [EW00, théorème 4.4]. De plus, [TW02] donne un algorithme permettant de décider si une propriété de PLTL peut être exprimée dans le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$.

Contrairement au cas général, l'ajout du passé dans les fragments $\mathrm{L}(\mathbf{F})$ et $\mathrm{L}(\mathbf{F}, \mathbf{X})$ augmente l'expressivité « initiale », comme en témoigne le résultat suivant :

Proposition 1.24
Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ *est strictement plus expressif que le fragment* $\mathrm{L}(\mathbf{F})$. *Le fragment* $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$ *est strictement plus expressif que le fragment* $\mathrm{L}(\mathbf{F}, \mathbf{X})$.

Démonstration :
Pour le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$, nous considérons, pour un entier m fixé[2], les chemins

$$\pi = a(a^m \cdot b \cdot a^m \cdot c)^\omega \quad \text{et} \quad \pi' = a(a^m \cdot c \cdot a^m \cdot b)^\omega$$

et nous allons montrer que, initialement, ils satisfont exactement les mêmes formules de $\mathrm{L}(\mathbf{F})$. Nous utiliserons le fait que

$$\forall \psi \in \mathrm{L}(\mathbf{F}).\ \forall i \in \mathbb{N}^*. \quad \pi, i \models \psi \ \Leftrightarrow\ \pi', i+m+1 \models \psi \quad \text{et} \quad \pi', i \models \psi \ \Leftrightarrow\ \pi, i+m+1 \models \psi,$$

puisque les suffixes π^i et π'^{i+m+1} d'une part, et π'^i et π^{i+m+1} d'autre part, sont identiques. Nous montrons alors que, si $i \leqslant m$, et si $\pi, i \models \varphi$ pour une formule $\varphi \in \mathrm{L}(\mathbf{F})$, alors $\pi', i \models \psi$. La preuve se fait par induction sur la structure de φ :

- Si φ est une proposition atomique, le résultat est évident ;
- si φ est une combinaison booléenne de sous-formules vérifiant, par hypothèse d'induction, la propriété, alors φ vérifie la condition également ;
- si $\varphi = \mathbf{F}\,\psi$, alors il existe un entier j tel que $\pi, j \models \psi$. Si $j \leqslant m$, alors $\pi', j \models \psi$ par hypothèse d'induction. Sinon, $\pi', j+m+1 \models \psi$. Dans les deux cas, $\pi', i \models \mathbf{F}\,\psi$.

Par symétrie, on obtient l'implication inverse. Donc π et π' satisfont (initialement) exactement les mêmes formules de $\mathrm{L}(\mathbf{F})$. Par contre, la formule $\mathbf{G}\,(a \vee \mathbf{F}^{-1}\,b)$ est vrai initialement le long de π, mais pas le long de π'. D'où le résultat.

Pour le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$, le raisonnement est similaire, mais un peu plus compliqué. Soit m un entier. Nous considérons toujours les chemins définis précédemment, et nous allons montrer que ces deux chemins ne peuvent pas être distingués par une formule φ de $\mathrm{L}(\mathbf{F}, \mathbf{X})$ si $h_{\mathbf{X}}(\varphi) \leqslant m$. Plus précisément, nous montrons le résultat suivant :

$$\forall m \geqslant 0.\ \forall k \leqslant m.\ \forall \varphi \in \mathrm{L}(\mathbf{F}, \mathbf{X}) \text{ t.q. } h_{\mathbf{X}}(\varphi) \leqslant k.\ \forall i \leqslant m - h_{\mathbf{X}}(\varphi). \quad \pi, i \models \varphi \ \Leftrightarrow\ \pi', i \models \varphi.$$

Le cas $k = 0$ a été traité ci-dessus. Supposons maintenant le résultat vrai pour un entier k donné, et considérons une formule φ, telle que $h_{\mathbf{X}}(\varphi) \leqslant k+1$. Soit i un entier inférieur à $m - h_{\mathbf{X}}(\varphi)$, tel que $\pi, i \models \varphi$. Alors

- si φ est une proposition atomique, le résultat est évident, car la i-ème position de π' est étiquetée de la même façon que la i-ème position de π, puisque $i \leqslant m$;
- si φ est une combinaison booléenne de sous-formules, par induction, le résultat est vrai ;
- si $\varphi = \mathbf{X}\,\psi$, alors $\pi, i+1 \models \psi$. Alors $h_{\mathbf{X}}(\psi) \leqslant k-1$. On peut donc appliquer l'hypothèse de la récurrence sur k : on a bien $i+1 \leqslant m - h_{\mathbf{X}}(\psi)$, donc $\pi', i+1 \models \psi$. Donc $\pi', i \models \varphi$;

[2]on peut prendre $m = 0$ pour cette partie de la preuve, mais le cas général nous servira dans la suite de la preuve, pour le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$.

– si $\varphi = \mathbf{F}\,\psi$, alors il existe une position j supérieure à i telle que $\pi, j \models \psi$. Si $j \leqslant m - h_{\mathbf{X}}(\psi)$, on peut appliquer l'hypothèse d'induction à ψ. Sinon, on remarque que π^j est le même chemin que π'^{j+m+1}. Comme ψ est une formule « pur futur », $\pi, j \models \psi$ est équivalent à $\pi', j+m+1 \models \psi$. On a alors bien entendu $i \leqslant j+m+1$, donc $\pi', i \models \mathbf{F}\,\psi$.

Par symétrie, on obtient l'implication inverse. Les chemins π et π' ne peuvent donc pas être distingués par une formule de $\mathrm{L}(\mathbf{F}, \mathbf{X})$ dont la hauteur temporelle par rapport à $\mathbf{X}$ est inférieure à m. Par contre, la formule $\mathbf{G}\,(a \vee \mathbf{F}^{-1}\,b)$ est vraie initialement le long de π mais pas le long de π'. □

1.2.3.2 Fragments « stutter-invariants »

Définition 1.25
Soient π et π' deux chemins. Ces deux chemins sont dits stutter-équivalents *s'il existe deux suites infinies strictement croissantes (i_k) et (j_k), vérifiant $i_0 = j_0 = 0$, et telles que pour tout entier k, les états $\pi_{i_k}, \pi_{i_k+1}, \ldots, \pi_{i_{k+1}-1}, \pi'_{j_k}, \pi'_{j_k+1}, \ldots, \pi'_{j_{k+1}-1}$ sont identiques.*

Définition 1.26
Une formule φ de PLTL *est* stutter-invariante *si elle ne peut pas distinguer deux chemins stutter-équivalents, c'est-à-dire que, si π et π' sont deux chemins stutter-équivalents, et si (i_k) et (j_k) sont deux suites qui « témoignent » de la stutter-équivalence de π et π', alors pour tout k,*

$$\pi, i_k \models \varphi \quad \Leftrightarrow \quad \pi, j_k \models \varphi.$$

On a alors le résultat suivant, issu de [PW97] :

Proposition 1.27
Une formule de PLTL *est stutter-invariante si, et seulement si, elle peut être exprimée dans le fragment $\mathrm{L}(\mathbf{U}, \mathbf{S})$, ou, de manière équivalente, dans le fragment $\mathrm{L}(\mathbf{U})$.*

[PW97] montre ce résultat en donnant une traduction effective. Cette traduction est cependant exponentielle. [Ete00] a proposé une traduction qui reste exponentielle, mais elle est linéaire si on autorise de « partager » les sous-formules, *i.e.* si on ne tient compte qu'une seule fois de la taille de sous-formules identiques. On peut ainsi « écrire » la formule équivalente en espace polynomial, et cela donne un algorithme en espace polynomial pour tester si une formule est stutter-invariante.

Par ailleurs, on a également le résultat suivant :

Proposition 1.28
Le fragment $\mathrm{L}(\mathbf{U})$ est strictement plus expressif que le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$.

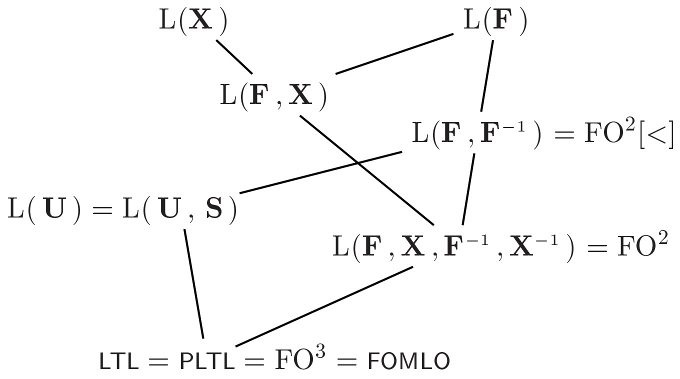
Démonstration :

Il est en effet clair que toute formule de $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$, étant stutter-invariante, peut être exprimée dans $\mathrm{L}(\mathbf{U})$. Inversement, [Lar94, corollaire 4.2.9] prouve que la propriété

$\mathbf{F}\,(a \wedge b\,\mathbf{U}\,c)$ ne peut pas être exprimée dans $\mathrm{L}(\mathbf{F}\,, \mathbf{X}\,, \mathbf{F}^{-1}\,, \mathbf{X}^{-1}\,)$. *A fortiori*, elle ne peut pas être exprimée dans $\mathrm{L}(\mathbf{F}\,, \mathbf{F}^{-1}\,)$. □

Pour résumer cette étude de l'expressivité, nous pouvons dresser le diagramme de la figure 1.2, qui contient l'ensemble des résultats que nous avons énoncés dans cette section.

Figure 1.2 Expressivité comparée de quelques fragments de PLTL



Les traits reliant deux fragments indiquent que le fragment du haut est strictement moins expressif que le fragment du bas.

1.2.4 Concision de PLTL

Nous avons vu précédemment que PLTL et LTL ont la même expressivité : toute formule de PLTL peut être transformée en une formule de LTL initialement équivalente. Cependant, les traductions connues ne sont pas efficaces, la meilleure donnant un résultat de taille triplement exponentielle. Nous allons voir ici que, de manière générale, il ne peut pas exister de traduction « efficace », *i.e.* qu'il existe des cas où une explosion exponentielle est inévitable.

1.2.4.1 PLTL peut être exponentiellement plus concis que LTL

Pour prouver ce résultat, nous allons exhiber une propriété dépendant d'un paramètre n, exprimable en PLTL par une formule de taille linéaire en n, mais telle que toute formule équivalente de LTL sera de taille exponentielle en n.

La propriété est celle de la proposition suivante :

Théorème 1.29 ([EVW97, théorème 2])
Soient $p_0, p_1, \ldots, p_n$ des propositions atomiques, et π un chemin étiqueté par ces propositions. On note $P_1(x), P_2(x), \ldots, P_n(x)$ les prédicats correspondant aux propositions, et φ_n la formule du premier ordre suivante :

$$\varphi_n \stackrel{\text{def}}{=} \forall x \forall y.\ \Big(\bigwedge_{i>0} (P_i(x) \ \Leftrightarrow\ P_i(y)) \Rightarrow (P_0(x) \ \Leftrightarrow\ P_0(y)) \Big)$$

Cette formule exprime que si deux positions quelconques ont les mêmes valeurs pour les propositions p_1 jusqu'à p_n, alors elles ont également la même valeur pour la proposition p_0.

Alors toute formule de PLTL *équivalente à cette propriété est de taille* $\Omega(2^n)$.

Démonstration :

Pour prouver ce résultat, nous prenons pour Prop l'ensemble des propositions atomiques $p_0, p_1, \ldots, p_n$. Pour chaque valeur de n, nous notons $L_n = \{u \in (\{\top, \bot\}^{\mathrm{Prop}})^\omega \mid u \models \varphi_n\}$. D'après [VW86], nous savons qu'un tel langage est reconnu par un automate de Büchi (généralisé) ayant $2^{O(|\varphi_n|)}$ reconnaissant le même langage.

Soit $a_0, a_1, \ldots, a_{2^n-1}$ une suite contenant (une et une seule fois) tous les sous-ensembles de $\{p_1, \ldots, p_n\}$. À chaque sous-ensemble K de $\{0, 1, \ldots, 2^n - 1\}$, on associe le mot $w_K = b_0 \cdot b_1 \cdots b_{2^n-1}$ défini par

$$\begin{cases} b_i \stackrel{\text{def}}{=} a_i & \text{si } i \in K \\ b_i \stackrel{\text{def}}{=} a_i \cup \{p_0\} & \text{sinon.} \end{cases}$$

On définit ainsi 2^{2^n} mots différents de taille 2^n, puisqu'il existe 2^{2^n} façons de choisir l'ensemble K, et que chacune conduit à un mot différent. Nous identifions, dans la suite, un sous-ensemble de Prop avec la fonction de Prop dans $\{\top, \bot\}$ qui envoient les éléments du sous-ensemble sur $\top$ et les autres sur $\bot$.

Si K et K' sont deux sous-ensembles de $\{0, 1, \ldots, 2^n - 1\}$ différents. Alors w_K^ω satisfait φ_n, puisque si deux « états » ont la même valuation pour les propositions $p_1, \ldots, p_n$, elles correspondent au même a_i. Elles ont donc la même valuation pour p_0 (cette valuation dépendant du fait que l'indice i appartient à K ou pas). Par contre, le mot $w_{K'} \cdot w_K^\omega$ ne satisfait pas φ_n : en effet, puisque K est différent de K', il existe un élément i appartenant à l'un mais pas à l'autre. La i-ème lettre de w_K a donc la même valuation que la i-ème lettre de $w_{K'}$ pour les propositions $p_1, \ldots, p_n$, mais pas pour p_0. D'où le fait que $w_{K'} \cdot w_K^\omega$ ne satisfasse pas φ_n.

Considérons maintenant un automate de Büchi $\mathcal{B}$ reconnaissant exactement le langage L_n. Cet automate doit accepter le mot w_K^ω, mais il doit refuser le mot $w_{K'} \cdot w_K^\omega$. Donc nécessairement, il ne peut pas se trouver dans le même état après avoir lu le mot w_K et après avoir lu le mot $w_{K'}$ (puisque dans le cas contraire, les deux mots w_K^ω et $W_{K'} \cdot w_K^\omega$ seraient soit tous les deux acceptés, soit tous les deux refusés). L'automate doit donc avoir au moins autant d'états qu'il y a de choix possibles pour K, donc au moins 2^{2^n}. La formule φ_n est donc au moins de taille 2^n. $\square$

Remarquons, au passage, qu'il existe effectivement des formules de PLTL exprimant la propriété φ_n : il suffit, par exemple, d'énumérer toutes les valuations possibles pour les propositions p_i. Par exemple :

$$\bigwedge_{q_0 \in \{\top,\bot\}} \bigwedge_{q_1 \in \{\top,\bot\}} \cdots \bigwedge_{q_n \in \{\top,\bot\}} \Big(\mathbf{F}\,((p_1 = q_1) \wedge \ldots \wedge (p_n = q_n) \wedge (p_0 = q_0))\Big) \Rightarrow$$
$$\mathbf{G}\,\big(((p_1 = q_1) \wedge \ldots \wedge (p_n = q_n)) \Rightarrow (p_0 = q_0)\big)$$

Corollaire 1.30
Soient $p_0, p_1, \ldots, p_n$ des propositions atomiques, et π un chemin étiqueté par ces propositions. On note $P_1(x), P_2(x), \ldots, P_n(x)$ les prédicats correspondant aux propositions, et ψ_n la formule du premier ordre suivante :

$$\psi_n \stackrel{\text{def}}{=} \forall x. \Big(\bigwedge_{i>0} (P_i(x) \;\Leftrightarrow\; P_i(0)) \Rightarrow (P_0(x) \;\Leftrightarrow\; P_0(0)) \Big)$$

Cette formule exprime que si une position quelconque a les mêmes valeurs que la position initiale pour les propositions p_1 jusqu'à p_n, alors elle a également la même valeur que la position initiale pour la proposition p_0.

Alors toute formule de LTL initialement équivalente à cette propriété est de taille au moins $2^{\Omega(n)}$.

Démonstration :

Soit en effet ψ'_n une formule de LTL équivalente à ψ_n. Considérons la formule $\varphi'_n \stackrel{\text{def}}{=} \mathbf{G}\,\psi'_n$. Cette formule de LTL est alors initialement équivalente à la formule φ_n du théorème 1.29 (d'après la proposition 1.8), et est donc de taille $2^{\Omega(n)}$, et donc ψ'_n est de taille $2^{\Omega(n)}$. □

On en déduit le théorème suivant :

Théorème 1.31 ([LMS02b, théorème 3.1])
Il existe une famille de formules (φ_n) de PLTL, telle que $|\varphi_n| = O(n)$, et telle que pour toute famille de formules (ψ_n) de LTL, si $\psi_n \equiv_i \varphi_n$ quel que soit n, alors $|\psi_n|$ est en $2^{\Omega(n)}$.

Démonstration :

Nous avons vu, au corollaire précédent, que la propriété « si une position quelconque a les mêmes valeurs que la position initiale pour les propositions p_1 jusqu'à p_n, alors elle a également la même valeur pour la proposition p_0 » ne peut s'exprimer en LTL qu'avec une formule de taille exponentielle en n.

Posons

$$\xi_n = \mathbf{G}\Big(\big(\bigwedge_{i=1}^{n} \big(p_i \;\Leftrightarrow\; \mathbf{F}^{-1}(\neg \mathbf{X}^{-1}\top \wedge p_i)\big)\big) \Rightarrow \big(p_0 \;\Leftrightarrow\; \mathbf{F}^{-1}(\neg \mathbf{X}^{-1}\top \wedge p_0)\big)\Big).$$

On a bien $\xi_n \equiv_i \psi_n$, et la taille de ξ_n est polynomiale. □

La preuve ci-dessus présente le défaut de faire intervenir un nombre non borné de propositions atomiques. Cependant, d'après [DS02], on peut se ramener à une seule proposition atomique (au prix d'une hauteur temporelle plus importante).

Ce résultat de concision peut être renforcé dans le sens suivant : l'explosion de la taille de la formule est accompagné d'une explosion du nombre de sous-formules. En d'autres termes, cette explosion ne peut pas être « réduite » en représentant la formule sous forme de graphe orienté acyclique, comme à la figure 1.1, page 20. En effet, dans la construction de l'automate de Büchi équivalent à une formule de PLTL, les états sont des sous-ensembles de l'ensemble des sous-formules. S'il y avait un nombre polynomial de sous-formules, l'automate de Büchi aurait un nombre simplement exponentiel d'états, alors que nous avons prouvé que cela n'est pas possible.

1.2.4.2 Concision comparée de certains fragments

La démonstration précédente n'utilise que le fragment $L(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$. On en déduit que $L(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$ peut être exponentiellement plus succinct que $L(\mathbf{F}, \mathbf{X})$, en plus d'être plus expressif.

De plus, ξ_n peut s'exprimer sans la modalité $\mathbf{X}^{-1}$. En effet, cette modalité ne sert qu'à « chercher » l'état initial, ce qui peut être fait avec la modalité $\mathbf{I}$, qui est en fait l'abréviation de $\mathbf{F}^{-1}\mathbf{G}^{-1}$. On peut donc écrire

$$\xi_n = \mathbf{G}\Big(\big(\bigwedge_{i=1}^{n}(p_i \;\Leftrightarrow\; \mathbf{F}^{-1}\mathbf{G}^{-1}p_i)\big) \Rightarrow \big(p_0 \;\Leftrightarrow\; \mathbf{F}^{-1}\mathbf{G}^{-1}p_0\big)\Big).$$

Donc, le fragment $L(\mathbf{F}, \mathbf{F}^{-1})$ peut aussi être exponentiellement plus concis que le fragment $L(\mathbf{F})$, en plus d'être plus expressif.

1.3 Complexité de la vérification de propriétés de PLTL

1.3.1 Les problèmes de vérification

Nous avons vu, dans ce qui précède, que l'utilisation du passé simplifie l'écriture des formules de logique temporelle. Les modalités du passé permettent de rester assez proche du langage naturel, et d'obtenir des formules plus concises.

Cette troisième section, la plus importante, a pour but d'étudier le coût, en termes de complexité des problèmes de vérification, de l'ajout des modalités du passé. Nous allons, dans un premier temps, définir les problèmes de vérification que nous considérerons, puis étudier la complexité de ces problèmes pour les différents fragments que nous avons définis précédemment.

1.3.1.1 Validité et satisfaisabilité

Le problème de la validité est le suivant : « étant donnée une formule φ, est-elle vraie le long de tout chemin ? » Plus formellement :

Définition 1.32
Soit φ une formule de PLTL*. On dira que φ est* (globalement) valide *si, pour tout chemin i et toute position i, $\pi, i \models \varphi$. On dira que φ est* initialement valide *si, pour tout chemin π, $\pi \models \varphi$.*

Intuitivement, dire que « φ est valide » signifie donc que φ est vraie partout, le long de tous les chemins considérés. Dire que « φ est initialement valide » signifie que φ est vraie dans l'état initial de tous ces chemins.

On peut donner une autre caractérisation de la validité : une formule φ est globalement (respectivement initialement) valide si, et seulement si, elle est globalement (respectivement initialement) équivalente à $\top$.

Bien entendu, il découle de la proposition 1.17 que ces deux notions coïncident pour une formule de LTL.

Le problème de la satisfaisabilité est la version « existentielle » de la validité : une formule est satisfaisable s'il existe un chemin qui la satisfait.

Définition 1.33
Soit φ une formule de PLTL. *La formule φ est dite* (globalement) satisfaisable *s'il existe un chemin π et une position i tels que $\pi, i \models \varphi$. De la même façon, φ est* initialement satisfaisable *s'il existe un chemin π tel que $\pi \models \varphi$.*

Cette notion est en fait la notion duale de la validité : une formule est globalement (respectivement initialement) satisfaisable si, et seulement si, sa négation n'est pas globalement (respectivement initialement) valide. Comme précédemment, les notions de satisfaisabilité initiale et de satisfaisabilité globale coïncident pour la logique LTL.

Ces définitions donnent lieu à quatre problèmes de vérification :

Définition 1.34

VAL	*entrée :*	*une formule φ de* PLTL
	sortie :	*est-ce que φ est valide ?*
VAL_i *:*	*entrée :*	*une formule φ de* PLTL
	sortie :	*est-ce que φ est initialement valide ?*
SAT *:*	*entrée :*	*une formule φ de* PLTL
	sortie :	*est-ce que φ est satisfaisable ?*
SAT_i *:*	*entrée :*	*une formule φ de* PLTL
	sortie :	*est-ce que φ est initialement satisfaisable ?*

Parmi ces quatre problèmes, le plus intéressant, du point de vue de la vérification, est le problème de la satisfaisabilité initiale. Cependant, tout comme nous avons vu que la satisfaisabilité et la validité sont des notions duales, ces problèmes sont assez proches les uns des autres. On a en effet les équivalences suivantes :

Proposition 1.35
Soit φ une formule de PLTL. *Alors :*
- *φ est initialement valide (respectivement satisfaisable) si, et seulement si,* $\mathbf{I}\,\varphi$ *est globalement valide (respectivement satisfaisable) ;*
- *φ est globalement valide si, et seulement si,* $\mathbf{G}\,\varphi$ *est initialement valide ;*
- *φ est globalement satisfaisable si, et seulement si,* $\mathbf{F}\,\varphi$ *est initialement satisfaisable.*

Ces équivalences découlent directement des résultats de la proposition 1.18.

[MP88] discute des avantages et inconvénients des deux approches (la version initiale et la version globale des problèmes de vérification), et préconise l'utilisation des versions initiales. Bien entendu, ces discussions n'ont de sens que dans le cadre de l'étude de la logique temporelle avec passé, et la question a donc souvent été esquivée du fait que la plupart des recherches se sont portées sur la logique « pure futur ».

1.3.1.2 Le model checking

Dans la pratique, il arrive souvent d'avoir à vérifier qu'une propriété est satisfaite ou valide pour un système informatique donné, un algorithme, un programme, ... Les notions de validité et de satisfaisabilité ne se rapportent pas à une exécution ou à un modèle donné, mais à toutes les exécutions que l'on peut construire à partir des propositions atomiques de Prop.

Afin d'appliquer des méthodes formelles à un système réactif, on le transforme tout d'abord en un modèle formel ; dans toute la suite, nous utiliserons comme modèles des *structures de Kripke*. Nous définirons ensuite les problèmes de validité et de satisfaisabilité *restreints* aux exécutions de ces modèles.

Définition 1.36
Soit Prop *un ensemble de propositions atomiques. Une* structure de Kripke *est un 4-uplet* (S, S_0, l, R) *où :*
- $S = \{s_1, s_2, \ldots, s_k\}$ *est l'ensemble (fini) des états de la structure ;*
- $S_0 \subseteq S$ *est l'ensemble des états initiaux ;*
- $l \in (\{\top, \bot\}^{\mathrm{Prop}})^S$ *associe, à chaque état de* S*, une fonction indiquant quelles propositions atomiques sont vérifiées dans cet état ;*
- $R \in S \times S$ *est la relation de transition. Cette relation est totale, c'est-à-dire que, pour tout état* s*, il existe un état* s' *tel que* $(s, s') \in R$*).*

On représente couramment les structures de Kripke sous forme de graphe orienté dont les états sont étiquetés par les propositions atomiques qu'il vérifie. Voir par exemple la structure de Kripke de la figure 1.6, page 51.

Définition 1.37
Soit $K = (S, S_0, l, R)$ *une structure de Kripke. Un* chemin issu de la structure K *est une suite* $p_0, p_1, \ldots$ *d'états de* S *vérifiant la propriété suivante : pour tout entier naturel* i*,* (p_i, p_{i+1}) *est un élément de* R*. Un chemin est dit* initial *si, de plus,* p_0 *appartient à* S_0*. On parlera également d'*exécution *de la structure de Kripke dans ce cas.*

La relation de transition étant totale, on peut toujours « compléter » un chemin fini en un chemin infini.

On remarque alors que la suite π définie par $\pi_i = l(p_i)$ est un chemin au sens de la définition 1.6. Pour simplifier, lorsqu'il n'y aura pas d'ambiguïté, nous dirons qu'un chemin $(\pi_0, \pi_1, \ldots) \in (\{\top, \bot\}^{\mathrm{Prop}})^{\mathrm{dom}(\pi)}$ *est* issu de la structure de Kripke $K = (S, S_0, l, R)$ s'il existe un chemin $(p_0, p_1, \ldots) \in S^{\mathrm{dom}(\pi)}$ issu de K tel que, pour tout entier i, $\pi_i = l(p_i)$.

Nous noterons Π_K l'ensemble des chemins (infinis) issus d'une structure de Kripke K, et Σ_K l'ensemble des exécutions (infinies) issues de K.

De manière générale, le problème du *model checking* est alors le suivant : étant données une structure de Kripke K et une formule de PLTL φ, la formule φ est-elle vraie dans la structure K ?

Nous définissons en fait quatre problèmes de *model checking* :

Définition 1.38

$\mathrm{MC}^{\forall}$:	*entrée :*	*une formule* φ *de* PLTL *et une structure de Kripke* K
	sortie :	*est-ce que* φ *est valide pour* Σ_K *?*
$\mathrm{MC}_i^{\forall}$:	*entrée :*	*une formule* φ *de* PLTL *et une structure de Kripke* K
	sortie :	*est-ce que* φ *est initialement valide pour* Σ_K *?*
$\mathrm{MC}^{\exists}$:	*entrée :*	*une formule* φ *de* PLTL *et une structure de Kripke* K
	sortie :	*est-ce que* φ *est satisfaisable pour* Σ_K *?*
$\mathrm{MC}_i^{\exists}$:	*entrée :*	*une formule* φ *de* PLTL *et une structure de Kripke* K
	sortie :	*est-ce que* φ *est initialement satisfaisable pour* Σ_K *?*

Le problème le plus intéressant du point de vue de la vérifiction est le problème du model checking universel initial. Mais comme pour les problèmes de satisfaisabilité et de validité, ces quatre problèmes de model checking sont très liés puisque ce sont juste des restrictions des quatre problèmes de la section 1.3.1.1 à un ensemble de chemins. On a donc l'équivalent de la proposition 1.35 pour ces problèmes de model checking.

1.3.1.3 Relations entre les problèmes de satisfaisabilité et de model checking

Les deux « types » de problèmes — satisfaisabilité et model checking — peuvent être réduits l'un à l'autre.

La réduction de la satisfaisabilité au model checking se fait en définissant une structure de Kripke *universelle*, qui contiendra tous les chemins que l'on peut construire à partir des propositions atomiques de Prop.

Définition 1.39
Soit Prop *un ensemble de propositions atomiques. La* structure de Kripke *universelle sur* Prop *est la structure de Kripke* $K_{\mathrm{Prop}} = (S, S, l, R)$ *définie comme suit :*
- $S = \{\top, \bot\}^{\mathrm{Prop}}$ *;*
- $l = Id$ *;*
- $R = S \times S$.

On a alors les équivalences suivantes :

Proposition 1.40
Soit φ *une formule de* PLTL, *et* Prop *l'ensemble des propositions atomiques utilisées par* φ. *On a alors les équivalences suivantes :*

$$\mathrm{VAL}(\varphi) \equiv \mathrm{MC}^{\forall}(\varphi, K_{\mathrm{Prop}}) \qquad \mathrm{VAL}_i(\varphi) \equiv \mathrm{MC}_i^{\forall}(\varphi, K_{\mathrm{Prop}})$$
$$\mathrm{SAT}(\varphi) \equiv \mathrm{MC}^{\exists}(\varphi, K_{\mathrm{Prop}}) \qquad \mathrm{SAT}_i(\varphi) \equiv \mathrm{MC}_i^{\exists}(\varphi, K_{\mathrm{Prop}})$$

(L'équivalence $\equiv$ *entre deux problèmes signifie qu'ils retournent le même résultat.)*

Ces résultats montrent donc que les problèmes de validité et de satisfaisabilité peuvent se réduire à des problèmes de model checking. Cependant, de manière générale, cette réduction est exponentielle, puisque Prop peut être de la même taille que φ, et que K_{Prop} est de taille exponentielle dans la taille de Prop.

Inversement, on peut réduire les problèmes de model checking aux problèmes de satisfaisabilité. À partir d'une structure de Kripke K, nous allons pour ce faire écrire une formule signifiant précisément qu'un chemin est un chemin de K :

Proposition 1.41
Soit $K = (S, S_0, l, R)$ une structure de Kripke sur un ensemble de propositions atomiques Prop. *Il existe une formule φ_K de* LTL *telle que*

$$\forall \pi \in (\{\top, \bot\}^{\mathrm{Prop}})^{\mathbb{N}}.\ (\pi \in \Sigma_K \;\Leftrightarrow\; \pi \models \varphi_K)$$

Démonstration :

La formule φ_K devra s'assurer que les quatre propriétés suivantes sont satisfaites :
- chaque état de π correspond bien à un état de S ;
- le premier état appartient à l'ensemble S_0 ;
- l'étiquetage, par la fonction l, de chaque état correspond bien à l'étiquetage d'un état de la structure de Kripke ;
- les transitions sont bien respectées.

Pour ce faire, nous définissons, pour chaque état $s \in S$, une nouvelle proposition atomique P_s, n'appartenant pas à Prop, et nous considérons un nouvel ensemble de propositions atomiques $\mathrm{Prop}' = \mathrm{Prop} \cup \{P_s \mid s \in S\}$. Nous devrons donc nous assurer que chaque état de π est étiqueté par une, et une seule, proposition P_s.

Nous définissons alors les formules suivantes, que nous réutiliserons à plusieurs occasions par la suite :

$$\psi^{state} \stackrel{\text{def}}{=} \Big(\bigvee_{s \in S} P_s\Big) \wedge \neg\Big(\bigvee_{\substack{s,s' \in S \\ s \neq s'}} P_s \wedge P_{s'}\Big) \tag{1.3}$$

$$\psi^{label} \stackrel{\text{def}}{=} \bigwedge_{s \in S} \Big(P_s \Rightarrow \Big(\bigwedge_{\substack{p \in \mathrm{Prop} \\ l(s)(p)=1}} p \wedge \bigwedge_{\substack{p \in \mathrm{Prop} \\ l(s)(p)=0}} \neg p\Big)\Big) \tag{1.4}$$

Nous pouvons maintenant écrire les propriétés énoncées ci-dessus :

$$\begin{aligned}
\varphi_K^{init} &\stackrel{\text{def}}{=} \Big(\bigvee_{s \in S_0} P_s\Big) \\
\varphi_K^{state} &\stackrel{\text{def}}{=} \mathbf{G}\, \psi^{state} \\
\varphi_K^{label} &\stackrel{\text{def}}{=} \mathbf{G}\, \psi^{label} \\
\varphi_K^{trans} &\stackrel{\text{def}}{=} \mathbf{G}\Big(\bigvee_{(s,s') \in R} P_s \wedge \mathbf{X}\, P_{s'}\Big)
\end{aligned}$$

Nous noterons

$$\varphi_K \stackrel{\text{def}}{=} \varphi_K^{init} \wedge \varphi_K^{state} \wedge \varphi_K^{label} \wedge \varphi_K^{trans}.$$

Il est clair que, d'une part, un chemin initial issu de la structure de Kripke K satisfait φ_K, et que, d'autre part, un chemin satisfaisant φ_K est un chemin initial de la structure de Kripke K. □

Proposition 1.42
Soit φ une formule de PLTL, et K une structure de Kripke. On a alors les équivalences suivantes :

$$\mathrm{MC}^{\forall}(\varphi, K) \equiv \mathrm{VAL}((\mathbf{I}\,\varphi_K) \Rightarrow \varphi) \qquad \mathrm{MC}_i^{\forall}(\varphi, K) \equiv \mathrm{VAL}_i(\varphi_K \Rightarrow \varphi)$$
$$\mathrm{MC}^{\exists}(\varphi, K) \equiv \mathrm{SAT}(\varphi \wedge \mathbf{I}\,\varphi_K) \qquad \mathrm{MC}_i^{\exists}(\varphi, K) \equiv \mathrm{SAT}_i(\varphi \wedge \varphi_K)$$

Ces résultats découlent directement de la proposition précédente. On remarque, par ailleurs, que la formule φ_K a une taille quadratique en la taille de la structure de Kripke K. La réduction est donc polynomiale.

1.3.2 Le model checking d'un chemin

Avant d'étudier le problème général du model checking d'une structure de Kripke, nous allons commencer par étudier le cas, plus simple, où la structure de Kripke ne contient qu'une seule exécution. Nous verrons qu'il existe un algorithme en temps polynomial pour la vérification d'une formule de PLTL dans une telle structure de Kripke.

1.3.2.1 Existence d'un témoin ultimement périodique

Une structure de Kripke ne contenant qu'une seule exécution (infinie) est nécessairement uns structure ultimement périodique : elle est constituée d'une partie initiale, que l'on parcourt une fois, et d'une partie périodique, que l'on parcourt ensuite une infinité de fois. Nous appellerons *boucle* une telle structure :

Définition 1.43 ([Mar02])
Une boucle est une structure de Kripke $L = (S, \{s_0\}, l, R)$ pour laquelle il existe deux entiers m et p, et une bijection $t\colon S \to [\![0, m+p-1]\!]$, tels que
- *$t(s_0) = 0$;*
- *pour tout entier i compris entre 0 et $m+p-2$, $(t^{-1}(i), t^{-1}(i+1)) \in R$, et de plus, $(t^{-1}(m+p-1), t^{-1}(m)) \in R$.*

Le type d'une boucle L est le couple (m, p) satisfaisant les conditions de la définition précédente et dont la somme $m+p$ est minimale. Cette somme est appelée taille *de la boucle. La* partie initiale *de L est le chemin $(t^{-1}(i))_{i<m}$, et la* partie périodique *est le chemin $(t^{-1}(i))_{i\in[\![m,m+p-1]\!]}$.*

On peut montrer assez facilement que la définition du type d'une boucle n'est pas ambiguë, *i.e.* que le type d'une boucle est unique. En effet, si (m, p) et (n, q) sont deux couples « possibles » pour être le type d'une boucle L, et donc en particulier minimisent la somme des longueurs de la partie initiale et de la partie périodique, alors $(\min(m, n), \mathrm{pgcd}(p, q))$ convient également. Puisque $\min(m, n) \leqslant m$ et $\mathrm{pgcd}(p, q) \leqslant p$, et que par ailleurs $m + p \leqslant \min(m, n) + \mathrm{pgcd}(p, q)$, on a $m = \min(m, n)$ et $p = \mathrm{pgcd}(p, q)$.

Les mêmes égalités peuvent être obtenues pour n et q par un raisonnement similaire. Donc nécessairement $m = n$ et $p = q$.

Il n'existe qu'un chemin initial issu d'une boucle L, au sens de la définition 1.37 ; nous l'appellerons *chemin associé* à la boucle L, et nous le noterons π_L.

Définition 1.44
Un sous-chemin *d'un chemin π est un chemin π' tel qu'il existe une application croissante f, de* $\mathrm{dom}(\pi')$ *dans* $\mathrm{dom}(\pi)$, *vérifiant la propriété suivante :* $\forall i \in \mathrm{dom}(\pi').\ \pi'_{f(i)} = \pi_i$.

Une sous-boucle d'une boucle L est une boucle L' dont le chemin associé est un sous-chemin du chemin associé à L. La sous-boucle L' est dite acceptable *si ses parties initiale et périodique sont des sous-chemins des parties initiale et périodique de L, respectivement. Nous noterons $L' \preccurlyeq_f L$ si L' est une sous-boucle acceptable de L, la fonction f étant la fonction reliant les chemins π_L et $\pi_{L'}$.*

La notion d'*acceptabilité* est importante ici pour contrôler la taille de la sous-boucle. En effet, la taille d'une sous-boucle peut être plus grande que la taille de la boucle initiale. Pour une sous-boucle acceptable, la taille est toujours inférieure à la taille de la boucle de départ. Par exemple, la boucle $aababba(aab)^\omega$ de taille 10 est une sous-boucle de la boucle $(ab)^\omega$, qui est de taille 2.

Proposition 1.45
Soient L une boucle de type (m,p), et $L' \preccurlyeq_f L$ de type (m',p'). Alors

$$\forall k, j \in \mathbb{N}. \quad j \in [\![m + kp, m + (k+1)p]\!] \iff f^{-1}(j) \in [\![m' + kp', m' + (k+1)p']\!]$$

Démonstration :
Ce résultat traduit simplement le fait que la partie périodique de L' est un sous-chemin de la partie périodique de L. Cela s'exprime par l'égalité suivante, valable pour tout entier i inférieur à p, et pour tout entier k :

$$f^{-1}(m - 1 + i + kp) = f^{-1}(m - 1 + i) + kp'.$$

On en déduit facilement l'équivalence ci-dessus. □

L'intérêt des boucles vient essentiellement du fait que le cas général se ramène au cas des boucles, grâce au résultat suivant :

Théorème 1.46 ([SC85, théorème 4.7])
Soit φ une formule satisfaisable de PLTL. *Alors φ est satisfaisable dans une boucle.*

Ce résultat est prouvé pour L(**U** , **X** , **S**) et pour la satisfaisabilité initiale dans [SC85], mais il s'étend bien entendu aisément à PLTL (puisque l'on peut traduire PLTL dans LTL), et à la satisfaisabilité globale grâce à la proposition 1.35.

On peut montrer que ce résultat s'étend également au problème du model checking :

Théorème 1.47
Soit φ une formule de PLTL, *et K une structure de Kripke. Si φ est vraie le long d'un chemin de K, alors φ est également vraie le long d'un chemin* ultimement périodique *de K.*

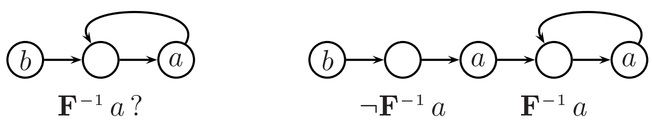
La preuve de ce résultat se fait simplement en se ramenant au problème de la satisfaisabilité, grâce à la proposition 1.41, puis en appliquant le théorème précédent.

1.3.2.2 Nécessité de « dérouler » la boucle

Pour vérifier qu'une formule φ est satisfaite le long d'une boucle, nous aurons simplement à parcourir la boucle, et à étiqueter les états de cette boucle par les sous-formules de φ. Ainsi, de manière inductive, nous connaîtrons l'ensemble des états de la boucle qui satisfont la formule.

Cependant, comme le prouve l'exemple suivant, cet algorithme n'est pas correct. En effet, si on considère l'exemple de la figure 1.3, l'étiquetage du premier état de la partie périodique par la sous-formule $\mathbf{F}^{-1}\,a$ est ambigu, car cet état a deux prédécesseurs. Lorsqu'on

Figure 1.3 Nécessité de dérouler la boucle



a déroulé la boucle, l'étiquetage n'est plus ambigu. La question est donc maintenant de généraliser ce résultat, et de savoir combien de fois il est suffisant de dérouler la boucle. Cela dépend en fait du nombre de modalités du passé qui sont emboîtées dans la formule.

Théorème 1.48 ([Mar02, lemme 2.4])
Soient φ une formule de PLTL, et L une boucle de type (m, p). Alors, pour tout entier k supérieur ou égal à $m + h_P(\varphi) \cdot p$ et pour toute sous-formule ψ de φ, on a

$$\pi_L, k \models \psi \;\Leftrightarrow\; \pi_L, k+p \models \psi.$$

Démonstration :
Si $p = 0$, le résultat est trivial. Nous allons donc supposer que ce n'est pas le cas. La preuve du résultat se fait alors par induction sur la structure de la sous-formule ψ.

- Le résultat est évidemment vrai pour les propositions atomiques, puisque les positions k et $k+p$ le long de π_L correspondent au même état de la boucle L ;
- si $\psi = \neg\psi_1$ (respectivement $\psi = \psi_1 \vee \psi_2$), en supposant le résultat vrai pour ψ_1 (respectivement ψ_1 et ψ_2), le résultat est immédiat également ;
- si $\psi = \mathbf{X}\,\psi_1$, le résultat provient immédiatement de l'hypothèse d'induction appliquée à la position $k+1$ et à la sous-formule ψ_1 ;
- si $\psi = \psi_1\,\mathbf{U}\,\psi_2$, de la même façon, c'est une conséquence directe de l'application de l'hypothèse d'induction ;
- si $\psi = \mathbf{X}^{-1}\,\psi_1$, puisque l'on suppose que p n'est pas nul, on a

$$k - 1 \geqslant m + h_P(\psi) \cdot p - 1 = m + (h_P(\psi_1) + 1) \cdot p - 1 \geqslant m + h_P(\psi_1) \cdot p \geqslant 0$$

puisque p est non nul. On a donc

$$\begin{aligned}\pi_L, k \models \mathbf{X}^{-1}\,\psi_1 &\Leftrightarrow \pi_L, k-1 \models \psi_1\\ &\Leftrightarrow \pi_L, k-1+p \models \psi_1\\ &\Leftrightarrow \pi_L, k+p \models \mathbf{X}^{-1}\,\psi_1.\end{aligned}$$

La première équivalence fait intervenir le fait que $k-1$ est positif ou nul, la deuxième provient de l'hypothèse de récurrence ;

- si $\psi = \psi_1\,\mathbf{S}\,\psi_2$, nous allons montrer séparément les deux implications :
  - supposons tout d'abord que $\pi_L, k \models \psi_1\,\mathbf{S}\,\psi_2$. Il existe alors $k' \leqslant k$ tel que $\pi_L, k' \models \psi_2$ et, pour tout j compris entre $k'+1$ et k inclus, $\pi_L, j \models \psi_1$. Deux cas peuvent se présenter :
    - soit k' est strictement inférieur à $k-p$, auquel cas toutes les positions entre $k-p$ et k satisfont ψ_1. Comme $h_P(\psi_1) \leqslant h_P(\psi) - 1$, on en déduit que toutes les positions entre k et $k+p$ satisfont ψ_1 également. La position k' est donc un témoin du fait que $\pi_L, k+p \models \psi$;
    - soit k' est supérieur ou égal à $k-p$, et dans ce cas, il suffit d'appliquer l'hypothèse d'induction à l'état k' et à la formule ψ_2 d'une part (on peut le faire puisque $h_P(\psi_2) \leqslant h_P(\psi) - 1$), et aux états j et à la formule ψ_1 d'autre part, pour les valeurs de j comprises entre $k'+1$ et k ;
  - la réciproque se montre de la même façon, en distinguant le cas où k' est inférieur à k du cas où k' est supérieur à k. □

1.3.2.3 Model checking d'une boucle

Nous allons pouvoir maintenant définir précisément notre algorithme de model checking d'une boucle par étiquetage. Il est possible d'adapter un algorithme d'étiquetage utilisé pour la vérification de formules de logiques du temps arborescent [CES86].

Cependant, nous allons prendre une autre approche, intéressante en elle-même : nous allons montrer que le model checking d'une boucle se ramène au model checking d'un chemin fini. Plus précisément :

Théorème 1.49
Soit φ une formule de PLTL *et L une boucle. Il existe alors une formule φ' de* PLTL *et un chemin fini π, constructibles de manière effective en temps polynomial, et de taille polynomiale dans la taille des données, tels que*

$$\pi_L, 0 \models \varphi \quad\Leftrightarrow\quad \pi, 0 \models \varphi'.$$

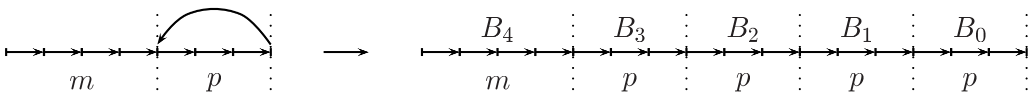
Démonstration :
Si la boucle L de départ est un chemin fini (*i.e.* si elle est de type $(m, 0)$), alors le résultat est trivial.

Dans les autres cas, nous notons $L = (\llbracket 0, m+p-1 \rrbracket, \{0\}, l, R)$ cette boucle de type (m, p) avec $p \neq 0$, et nous prenons une formule φ de PLTL. Posons $H(\psi) \stackrel{\text{def}}{=} h_P(\psi) + h_F(\psi)$, pour toute formule ψ de PLTL. Nous ajoutons à Prop les propositions B_i pour $i \in \llbracket 0, H(\varphi)+1 \rrbracket$, et nous construisons la boucle $L' = (S', \{0\}, l', R')$ de type $(m + (H(\varphi)+1)p, 0)$ dont la fonction d'étiquetage est définie de la façon suivante :

$$\begin{aligned}
l'(i)(a) = \top \;\Leftrightarrow\; l(i)(a) &= \top && \text{pour tout } i < m \text{ et } a \in \text{Prop} \\
l'(i)(B_{H(\varphi)+1}) &= \top && \text{pour tout } i < m \\
l'(i)(B_k) &= \bot && \text{pour tout } i < m \text{ et } k < H(\varphi)+1 \\
l'(m+kp+i)(a) = \top \;\Leftrightarrow\; l(m+i)(a) &= \top && \text{pour tout } i < p,\ k \leqslant H(\varphi) \text{ et } a \in \text{Prop} \\
l'(m+kp+i)(B_{H(\varphi)-k}) &= \top && \text{pour tout } i < p,\ k \leqslant H(\varphi) \\
l'(m+kp+i)(B_{H(\varphi)-j}) &= \bot && \text{pour tout } i < p,\ k \leqslant H(\varphi) \text{ et } k \neq j
\end{aligned}$$

Cette construction est représentée sur la figure 1.4. L'étiquetage par les propositions B_i

Figure 1.4 La transformation d'un chemin infini en un chemin fini (avec $H(\varphi) = 3$).



vérifie alors l'équivalence suivante :

$$\pi_s, k \models B_i \quad \Leftrightarrow \quad m + (H(\varphi) - i) \cdot p \leqslant k < m + (H(\varphi) - i + 1) \cdot p. \tag{1.5}$$

La formule φ' est construite récursivement à partir de φ, en remplaçant chaque sous-formule $\psi = \alpha \,\mathbf{U}\, \beta$ de φ par la sous-formule $\psi' = \alpha' \,\mathbf{U}\, (\beta' \wedge B_{\geqslant h_F(\psi)})$, où $B_{\geqslant i} \stackrel{\text{def}}{=} \bigwedge_{j \geqslant i} B_j$.

On a alors le résultat suivant :

Lemme 1.50
Pour toute sous-formule ψ de φ, et pour tout $i < m + (H(\varphi) - h_F(\psi))p$, on a l'équivalence suivante :

$$\pi_L, i \models \psi \quad \Leftrightarrow \quad \pi_S, i \models \psi'.$$

Nous prouvons ce lemme par induction structurelle sur la sous-formule ψ. Soit i un entier strictement inférieur à $m + (H(\varphi) - h_F(\psi))p$. Alors,

- si ψ est une proposition atomique de Prop, le résultat est immédiat ;
- si ψ est une combinaison booléenne de sous-formules, le résultat découle aisément de l'hypothèse d'induction appliquée aux sous-formules ;
- si $\psi = \alpha \,\mathbf{U}\, \beta$, nous allons montrer les deux implications :
  - supposons tout d'abord que $\pi_L, i \models \psi$. Il existe alors un entier k supérieur à i tel que $\pi_L, k \models \beta$ et tel que, pour tout entier $i \leqslant j < k$, $\pi_L, j \models \alpha$.
    D'après le théorème 1.48, on sait que k peut être choisi strictement inférieur à $m + (h_P(\beta) + 1)p$, à moins que i soit supérieur à cette valeur, auquel cas k pourra être choisi inférieur à $i + p$ (toujours d'après le théorème 1.48). Nous traitons les deux éventualités séparément :
  - si $k < m + (h_P(\beta) + 1)p$: on sait que $h_P(\beta) + 1 \leqslant H(\varphi) - h_F(\beta)$ d'une part, mais également que $h_P(\beta) + 1 \leqslant H(\varphi) - h_F(\alpha)$. Ces deux inégalités s'obtiennent

en comparant les hauteurs temporelles du futur de φ avec celles de α et β. Nous pouvons alors appliquer l'hypothèse d'induction à la position k pour la formule β, et à toutes les positions j comprises entre i et $k-1$ pour la formule α. Enfin, d'après l'équation (1.5), on obtient que $\pi_S, k \models B_{\geqslant h_F(\beta)+1}$ et $\pi_S, k \models B_{\geqslant h_F(\alpha)+1}$. Donc en particulier $\pi_S, k \models B_{\geqslant h_F(\psi)}$. Globalement, donc, $\pi_S, i \models \alpha\,\mathbf{U}\,(\beta \wedge B_{\geqslant h_F(\psi)})$.
    - si $k \leqslant i+p$: alors $k < m+(H(\varphi)-h_F(\psi)+1)p$, d'où $k < m+(H(\varphi)-h_F(\beta))p$ et $k < m+(H(\varphi)-h_F(\alpha))p$. Dès lors, le raisonnement est identique au cas précédent, et on obtient bien le résultat voulu.
  - inversement, supposons que $\pi_S, i \models \psi'$. Alors il existe un entier $k \geqslant i$ tel que $\pi_S, k \models \beta' \wedge B_{\geqslant h_F(\psi)}$ et tel que, pour tout entier j entre i et $k-1$, $\pi_S, j \models \alpha'$. On en déduit que $k < m+(H(\varphi)-h_F(\psi)+1)p$. En particulier, $k < m+(H(\varphi)-h_F(\beta))p$, et $j < m+(H(\varphi)-h_F(\alpha))p$ pour tout j compris entre i et $k-1$. On peut donc appliquer l'hypothèse d'induction pour les sous-formules α et β. On obtient alors $\pi_L, k \models \beta$ et $\pi_L, j \models \alpha$ pour tout j entre i et $k-1$. Donc $\pi_L, i \models \alpha\,\mathbf{U}\,\beta$, comme nous voulions le prouver.
- si $\psi = \mathbf{X}\,\alpha$, comme $i < m+(H(\varphi)-h_F(\psi))p$, et $h_F(\psi) = h_F(\alpha)+1$, on a $i+1 < m+(H(\varphi)-h_F(\alpha))p$, car p est supposé non nul. On peut donc appliquer l'hypothèse de récurrence à la sous-formule α et à la position $i+1$:

$$\pi_L, i+1 \models \alpha \quad \Leftrightarrow \quad \pi_S, i+1 \models \alpha'.$$

  On en déduit immédiatement l'équivalence recherchée.
- si $\varphi = \alpha\,\mathbf{S}\,\beta$, nous allons montrer les deux implications :
  - si $\pi_L, i \models \alpha\,\mathbf{S}\,\beta$, alors il existe $k \leqslant i$ tel que $\pi_L, k \models \beta$ et, pour tout j entre $k+1$ et i, $\pi_L, j \models \alpha$. De plus, $h_F(\alpha)$ et $h_F(\beta)$ sont inférieures à $h_F(\psi)$. Donc $k < m+(H(\varphi)-h_F(\beta))p$ et, pour tout j entre $k+1$ et i, $j < m+(H(\varphi)-h_F(\alpha))p$. On peut donc appliquer l'hypothèse d'induction pour ces entiers et les formules correspondantes. On obtient que $\pi_S, k \models \beta'$ et $\pi_S, j \models \alpha'$ pour les positions j entre $k+1$ et i, donc que $\pi_S, i \models \alpha'\,\mathbf{S}\,\beta'$;
  - avec les mêmes arguments, on prouve l'implication inverse ;
- si $\psi = \mathbf{X}^{-1}\,\alpha$, comme $i < m+(H(\varphi)-h_F(\psi))p$, on a $i-1 < m+(H(\varphi)-h_F(\alpha))p$. On a donc, d'après l'hypothèse d'induction,

$$\pi_L, i-1 \models \alpha \quad \Leftrightarrow \quad \pi_S, i-1 \models \alpha'.$$

  D'où l'équivalence recherchée dans ce dernier cas. □

Une fois ramené à un chemin fini, le model checking d'une formule de PLTL n'est pas un problème compliqué. La figure 1.5 propose un algorithme d'étiquetage pour le model checking d'un chemin fini π, de longueur n, pour une formule $\varphi \in$ PLTL. Cet algorithme remplit un tableau s à deux entrées, de telle sorte que $s[i, \psi]$ valle $\top$ si $\pi_S, i \models \psi$, et $\perp$ sinon. Pour ce faire, l'algorithme procède récursivement sur la structure de la formule φ, en commençant par propositions atomiques, et jusqu'à la formule φ elle-même.

Cette procédure récursive permet de calculer les positions de π vérifiant la formule φ, et donc de résoudre le problème du model checking. Elle s'exécute en temps linéaire dans

Figure 1.5 L'algorithme de model checking d'un chemin fini

```
procedure etiquette(π, φ)
  F := {sous-formules de φ};
  pour toutes les sous-formules ψ ∈ F faire
    dans le cas où ψ est de la forme
      ⊤        : pour i allant de 0 à n faire s[i,ψ] := ⊤;
      ⊥        : pour i allant de 0 à n faire s[i,ψ] := ⊥;
      p ∈ Prop : pour i allant de 0 à n faire
                   si π_i(p) = 1 alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
      ¬ψ1      : si s[0,ψ1] = ∅ alors etiquette(π, ψ1);
                 pour i allant de 0 à n faire
                   si s[i,ψ1] = ⊤ alors s[i,ψ] := ⊥ sinon s[i,ψ] := ⊤;
      ψ1 ∨ ψ2  : si s[0,ψ1] = ∅ alors  etiquette(π, ψ1);
                 si s[0,ψ2] = ∅ alors  etiquette(π, ψ2);
                 pour i allant de 0 à n faire
                   si (s[i,ψ1] = ⊤ ou s[i,ψ2] = ⊤)
                     alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
      X ψ1     : si s[0,ψ1] = ∅ alors etiquette(π, ψ1);
                 pour i allant de 0 à n − 1 faire
                   si s[i+1,ψ1] = ⊤
                     alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
                 s[n,ψ] := ⊥;
      ψ1 U ψ2  : si s[0,ψ1] = ∅ alors etiquette(π, ψ1);
                 si s[0,ψ2] = ∅ alors etiquette(π, ψ2);
                 pour i allant de n à 0 en décrémentant faire
                   si (s[i,ψ2] = ⊤ ou (i < n et s[i+1,ψ] = ⊤ et s[i,ψ1] = ⊤))
                     alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
      X^-1 ψ1  : si s[0,ψ1] = ∅ alors etiquette(π, ψ1);
                 pour i allant de 1 à n faire
                   si s[i−1,ψ1] = ⊤
                     alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
                 s[0,ψ] := ⊥;
      ψ1 S ψ2  : si s[0,ψ1] = ∅ alors etiquette(π, ψ1);
                 si s[0,ψ2] = ∅ alors etiquette(π, ψ2);
                 pour i allant de 0 à n faire
                   si (s[i,ψ2] = ⊤ ou (i > 0 et s[i−1,ψ] = ⊤ et s[i,ψ1] = ⊤))
                     alors s[i,ψ] := ⊤ sinon s[i,ψ] := ⊥;
```

la taille de la formule et dans la taille du chemin. La construction du chemin fini π et de la formule φ' à partir de la boucle L et de la formule φ se font également en temps linéaire. Globalement, le model checking d'un chemin pour une formule de PLTL se fait en temps polynomial dans la taille du chemin et de la formule.

À notre connaissance, la classe de complexité du problème du model checking d'une boucle pour une formule de LTL ou PLTL n'est pas connue. C'est cependant un problème qui apparaît naturellement dans l'analyse de « logs », lorsqu'on souhaite par exemple comprendre les raisons d'une erreur dans un programme, ou détecter des tentatives d'intrusions [RGL01].

1.3.3 Les problèmes NP-complets

Nous allons dresser ici la liste des fragments pour lesquels certains des problèmes de vérification que nous avons définis sont NP-complets. Nous savons qu'il en existe : [SC85] a prouvé que le model checking existentiel et la satisfaisabilité des fragments $\mathrm{L}(\mathbf{F})$ et $\mathrm{L}^{+}(\mathbf{F},\mathbf{X})$ sont NP-complets. [EVW97] a prouvé que la satisfaisabilité de la logique $\mathrm{L}(\mathbf{F},\mathbf{F}^{-1})$ est NP-complète.

Nous allons tout d'abord montrer que la classe NP est, à quelques rares exceptions près, la borne inférieure de la complexité des problèmes de vérification pour la logique du temps linéaire : les problèmes non triviaux sont NP-durs. Nous verrons ensuite les problèmes pour lesquels cette borne inférieure peut être atteinte, c'est-à-dire pour lesquels il existe un algorithme non déterministe en temps polynomial.

La plupart des résultats de cette section proviennent de [Mar02].

1.3.3.1 Les bornes inférieures

Pour la satisfaisabilité. Il est clair que la satisfaisabilité de tout fragment de PLTL est un problème NP-dur, puisque le problème 3-SAT est un sous-problème du problème de la satisfaisabilité d'une formule de logique temporelle. La satisfaisabilité d'une formule de PLTL est donc NP-dure.

Pour la validité. Une formule φ est valide si, et seulement si, sa négation n'est pas satisfaisable. Par dualité, on en déduit que la validité est co-NP-dure pour tout fragment de PLTL.

Pour le model checking existentiel. Les arguments précédents ne s'appliquent plus, puisqu'il s'agit ici d'*évaluer* une formule le long d'un chemin d'une structure de Kripke. Cependant, sans opérateur temporel, ou en considérant le model checking existentiel initial avec uniquement des modalités du passé, le problème revient à évaluer une formule booléenne dans un état. Ce problème est NC^1-complet [3] [Bus93]. Nous allons voir que, dans les autres cas, le problème devient NP-dur.

[3] La classe NC^1, qui est égale à la classe ALOGTIME, est une sous-classe de la classe LOGSPACE. Elle contient les langages reconnaissables par des classes de circuits booléens « ALOGTIME-uniformes » de taille polynomiale et de profondeur logarithmique. Voir [Joh90] pour de plus amples détails.

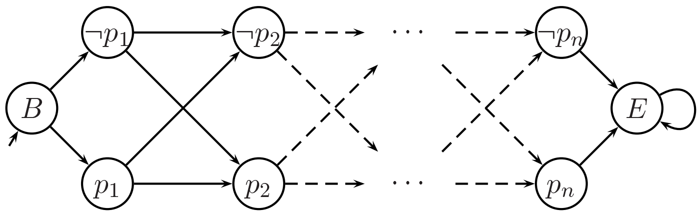
Théorème 1.51
Le model checking existentiel initial est NP*-dur pour* $L^+(\mathbf{F})$, $L^+(\mathbf{G})$ *et* $L^+(\mathbf{X})$.

Démonstration :
Nous montrons que le problème 3-SAT se réduit au model checking de ces logiques. Nous considérons pour cela un ensemble $\mathrm{Prop} = \{p_1, p_2, \ldots, p_n\}$ de propositions atomiques, et une instance de 3-SAT :

$$\bigwedge_i \bigvee_{j=1}^{3} \varepsilon_{i,j}\ p_{k_{i,j}}.$$

Dans cette formule, $\varepsilon_{i,j}$ est soit vide, soit égal à $\neg$. Nous associons à cette instance la structure de Kripke de la figure 1.6. Un chemin dans cette structure, partant de l'état initial, définit de manière évidente une valuation des propositions atomiques de Prop.

Figure 1.6 La structure de Kripke associée à une instance de 3-SAT



Alors l'instance de 3-SAT est satisfaisable si, et seulement si, l'une des trois formules suivantes est vraie initialement le long d'un chemin de la structure de Kripke de la figure 1.6 :

$$\varphi_{\mathbf{F}} \stackrel{\text{def}}{=} \bigwedge_i \bigvee_{j=1}^{3} \mathbf{F}\,(\varepsilon_{i,j}\ p_{k_{i,j}})$$

$$\varphi_{\mathbf{G}} \stackrel{\text{def}}{=} \bigwedge_i \bigvee_{j=1}^{3} \mathbf{G}\,(p_{k_{i,j}} \Rightarrow \varepsilon_{i,j}\ \top)$$

$$\varphi_{\mathbf{X}} \stackrel{\text{def}}{=} \bigwedge_i \bigvee_{j=1}^{3} \mathbf{X}^{k_{i,j}}(\varepsilon_{i,j}\ p_{k_{i,j}})$$

La preuve de cette équivalence est évidente. □

Théorème 1.52
Le model checking existentiel global est NP*-dur pour* $L^+(\mathbf{F})$, $L^+(\mathbf{G})$, $L^+(\mathbf{X})$, $L^+(\mathbf{F}^{-1})$, $L^+(\mathbf{G}^{-1})$ *et* $L^+(\mathbf{X}^{-1})$.

Démonstration :
Pour les trois fragments « du futur », la réduction est semblable à la précédente :

l'instance de 3-SAT est satisfaisable si, et seulement si, l'une des trois formules suivantes est vraie, à une certaine position, le long d'un chemin de la structure de Kripke de la figure 1.6 :

$$\varphi'_{\mathbf{F}} \stackrel{\text{def}}{=} B \wedge \varphi_{\mathbf{F}} \qquad \varphi'_{\mathbf{G}} \stackrel{\text{def}}{=} B \wedge \varphi_{\mathbf{G}} \qquad \varphi'_{\mathbf{X}} \stackrel{\text{def}}{=} B \wedge \varphi_{\mathbf{X}}$$

Pour les fragments du passé, nous ferons toujours la même réduction, mais avec des formules utilisant des modalités du passé :

$$\begin{aligned}
\varphi_{\mathbf{F}^{-1}} &\stackrel{\text{def}}{=} E \wedge \bigwedge_i \bigvee_{j=1}^{3} \mathbf{F}^{-1}\,(\varepsilon_{i,j}\ p_{k_{i,j}}) \\
\varphi_{\mathbf{G}^{-1}} &\stackrel{\text{def}}{=} E \wedge \bigwedge_i \bigvee_{j=1}^{3} \mathbf{G}^{-1}\,(p_{k_{i,j}} \Rightarrow \varepsilon_{i,j}\ \top) \\
\varphi_{\mathbf{X}^{-1}} &\stackrel{\text{def}}{=} E \wedge \bigwedge_i \bigvee_{j=1}^{3} (\mathbf{X}^{-1})^{n+1-k_{i,j}}(\varepsilon_{i,j}\ p_{k_{i,j}})
\end{aligned}$$

Là encore, les preuves de ces équivalences sont immédiates. □

Pour le model checking universel. Tout comme pour le model checking existentiel, le model checking universel initial est NC^1-complet dans le cas où on n'utilise pas de modalités, ou uniquement des modalités du passé. En effet, cela peut nécessiter d'évaluer la formule dans tous les états initiaux, mais cela pourra toujours être décidé par une classe de circuits booléens de taille polynomiale et de hauteur logarithmique.

Dans tous les autres cas, le model checking universel est co-NP-dur :

Théorème 1.53
Le model checking universel initial pour $\mathrm{L}^+(\mathbf{F})$, $\mathrm{L}^+(\mathbf{G})$ *et* $\mathrm{L}^+(\mathbf{X})$, *et le model checking universel global pour* $\mathrm{L}^+(\mathbf{F})$, $\mathrm{L}^+(\mathbf{G})$, $\mathrm{L}^+(\mathbf{X})$, $\mathrm{L}^+(\mathbf{F}^{-1})$, $\mathrm{L}^+(\mathbf{G}^{-1})$ *et* $\mathrm{L}^+(\mathbf{X}^{-1})$, *sont des problèmes co-*NP*-durs.*

Démonstration :

Ces problèmes se ramènent aux précédents par dualité : étant donnée une formule φ, vérifier qu'elle est vraie initialement le long de tous les chemins de la structure revient à vérifier qu'il n'existe pas de chemin le long de laquelle sa négation soit initialement vraie. Le model checking universel initial de $\mathrm{L}^+(\mathbf{F})$ est donc le problème dual du model checking existentiel initial de $\mathrm{L}^+(\mathbf{G})$ (puisque la négation d'une formule de $\mathrm{L}^+(\mathbf{F})$ est une formule de $\mathrm{L}^+(\mathbf{G})$). Ainsi, par exemple, l'instance du problème 3-SAT de la démonstration du théorème 1.51 est satisfaisable si, et seulement si, il n'est pas vrai que tous les chemins de la structure satisfont la formule

$$\bigvee_i \bigwedge_{j=1}^{3} \mathbf{G}\,(\neg\varepsilon_{i,j}\ p_{k_{i,j}}).$$

De même, vérifier que φ est vraie à une certaine position le long de tous les chemins revient à vérifier qu'il n'existe pas de chemin le long duquel $\neg\varphi$ soit vraie à une certaine position. □

1.3.3.2 Algorithmes NP pour certains fragments de PLTL

Avant de donner ces algorithmes, nous avons besoin de définir plus précisément ce que nous entendons par « sous-formule », et d'étendre un peu cette notion.

Sous-formules, clôture de Fischer-Ladner. Nous avons déjà parlé de sous-formules d'une formule, mais sans en donner une signification précise. Dans la suite, nous utiliserons encore cette notion, mais parfois dans un sens un peu différent : nous aurons aussi besoin de considérer la négation de ces sous-formules.

Définition 1.54
Soit φ une formule de PLTL*. L*'ensemble des sous-formules de φ*, noté* $\mathrm{SubF}(\varphi)$*, est le plus petit ensemble de formules contenant φ et vérifiant les propriétés suivantes :*
– *si une formule $\neg\psi \in \mathrm{SubF}(\varphi)$, alors $\psi \in \mathrm{SubF}(\varphi)$;*
– *si une formule $\psi_1 \vee \psi_2 \in \mathrm{SubF}(\varphi)$, alors ψ_1 et ψ_2 appartiennent également à $\mathrm{SubF}(\varphi)$;*
– *si une formule $\psi_1 \wedge \psi_2 \in \mathrm{SubF}(\varphi)$, alors ψ_1 et ψ_2 appartiennent également à $\mathrm{SubF}(\varphi)$;*
– *pour toute modalité $\mathbf{M}$ d'arité a, si $\mathbf{M}(\psi_1, \ldots, \psi_a)$ appartient à $\mathrm{SubF}(\varphi)$, alors les formules $\psi_1, \ldots, \psi_a$ appartiennent également à $\mathrm{SubF}(\varphi)$.*

Cette définition est la définition « intuitive » de ce qu'est une sous-formule : c'est une partie de la formule de départ.

Cependant, en vertu de l'équivalence $\neg\neg\varphi \equiv \varphi$, il serait « légitime » d'ajouter la négation de chacune des sous-formules. On obtient ainsi la clôture de Fischer-Ladner.

Définition 1.55
Soit φ une formule de PLTL*. La* clôture de Fischer-Ladner*, noté* $\mathrm{FL}(\varphi)$*, est le plus petit ensemble de formules contenant φ et vérifiant les propriétés suivantes :*
– *si une formule $\neg\psi \in \mathrm{FL}(\varphi)$, alors $\psi \in \mathrm{FL}(\varphi)$, en identifiant toujours $\neg\neg\psi$ à ψ. En particulier, donc, si une formule appartient à $\mathrm{FL}(\varphi)$, sa négation y appartient aussi ;*
– *si une formule $\psi_1 \vee \psi_2 \in \mathrm{FL}(\varphi)$, alors ψ_1 et ψ_2 appartiennent également à $\mathrm{FL}(\varphi)$;*
– *si une formule $\psi_1 \wedge \psi_2 \in \mathrm{FL}(\varphi)$, alors ψ_1 et ψ_2 appartiennent également à $\mathrm{FL}(\varphi)$;*
– *pour toute modalité $\mathbf{M}$ d'arité a, si $\mathbf{M}(\psi_1, \ldots, \psi_a)$ appartient à $\mathrm{FL}(\varphi)$, alors les formules $\psi_1, \ldots, \psi_a$ appartiennent également à $\mathrm{FL}(\varphi)$.*

On peut montrer par induction que le cardinal de $\mathrm{FL}(\varphi)$ est majoré par deux fois la taille de la formule φ.

Nous allons maintenant dresser la liste des problèmes NP-complets parmi ceux que nous avons définis, en les classant par fragments.

Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$**.** Nous allons montrer que la logique $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ possède un témoin polynomial. Ce résultat est énoncé dans [EVW02], mais nous en utiliserons une version un peu plus forte, afin de pouvoir l'appliquer au problème du model checking.

Lemme 1.56
Soit φ une formule satisfaisable de $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$, et L une boucle pour laquelle il existe $i \in \mathbb{N}$ tel que $\pi_L, i \models \varphi$. Il existe alors une sous-boucle acceptable L' de L vérifiant les conditions suivantes :

- *L' est de taille polynomiale en la taille de φ ;*
- *pour toute boucle L'' vérifiant $L' \preccurlyeq L'' \preccurlyeq_f L$, on a $\pi_{L''}, f^{-1}(i) \models \varphi$.*

Démonstration :

Posons $h = h_P(\varphi)$, et notons (m, p) le type de L.

Pour chaque formule de la forme $\mathbf{F}^{-1}\psi$ dans $\mathrm{FL}(\varphi)$, s'il existe une position k telle que $\pi_L, k \models \psi$, nous savons que cette position peut être choisie entre 0 et $m + (h+1)p$, d'après le théorème 1.48. Si une telle position existe, nous posons alors

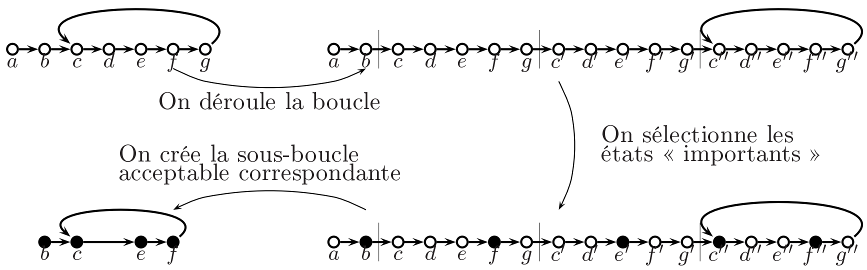
$$i_{\mathbf{F}^{-1}\psi} \stackrel{\text{def}}{=} \min\{k \mid \pi_L, i \models \psi\}.$$

De la même façon, pour les formules de la forme $\mathbf{F}\psi$ dans $\mathrm{FL}(\varphi)$, s'il existe une position k telle que $\pi_L, k \models \psi$, cette position peut être choisie entre 0 et $m + (h+1)p$. Dans ce cas, nous posons

$$i_{\mathbf{F}\psi} \stackrel{\text{def}}{=} \max\{k < m + (h_P(\varphi) + 1) \cdot p \mid \pi_L, i \models \psi\}.$$

Ces positions étant « choisies », nous construisons maintenant la sous-boucle acceptable L' de L contenant exactement les états correspondants (ainsi que l'état i dans lequel on souhaite vérifier la propriété). La figure 1.7 donne une idée intuitive de la construction.

Figure 1.7 Construction de L'



Formellement, en supposant que les états de la boucle L sont numérotés dans l'ordre de 0 à $m + p - 1$, la boucle $L' = (S', \{s'_0\}, l_{|S'}, R')$ est définie de la façon suivante :

- pour chaque formule $\xi \in \mathrm{FL}(\varphi)$ de la forme $\mathbf{F}\psi$ ou $\mathbf{F}^{-1}\psi$, si $i_\xi < m + p$, on définit $j_\xi = i_\xi$. Sinon, si i_ξ est supérieur à $m + p$, on prend pour j_ξ l'unique entier congru à i_ξ modulo p et compris entre m et $m + p - 1$. Pour chacune des valeur ainsi calculée, ainsi que pour l'état i dans lequel on vérifie la propriété φ, on conserve le j_ξ-ème état de L ;

– l'état initial est l'état « minimal » ;
– l'étiquetage est le même que pour L (restreint, bien entendu, aux états de L') ;
– la relation de transition est composée exactement des couples $(x, y) \in S'^2$ tels qu'il existe une suite d'états $x = s_0, s_1, s_2, \ldots, s_p = y$ de S telle que $s_1, \ldots, s_{p-1}$ n'appartiennent pas à S' et vérifiant $(s_0, s_1) \in R$, $(s_1, s_2) \in R$, …, $(s_{p-1}, s_p) \in R$.

Il est clair que la boucle ainsi construite est de taille polynomiale (c'est une sous-boucle acceptable de L). Soit L'' une sous-boucle de L telle que L' soit une sous-boucle de L''. Nous notons (m'', p'') son type, et f la fonction telle que $L'' \preccurlyeq_f L$. Montrons que cette boucle est un témoin de la formule φ. Pour cela, nous prouvons la propriété suivante :

$$\forall \psi \in \mathrm{FL}(\varphi).\ \forall j \in \mathbb{N}. \quad \pi_{L''}, j \models \psi \ \Leftrightarrow\ \pi_L, f(j) \models \psi.$$

Cette propriété se prouve par induction sur la structure de ψ :
– si ψ est une proposition atomique, le résultat est immédiat puisque l'étiquetage du j-ème état de $\pi_{L''}$ est le même que celui du $f(j)$-ème état de π_L ;
– si $\psi = \neg\psi_1$, l'équivalence étant supposé vérifiée pour ψ_1 par hypothèse d'induction, il est vrai également pour ψ ;
– si $\psi = \psi_1 \vee \psi_2$, en supposant, par hypothèse d'induction, que le résultat est vrai pour ψ_1 ou ψ_2, il est vrai également pour ψ ;
– si $\psi = \mathbf{F}\,\psi_1$, nous allons montrer les deux implications :
  – supposons que $\pi_{L''}, j \models \mathbf{F}\,\psi_1$. Il existe alors un entier i supérieur à j tel que $\pi_{L''}, i \models \psi_1$. Par hypothèse d'induction, et puisque f est croissante, on obtient un entier $f(i) \geqslant f(j)$ pour lequel $\pi_L, f(i) \models \psi_1$. Donc $\pi_L, f(j) \models \mathbf{F}\,\psi_1$;
  – inversement, supposons que $\pi_L, f(j) \models \mathbf{F}\,\psi_1$. Il existe alors un entier $i \geqslant f(j)$ tel que $\pi_L, i \models \psi_1$. Nous devons encore distinguer deux possibilités :
    – si $i \leqslant m + (h_P(\varphi) + 1)p - 1$, alors nécessairement $i \leqslant i_{\mathbf{F}\,\psi_1}$, par construction de $i_{\mathbf{F}\,\psi_1}$. Par hypothèse d'induction, nous savons que $\pi_{L''}, f^{-1}(i_{\mathbf{F}\,\psi_1}) \models \psi_1$. Comme f est croissante, $f^{-1}(i_{\mathbf{F}\,\psi_1})$ est plus grand que $f^{-1}(i)$, lui-même plus grand que j. Donc $\pi_{L''}, j \models \mathbf{F}\,\psi_1$;
    – si $i \geqslant m + (h_P(\varphi) + 1)p$, alors d'après le théorème 1.48, on sait que l'entier k, congru à i modulo p et compris entre $m + h_P(\varphi) \cdot p$ et $m + (h_P(\varphi) + 1) \cdot p - 1$, satisfait ψ_1. Cet entier k est donc nécessairement inférieur à $i_{\mathbf{F}\,\psi_1}$. On en déduit que $i_{\mathbf{F}\,\psi_1}$ est supérieur où égal à $m + h_P(\varphi) \cdot p$, et donc que $f^{-1}(i_{\mathbf{F}\,\psi_1})$ est supérieur ou égal à $m'' + h_P(\varphi) \cdot p''$. Par hypothèse d'induction, $\pi_{L''}, f^{-1}(i_{\mathbf{F}\,\psi_1}) \models \psi_1$, et d'après le théorème 1.48, appliqué à la boucle L', pour tout entier naturel n, $\pi_{L''}, (f^{-1}(i_{\mathbf{F}\,\psi_1}) + n \cdot p'') \models \psi_1$. On en déduit que ψ_1 est vraie infiniment souvent le long de $\pi_{L''}$, et donc que toutes les positions de $\pi_{L''}$ satisfont $\mathbf{F}\,\psi_1$. En particulier la position j.
– si $\psi = \mathbf{F}^{-1}\,\psi_1$, nous montrons également les deux implications :
  – si $\pi_{L''}, j \models \mathbf{F}^{-1}\,\psi_1$, alors il existe $i \leqslant j$ tel que $\pi_{L''}, i \models \psi_1$. Par induction, $\pi_L, f(i) \models \psi_1$. Puisque la fonction f est croissante, cela fournit un témoin du fait que $\pi_L, f(j) \models \mathbf{F}^{-1}\,\psi_1$;
  – si $\pi_L, f(j) \models \mathbf{F}^{-1}\,\psi_1$, alors il existe $i \leqslant f(j)$ tel que $\pi_L, i \models \psi_1$. Nécessairement, $i_{\mathbf{F}^{-1}\,\psi_1} \leqslant i \leqslant f(j)$. Par induction, $\pi_{L''}, f^{-1}(i_{\mathbf{F}^{-1}\,\pi_1}) \models \psi_1$, et comme f est croissante, $f^{-1}(i_{\mathbf{F}^{-1}\,\pi_1}) \leqslant j$. Donc $\pi_{L''}, j \models \mathbf{F}^{-1}\,\psi_1$. □

Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ étant stable par négation, ce résultat suffit à prouver que tous les problèmes que nous considérons sont (co-)NP-faciles :

- pour la satisfaisabilité (initiale ou globale), c'est immédiat : il suffit de « deviner » un chemin, puis de vérifier qu'il répond bien au problème ;
- pour la validité, tester que φ est valide revient à tester que $\neg\varphi$ n'est pas satisfaisable. Comme $\neg\varphi$ appartient toujours à $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ si φ y appartient, tester que $\neg\varphi$ n'est pas satisfaisable se fait par un algorithme co-NP ;
- pour le model checking existentiel, il suffit là aussi de « deviner » les états à conserver. Le fait que le résultat reste vrai pour toute sous-boucle acceptable de la boucle initiale contenant ces états nous assure que l'on peut trouver un témoin ultimement périodique polynomial dans la structure de Kripke que l'on étudie. On peut également fournir un autre algorithme pour le model checking existentiel d'une formule de $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$, en modifiant légèrement la preuve de la proposition 1.41 pour qu'elle n'utilise que les modalités $\mathbf{F}$ et $\mathbf{F}^{-1}$, et en se ramenant ainsi au problème de la satisfaisabilité.
- pour le model checking universel, c'est le problème dual du précédent, puisque $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ est stable par négation. Ce problème est donc co-NP-facile.

Théorème 1.57
Les huit problèmes que nous étudions sont (co-)NP-complets pour la fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$.

Le fragment $\mathrm{L}(\mathbf{X}, \mathbf{X}^{-1})$. Une formule φ appartenant à $\mathrm{L}(\mathbf{X}, \mathbf{X}^{-1})$ fait intervenir au plus $h_P(\varphi) + h_F(\varphi)$. Elle admet donc trivialement un témoin polynomial. Il existe donc de manière évidente des algorithmes (co-)NP-complets pour tous les problèmes que nous étudions relativement à ce fragment.

Le fragment $\mathrm{L}(\mathbf{X}, \mathbf{S}, \mathbf{X}^{-1})$. Il est évident qu'une formule $\varphi \in \mathrm{L}(\mathbf{X}, \mathbf{S}, \mathbf{X}^{-1})$ ne fait intervenir au plus que $h_F(\varphi)$ états postérieurs à l'état courant. En particulier, lorsqu'on évalue une telle formule initialement, elle ne fait intervenir au plus que $h_F(\varphi)$ états.

Cela nous fournit un algorithme NP pour la satisfaisabilité et le model checking existentiel initiaux, et un algorithme co-NP pour la validité et le model checking universel initiaux également.

Théorème 1.58
Les quatre problèmes « initiaux » que nous considérons sont NP-complets ou co-NP-complets (suivant qu'ils sont « existentiels » ou « universels ») pour $\mathrm{L}(\mathbf{X}, \mathbf{S}, \mathbf{X}^{-1})$.

Cependant, comme nous le verrons plus loin, à l'exception des problèmes de validité, les problèmes de vérification du fragment $\mathrm{L}^+(\mathbf{U})$ sont PSPACE-durs. Par « symétrie », nous verrons que les problèmes « globaux » sont PSPACE-durs pour tous les fragments considérés contenant la modalité $\mathbf{S}$.

Le fragment $\mathrm{L}^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$. On se base sur le résultat suivant :

Lemme 1.59
Soit φ une formule satisfaisable de $\mathrm{L}^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$, et L une boucle pour laquelle il existe $i \in \mathbb{N}$ tel que $\pi_L, i \models \varphi$. Il existe alors une sous-boucle acceptable L' de L vérifiant les conditions suivantes :
- *L' est de taille polynomiale en la taille de φ ;*
- *pour toute boucle L'' vérifiant $L' \preccurlyeq L'' \preccurlyeq_f L$, on a $\pi_{L''}, f^{-1}(i) \models \varphi$.*

Démonstration :
La preuve est semblable à celle du lemme 1.56 : étant donnée une boucle témoin L, nous allons en extraire une sous-boucle L' en sélectionnant certains états « témoins ».

Ces états témoins sont construits en créant une liste $S \cup T$ de couples (j, ψ) vérifiant la propriété suivante :

$$\forall j < m + (h_P(\varphi) + 1) \cdot p.\ \forall \psi \in \mathrm{SubF}(\varphi). \quad (j, \psi) \in S \cup T \Rightarrow \pi_L, j \models \psi. \tag{1.6}$$

Nous considérons donc une formule satisfaisable φ de $\mathrm{L}^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$, une boucle L et un entier i tels que $\pi_L, i \models \varphi$. Nous construisons alors S de manière récursive comme suit :
- initialement, $S = \{(i, \varphi)\}$ et $T = \varnothing$. La propriété (1.6) est évidemment vérifiée ;
- tant que S contient des couples (j, ψ) dans lesquels ψ n'est pas (la négation d') une proposition atomique, alors on supprime (j, ψ) de S, on le met dans T, et :
  - si $\psi = \psi_1 \vee \psi_2$, alors soit $\pi_L, j \models \psi_1$, soit $\pi_L, j \models \psi_2$. On place alors (j, ψ_1) ou (j, ψ_2) dans S, suivant le cas. La propriété (1.6) reste vraie ;
  - si $\psi = \psi_1 \wedge \psi_2$, alors $\pi_L, j \models \psi_1$ et $\pi_L, j \models \psi_2$. On ajoute donc (j, ψ_1) et (j, ψ_2) dans S, et la propriété (1.6) est toujours satisfaite ;
  - si $\psi = \mathbf{F}\,\psi_1$, la propriété (1.6) étant vérifiée pour ce couple, on a $\pi_L, j \models \mathbf{F}\,\psi_1$. Il existe donc un entier i supérieur à j tel que $\pi_L, i \models \psi_1$. Si i est supérieur à $m + (h_P(\varphi) + 1) \cdot p$, on sait alors, d'après le théorème 1.48, que l'on peut trouver une position i', comprise entre $m + h_P(\varphi) \cdot p$ et $m + (h_P(\varphi) + 1) \cdot p - 1$, qui vérifie la même formule ψ_1. On ajoute donc (i, ψ_1) ou (i', ψ_1) à S, selon le cas ;
  - si $\psi = \mathbf{X}\,\psi_1$, alors $\pi_L, j \models \mathbf{X}\,\psi_1$. La position $j + 1$ existe donc dans le chemin π_L. Si $j < m + (h_P(\varphi) + 1) \cdot p$, on ajoute $(j + 1, \psi_1)$ à S, sinon, on ajoute $(j + 1 - p, \psi_1)$. La propriété (1.6) est conservée, grâce au théorème 1.48 pour le deuxième cas ;
  - si $\psi = \mathbf{F}^{-1}\,\psi_1$, il existe une position i inférieure à j dans laquelle ψ_1 est satisfaite. On ajoute donc (i, ψ_1) à S ;
  - si $\psi = \mathbf{X}^{-1}\,\psi_1$, de la même façon, on ajoute $(j - 1, \psi_1)$ dans S.

Il est clair que ce processus termine, puisque la somme des longueurs des formules apparaissant dans S diminue strictement à chaque étape. Le processus précédent se fait donc en au plus $|\varphi|$ étapes. Comme à chaque étape, la taille de $S \cup T$ augmente au plus de deux éléments, il y a, à la fin, au plus $2 \cdot |\varphi|$ éléments dans $S \cup T$.

Nous considérons maintenant la sous-boucle acceptable L' de L construite comme dans la preuve du lemme 1.56 (*cf.* figure 1.7). Soit alors L'' une sous-boucle acceptable de L dont L' est une sous-boucle, et nous notons f la fonction telle que $L'' \preccurlyeq_f L$. Nous prouvons alors que $\pi_{L''}, f^{-1}(i) \models \varphi$, en prouvant, par induction, que pour tout couple $(j, \psi) \in S \cup T$, $\pi_{L''}, f^{-1}(j) \models \psi$:

- si ψ est une proposition atomique, le résultat est évident, puisque la position $f^{-1}(j)$ dans $\pi_{L''}$ correspond au même état que la position j dans le chemin π_L ;
- si $\psi = \psi_1 \vee \psi_2$ (respectivement $\psi = \psi_1 \wedge \psi_2$), alors par construction, (j, ψ_1) ou (respectivement et) (j, ψ_2) appartiennent à $S \cup T$, et par hypothèse d'induction, on a le résultat souhaité ;
- si $\psi = \mathbf{X}\,\psi_1$, alors $(j+1, \psi_1)$, ou $(j+1-p, \psi_1)$, appartient à $S \cup T$. Dans le premier cas, comme f est strictement croissante, et comme j et $j+1$ apparaissent dans $S \cup T$, $f^{-1}(j+1) = f^{-1}(j)+1$. Par hypothèse d'induction, $\pi_{L''}, f^{-1}(j+1) \models \psi_1$, et d'après ce qui précède, $\pi_{L''}, f^{-1}(j) \models \mathbf{X}\,\psi_1$. Si on est dans le deuxième cas, c'est alors que j est égal à $m+(h_P(\varphi)+1)p-1$, et $j+1-p$ est égal à $m+h_P(\varphi)p$. Ces deux états étant conservés dans la sous-boucle L', on en déduit que $f^{-1}(j+1-p) = m' + h_P(\varphi)p'$ et $f^{-1}(j) = m' + (h_P(\varphi)+1)p' - 1$. Par hypothèse d'induction, $\pi_{L''}, f^{-1}(j+1-p) \models \psi_1$; Puisque $f^{-1}(j+1-p) = m' + h_P(\varphi)p'$, on peut appliquer le théorème 1.48, et on obtient que $\pi_{L''}, m' + (h_P(\varphi)+1)p' \models \psi_1$. Il s'ensuit immédiatement que $\pi_{L''}, m' + (h_P(\varphi)+1)p' - 1 \models \mathbf{X}\,\psi_1$, ce qui constitue le résultat voulu ;
- si $\psi = \mathbf{F}\,\psi_1$, il nous faut aussi distinguer les deux possibilités. Le raisonnement est similaire au précédent : puisque $(j, \mathbf{F}\,\psi_1)$ appartient à $S \cup T$, il existe dans $S \cup T$ un couple de la forme (i, ψ_1), avec soit $i \geqslant j$, soit $i, j \in [\![m+h_P(\varphi)p, m+(h_P(\varphi)+1)p-1]\!]$. Dans le premier cas, l'hypothèse d'induction permet d'aboutir à la conclusion recherchée ; dans le second cas, on sait qu'il existe un couple (i', ψ_1) dans $S \cup T$ tel que i' soit compris entre $m + h_P(\varphi)p$ et $m + (h_P(\varphi)+1)p$. L'hypothèse d'induction et le théorème 1.48 permettent de conclure ;
- si $\psi = \mathbf{X}^{-1}\,\psi_1$, le résultat provient directement de l'hypothèse d'induction ;
- si $\psi = \mathbf{F}^{-1}\,\psi_1$ également. □

Ce résultat nous permet d'obtenir un algorithme NP pour les problèmes « existentiels » : satisfaisabilité initiale et globale, et model checking existentiel initial et global. Les problèmes « universels » correspondants sont cependant plus complexes : d'après [SC85], le model checking existentiel initial de $\mathrm{L}^+(\mathbf{G}, \mathbf{X})$ est PSPACE-dur. Par dualité, on en déduit que le model checking universel initial du fragment $\mathrm{L}^+(\mathbf{F}, \mathbf{X})$ est (co-)PSPACE-dur, donc PSPACE-dur.

Théorème 1.60
Les quatre problèmes « existentiels » que nous étudions sont NP*-complets pour le fragment* $\mathrm{L}^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$.

Par dualité, à partir de ce théorème, on obtient que les quatre problèmes « universels » pour le fragment $\mathrm{L}^+(\mathbf{G}, \mathbf{X}, \mathbf{G}^{-1}, \mathbf{X}^{-1})$ sont co-NP-durs.

Le fragment $\mathrm{L}^+(\mathbf{G}, \mathbf{S}, \mathbf{X}^{-1})$. Nous étudions ici le problème de la satisfaisabilité initiale du fragment $\mathrm{L}^+(\mathbf{G}, \mathbf{S}, \mathbf{X}^{-1})$. On a les propriétés suivantes :

Lemme 1.61
Soit φ une formule de $\mathrm{L}^+(\mathbf{G}, \mathbf{S}, \mathbf{X}^{-1})$ construite à partir des propositions de Prop. *Soient a un état et π un chemin, tous deux étiquetés par des propositions de* Prop. *Alors*
- *si $a^\omega, 0 \models \varphi$, alors pour tout entier i, $a^\omega, i \models \varphi$;*

– *si* $\pi, 0 \models \varphi$*, alors* $(\pi_0)^\omega, 0 \models \varphi$.

Démonstration :

Nous prouvons séparément les deux implications : pour la première, supposons que $a^\omega, 0 \models \varphi$. Nous procédons par induction structurelle sur φ :

– si φ est une proposition atomique, le résultat est évident ;
– si φ est une conjonction ou une disjonction de deux sous-formules, alors l'hypothèse d'induction, appliquée aux sous-formules, donne le résultat ;
– si $\varphi = \mathbf{G}\,\psi$, alors le fait que $a^\omega, 0 \models \mathbf{G}\,\psi$ entraîne que $a^\omega, 0 \models \psi$. L'hypothèse d'induction assure que pour tout i, $a^\omega, i \models \psi$, et donc $a^\omega, i \models \mathbf{G}\,\psi$;
– si $\varphi = \varphi_1 \,\mathbf{S}\, \varphi_2$, on sait alors que $a^\omega, 0 \models \varphi_2$. L'hypothèse d'induction nous assure donc que φ_2 est satisfaite à toutes les positions du chemin a^ω. A fortiori, $\varphi_1 \,\mathbf{S}\, \varphi_2$ est vrai à toutes les positions le long de ce chemin ;
– le cas où $\varphi = \mathbf{X}^{-1}\,\psi$ n'est pas possible, puisqu'une telle formule ne peut pas être vraie dans l'état initial.

Montrons maintenant la deuxième implication, toujours par induction :

– si φ est une proposition atomique, le résultat est immédiat ;
– si φ est une conjonction ou une disjonction, c'est immédiat également ;
– si $\varphi = \mathbf{G}\,\psi$, on a alors $\pi, 0 \models \mathbf{G}\,\psi$, donc en particulier $\pi, 0 \models \psi$. On en déduit, par l'hypothèse d'induction, que $(\pi_0)^\omega, 0 \models \psi$. D'après le résultat précédent, on obtient que, pour tout i, $(\pi_0)^\omega, i \models \psi$, et donc $(\pi_0)^\omega, 0 \models \mathbf{G}\,\psi$;
– si $\varphi = \varphi_1 \,\mathbf{S}\, \varphi_2$, alors le fait que $\pi, 0 \models \varphi_1 \,\mathbf{S}\, \varphi_2$ entraîne que $\pi, 0 \models \varphi_2$, donc que $(\pi_0)^\omega \models \varphi_2$. Donc nécessairement, $(\pi_0)^\omega \models \varphi_1 \,\mathbf{S}\, \varphi_2$;
– si $\varphi = \mathbf{X}^{-1}\,\psi$, l'hypothèse $\pi, 0 \models \varphi$ ne peut pas être vraie.

$\square$

On en déduit immédiatement le résultat suivant :

Théorème 1.62
Le problème de la satisfaisabilité initiale du fragment $\mathrm{L}^+(\mathbf{G}, \mathbf{S}, \mathbf{X}^{-1})$ *est* NP*-complet.*

Nous verrons plus tard que les autres problèmes concernant ce fragment (et même des fragments plus petits) sont PSPACE-complets.

Le fragment $\mathrm{L}^+(\mathbf{U}, \mathbf{S})$. Une preuve similaire à la précédente nous permet de montrer que les problèmes de validité du fragment $\mathrm{L}^+(\mathbf{U}, \mathbf{S})$ sont co-NP-complets :

Lemme 1.63
Soient $\varphi \in \mathrm{L}^+(\mathbf{U}, \mathbf{S})$*,* a *un état et* π *un chemin, tous trois construits sur l'ensemble des propositions atomiques* Prop*. Alors*

– *s'il existe un entier* i *tel que* $a^\omega, i \not\models \varphi$*, alors pour tout entier* i*,* $a^\omega, i \not\models \varphi$ *;*
– *s'il existe un entier* i *tel que* $\pi, i \not\models \varphi$*, alors* $(\pi_i)^\omega, i \not\models \varphi$.

Démonstration :

Comme précédemment : pour la première implication,

– si φ est une proposition atomique, ou une conjonction ou disjonction de deux sous-formules, le résultat est immédiat ;
– si $\varphi = \varphi_1 \mathbf{U} \varphi_2$, alors le fait que $a^\omega, i \not\models \varphi_1 \mathbf{U} \varphi_2$ entraîne, en particulier, que $a^\omega, i \not\models \varphi_2$. L'hypothèse d'induction nous assure qu'aucune position du chemin a^ω ne satisfait φ_2, et on en déduit qu'aucun état ne peut satisfaire $\varphi_1 \mathbf{U} \varphi_2$;
– si $\varphi = \varphi_1 \mathbf{S} \varphi_2$, alors comme $a^\omega, i \not\models \varphi_1 \mathbf{S} \varphi_2$, on en déduit en particulier que $a^\omega, i \not\models \varphi_2$. Par induction, $a^\omega, i \not\models \varphi_2$ quel que soit i, et donc $a^\omega, i \not\models \varphi_1 \mathbf{S} \varphi_2$.

La deuxième implication se prouve de la même façon :
– le résultat est évident si φ est une proposition atomique, ou une conjonction ou une disjonction de sous-formules ;
– si $\varphi = \varphi_1 \mathbf{U} \varphi_2$, alors, du fait que $\pi, i \not\models \varphi_1 \mathbf{U} \varphi_2$, on déduit que $\pi, i \not\models \varphi_2$. Par induction, $(\pi_i)^\omega, i \not\models \varphi_2$, et d'après la première implication, pour tout j, $(\pi_i)^\omega, j \not\models \varphi_2$. Il est donc impossible que $\varphi_1 \mathbf{U} \varphi_2$ soit vrai dans la position i de $(\pi_i)^\omega$;
– de la même manière, si $\varphi = \varphi_1 \mathbf{S} \varphi_2$, si $\pi, i \not\models \varphi_1 \mathbf{S} \varphi_2$, alors $\pi, i \not\models \varphi_2$. D'où $(\pi_i)^\omega, i \not\models \varphi_2$, et $(\pi_i)^\omega, j \not\models \varphi_2$ quel que soit j. Donc $(\pi_i)^\omega, i \not\models \varphi_1 \mathbf{S} \varphi_2$

□

Il existe donc un contre-exemple de taille polynomiale pour toute formule de $\mathrm{L}^+(\mathbf{U}, \mathbf{S})$ non valide. On en déduit le théorème suivant :

Théorème 1.64
Les problèmes de validité (initiale et globale) du fragment $\mathrm{L}^+(\mathbf{U}, \mathbf{S})$ *sont co-*NP*-complets.*

En résumé, nous avons prouvé que les problèmes suivants sont NP-complets (ou co-NP-complets pour les problèmes universels) :
– tous les problèmes de validité, de satisfaisabilité et de model checking pour les fragments $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1})$ et $\mathrm{L}(\mathbf{X}, \mathbf{X}^{-1})$;
– les versions « initiales » des problèmes de validité, satisfaisabilité et model checking pour le fragment $\mathrm{L}(\mathbf{X}, \mathbf{S}, \mathbf{X}^{-1})$;
– les problèmes de satisfaisabilité et de model checking existentiel pour le fragment $\mathrm{L}^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$;
– par dualité, les problèmes de validité et de model checking universel pour le fragment $\mathrm{L}^+(\mathbf{G}, \mathbf{X}, \mathbf{G}^{-1}, \mathbf{X}^{-1})$;
– le problème de satisfaisabilité initiale de $\mathrm{L}^+(\mathbf{G}, \mathbf{S}, \mathbf{X}^{-1})$;
– les problèmes de validité de $\mathrm{L}^+(\mathbf{U}, \mathbf{S})$;

Bien entendu, ces résultats restent vrais pour les fragments non triviaux des logiques citées ci-dessus.

1.3.4 Les problèmes PSPACE-complets

Nous allons voir dans cette section qu'il existe un algorithme en espace polynomial pour les problèmes de vérification de la logique PLTL. Cela établit une borne supérieure pour la complexité de tous les problèmes que nous avons définis. Nous verrons ensuite que cette borne supérieure ne peut pas être améliorée pour tous les fragments que nous n'avons pas encore traités.

1.3.4.1 La borne supérieure

Un algorithme « naïf ». Dans [SC85], il est prouvé que la satisfaisabilité initiale et le model checking universel initial sont PSPACE-faciles pour L($\mathbf{U}$, $\mathbf{X}$, $\mathbf{S}$). Ce résultat est prouvé par l'intermédiaire d'un lemme affirmant que toute formule de PLTL admet un témoin ultimement périodique de taille exponentielle. [SC85] fournit ensuite un algorithme « à la volée » pour vérifier qu'un chemin ultimement périodique, choisi de manière non déterministe, satisfait la propriété à vérifier. Le terme « à la volée » signifie qu'il n'est pas nécessaire de stocker l'intégralité du chemin, et qu'il suffit de « deviner » les états les uns à la suite des autres. De cette façon, l'information à mémoriser est de taille polynomiale, et l'algorithme est dans PSPACE.

Par ailleurs, il est facile d'étendre ces résultats à la logique PLTL.

Algorithmes utilisant des tableaux. [LPZ85] fournit un nouvel algorithme pour la satisfaisabilité de PLTL. Étant donnée une formule φ, cet algorithme construit un graphe G dont les chemins, sous certaines conditions, satisfont initialement la formule φ. Cet algorithme nécessite cependant la construction de tout le graphe, et utilise donc un espace exponentiel. [KMMP93] a adapté cette méthode afin de ne construire qu'une partie du graphe. Mais l'algorithme proposé nécessite toujours un espace exponentiel.

Algorithmes utilisant la théorie des automates. Les méthodes à base d'automates sont assez similaires à celles utilisant des tableaux. Elles ont été développées principalement dans [WVS83], [VW94] et [Var96], pour la logique LTL.

- Dans [WVS83], les auteurs traduisent une formule φ de LTL en un automate de Büchi (la condition d'acceptation de l'automate de Büchi servant à s'assurer que les parties droites des modalités $\mathbf{U}$ sont effectivement satisfaites). L'automate est de taille exponentielle, mais il n'est pas nécessaire de le construire intégralement, puisque l'algorithme servant à tester la vacuité du langage reconnu par un automate de Büchi peut se faire « à la volée » [VW94] ;
- dans [Var96], la formule est transformée en un automate de Büchi *alternant*, de taille polynomiale, et pour lequel le test de la vacuité est dans PSPACE.

Dans les deux cas, l'algorithme obtenu termine en temps exponentiel, et utilise un espace polynomial.

Nous avons donc, de façon générale, le résultat suivant :

Théorème 1.65
Les huit problèmes de vérification que nous étudions sont PSPACE-*faciles pour la logique* PLTL *et tous ses fragments.*

1.3.4.2 Problèmes de pavage

Nous allons, dans toute la fin de ce chapitre, établir des réductions (utilisant un espace logarithmique) de problèmes PSPACE-complets à des problèmes de vérification de fragments de PLTL. Cela prouvera que les problèmes de vérification considérés sont PSPACE-durs, donc PSPACE-complets au vu de ce qui précède.

Les problèmes PSPACE-complets de départ seront des problèmes de pavage d'une grille extraits de [Har85], et adaptés à la situation.

Pour définir ces problèmes, nous aurons besoin des quelques définitions suivantes :

Définition 1.66
Soit C un ensemble, dont les éléments $c_0, \dots, c_p$ sont appelées couleurs. *Un* type de domino *est un élément de C^4, de la forme $d_i = (\mathrm{left}_i, \mathrm{right}_i, \mathrm{up}_i, \mathrm{down}_i)$. Nous définissons, sur l'ensemble des types de dominos, quatre fonctions* left, right, up *et* down, *retournant respectivement le premier, deuxième, troisième et quatrième élément de leur argument.*

Étant donné un ensemble T de types de dominos, une grille pavée *(ou un* pavage*)* de largeur m et de hauteur n *est une application G de $[\![1,m]\!] \times [\![1,n]\!]$ dans T. Une grille pavée est dite* correcte *si elle vérifie les deux conditions suivantes :*

- *pour tout entier i compris entre 1 et $m-1$, et pour tout j entre 1 et n,* $\mathrm{right}(G(i,j)) = \mathrm{left}(G(i,j))$ *;*
- *pour tout entier i compris entre 1 et m, et pour tout j entre 1 et $n-1$,* $\mathrm{up}(G(i,j)) = \mathrm{down}(G(i,j))$ *;*

Enfin, une grille pavée correcte G est infinie *lorsque deux de ses lignes sont identiques*[4]. *Une grille pavée correcte G, de $[\![1,m]\!] \times [\![1,n]\!]$ dans T, est* maximale *si elle est infinie ou s'il n'existe pas de grille pavée correcte G' de $[\![1,m]\!] \times [\![1,n+1]\!]$ dans T telle que $G'_{|[\![1,m]\!]\times[\![1,n]\!]} = G$.*

Le problème de départ est le suivant :

Définition 1.67 ([Har83, problème B2])

Pavage1 *:*	*entrée :*	*– un ensemble T de types de dominos* *– un entier m (en unaire)* *– deux couleurs c_d et c_f*
	sortie :	*existe-t-il un entier n et une grille pavée correcte G, de largeur m et de hauteur n, telle que* $\mathrm{down}(G(1,1)) = c_d$ *et* $\mathrm{up}(G(1,n)) = c_f$ *?*

Théorème 1.68 ([Har83])
Le problème Pavage1 *est* PSPACE*-complet.*

Ce résultat se prouve assez simplement, d'une part par un algorithme non déterministe « à la volée » en espace polynômial, et d'autre part par une réduction au problème de l'acceptation d'un mot par une machine de Turing polynômiale. Des présentations plus détaillées peuvent être trouvés à ce sujet dans [vEB83] et [vEB97].

Nous utiliserons plus facilement le problème suivant :

[4]C'est à dire qu'il existe deux entiers distincts i et j tels que $\forall k \in [\![1,m]\!].\ G(k,i) = G(k,j)$.

Définition 1.69

Pavage2 : *entrée :*
- *un ensemble T de types de dominos*
- *un entier m (en unaire)*
- *deux couleurs c_d et c_f*

sortie : *existe-t-il un entier n et une grille pavée correcte G, de largeur m et de hauteur n, et telle que* $\mathrm{down}(G(1,1)) = c_d$ *et* $\mathrm{up}(G(m,n)) = c_f$ *?*

La seule différence avec Pavage1 est que c'est la couleur $\mathrm{up}(G(m,n))$ qui est imposée ici, et non pas la couleur $\mathrm{up}(G(1,n))$. Ce nouveau problème est évidemment PSPACE-complet lui aussi.

Définition 1.70

Pavage3 : *entrée :*
- *un ensemble T de types de dominos*
- *un entier m (en unaire)*
- *deux couleurs c_d et c_f*

sortie : *existe-t-il une grille pavée correcte infinie G, de largeur m, vérifiant* $\mathrm{down}(G(1,1)) = c_d$ *et telle que, pour tout entier n,* $\mathrm{up}(G(m,n)) \neq c_f$ *?*

On peut facilement coder le problème de la non-vacuité du langage reconnu par une machine de Turing à ce problème, et inversement, on se convainc facilement qu'il existe un algorithme (non déterministe) en espace polynômial pour résoudre de problème.

Les problèmes suivants sont les problèmes duaux : ce sont donc des problèmes « universels », dans le sens où ils expriment une propriété sur tous les pavages. Ils nous serviront lors de réductions à nos problèmes universels :

Définition 1.71

Pavage4 : *entrée :*
- *un ensemble T de types de dominos*
- *un entier m (en unaire)*
- *deux couleurs c_d et c_f*

sortie : *est-ce que, pour toutes les grilles pavées correctes infinies G, de largeur m, vérifiant* $\mathrm{down}(G(1,1)) = c_d$*, il existe un entier n tel que* $\mathrm{up}(G(m,n)) = c_f$ *?*

C'est le problème dual du problème Pavage3. Puisque PSPACE = co-PSPACE, ce problème est lui aussi PSPACE-complet.

Définition 1.72

Pavage5 : *entrée :*
- *un ensemble T de types de dominos*
- *un entier m (en unaire)*
- *deux couleurs c_d et c_f*

sortie : *est-ce que, pour toutes les grilles pavées correctes infinies G, de largeur m, vérifiant* $\mathrm{down}(G(1,1)) = c_d$ *et pour tout entier n, on a* $\mathrm{up}(G(m,n)) \neq c_f$ *?*

Ce problème est le dual du problème Pavage2.

1.3.4.3 PSPACE-dureté pour certains fragments de PLTL

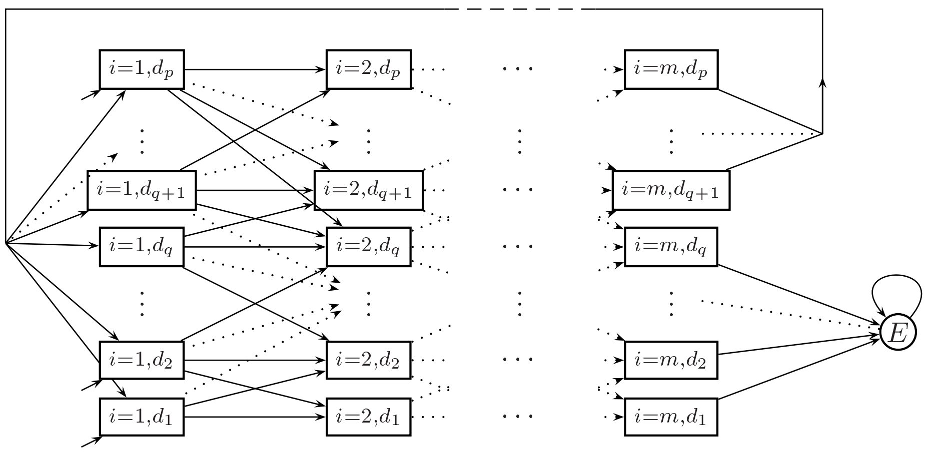
Le fragment $L^+(\mathbf{U})$. Nous avons vu que les problèmes de validité pour le fragment $L^+(\mathbf{U}, \mathbf{S})$ sont co-NP-complets. Nous verrons ci-dessous que les autres problèmes sont PSPACE-durs, même sans utiliser la modalité **S**.

Lemme 1.73
Le problème Pavage2 *se réduit au model checking existentiel initial du fragment* $L^+(\mathbf{U})$.

Démonstration :

Soit (C, T, m, c_d, c_f) une instance du problème de pavage. Nous notons $d_1, d_2, \ldots, d_q$ les éléments de T. À cette instance, nous associons le structure de Kripke représentée à la figure 1.8. Dans cette structure, les états initiaux sont ceux étiquetés par $i = 1$ et par un

Figure 1.8 La structure de Kripke K_1 associée au problème Pavage2



type de domino d tel que $\text{down}(d) = c_d$. Les états qui ont une transition vers l'état E sont ceux qui sont étiquetés par $i = m$ et par un type de domino d tel que $\text{up}(d) = c_f$.

Formellement, cette structure de Kripke $K_1 = (S, S_0, l, R)$ est définie à partir de

$$\text{Prop} \stackrel{\text{def}}{=} \{E\} \cup \{i = k \mid k \in [\![1, m]\!]\} \cup \{d_l \mid l \in [\![1, n]\!]\}$$

par les relations suivantes :

- $S = ([\![1, m]\!] \times [\![1, n]\!]) \cup \{E\}$;
- $S_0 = S \cap (\{1\} \times \{i \in [\![1, n]\!] \mid \text{down}(d_i) = c_d\})$;
- $l(E) = \{E\}$, et $l(k, l) = \{i = k, d_l\}$;
- $R = \{((k, l), (k+1, l')) \mid k \in [\![1, m-1]\!],\ l \in [\![1, n]\!]\}$
  $\cup \{((m, l), (1, l')) \mid l \in [\![1, n]\!]\}$
  $\cup \{((m, l), E) \mid \text{up}(d_l) = c_f\}$

À tout chemin dans cette structure de Kripke, on peut associer une application de $[\![1, m]\!] \times [\![1, n]\!]$ dans T de manière évidente : chaque portion de chemin d'un état étiqueté par $i = 1$ jusqu'à un état étiqueté par $i = m$ représente le contenu d'une ligne de la grille pavée. Tout chemin se terminant dans l'état E correspond donc à une grille pavée dont la couleur down du domino inférieur gauche est c_d, par construction des états initiaux, et dont la couleur up du domino supérieur droit est c_f.

Afin de compléter la réduction, il nous reste donc à imposer à ce chemin de correspondre à une grille pavée *correcte*. Cela peut être fait au moyen des deux formules de $\mathrm{L}^+(\mathbf{U})$ ci-dessous :

$$\varphi_{horiz} \stackrel{\text{def}}{=} \left(\bigwedge_{k=1}^{m-1} \bigwedge_{d \in T} (i = k \ \wedge \ d) \Rightarrow \big(i = k \ \mathbf{U}\,(i = k+1 \ \wedge \bigvee_{\substack{d' \in T \\ \mathrm{left}(d') = \mathrm{right}(d)}} d')\big) \right) \mathbf{U}\, E$$

$$\varphi_{vert} \stackrel{\text{def}}{=} \left(\bigwedge_{k=1}^{m} \bigwedge_{d \in T} (i = k \ \wedge \ d) \Rightarrow \big(i = k \ \mathbf{U}\,(\neg(i = k) \ \wedge \right. \\ \left. ((\neg i = k) \ \mathbf{U} \ (E \ \vee (i = k \ \wedge \bigvee_{\substack{d' \in T \\ \mathrm{down}(d') = \mathrm{up}(d)}} d'))))\big) \right) \mathbf{U}\, E$$

On se convainc facilement que

- d'une part, si le problème Pavage2 a une solution pour l'instance de départ, alors toute grille pavée correcte solution du problème correspond à un chemin dans K_1 satisfaisant les formules φ_{horiz} et φ_{vert} ;
- d'autre part, un chemin dans K_1 vérifiant φ_{horiz} et φ_{vert} correspond à une grille pavée correcte et vérifiant les conditions initiale et finale du problème Pavage2. ☐

Lemme 1.74
Le problème Pavage2 *se réduit à la satisfaisabilité initiale de* $\mathrm{L}^+(\mathbf{U})$.

Démonstration :
La proposition 1.42 de la page 43 nous fournit une réduction du problème du model checking existentiel initial à celui de la satisfaisabilité initiale. Ce résultat est basé sur le codage d'une structure de Kripke en une formule de PLTL « équivalente » (*cf.* proposition 1.41). Cette réduction n'est cependant pas applicable si on se restreint, comme ici, au fragment $\mathrm{L}^+(\mathbf{U})$: elle nécessite en effet l'utilisation des modalités $\mathbf{X}$ et $\mathbf{G}$.

L'utilisation du $\mathbf{X}$ peut être évitée en quotientant les chemins par la relation de stutter-équivalence (*cf.* définition 1.25). L'utilisation du $\mathbf{G}$, par contre, ne peut pas être contournée.

Nous allons donc devoir revenir au problème Pavage2 de départ, et le réduire directement à la satisfaisabilité de $\mathrm{L}^+(\mathbf{U})$. Pour ce faire, nous réutilisons les formules φ_{horiz} et φ_{vert} de la preuve précédente, et nous allons coder la structure de Kripke K_1 en une formule de $\mathrm{L}^+(\mathbf{U})$. Cela nous évite le problème du $\mathbf{G}$ évoqué précédemment, puisque nous ne nous intéressons, dans cette structure, qu'aux chemins se terminant dans l'état E. Les formules du type $\mathbf{G}\,\varphi$ de la preuve de la proposition 1.41 seront donc codées par des formules du type $\varphi\,\mathbf{U}\,E$.

Pour coder la structure de Kripke de la figure 1.8, nous ajoutons une proposition atomique P_s pour chaque état de S. En réutilisant les formules ψ^{state} et ψ^{label} de la preuve de la proposition 1.41, page 42, nous définissons alors les formules suivantes :

$$
\begin{aligned}
\varphi_{K_1}^{init} &\stackrel{\text{def}}{=} \Big(\bigvee_{s\in S_0} P_s\Big)\\
\varphi_{K_1}^{state} &\stackrel{\text{def}}{=} \psi^{state}\,\mathbf{U}\ E\\
\varphi_{K_1}^{label} &\stackrel{\text{def}}{=} \psi^{label}\,\mathbf{U}\ E\\
\varphi_{K_1}^{trans} &\stackrel{\text{def}}{=} \Big(\bigvee_{(s,s')\in R} (P_s\wedge\ (P_s\,\mathbf{U}\,P_{s'}))\Big)\,\mathbf{U}\ E
\end{aligned}
$$

Nous noterons donc

$$\varphi_{K_1} \stackrel{\text{def}}{=} \varphi_{K_1}^{init}\wedge\varphi_{K_1}^{state}\wedge\varphi_{K_1}^{label}\wedge\varphi_{K_1}^{trans}.$$

Alors tout chemin satisfaisant $\varphi_{K_1}\wedge\varphi_{horiz}\wedge\varphi_{vert}$ correspond à une grille pavée correcte satisfaisant les conditions du problème Pavage2. Inversement, si le problème Pavage2 admet une solution, on peut associer à cette solution un chemin vérifiant cette formule. □

De ces résultats, on déduit le théorème suivant :

Théorème 1.75
Les quatre problèmes « existentiels » sont PSPACE*-durs pour le fragment* $\mathrm{L}^+(\,\mathbf{U}\,)$.

C'est clair pour la satisfaisabilité initiale et le model checking existentiel initial. Pour la version « globale » de ces problème, le résultat découle de la proposition 1.8, puisque $\mathrm{L}^+(\,\mathbf{U}\,)$ est inclus dans LTL.

Passons maintenant aux problèmes universels.

Lemme 1.76
Le problème Pavage4 *se réduit au model checking universel initial de* $\mathrm{L}^+(\,\mathbf{U}\,)$.

Démonstration :
Pour cette preuve, nous considérons toujours la structure de Kripke décrite à la figure 1.8. Le problème Pavage4 est vrai si, et seulement si, tous les chemins (infinis) qui représentent une grille pavée correcte se terminent dans l'état E. Un chemin de la structure de Kripke doit donc soit atteindre E, soit être « incorrect » à un endroit. Nous définissons donc les formules suivantes :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{k=1}^{m-1}\bigvee_{d\in T} \top\,\mathbf{U}\ (i=k\ \wedge\ d\ \wedge (i=k\ \ \mathbf{U}\,(i=k+1\ \wedge\neg \bigvee_{\substack{d'\in T\\ \mathrm{left}(d')=\mathrm{right}(d)}} d')))$$

$$
\begin{aligned}
\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{k=1}^{m}\bigvee_{d\in T} \top\,\mathbf{U}\ (i=k\ \wedge\ d\ \wedge (i=k\ \ \mathbf{U}\,(\neg i=k\ \wedge\\
(\neg i=k)\,\mathbf{U}\,(i=k\ \wedge\neg \bigvee_{\substack{d'\in T\\ \mathrm{down}(d')=\mathrm{up}(d)}} d')))
\end{aligned}
$$

Un chemin satisfaisant $\varphi_{\neg horiz} \vee \varphi_{\neg vert}$ correspond donc à une grille pavée *non correcte*. Le fait que toute grille pavée correcte vérifiant la condition initiale vérifie aussi la condition finale est équivalent au fait que tous les chemins infinis de la structure de Kripke K_1 vérifient la formule $\varphi_{\neg horiz} \vee \varphi_{\neg vert} \vee \top \, \mathbf{U} \, E$. On a ainsi réduit le problème Pavage4 à un problème de model checking universel initial d'une formule de $\mathrm{L}^+(\,\mathbf{U}\,)$. □

On en déduit le théorème suivant :

Théorème 1.77
Les problèmes de model checking universels du fragment $\mathrm{L}^+(\,\mathbf{U}\,)$ *sont* PSPACE*-complets.*

Nous avons vu précédemment (théorème 1.64, page 60) que les problèmes de validité pour $\mathrm{L}^+(\,\mathbf{U}\,)$ sont co-NP-complets, ce qui complète l'étude de ce fragment.

Les fragments $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ et $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$. Nous savons que les problèmes de type « existentiel » pour $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,,\mathbf{F}^{-1}\,,\mathbf{X}^{-1}\,)$ sont NP-complets (*cf.* théorème 1.60, page 58). Nous en avons déduit que les problèmes « universels » pour $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,,\mathbf{G}^{-1}\,,\mathbf{X}^{-1}\,)$ sont co-NP-complets. Nous allons voir ici que les problèmes universels pour les fragments $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ et les problèmes existentiels pour $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ sont PSPACE-complets. Nous verrons ensuite des résultats similaires pour les fragments $\mathrm{L}_s^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$ et $\mathrm{L}_s^+(\mathbf{G}\,,\mathbf{X}^{-1}\,)$.

Lemme 1.78
Le problème Pavage4 *se réduit au model checking universel de* $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$.

Démonstration :
La réduction est similaire à celle de la preuve du lemme 1.74. Nous avons donc simplement à écrire des formules de $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ équivalentes aux formules $\varphi_{\neg horiz}$ et $\varphi_{\neg vert}$ de cette preuve :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{d \in T} \mathbf{F}\left(d \;\wedge \mathbf{X}\,(\neg E) \wedge \mathbf{X}\left(\neg \bigvee_{\substack{d' \in T \\ \mathrm{left}(d')=\mathrm{right}(d)}} d'\right)\right)$$

$$\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{d \in T} \mathbf{F}\left(d \;\wedge \mathbf{X}^{\,m}(\neg E) \wedge \mathbf{X}^{\,m}\left(\neg \bigvee_{\substack{d' \in T \\ \mathrm{down}(d')=\mathrm{up}(d)}} d'\right)\right)$$

Nous obtenons alors l'équivalence suivante : tous les chemins (infinis) vérifient $\varphi_{\neg horiz} \vee \varphi_{\neg vert} \vee \mathbf{F}\, E$ si, et seulement si, toutes les grilles pavées correctes infinies vérifiant la condition initiale (la couleur down du domino inférieur gauche est c_0) vérifient la condition finale (la couleur up de l'un des dominos de droite est c_1). □

Théorème 1.79
Le model checking universel du fragment $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ *est* PSPACE*-complet.*

Comme pour $\mathrm{L}^+(\,\mathbf{U}\,)$, $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ est un fragment de LTL, et la proposition 1.8 s'applique : chaque problème « global » est équivalent au problème initial correspondant.

On en déduit immédiatement, par dualité, le théorème suivant :

Théorème 1.80
Le model checking existentiel du fragment $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ *est* PSPACE*-complet.*

La proposition 1.41 de la page 42 fournit une réduction de la satisfaisabilité au model checking existentiel. Cette réduction n'utilise que les modalités $\mathbf{G}$ et $\mathbf{X}$. On en déduit le théorème suivant :

Théorème 1.81
Le problème de la satisfaisabilité du fragment $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ *est* PSPACE*-complet.*

Par dualité, nous obtenons enfin :

Théorème 1.82
Le problème de la validité du fragment $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ *est* PSPACE*-complet.*

Les fragments $\mathrm{L}_s^+(\mathbf{G}\,,\mathbf{X}^{-1}\,)$ et $\mathrm{L}_s^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$. Les réductions pour ces fragments sont similaires aux réductions pour les fragments $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ et $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$. D'après le théorème 1.60, page 58, nous savons que les problèmes « existentiels » pour $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$ sont NP-complets, et que les problèmes « universels » pour $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}^{-1}\,)$ sont co-NP-complets.

Lemme 1.83
Le problème Pavage4 *se réduit au model checking universel initial de* $\mathrm{L}_s^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$.

Démonstration :
Il suffit d'adapter la preuve du lemme 1.78, en réécrivant les formules $\varphi_{\neg horiz}$ et $\varphi_{\neg vert}$:

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{d\in T} \mathbf{F}\left(\mathbf{X}^{-1}\, d\ \wedge (\neg E) \wedge \left(\neg \bigvee_{\substack{d'\in T\\ \mathrm{left}(d')=\mathrm{right}(d)}} d'\right)\right)$$

$$\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{d\in T} \mathbf{F}\left((\mathbf{X}^{-1}\,)^m d\ \wedge (\neg E) \wedge \left(\neg \bigvee_{\substack{d'\in T\\ \mathrm{down}(d')=\mathrm{up}(d)}} d'\right)\right)$$

□

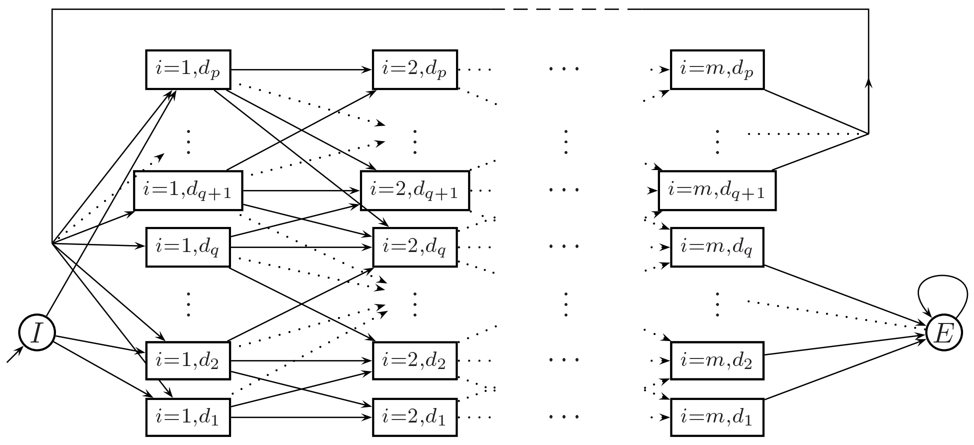
Lemme 1.84
Le problème Pavage4 *se réduit au model checking universel global de* $\mathrm{L}_s^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$.

Démonstration :
Pour cette réduction, nous allons adapter légèrement la structure, afin de pouvoir détecter le premier état. Pour ce faire, nous considérons la structure de la figure 1.9, qui utilise des propositions atomiques de $\mathrm{Prop} \cup \{I\}$:

Nous conservons les formules $\varphi_{\neg horiz}$ et $\varphi_{\neg vert}$ de la preuve précédente. On a alors l'équivalence suivante : le problème Pavage4 est satisfait si, et seulement si, tous les chemins infinis, à toutes les positions, vérifient la formule suivante :

$$I \vee \mathbf{X}^{-1}\,\neg I \vee \varphi_{\neg vert} \vee \varphi_{\neg horiz} \vee \mathbf{F}\,E.$$

Figure 1.9 La structure de Kripke K_2.



La formule $I \vee \mathbf{X}^{-1}\neg I$ est vraie partout sauf dans le deuxième état de toute exécution de la structure de Kripke K_3. La formule ci-dessus impose donc que $\varphi_{\neg vert} \vee \varphi_{\neg horiz} \vee \mathbf{F}\,E$ soit vérifiée au premier passage dans un état étiqueté par $i = 1$, pour tout chemin. □

Théorème 1.85
Le model checking universel de $\mathrm{L}^+_s(\mathbf{F}\,, \mathbf{X}^{-1}\,)$ *est* PSPACE*-complet.*

On a le résultat dual suivant :

Théorème 1.86
Le model checking existentiel de $\mathrm{L}^+_s(\mathbf{G}\,, \mathbf{X}^{-1}\,)$ *est* PSPACE*-complet.*

Démonstration :

Ce théorème ne peut pas se déduire du précédent par dualité, parce que $\mathbf{X}^{-1}$ n'est pas exactement son propre dual ($\neg\mathbf{X}^{-1}\top$ n'est pas équivalent à $\mathbf{X}^{-1}\neg\top$).

Nous allons donc réduire le problème Pavage3 à ces deux problèmes (initial et global) de model checking. Le problème Pavage3 admet une solution si, et seulement si, il existe un chemin infini dans la structure de Kripke K_2 qui soit correct et ne passe jamais dans l'état E. Nous écrivons donc :

$$\varphi_{horiz} \stackrel{\text{def}}{=} \mathbf{G}\left(\bigwedge_{d\in T}\big((d \wedge \neg(i=1)) \Rightarrow \bigvee_{\substack{d'\in T\\ \mathrm{right}(d')=\mathrm{left}(d)}} d'\big)\right)$$

$$\varphi_{vert} \stackrel{\text{def}}{=} \mathbf{G}\left(I \vee (\bigvee_{i=1}^{m}(\mathbf{X}^{-1}\,)^i I) \vee \bigwedge_{d\in T}\big(d \Rightarrow \bigvee_{\substack{d'\in T\\ \mathrm{up}(d')=\mathrm{down}(d)}}(\mathbf{X}^{-1}\,)^m d'\big)\right)$$

Alors Pavage3 admet une solution si, et seulement si, il existe un chemin dans K_2 vérifiant initialement $\varphi_{horiz} \wedge \varphi_{vert} \wedge \mathbf{G}\,\neg E$. Ce qui est également équivalent à l'existence d'un chemin π dans K_2 et d'une position i tels que $\pi, i \models I \wedge \varphi_{horiz} \wedge \varphi_{vert} \wedge \mathbf{G}\,\neg E$.

Nous avons donc réduit le problème Pavage3 aux deux problèmes de model checking que nous considérions. Ces deux problèmes sont donc PSPACE-durs, et donc PSPACE-complets d'après le théorème 1.65. □

Nous avons vu précédemment que la satisfaisabilité initiale de $\mathrm{L}^+(\mathbf{G}, \mathbf{X}^{-1})$ est NP-complète. Ce n'est pas le cas pour la satisfaisabilité globale, qui est bien un problème PSPACE-complet :

Théorème 1.87
La satisfaisabilité globale du fragment $\mathrm{L}^+_s(\mathbf{G}, \mathbf{X}^{-1})$ *est* PSPACE*-complète.*

Démonstration :
Nous allons utiliser la réduction de la preuve précédente, en codant, en plus, la structure de Kripke K_2 à l'aide d'une formule de $\mathrm{L}^+_s(\mathbf{G}, \mathbf{X}^{-1})$. Cette preuve suit le même cheminement que la preuve du lemme 1.74 : nous ajoutons les propositions atomiques $P_{(k,d)}$ pour chaque entier $k \leqslant m$ et pour chaque $d \in T$, ainsi que P_I et P_E. Nous notons S l'ensemble des états de la structure de Kripke K_2 : $S = \{(k,d) \mid 1 \leqslant k \leqslant m, d \in T\} \cup \{I, E\}$. Nous notons R la relation de transition, et l la fonction d'étiquetage des états. Nous écrivons alors les formules suivantes :

$$\begin{aligned}
\varphi_{K_2}^{init} &\stackrel{\text{def}}{=} \mathbf{X}^{-1}\, P_I \wedge \bigvee_{\substack{d \in T \\ \text{down}(d) = c_d}} P_{1,d} \\
\varphi_{K_2}^{state} &\stackrel{\text{def}}{=} \mathbf{G}\,\mathbf{X}^{-1}\, \psi^{state} \\
\varphi_{K_2}^{label} &\stackrel{\text{def}}{=} \mathbf{G}\,\mathbf{X}^{-1}\, \psi^{label} \\
\varphi_{K_2}^{trans} &\stackrel{\text{def}}{=} \mathbf{G}\Big(\bigvee_{(s,s') \in R} (P_{s'} \wedge \mathbf{X}^{-1}\, P_s) \wedge \mathbf{G}\,\neg P_I \Big)
\end{aligned}$$

Nous noterons

$$\varphi_{K_2} \stackrel{\text{def}}{=} \varphi_{K_2}^{init} \wedge \varphi_{K_2}^{state} \wedge \varphi_{K_2}^{label} \wedge \varphi_{K_2}^{trans}.$$

Intuitivement, le fait de considérer la satisfaisabilité globale, et non initiale, nous permet d'« initialiser » le chemin (avec $\mathbf{X}^{-1}\, P_I$) et de s'assurer qu'on ne reste pas infiniment souvent dans cet état initial (avec $\mathbf{G}\,\neg P_I$).

Maintenant, le problème Pavage3 admet une solution si, et seulement si, la formule suivante est (globalement) satisfaisable :

$$\varphi_{K_2} \wedge \varphi_{horiz} \wedge \varphi_{vert} \wedge \mathbf{G}\,\neg E$$

où φ_{horiz} et φ_{vert} sont les formules de la preuve précédente. □

Pour en terminer avec ces fragments, nous montrons le résultat suivant :

Lemme 1.88
Le problème Pavage4 *se réduit aux problèmes validité initiale et globale de* $\mathrm{L}^+(\mathbf{F}, \mathbf{X}^{-1})$.

Démonstration :
Nous allons, pour cette réduction, exprimer que tous les chemins (infinis) vérifient toujours l'une des conditions suivantes :
- soit ce n'est pas un chemin de la structure de Kripke K_2 ;
- soit ce chemin ne correspond pas à un pavage correct ;
- soit ce chemin arrive en E.

Si tous les chemins vérifient l'une de ces trois conditions, alors tous les chemins correspondant à un pavage correct infini satisfaisant la condition initiale vérifieront la condition finale. Inversement, si tous les pavages corrects infinis satisfaisant la condition initiale vérifient la condition finale, alors tous les chemins vérifieront l'une des conditions énoncées ci-dessus.

Il nous reste donc à exprimer, avec des formules de $\mathrm{L}^+(\mathbf{F}, \mathbf{X}^{-1})$, les conditions ci-dessus :
- pour exprimer qu'un chemin ne correspond pas à un chemin de la structure de Kripke K_2, nous écrirons les quatre formules suivantes :

$$\begin{aligned}
\varphi^{init}_{\neg K_2} &\stackrel{\text{def}}{=} \neg P_I \\
\varphi^{state}_{\neg K_2} &\stackrel{\text{def}}{=} \mathbf{F}\, \neg \psi^{state} \\
\varphi^{label}_{\neg K_2} &\stackrel{\text{def}}{=} \mathbf{F}\, \neg \psi^{label} \\
\varphi^{trans}_{\neg K_2} &\stackrel{\text{def}}{=} \mathbf{F}\, \Big(\big(\neg P_I \wedge \bigwedge_{(s,s')\in R} (\neg P_{s'} \vee \mathbf{X}^{-1} \neg P_s)\big) \vee (P_I \wedge \mathbf{X}^{-1} \top) \Big)
\end{aligned}$$

  La disjonction de des quatre formules, notée $\varphi_{\neg K_2}$, est initialement satisfaite exactement pour les chemins qui ne sont pas issus de la structure K_2.
- pour exprimer qu'un chemin, supposé être issu de K_2, ne représente pas un pavage correct, nous écrirons :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{d\in T} \mathbf{F}\, \Big(\mathbf{X}^{-1} d \ \wedge (\neg E) \wedge \Big(\neg \bigvee_{\substack{d'\in T \\ \mathrm{left}(d')=\mathrm{right}(d)}} d' \Big) \Big)$$

$$\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{d\in T} \mathbf{F}\, \Big((\mathbf{X}^{-1})^m d \ \wedge (\neg E) \wedge \Big(\neg \bigvee_{\substack{d'\in T \\ \mathrm{down}(d')=\mathrm{up}(d)}} d' \Big) \Big)$$

- enfin, pour exprimer que notre chemin, supposé représenter un pavage correct, arrive bien dans l'état E, il suffit d'écrire $\mathbf{F}\, E$.

Si tous les chemins vérifient, initialement, la disjonction de ces formules, soit G une grille pavée correcte infinie vérifiant la condition initiale $\mathrm{down}(G(1,1)) = c_d$. On associe à cette grille le chemin correspondant dans K_2. Ce chemin ne vérifie ni $\varphi_{\neg K_2}$, ni $\varphi_{\neg horiz} \vee \varphi_{\neg vert}$. D'après l'hypothèse, ce chemin passe dans l'état E.

Inversement, supposons que tous les pavages corrects infinis satisfaisant la condition initiale satisfont également la condition finale. Soit π un chemin. Si ce n'est pas un chemin issu de K_2, il vérifie initialement $\varphi_{\neg K_2}$. Autrement, ce chemin peut être associé à un pavage. Si ce pavage n'est pas correct, alors le chemin vérifie $\varphi_{\neg horiz} \vee \varphi_{\neg vert}$ initialement. Si le pavage est correct, et si le chemin ne vérifie pas $\mathbf{F}\, E$ initialement, alors le pavage est infini et ne vérifie pas la condition finale. Ce qui contredit l'hypothèse.

Cette réduction est également valable pour la validité globale : supposons que tous les chemins, à toutes les positions, vérifient la disjonction de ces formules. Alors $\varphi_{\neg K_2}$ est satisfaite dans tous les états, sauf l'état initial, puisque tout état non initial vérifie soit $\neg P_I$, soit $P_I \wedge \mathbf{X}^{-1}\, \top$. Il ne nous reste donc qu'à traiter le cas de l'état initial, ce qui se fait exactement de la même manière que précédemment.

Réciproquement, supposons que tous les pavages corrects infinis satisfaisant la condition initiale satisfont également la condition finale, et considérons un chemin π. Les états non initiaux vérifient nécessairement la formule $\varphi_{\neg K_2}$, pour la même raison que précédemment. Et le cas de l'état initial se traite comme ci-dessus. □

Théorème 1.89
Les problèmes de validité sont PSPACE*-complets pour le fragment* $\mathrm{L}^+_s(\mathbf{F}\,, \mathbf{X}^{-1}\,)$.

Les fragments $\mathrm{L}^+_s(\mathbf{F}\,, \mathbf{S}\,)$ et $\mathrm{L}^+_s(\mathbf{G}\,, \mathbf{S}\,)$. Nous allons voir dans un premier temps que, symétriquement au cas de $\mathrm{L}^+(\,\mathbf{U}\,)$, la version globale des problèmes de satisfaisabilité et de model checking des fragments contenant $\mathbf{S}$ sont PSPACE-durs.

Lemme 1.90
Le problème Pavage2 *se réduit au model checking existentiel global de* $\mathrm{L}^+(\,\mathbf{S}\,)$.

Démonstration :

Nous utilisons la structure de Kripke K_2 de la page 1.9. Il existe un pavage correct satisfaisant les conditions initiale et finale si, et seulement si, il existe un chemin dans K_2 satisfaisant les conditions suivantes :

$$\varphi_{horiz} \stackrel{\text{def}}{=} E \wedge E\, \mathbf{S} \Big(\neg E \wedge (\bigwedge_{k=2}^{m} \bigwedge_{d\in T} (i = k\ \wedge\ d) \Rightarrow \big(i = k\ \ \mathbf{S}\,(i = k-1\ \wedge \bigvee_{\substack{d'\in T\\ \mathrm{right}(d')=\mathrm{left}(d)}} d')\big)\big)\, \mathbf{S}\, I\Big)$$

$$\begin{aligned}\varphi_{vert} \stackrel{\text{def}}{=} E \wedge E\, \mathbf{S} \Bigg(&\neg E \wedge (\bigwedge_{k=1}^{m} \bigwedge_{d\in T} (i = k\ \wedge\ d) \Rightarrow \big(i = k\ \ \mathbf{S}\,(\neg(i = k) \wedge \\ &\qquad ((\neg i = k)\ \ \mathbf{S}\,(I \vee (i = k\ \wedge \bigvee_{\substack{d'\in T\\ \mathrm{up}(d')=\mathrm{down}(d)}} d')))))\big)\, \mathbf{S}\, I\Bigg)\end{aligned}$$

Ces formules sont, à peu de choses près, « symétriques » à celle que nous avions utilisées dans la preuve du lemme 1.73. On se convainc facilement que le problème Pavage2 admet une solution si, et seulement si, il existe un chemin π issu de K_2, et une position i tels que la conjonction des deux formules ci-dessus est satisfaite à la position i le long de π. □

Ce résultat s'étend à la satisfaisabilité globale, en ajoutant aux formules de la démonstration précédente un codage de la structure de Kripke K_2 :

Lemme 1.91
Le problème Pavage2 *se réduit à la satisfaisabilité globale de* $\mathrm{L}^+(\mathbf{S})$.

Démonstration :
Pour chaque état s de K_2, nous ajoutons une proposition atomique, P_s. Le codage de la structure de Kripke K_2 s'écrit alors comme suit :

$$\begin{aligned}
\varphi_{K_2}^{init} &\stackrel{\text{def}}{=} E \wedge \neg P_I \wedge \top\, \mathbf{S}\ (P_I \wedge \psi^{state} \wedge \psi^{label}) \\
\varphi_{K_2}^{state} &\stackrel{\text{def}}{=} \psi^{state}\, \mathbf{S}\ (P_I \wedge \psi^{state} \wedge \psi^{label}) \\
\varphi_{K_2}^{label} &\stackrel{\text{def}}{=} \psi^{label}\, \mathbf{S}\ (P_I \wedge \psi^{state} \wedge \psi^{label}) \\
\varphi_{K_2}^{trans} &\stackrel{\text{def}}{=} \Big(\bigvee_{(s,s')\in R} (P_{s'} \wedge\ (P_{s'}\, \mathbf{S}\ P_s)) \Big)\, \mathbf{S}\ (P_I \wedge \psi^{state} \wedge \psi^{label})
\end{aligned}$$

Nous noterons donc

$$\varphi_{K_2} \stackrel{\text{def}}{=} \varphi_{K_2}^{init} \wedge \varphi_{K_2}^{state} \wedge \varphi_{K_2}^{label} \wedge \varphi_{K_2}^{trans}.$$

Alors il existe un chemin π et une position i dans laquelle $\varphi_{K_2} \wedge \varphi_{horiz} \wedge \varphi_{vert}$ est satisfaite si, et seulement si, le problème Pavage2 admet une solution. □

On a donc le théorème suivant :

Théorème 1.92
Les problèmes existentiels globaux concernant le fragment $\mathrm{L}^+(\mathbf{S})$ *sont* PSPACE*-durs.*

Nous allons maintenant prouver que c'est le cas aussi pour le model checking universel global, de la même manière que pour le fragment $\mathrm{L}^+(\mathbf{U})$:

Lemme 1.93
Le problème Pavage4 *se réduit au problème du model checking universel global du fragment* $\mathrm{L}^+(\mathbf{S})$.

Démonstration :
Nous utilisons simplement des formules « symétriques » aux formules écrites dans la preuve du lemme 1.76. L'idée de la preuve est exactement la même, mais nous raisonnons cette fois sur la structure K_2. Nous écrivons donc les formules suivantes :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{k=1}^{m-1} \bigvee_{d\in T} ((i = k+1 \wedge d) \wedge (i = k+1\ \ \mathbf{S}\,(i = k \wedge \neg \bigvee_{\substack{d'\in T \\ \mathrm{right}(d')=\mathrm{left}(d)}} d')))$$

$$\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{k=1}^{m} \bigvee_{d\in T} ((i = k \wedge d) \wedge (i = k\ \ \mathbf{S}\,((\neg i = k) \wedge ((\neg i = k)\, \mathbf{S}\ \ i = k \wedge \neg \bigvee_{\substack{d'\in T \\ \mathrm{down}(d')=\mathrm{up}(d)}} d'))))$$

□

Tous les problèmes concernant $\mathrm{L}^+(\mathbf{S})$ sont donc classifiés, puisque nous savons déjà que la validité est co-NP-complète, et que les problèmes initiaux sont NC^1-complets.

Nous allons maintenant ajouter une modalité du futur à la modalité **S**. Les seuls cas qu'il nous reste à traiter sont l'ajout de **F** et l'ajout de **G**, puisque les résultats concernant l'ajout de **U** ou **X** peuvent être déduits de tout ce qui précède.

Lemme 1.94
Le problème Pavage2 *se réduit au model checking existentiel initial de* $\mathrm{L}^+_s(\mathbf{F}, \mathbf{S})$.

Démonstration :
Nous reprenons la preuve du lemme 1.90, en modifiant légèrement les formules utilisées. Nous notons

$$\psi_{horiz} \stackrel{\text{def}}{=} \neg E \wedge \Big(\bigwedge_{k=2}^{m} \bigwedge_{d \in T} (i = k \ \wedge\ d) \Rightarrow \big(i = k \ \mathbf{S}\, (i = k-1 \ \wedge \bigvee_{\substack{d' \in T \\ \mathrm{right}(d') = \mathrm{left}(d)}} d')\big)\Big)\, \mathbf{S}\, I$$

$$\psi_{vert} \stackrel{\text{def}}{=} \neg E \wedge \Big(\bigwedge_{k=1}^{m} \bigwedge_{d \in T} (i = k \ \wedge\ d) \Rightarrow \big(i = k \ \mathbf{S}\, (\neg(i = k) \wedge ((\neg i = k)\ \mathbf{S}\, (I \vee (i = k \ \wedge \bigvee_{\substack{d' \in T \\ \mathrm{up}(d') = \mathrm{down}(d)}} d')))))\Big)\, \mathbf{S}\, I$$

Nous posons alors

$$\varphi_{horiz} \stackrel{\text{def}}{=} \mathbf{F}\, (E \wedge E\, \mathbf{S}\, \psi_{horiz}) \qquad \varphi_{vert} \stackrel{\text{def}}{=} \mathbf{F}\, (E \wedge E\, \mathbf{S}\, \psi_{vert})$$

La conjonction de ces deux formules est satisfaite dans l'état initial de la structure K_2 si, et seulement si, le problème Pavage2 admet une solution. □

Lemme 1.95
Le problème Pavage3 *se réduit au model checking existentiel initial de* $\mathrm{L}^+_s(\mathbf{G}, \mathbf{S})$.

Démonstration :
En reprenant les notations de la preuve précédente, et en écrivant

$$\varphi_{horiz} \stackrel{\text{def}}{=} \mathbf{G}\, (\psi_{horiz}) \qquad \varphi_{vert} \stackrel{\text{def}}{=} \mathbf{G}\, (\psi_{vert})$$

Le problème Pavage3 a une solution si, et seulement si, il existe un chemin issu de K_2 vérifiant initialement $\varphi_{horiz} \wedge \varphi_{vert}$. □

Lemme 1.96
Le problème Pavage4 *se réduit au model checking universel initial de* $\mathrm{L}^+_s(\mathbf{F}, \mathbf{S})$.

Démonstration :
Nous devons exprimer le fait que le pavage associé à un chemin n'est pas correct. Nous

écrirons pour cela :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \bigvee_{k=2}^{m} \bigvee_{d \in T} \mathbf{F} \Big(i = k \ \wedge (i = k \ \mathbf{S} \, (i = k-1 \ \wedge d)) \wedge \Big(\neg \bigvee_{\substack{d' \in T \\ \mathrm{left}(d') = \mathrm{right}(d)}} d' \Big) \Big)$$

$$\varphi_{\neg vert} \stackrel{\text{def}}{=} \bigvee_{k=1}^{m} \bigvee_{d \in T} \mathbf{F} \Big(i = k \ \wedge (i = k \ \mathbf{S} \, (\neg i = k \ \wedge (\neg i = k) \, \mathbf{S} \, (i = k \ \wedge d))) \wedge \Big(\neg \bigvee_{\substack{d' \in T \\ \mathrm{down}(d') = \mathrm{up}(d)}} d' \Big) \Big)$$

Alors tous les chemins vérifient la disjonction $\varphi_{\neg horiz} \vee \varphi_{\neg vert} \vee \mathbf{F}\, E$ si, et seulement si, les pavages corrects vérifient la condition finale. □

Lemme 1.97
Le problème Pavage5 *se réduit au model checking universel initial de* $\mathrm{L}_s^+(\mathbf{G}\,, \mathbf{S}\,)$.

Démonstration :
Nous allons exprimer que tous les chemins de K_2 vérifient la propriété suivante : soit ils n'arrivent jamais dans l'état E, soit ils y arrivent mais ne correspondent alors pas à un chemin correct. Nous écrivons donc les formules suivantes :

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \mathbf{G} \Big(E \Rightarrow \top \, \mathbf{S} \Big(\bigvee_{k=2}^{m} \bigvee_{d \in T} i = k \ \wedge (i = k \ \mathbf{S} \, (i = k-1 \ \wedge d)) \wedge \Big(\neg \bigvee_{\substack{d' \in T \\ \mathrm{left}(d') = \mathrm{right}(d)}} d' \Big) \Big) \Big)$$

$$\varphi_{\neg horiz} \stackrel{\text{def}}{=} \mathbf{G} \Big(E \Rightarrow \top \, \mathbf{S} \Big(\bigvee_{k=1}^{m} \bigvee_{d \in T} i = k \ \wedge (i = k \ \mathbf{S} \, (\neg i = k \ \wedge (\neg i = k) \, \mathbf{S} \, (i = k \ \wedge d))) \wedge \Big(\neg \bigvee_{\substack{d' \in T \\ \mathrm{down}(d') = \mathrm{up}(d)}} d' \Big) \Big) \Big)$$

□

Lemme 1.98
Le problème Pavage4 *se réduit à la satisfaisabilité initiale de* $\mathrm{L}_s^+(\mathbf{F}\,, \mathbf{S}\,)$.

Démonstration :
Pour cette preuve, nous partons de la preuve du lemme 1.94, et nous écrivons une

formule satisfaite exactement par les chemins issus de K_2 :

$$\begin{aligned}
\varphi_{K_2}^{init} &\stackrel{\text{def}}{=} P_I \wedge \psi^{state} \wedge \psi^{label} \wedge \mathbf{F}\, E\\
\varphi_{K_2}^{state} &\stackrel{\text{def}}{=} \mathbf{F}\,(E \wedge (\psi^{state}\,\mathbf{S}\,(P_I \wedge \psi^{state} \wedge \psi^{label})))\\
\varphi_{K_2}^{label} &\stackrel{\text{def}}{=} \mathbf{F}\,(E \wedge (\psi^{label}\,\mathbf{S}\,(P_I \wedge \psi^{state} \wedge \psi^{label})))\\
\varphi_{K_2}^{trans} &\stackrel{\text{def}}{=} \mathbf{F}\,\Big(E \wedge \Big(\bigvee_{(s,s')\in R} (P_{s'} \wedge\ (P_{s'}\,\mathbf{S}\,P_s))\Big)\,\mathbf{S}\,(P_I \wedge \psi^{state} \wedge \psi^{label})\Big)
\end{aligned}$$

Nous noterons donc

$$\varphi_{K_2} \stackrel{\text{def}}{=} \varphi_{K_2}^{init} \wedge \varphi_{K_2}^{state} \wedge \varphi_{K_2}^{label} \wedge \varphi_{K_2}^{trans}.$$

En reprenant les formules du lemme 1.94, il existe un chemin vérifiant $\varphi_{K_2} \wedge \varphi_{horiz} \wedge \varphi_{vert}$ si, et seulement si, il existe un pavage correct qui satisfait les conditions initiale et finale.

□

En résumé, nous avons prouvé que les problèmes suivants sont PSPACE-complets :

– tous les problèmes de satisfaisabilité et de model checking pour les fragments $\mathrm{L}^+(\,\mathbf{U}\,)$ et $\mathrm{L}^+(\,\mathbf{S}\,)$;
– les problèmes de validité et de model checking universel pour les fragments $\mathrm{L}^+(\mathbf{F}\,,\mathbf{X}\,)$ et $\mathrm{L}_s^+(\mathbf{F}\,,\mathbf{X}^{-1}\,)$;
– les problèmes de satisfaisabilité et de model checking existentiel pour les fragments $\mathrm{L}^+(\mathbf{G}\,,\mathbf{X}\,)$ et $\mathrm{L}_s^+(\mathbf{G}\,,\mathbf{X}^{-1}\,)$;
– les problèmes de model checking des fragments $\mathrm{L}_s^+(\mathbf{F}\,,\,\mathbf{S}\,)$ et $\mathrm{L}_s^+(\mathbf{G}\,,\,\mathbf{S}\,)$;
– le problème de satisfaisabilité initiale du fragment $\mathrm{L}_s^+(\mathbf{F}\,,\,\mathbf{S}\,)$.

Bien entendu, ces résultats restent vrais pour les logiques contenant les fragments cités ci-dessus.

Le tableau de la figure 1.10 récapitule les principaux résultats que nous avons cités ou établis dans cette section. Ces résultats s'obtiennent assez facilement par « inclusion » ou par dualité, à partir de ceux que nous avons prouvés dans ce chapitre. Par exemple : le fragment $\mathrm{L}(\,\mathbf{U}\,)$ contenant le fragment $\mathrm{L}^+(\,\mathbf{U}\,)$, la satisfaisabilité de $\mathrm{L}(\,\mathbf{U}\,)$ est PSPACE-complète. Par dualité, on obtient que la validité de $\mathrm{L}(\,\mathbf{U}\,)$ est PSPACE-complète. Celle de $\mathrm{L}(\,\mathbf{U}\,,\,\mathbf{S}\,)$ est donc également PSPACE-complète.

Par ailleurs, nous n'avons pas distingués les problèmes initiaux et globaux dans ce tableau : pour tous les fragments qui y sont présentés, les deux types de problèmes ont la même complexité.

Conclusion

En comparant le tableau récapitulatif des résultats de complexité et le diagramme comparant l'expressivité des fragments de PLTL, nous constatons un résultat général : l'utilisation du passé dans la logique temporelle du temps linéaire apporte de l'expressivité mais n'augmente pas la complexité de la vérification.

L'apport d'expressivité est un fait nouveau : le passé n'a pas été très étudié jusqu'à maintenant principalement à cause du fait que le pouvoir d'expression de PLTL est le même

Figure 1.10 Récapitulatif des principaux résultats de complexité

<table>
<tr><td rowspan="2"></td><td colspan="2">model checking</td><td rowspan="2">satisfaisabilité</td><td rowspan="2">validité</td></tr>
<tr><td>existentiel</td><td>universel</td></tr>
<tr><td>$L(\mathbf{X})$</td><td colspan="4" rowspan="2">(co-)NP-complet</td></tr>
<tr><td>$L(\mathbf{F})$, $L(\mathbf{F}, \mathbf{F}^{-1})$</td></tr>
<tr><td>$L^+(\mathbf{F}, \mathbf{X})$</td><td rowspan="2"></td><td rowspan="7"></td><td rowspan="2"></td><td rowspan="2">PSPACE-c.</td></tr>
<tr><td>$L^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$</td></tr>
<tr><td>$L^+(\mathbf{U})$, $L^+(\mathbf{U}, \mathbf{S})$</td><td rowspan="5"></td><td rowspan="5">PSPACE-complet</td><td>co-NP-complet</td></tr>
<tr><td>$L(\mathbf{F}, \mathbf{X})$</td><td rowspan="4"></td></tr>
<tr><td>$L(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$</td></tr>
<tr><td>$L(\mathbf{U})$, $L(\mathbf{U}, \mathbf{S})$</td></tr>
<tr><td>LTL, PLTL</td></tr>
</table>

que celui de LTL. Nous avons vu, dans notre étude, que cette conclusion doit être nuancée, voire inversée, puisque d'une part l'ajout des modalités « symétriques » dans certains fragments restreints augmente le pouvoir d'expression, mais aussi parce que le passé apporte de la concision dans les formules, et permet de rester plus proche de la langue naturelle.

Concernant la complexité, nous avons réussi à classer tous les fragments que nous étudiions en utilisant un nombre assez restreint de techniques (toutes les réductions se font à partir de problèmes de pavage similaires, et les preuves de « *témoin polynomial* » sont toutes assez semblables). Cependant, nous n'avons pas réussi à caractériser simplement (par exemple, par un critère syntaxique simple) la complexité des fragments de PLTL.

Du point de vue de la complexité de la vérification, notre analyse confirme les résultats déjà connus sur le sujet : l'ajout de modalités du passé n'augmente pas la complexité des problèmes de vérification. Nous constatons en effet, pour l'ensemble des fragments que nous avons définis, que l'ajout des modalités « symétriques » des modalités autorisées dans un fragment n'augmente pas la complexité des problèmes de vérification que nous avons considérés. Nous constatons également que les fragments « stratifiés » ont toujours la même complexité que les fragments « généraux ».

En conclusion, cette étude semble conforter l'idée que l'ajout de modalités du passé ne coûte pas plus cher du point de vue de la vérification, alors qu'il simplifie indéniablement la spécification des propriétés, rendant parfois cette spécification exponentiellement plus courte.

Pour en savoir plus...

La logique temporelle du temps linéaire a déjà fait l'objet de très nombreux travaux. Le point de vue de l'expressivité a été assez peu abordé cependant. Quelques résultats d'expressivité en fonction des modalités autorisées sont donnés dans [Lar94, EVW02]. Récemment, [TW96, TW98, EW00, TW02] ont établi que les restrictions sur la hauteur temporelle de **U** et **S** engendrent une hiérarchie stricte, et fournissent un algorithme permettant de décider à quel niveau de la hiérarchie se trouve une propriété (*i.e.* combien d'emboîtements de **U** et **S** sont nécessaires pour exprimer la propriété). Enfin, [MMSKR94] donne un algorithme de traduction de la logique L(**U** , **S**) dans la logique L(**U**).

Concernant la complexité de la vérification de ces logiques et de leurs fragments, les premiers résultats proviennent de [SC85]. [RMD+92] prouve que la satisfaisabilité de L(**U** , **S**) est PSPACE-complète, et donne un algorithme de décision à base d'automates pour ce fragment. Le fragment L(**I** , **X**) est étudié dans [SR99] : sa validité est co-NP-complète et son model checking est PSPACE-complet. [DS02] étudie la complexité des fragments en fonction du nombre de propositions atomiques et de la hauteur temporelle des formules. Le problème du model checking d'un chemin (section 1.3.2) a, par contre, été peu étudié. En particulier, la complexité exacte de ce problème pour la logique PLTL n'est pas connu. Nous reviendrons sur ce problème au chapitre suivant.

Le passé a été étudié dans d'autres logiques temporelles : [KP95, LS95, LS00] étudient les différentes sémantiques possible pour les opérateurs du passé dans le cas du temps arborescent, ainsi que la complexité des logiques qui en résultent. [Var98b] étudie l'apport du passé dans le μ-calcul arborescent, et développe le concept d'« automates alternants bidirectionnels » pour traiter la vérification de cette logique. Enfin, [AH93] étudie l'ajout du passé dans la logique TLTL, qui est une logique temporelle temporisée du temps linéaire, et sur laquelle nous reviendrons au chapitre 4.

Du point de vue « pratique », plusieurs outils ont été développés :

– le model checker SPIN utilise la théorie des automates, et en particulier la transformation des formules de LTL en automates de Büchi dont nous avons déjà parlé à la section 1.2.2.2. [GPVW96] présente l'algorithme « à la volée » qui permet de ne construire que la partie « utile » de l'automate. Une version améliorée de l'algorithme de traduction est étudiée dans [GO01].
– le model checker SMV est un model checker « symbolique » présenté dans [McM93b], et utilisant la représentation par *binary decision diagrams* (BDD, *cf.* [Bry86]). Il était initialement orienté vers la logique du temps arborescent, mais [CGH94] donne une méthode permettant de l'adapter au model checking de propriétés de LTL.

Ces deux model checkers ne traitent que la logique LTL. Il n'existe pas, à notre connaissance, de model checker permettant de traiter les formules de PLTL. Quelques travaux ont néanmoins été écrits sur ce sujet : [KPR98] propose un algorithme de model checking de PLTL sur des structures de Kripke « avec équité ». [MS00] donne une construction d'un automate alternant (*cf.* section 2.2.1.1) « équivalent » à une formule de PLTL. Une construction efficace de l'automate de Büchi associé à une formule de PLTL, utilisant des automates aternant très faibles bidirectionnels, a été proposée dans [GO03]. Cette construction fournit un automate de Büchi généralisé à $O(2^{|\varphi|+1})$ états.

CHAPITRE 2

La logique NLTL

Introduction

Dans le premier chapitre, nous avons vu que l'ajout de modalités du passé dans la logique LTL pouvait, dans un certain sens, apporter de l'expressivité. Cependant, il peut être utile, dans certains cas, de ne pas tenir compte du passé, de l'« oublier ». Considérons par exemple une alarme A, chargée de « surveiller » un système S. Ce système S sera modélisé uniquement par deux états, un état `normal` et un état `probleme`. L'alarme A pourra être dans l'état `attente`, dans l'état `alarme`, ou dans l'état `reinit`. Le système global a alors six états. Il est modélisé à la figure 2.1. Si on veut spécifier que « à tout moment, si l'alarme se déclenche, alors il y a eu un problème auparavant », nous écrirons

$$\mathbf{G}\,(\texttt{alarme} \Rightarrow \mathbf{F}^{-1}\,\texttt{probleme}).$$

L'alarme pouvant être réinitialisée, nous voulons en fait exprimer que, après chaque réinitialisation, si l'alarme se déclenche, c'est qu'un problème est survenu :

$$\mathbf{G}\,(\texttt{reinit} \Rightarrow (\mathbf{G}\,(\texttt{alarme} \Rightarrow \mathbf{F}^{-1}\,\texttt{probleme})).$$

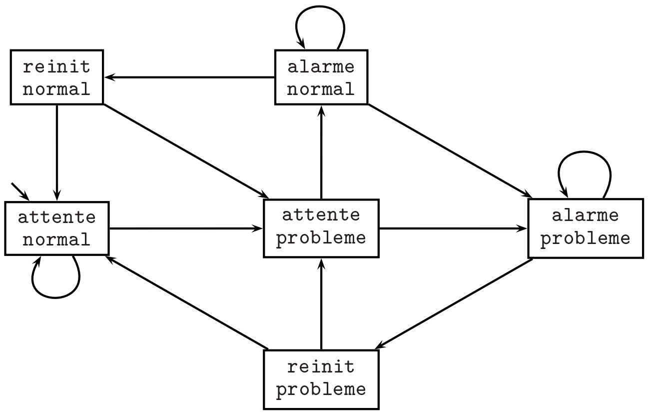
Cette propriété n'est pas la propriété que nous voulions exprimer, car un problème survenu *avant* la dernière réinitialisation peut servir de « témoin » pour la sous-formule $\mathbf{F}^{-1}\,\texttt{probleme}$. Nous allons étudier, dans ce chapitre, une méthode permettant de pallier ce problème.

2.1 Définition et premiers résultats

2.1.1 La logique NLTL

Nous commençons par définir un nouvel opérateur permettant, comme le suggère ce qui précède, de réinitialiser le passé.

Figure 2.1 Modélisation d'une alarme



2.1.1.1 Syntaxe et sémantique de NLTL

Afin d'« oublier le passé », nous allons utiliser l'opérateur $\mathbf{N}$, appelé *now*, défini dans [LS95]. Cet opérateur a été étudié principalement dans le cadre de la logique temporelle du temps arborescent, dans [LS00] notamment.

Définition 2.1 ([LS95])
L'opérateur $\mathbf{N}$ *est défini par la sémantique suivante :*

$$\pi, i \models \mathbf{N}\,\varphi \quad \Leftrightarrow \quad \pi^i, 0 \models \varphi.$$

Cet opérateur n'est pas une modalité au sens propre du terme, parce qu'une modalité ne modifie normalement que la position où l'on évalue une formule, et pas le chemin le long duquel on l'évalue.

La logique NLTL est définie en ajoutant cet opérateur à PLTL :

Définition 2.2
La logique NLTL *est la logique* $\mathrm{L}(\mathbf{U}, \mathbf{X}, \mathbf{S}, \mathbf{X}^{-1}, \mathbf{N})$.

Avec ce nouvel opérateur, on peut exprimer la propriété que nous voulions tester dans l'introduction :

$$\mathbf{G}\,(\texttt{reinit} \Rightarrow \mathbf{N}\,\mathbf{G}\,(\texttt{alarme} \Rightarrow \mathbf{F}^{-1}\,\texttt{probleme})).$$

L'opérateur $\mathbf{N}$ permet aussi d'exprimer la modalité $\mathbf{U}$ à partir de $\mathbf{F}$ et $\mathbf{F}^{-1}$ (pour la logique LTL) : pour toute formules φ et ψ de LTL,

$$\varphi\,\mathbf{U}\,\psi \equiv \mathbf{F}\,\varphi \wedge \mathbf{N}\,\mathbf{G}\,(\varphi \vee \mathbf{F}^{-1}\,\psi).$$

On notera que cette équivalence n'est pas vraie si φ ou ψ contiennent des modalités du passé, puisque l'opérateur $\mathbf{N}$ pourrait modifier leur évaluation. Notons également que l'opérateur $\mathbf{N}$ ne permet pas d'exprimer la modalité $\mathbf{S}$ avec $\mathbf{F}$ et $\mathbf{F}^{-1}$.

2.1.1.2 Propriétés de l'opérateur $\mathbf{N}$

Les résultats suivants présentent quelques caractéristiques de l'opérateur $\mathbf{N}$:

Proposition 2.3 ([LS95, Propositions 6.1 et 6.3])
Soient φ et ψ deux formules de NLTL. *Alors*

$$\begin{aligned} \mathbf{N}\,\neg\varphi &\equiv \neg\mathbf{N}\,\varphi & \qquad \mathbf{N}\,(\varphi \vee \psi) &\equiv \mathbf{N}\,\varphi \vee \mathbf{N}\,\psi \\ \mathbf{N}\,\mathbf{X}^{-1}\,\varphi &\equiv \bot & \mathbf{N}\,(\varphi\,\mathbf{S}\,\psi) &\equiv \mathbf{N}\,\psi \\ \mathbf{N}\,\mathbf{N}\,\varphi &\equiv \mathbf{N}\,\varphi & \mathbf{N}\,\psi &\equiv \psi \quad \text{si } \psi \in \mathsf{LTL} \end{aligned}$$

Toutes ces équivalences sont valables initialement et globalement. Elles se prouvent de manière immédiate en se ramenant à la sémantique des différents opérateurs.

L'opérateur $\mathbf{N}$ fournit également un lien entre les équivalences globale et initiale (de la même façon que la modalité $\mathbf{I}$, à la proposition 1.18) :

Proposition 2.4
Soient φ et ψ deux formules de NLTL. *On a alors l'équivalence suivante :*

$$\varphi \equiv_i \psi \text{ si, et seulement si, } \mathbf{N}\,\varphi \equiv_g \mathbf{N}\,\psi$$

Là encore, le résultat s'obtient très facilement en revenant aux définitions des équivalences et de l'opérateur $\mathbf{N}$.

2.1.2 Expressivité de NLTL

Nous allons voir dans cette section que, tout comme les modalités du passé, l'opérateur $\mathbf{N}$ n'apporte pas, à proprement parler, d'expressivité, mais permet d'écrire certaines spécifications de façon plus simple et plus concise.

2.1.2.1 Expressivité

Théorème 2.5 ([LS95])
La logique NLTL *n'est pas plus expressive que la logique* LTL.

Démonstration :
Le théorème 1.19, à la page 30, nous assure que toute formule φ de PLTL est initialement équivalente à une formule de LTL. Donc une formule $\mathbf{N}\,\varphi$ peut se réécrire en une formule globalement équivalente de la forme $\mathbf{N}\,\psi$ où ψ appartient à LTL, grâce à la proposition 2.4. Dans ce cas, l'opérateur $\mathbf{N}$ n'a aucun effet, puisque ψ ne dépend pas du passé (*cf.* proposition 1.8). En appliquant ces traductions à toutes les sous-formules du type $\mathbf{N}\,\varphi$, et grâce aux équivalences de la proposition 2.3, on traduit une formule de NLTL en une formule de PLTL globalement équivalente, qui peut à son tour être traduite en une formule de LTL initialement équivalente. □

2.1.2.2 Concision

Tout comme entre PLTL et LTL, nous allons mettre un évidence un saut de concision entre NLTL et PLTL. Plus précisément, on a le résultat suivant :

Théorème 2.6 ([LMS02b, théorème 3.3])
Il existe une famille de formules (φ_n) *de* NLTL *telle que* $|\varphi_n|$ *soit un polynôme en* n*, et telle que pour toute famille* (ψ_n) *de formules de* PLTL*, si* $\psi_n \equiv \varphi_n$ *quel que soit* n*, alors* $|\psi_n|$ *est en* $2^{\Omega(n)}$.

Démonstration :
La preuve est basée sur celle du théorème 1.31 : nous considérons la formule suivante :

$$\varphi_n = \mathbf{G}\,\mathbf{N}\,\mathbf{G}\Big(\big(\bigwedge_{i=1}^{n}\big(p_i \;\Leftrightarrow\; \mathbf{F}^{-1}(\neg\mathbf{X}^{-1}\top \wedge p_i)\big)\big) \Rightarrow \big(p_0 \;\Leftrightarrow\; \mathbf{F}^{-1}(\neg\mathbf{X}^{-1}\top \wedge p_0)\big)\Big).$$

Cette formule exprime la propriété que « toute paire d'état ayant les mêmes valeurs pour les propositions p_1 à p_n ont également même valeur pour la proposition p_0 ». On sait (d'après [EVW97]) que cette propriété ne peut s'exprimer dans PLTL qu'au moyen de formules de taille exponentielle en n. □

Il est important de noter que les sauts de concisions ne sont pas « cumulatifs », *i.e.* on ne peut pas affirmer, en alliant ce théorème au théorème 1.31, que le saut de concision entre NLTL et LTL est doublement exponentiel.

2.1.2.3 Les fragments

Théorème 2.7
Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{N})$ *a la même expressivité (en ce qui concerne l'équivalence initiale) que les fragments* $\mathrm{L}(\mathbf{U})$ *et* $\mathrm{L}(\mathbf{U}, \mathbf{S})$*, et peut être exponentiellement plus succinct.*

Démonstration :
Nous avons déjà vu que, si φ et ψ sont deux formules de LTL, on a l'équivalence

$$\varphi\,\mathbf{U}\,\psi \equiv \mathbf{F}\,\psi \wedge \mathbf{N}\,\mathbf{G}\,(\varphi \vee \mathbf{F}^{-1}\,\psi).$$

On sait par ailleurs que le fragment $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{N})$ peut se traduire dans LTL. D'après la proposition 1.27, et puisque toute formule de $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{N})$ est stutter-invariante, la formule de LTL équivalente sera également stutter-invariante, donc sera elle-même équivalente à une formule de $\mathrm{L}(\mathbf{U})$ (d'après un résultat de [PW97], dont nous avons déjà parlé à la page 34).

Concernant la concision, il suffit de remarquer que les formules φ_n de la preuve du théorème 2.6 peuvent s'écrire de manière équivalente :

$$\varphi_n = \mathbf{G}\,\mathbf{N}\,\mathbf{G}\Big(\big(\bigwedge_{i=1}^{n}\big(p_i \;\Leftrightarrow\; \mathbf{I}\,p_i\big)\big) \Rightarrow \big(p_0 \;\Leftrightarrow\; \mathbf{I}\,p_0\big)\Big).$$

□

Pour les mêmes raisons, on a le résultat suivant :

Théorème 2.8
Le fragment $\mathrm{L}(\mathbf{U}, \mathbf{S}, \mathbf{N})$ *a la même expressivité que* $\mathrm{L}(\mathbf{U})$ *et* $\mathrm{L}(\mathbf{U}, \mathbf{S})$ *pour l'équivalence initiale, mais il peut être exponentiellement plus succinct.*

2.1.3 Autres méthodes permettant d'oublier le passé

Il existe déjà plusieurs méthodes, plus ou moins directes, permettant de réinitialiser le passé. Nous en présentons quelques-unes ici. Nous allons voir que chacune de ces méthodes entraine une explosion de la complexité des problèmes de vérification.

2.1.3.1 La logique du premier ordre

Du fait que NLTL se traduit dans LTL, il découle que NLTL se traduit dans la logique du premier ordre (FOMLO). Il existe une traduction linéaire de NLTL dans FO^4, comme le montre la proposition suivante :

Proposition 2.9
Soit $\varphi \in$ NLTL. *Il existe une formule* $f_\varphi(i,j) \in \mathrm{FO}^4$, *de taille linéaire en la taille de* φ, *telle que*

$$\forall \pi.\ \forall i \in \mathbb{N}. \quad \pi^j, i-j \models \varphi \ \Leftrightarrow\ \pi \models f_\varphi(i,j).$$

Démonstration :
La construction se fait de la manière suivante :

$$\begin{aligned}
f_p(i,j) &\stackrel{\text{def}}{=} p\\
f_{\neg\varphi}(i,j) &\stackrel{\text{def}}{=} \neg f_\varphi(i,j)\\
f_{\varphi\vee\psi}(i,j) &\stackrel{\text{def}}{=} f_\varphi(i,j) \vee f_\psi(i,j)\\
f_{\varphi\,\mathbf{U}\,\psi}(i,j) &\stackrel{\text{def}}{=} \exists k.\ \big(k \geqslant i \wedge f_\psi(k,j) \wedge \forall i \leqslant l \leqslant k.\ f_\varphi(l,j)\big)\\
f_{\mathbf{X}\,\varphi}(i,j) &\stackrel{\text{def}}{=} f_\varphi(i+1,j)\\
f_{\varphi\,\mathbf{S}\,\psi}(i,j) &\stackrel{\text{def}}{=} \exists k.\ \big(j \leqslant k \leqslant i \wedge f_\psi(k,j) \wedge \forall k \leqslant l \leqslant i.\ f_\varphi(l,j)\big)\\
f_{\mathbf{X}^{-1}\varphi}(i,j) &\stackrel{\text{def}}{=} i > j \wedge f_\varphi(i-1,j)\\
f_{\mathbf{N}\,\varphi}(i,j) &\stackrel{\text{def}}{=} f_\varphi(i,i)
\end{aligned}$$

On peut montrer que chacune de ces définitions conserve l'équivalence que nous voulons prouver. La construction est donc correcte, et utilise au plus quatre variables. □

En fait, puisque NLTL se traduit dans LTL, toute formule de NLTL est équivalente à une formule de FO^3. Dans tous les cas, cela ne donne pas non plus une méthode efficace pour vérifier la logique NLTL, puisque la vérification de FO^3 est non élémentaire.

2.1.3.2 L'opérateur Chop

L'opérateur *chop*, noté **C** et défini dans [HKP82], a la sémantique suivante : $\pi, 0 \models \varphi\,\mathbf{C}\,\psi$ si, et seulement si, il existe un entier j tel que la portion (finie) de chemin comprise entre π_0 et π_{j-1} satisfait initialement ψ et que le suffixe π^j satisfait initialement φ. Cet opérateur n'a pas, à notre connaissance, été étudié avec des modalité du passé, ce qui explique que sa sémantique n'est définie que dans la position initiale du chemin.

Un algorithme pour la vérification de la logique L($\mathbf{U}$, $\mathbf{X}$, $\mathbf{C}$) est donné dans [RP86]. Cet algorithme, à base de tableaux, est non élémentaire. Cependant, d'après [HP85], la complexité de ce problème est non élémentaire.

Afin d'étudier cet opérateur avec des modalités du passé, nous devons en modifier la sémantique. Ce problème est évoqué à la fin de [RP86], mais aucune sémantique n'est proposée. Nous définissons l'opérateur $\Leftrightarrow$ avec la sémantique suivante : $\pi, i \models \varphi \Leftrightarrow \psi$ si, et seulement si, il existe $j \geqslant i$ tel que, si π' est le préfixe de π jusqu'à l'état π_{j-1}, on a $\pi^j, 0 \models \psi$ et, si $j > i$, $\pi', i \models \varphi$. Avec cet opérateur, on peut facilement coder l'opérateur $\mathbf{N}$, puisque $\mathbf{N}\,\varphi \equiv \bot \Leftrightarrow \varphi$. On peut également coder l'opérateur $\mathbf{C}$, puisque $\psi\,\mathbf{C}\,\varphi \equiv_i \psi \Leftrightarrow \varphi$. Il s'ensuit que la logique L($\mathbf{U}$, $\mathbf{X}$, $\mathbf{S}$, $\mathbf{X}^{-1}$, $\Leftrightarrow$) est non élémentaire.

2.1.3.3 LTL avec quantification sur les propositions

La logique temporelle linéaire avec quantification sur les propositions a été définie dans [Sis83]. Comme son nom l'indique, elle autorise, dans ses formules, à « quantifier » (universellement ou existentiellement) sur l'étiquetage, par certaines propositions, du chemin le long duquel on évalue la formule.

La syntaxe est donnée par

$$\mathsf{QLTL} \ni \varphi, \psi \ ::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\psi \mid \exists\, p.\varphi \mid \exists\, q.\varphi \mid \ \ldots \ \mid p \mid q \mid \ \ldots$$

où p, q, ... sont des éléments de Prop. La sémantique de $\exists$ est :

$$\pi, i \models \exists\, p.\varphi \quad \Leftrightarrow \quad \exists \pi'.\ \Big(\pi', i \models \varphi \text{ et } \forall q \in \text{Prop} \smallsetminus \{p\}.\ \forall j \in \mathbb{N}.\ \pi(j)(q) = \pi'(j)(q)\Big).$$

La formule $\exists\, p.\varphi$ peut se lire « il existe un étiquetage du chemin par la proposition p de telle sorte que φ soit vérifiée ».

L'opérateur $\forall$ est défini comme étant le dual de $\exists$:

$$\forall\, p.\varphi \equiv \neg\exists\, p.\neg\varphi$$

et signifie que φ est vérifiée quel que soit l'étiquetage choisi pour la proposition p.

[SVW87] montre que la vérification de cette logique est non élémentaire, et en étudie des restrictions. [Ete99] étudie les fragments EQLTL et AQLTL, définis de la façon suivante :

$$\mathsf{EQLTL} \ni \xi \ ::= \ \psi \mid \varphi \mid \ \ldots \ \mid \exists\, p.\xi \mid \exists\, q.\xi \mid \ \ldots$$

où $\psi, \psi, \ldots$ appartiennent à PLTL, et $p, q, \ldots$ appartiennent à Prop. De même,

$$\mathsf{AQLTL} \ni \xi \ ::= \ \psi \mid \varphi \mid \ \ldots \ \mid \forall\, p.\xi \mid \forall\, q.\xi \mid \ \ldots$$

Cela signifie qu'une formule de EQLTL (respectivement AQLTL) est une formule de LTL précédée d'une quantification existentielle (respectivement universelle) sur certaines propositions. La quantification ne peut pas, pour ces logiques, se trouver « à l'intérieur » de la formule.

On définit de même les logiques EQPLTL et AQPLTL, dans lesquelles les formules $\varphi, \psi, \dots$ appartiennent à PLTL.

On montre assez facilement que la vérification de toutes ces logiques peut se faire dans PSPACE.

Par ailleurs, on a les traductions suivantes :

$$\begin{aligned}
\exists\, q_1 \dots \exists\, q_p.\varphi(\alpha\, \mathbf{S}\, \beta, q_1, \dots, q_p) \ &\equiv_i \ \exists\, q_0.\exists\, q_1 \dots \exists\, q_p.\varphi(q_0, q_1, \dots, q_p) \wedge (q_0 \Leftrightarrow \beta) \wedge \\
&\qquad \mathbf{G}\,(q_0 \Rightarrow \mathbf{X}\,((\alpha \vee \beta) \Leftrightarrow q_0)) \wedge \mathbf{G}\,((\neg q_0) \Rightarrow \mathbf{X}\,(\beta \Leftrightarrow q_0)) \\
\exists\, q_1 \dots \exists\, q_p.\varphi(\mathbf{X}^{-1}\, \alpha, q_1, \dots, q_p) \ &\equiv_i \ \exists\, q_0.\exists\, q_1 \dots \exists\, q_p.\varphi(q_0, q_1, \dots, q_p) \wedge (\neg q_0) \wedge \\
&\qquad \mathbf{G}\,(\alpha \Leftrightarrow \mathbf{X}\, q_0).
\end{aligned}$$

Ces équivalences permettent de montrer que l'on peut traduire une formule de EQPLTL en une formule de EQLTL équivalente (ce qui n'est pas surprenant), mais surtout que la formule équivalente peut être de taille exponentielle en la taille de la formule de départ (en effet, chaque étape de la traduction multiplie au plus par quatre la taille de la formule, et il y a au plus $|\varphi|$étapes dans cette transformation).

On a également l'équivalence suivante :

$$\exists\, q_1 \dots \exists\, q_p.\varphi(\mathbf{N}\, \psi, q_1, \dots, q_p) \ \equiv_i \ \exists\, q_1 \dots \exists\, q_p.\varphi(\exists\, q'_1 \dots \exists\, q'_r.\psi'(q'_1, \dots, q'_r), q_1, \dots, q_p)$$

où $\psi \equiv_i \exists\, q'_1 \dots \exists\, q'_r.\psi'(q'_1, \dots, q'_r)$ avec $\psi' \in$ LTL, et donc $\mathbf{N}\, \psi \equiv \exists\, q'_1 \dots \exists\, q'_r.\psi'(q'_1, \dots, q'_r)$. Cependant, cette transformation nous oblige à emboîter des quantificateurs à l'intérieur des formules de LTL, et la formule équivalente, de taille exponentielle, n'appartient pas à EQLTL mais à QLTL.

Puisque la vérification de QLTL est non-élémentaire, cette traduction n'est pas intéressante dans l'optique de la vérification de formules de NLTL. Mais puisqu'une formule de NLTL peut se traduire dans LTL, elle peut, à plus forte raison, se traduire dans EQLTL. Nous pouvons cependant, là encore, prouver un résultat de concision sur cette traduction :

Proposition 2.10
Toute formule φ de NLTL *peut être transformée en une formule équivalente ψ de* EQPLTL*, mais dans certains cas, cette formule équivalente est nécessairement de taille $\Omega(2^{|\varphi|})$.*

Démonstration :
D'après [Ete99, Corollaire 2], une formule φ de EQLTL peut être traduite en un automate de Büchi de taille $2^{O(|\varphi|)}$ reconnaissant le même langage. Ce résultat reste vrai si la formule φ appartient à EQPLTL, puisqu'il consiste d'abord en une traduction de la formule « non quantifiée », puis en une « projection ».

D'après la preuve de [EVW97, Théorème 2], on sait que tout automate de Büchi exprimant la propriété φ_n du théorème 1.29 est de taille 2^{2^n}. S'il existe une formule φ de EQPLTL exprimant cette propriété, elle doit donc être de taille au moins exponentielle. □

Nous n'avons cependant pas d'algorithme de traduction plus efficace que celui présenté à la section 1.2.2.2, à la page 31. Cela ne nous donne donc pas un algorithme efficace pour la vérification de la logique NLTL.

2.1.3.4 Les logiques hybrides

Les logiques hybrides [Gor96, BT99, Bla00] sont une extension de la logique modale autorisant l'utilisation de « pointeurs » : il est ainsi possible de « marquer » un état au cours de la formule, puis d'y faire référence par la suite.

On peut, pour ce faire, utiliser des « nominaux » [BT99], *i.e.* des propositions atomiques particulières ne pouvant être vraies que dans un état de la structure. Grâce à cela, il est possible d'exprimer la formule $\alpha\,\mathbf{U}\,\beta$ avec les modalités $\mathbf{F}$ et $\mathbf{F}^{-1}$:

$$\alpha\,\mathbf{U}\,\beta = \downarrow x.\mathbf{F}\,(\beta \wedge \mathbf{G}^{-1}\,(\mathbf{F}^{-1}\,x \Rightarrow \alpha)).$$

De la même façon, nous pouvons coder l'opérateur $\mathbf{N}$: en effet, $\mathbf{N}\,\varphi$ est équivalente à $\downarrow x.\psi$ où ψ est la formule φ dans laquelle toutes les formules de la forme $\alpha\,\mathbf{S}\,\beta$ est remplacée par $(\alpha \wedge \mathbf{F}^{-1}\,x)\,\mathbf{S}\,(\beta \wedge \mathbf{F}^{-1}\,x)$, et les formules de la forme $\mathbf{X}^{-1}\,\alpha$ par $\mathbf{X}^{-1}\,(\alpha \wedge \mathbf{F}^{-1}\,x)$.

Ces extensions rendent cependant le problème de la satisfaisabilité indécidable (avec des nominaux [BS95] ou avec des pointeurs [Gor96]).

2.2 Complexité

La section précédente nous a montré que tenter de se ramener à d'autres méthodes afin de vérifier des propriétés de NLTL fournissait des algorithmes relativement complexes. Nous allons développer ici un algorithme en espace exponentiel, puis montrer qu'il est optimal.

2.2.1 Un algorithme EXPSPACE

Cette section a pour objet la présentation d'un algorithme de model checking pour la logique NLTL. Cet algorithme utilise la théorie des automates de Büchi alternants, que nous présentons plus en détails dans un premier temps.

2.2.1.1 Automates de Büchi alternants

Automates de Büchi. Nous utilisons ici les automates de Büchi avec « condition d'acceptation généralisée » [CVWY91] :

Définition 2.11
Un automate de Büchi est un quintuplet $\mathcal{B} = (E, Q, Q_0, \delta, \mathcal{F})$ où
- *E est un ensemble fini non vide appelé* alphabet *;*
- *Q est un ensemble fini non vide dont les éléments sont appelés les* états *de l'automate ;*
- *Q_0 est un sous-ensemble de Q contenant les* états initiaux *;*
- *$\delta \subseteq Q \times E \times Q$ est l'ensemble des transitions ;*
- *$\mathcal{F} \subseteq \mathcal{P}(Q)$ est l'ensemble des* ensembles d'états répétés. *La taille d'un automate de Büchi est la somme du nombre d'états et du nombre de transitions de cet automate.*

La sémantique est définie de la façon suivante : un mot infini $w \in E^\omega$ est accepté par $\mathcal{B}$ si, et seulement si, il existe une suite infinie (q_i) d'états de l'automate telle que
- *$q_0 \in Q_0$;*
- *pour tout $i \in \mathbb{N}$, $(q_i, w_i, q_{i+1}) \in \delta$;*

– $\{q \in Q \mid \mathrm{Card}\{i \in \mathbb{N} \mid q_i = q\} = \infty\} \cap F \neq \varnothing$ *pour tous les éléments* $F \in \mathcal{F}$.

Un tel automate est dit *déterministe* si, dans chaque état q et pour chaque lettre a de l'alphabet, il existe au plus un état q' tel que $(q, a, q') \in \delta$. De nombreux résultats existent sur le modèle des automates de Büchi [Büc62, McN66] : ils peuvent reconnaître exactement les langages définissables par la logique monadique du second ordre, *i.e.* exactement les langages de mots infinis définissables par des expressions régulières. De plus, la classe des automates de Büchi déterministes est strictement moins expressive que la classe des automates de Büchi.

On a par ailleurs le résultat suivant :

Théorème 2.12 ([VW94, théorème 2.4])
Le problème de la vacuité du langage reconnu par un automate de Büchi est NLOGSPACE-*complet.*

Automates de Büchi et PLTL. Comme nous l'avons déjà vu la page 31, il est possible d'associer à toute formule de PLTL un automate de Büchi « équivalent ». La construction suivante, issue de [LPZ85], ne construit pas un automate de Büchi au sens propre du terme, mais il est facile de s'y ramener.

Définition 2.13
Soit φ une formule de PLTL. *La* clôture de φ *est le plus petit ensemble de formules vérifiant les conditions suivantes :*
– $\top$, $\mathbf{X}^{-1}\,\top$ *et φ appartiennent à* $\mathrm{Cl}(\varphi)$ *;*
– $\psi \in \mathrm{Cl}(\varphi)$ *si, et seulement si,* $\neg\psi \in \mathrm{Cl}(\varphi)$, *en identifiant ψ et $\neg\neg\psi$;*
– *si $\psi_1 \vee \psi_2$ appartient à $\mathrm{Cl}(\varphi)$, alors ψ_1 et ψ_2 y appartiennent aussi ;*
– *si $\mathbf{X}\,\psi$ appartient à $\mathrm{Cl}(\varphi)$, alors ψ y appartient aussi ;*
– *si $\psi_1\,\mathbf{U}\,\psi_2$ appartient à $\mathrm{Cl}(\varphi)$, alors ψ_1, ψ_2 et $\mathbf{X}\,(\psi_1\,\mathbf{U}\,\psi_2)$ y apparitent aussi ;*
– *si $\mathbf{X}^{-1}\,\psi$ appartient à $\mathrm{Cl}(\varphi)$, alors ψ y appartient aussi ;*
– *si $\psi_1\,\mathbf{S}\,\psi_2$ appartient à $\mathrm{Cl}(\varphi)$, alors ψ_1, ψ_2 et $\mathbf{X}^{-1}\,(\psi_1\,\mathbf{S}\,\psi_2)$ y apparitent aussi ;*

On montre par induction que cet ensemble contient au plus $4 \cdot |\varphi|$ éléments. On définit alors un *atome* comme étant un ensemble cohérent de sous-formules :

Définition 2.14
Un atome *A est un sous-ensemble de $\mathrm{Cl}(\varphi)$ contenant $\top$ et vérifiant les conditions suivantes, pour toutes les sous-formules ψ_1 et ψ_2 de $\mathrm{Cl}(\varphi)$:*
– $\psi_1 \in A$ *si, et seulement si,* $\neg\psi_1 \notin A$ *;*
– $\psi_1 \vee \psi_2 \in A$ *si, et seulement si,* $\psi_1 \in A$ *ou* $\psi_1 \in A$ *;*
– $\psi_1\,\mathbf{U}\,\psi_2 \in A$ *si, et seulement si,* $\psi_2 \in A$ *ou* $\{\psi_1, \mathbf{X}\,(\psi_1\,\mathbf{U}\,\psi_2)\} \subseteq A$ *;*
– $\psi_1\,\mathbf{S}\,\psi_2 \in A$ *si, et seulement si,* $\psi_2 \in A$ *ou* $\{\psi_1, \mathbf{X}^{-1}\,(\psi_1\,\mathbf{S}\,\psi_2)\} \subseteq A$ *;*

Nous noterons At_φ l'ensemble de tous les atomes de la formules φ. Il est clair qu'il y a au plus $2^{4\cdot|\varphi|}$ atomes associés à une formule φ.

Dans la construction de l'automate de Büchi « équivalent » à une formule φ, les états de l'automate seront donc les atomes. Un atome sera initial s'il ne contient aucune formule de la forme $\mathbf{X}^{-1}\,\psi$. Un couple d'atomes (A, B) sera une transition de l'automate si, et seulement si, les deux conditions suivantes sont réunies :

- pour chaque formule de la forme $\mathbf{X}\,\psi \in \mathrm{Cl}(\varphi)$, $\mathbf{X}\,\psi \in A \quad\Leftrightarrow\quad \psi \in B$;
- pour chaque formule de la forme $\mathbf{X}^{-1}\,\psi \in \mathrm{Cl}(\varphi)$, $\psi \in A \quad\Leftrightarrow\quad \mathbf{X}^{-1}\,\psi \in B$;

Les ensemble d'états répétés sont construits de telle sorte que, pour toute sous-formule de la forme $\psi_1\,\mathbf{U}\,\psi_2$, on passe infiniment souvent dans un état satisfaisant ψ_2 ou $\neg(\psi_1\,\mathbf{U}\,\psi_2)$. Avec cette construction, la formule φ est satisfaisable si, et seulement si, il existe un chemin acceptant dans cet automate dont un des état contient φ. En utilisant des algorithmes d'accessibilité « à la volée », cette méthode nous donne un algorithme en espace polynômial pour la satisfaisabilité de PLTL.

De plus amples détails sur cette construction peuvent être trouvés dans [LPZ85], ou à la section 2.2.1.2 où cette construction est étendue à la logique NLTL. Un autre algorithme utilisant les automates de Büchi est donné dans [VW94].

Automates de Büchi alternants. Le non-déterminisme ajoute donc de l'expressivité en proposant un choix « existentiel ». [MH84] a proposé d'étendre le modèle des automates de Büchi non déterministes en leur ajoutant un choix « universel » (qui est donc plutôt une « obligation »). Étant donné un ensemble X, on notera $\mathcal{R}^+(X)$ l'ensemble des formules booléenne positives (*i.e.* n'utilisant pas la négation) construites sur l'ensemble $X \cup \{\top, \bot\}$. Étant donnée une formule θ de $\mathcal{R}^+(X)$, un sous-ensemble Y d'éléments de X *satisfait* θ si, et seulement si, la formule θ est vraie lorsque les éléments de Y sont remplacés par $\top$ et les éléments de $X \setminus Y$ par $\bot$.

Du fait qu'il nous faudra, lors de certaines transitions, considérer plusieurs successseurs, une exécution d'un automate alternant sera donc un arbre étiqueté et non plus un mot. Nous utiliserons la définition suivante lorsque nous parlerons d'arbre :

Définition 2.15
Un arbre *T est un triplet (N, ε, V) tel que*

- *(N, V) est un graphe orienté, connexe ;*
- *$\varepsilon \in N$ est un sommet particulier, appelé* racine *de l'arbre ;*
- *pour tout élément s de N, il existe un, et un seul, chemin partant de ε et arrivant en s.*

Un nœud *de l'arbre T est un élément de N. Le* fils *d'un nœud x est un nœud y tel que $(x, y) \in V$. Le* père *d'un nœud x est un nœud y tel que $(y, x) \in V$. La racine de l'arbre n'a donc pas de père, puisque sinon il existerait un cycle partant et revenant à la racine, et il existerait donc plusieurs chemins entre la racine et les autres nœuds. Un nœud y est un prédécesseur d'un nœud x s'il existe une suite de nœuds $z_0 = y, z_1, z_2, \ldots, z_p = x$ tels que, pour tout $i < p$, $(z_i, z_i + 1) \in V$. L'entier p est la* distance *entre un nœud x et son prédécesseur y. Enfin, la* profondeur *d'un nœud est la distance de ce nœud à la racine de l'arbre.*

Les arbres que nous utiliserons seront *étiquetés* :

Définition 2.16
Soit Σ un ensemble. Un arbre étiqueté sur Σ *est un couple (U, σ) où $U = (N, \varepsilon, V)$ est un arbre et $\sigma\colon N \to \Sigma$ est une fonction d'étiquetage des nœuds de U.*

Le modèle des *automates de Büchi alternants* (avec condition d'acceptance généralisée) est alors défini comme suit [Var95, Var96] :

Définition 2.17
Un automate de Büchi alternant *est un quintuplet* $\mathcal{A} = (E, Q, Q_0, \rho, \mathcal{F})$ *où*
- E *est l'alphabet ;*
- Q *est l'ensemble des états ;*
- Q_0 *est l'ensemble des états initiaux ;*
- $\rho \subseteq \mathcal{R}^+(Q)^{Q\times E}$ *est l'ensemble des « transitions » ;*
- $\mathcal{F} \subseteq \mathcal{P}(Q)$ *est l'ensemble des* ensembles d'états répétés*.*

Un mot infini $w \in E^\omega$ *est accepté par* $\mathcal{A}$ *si, et seulement si, il existe un arbre étiqueté sur* Q $T = ((N, \varepsilon, A), \sigma)$ *tel que*
- *la racine* ε *est étiquetée par un état de* Q_0 *;*
- *si* x *est un nœud de* T *de profondeur* i *et étiqueté par l'état* q*, et si* $\rho(q, w_i) = \theta$*, alors le nœud* x *a* k *fils* $x_1, \ldots, x_k$*, avec* $k \leqslant |Q|$*, et ces fils vérifient que l'ensemble* $\{\sigma(x_1), \ldots, \sigma(x_k)\}$ *satisfait* θ *;*
- *le long de toute branche infinie, l'ensemble des états répétés infiniment souvent intersecte chaque élément* $F \in \mathcal{F}$*.*

Un tel arbre est appelé arbre d'exécution *de l'automate* $\mathcal{A}$*.*

Il est clair que ce nouveau modèle est un extension du modèle des automates de Büchi non déterministes : si $\mathcal{B} = (E, Q, Q_0, \delta, \mathcal{F})$ est un automate de Büchi non déterministe, et si

$$\rho\colon (q, a) \mapsto \bigvee_{(q,a,q')\in\delta} q'$$

alors l'automate de Büchi alternant $\mathcal{A} = (E, Q, Q_0, \rho, \mathcal{F})$ reconnaît exactement le même langage que l'automate $\mathcal{B}$. Inversement, on a le résultat suivant :

Proposition 2.18 ([MH84])
Soit $\mathcal{A}$ *un automate de Büchi alternant à* n *états. Il existe alors un automate de Büchi non déterministe* $\mathcal{B}$ *avec* 2^n *états reconnaissant le même langage.*

Puisque la vacuité d'un automate de Büchi est décidable en espace logarithmique, le résultat précédent fournit un algorithme utilisant un espace polynomial. On a en fait le résultat suivant :

Théorème 2.19 ([Var96, proposition 21])
Le problème de la vacuité du langage reconnu par un automate de Büchi alternant est PSPACE*-complet.*

Automates alternants et LTL. Tout comme on peut associer à une formule de LTL un automate de Büchi, on peut également lui associer un automate de Büchi *alternant*. Cependant, pour que cela soit intéressant, il faut que l'automate résultant soit de taille polynômiale, puisque le test de la vacuité d'un automate de Büchi alternant est PSPACE-complet. [Var96] propose une telle construction. Pour définir l'automate alternant, nous aurons besoin de la définition suivante :

Définition 2.20
Soit φ une formule de LTL. *La formule « duale » de φ, notée $\overline{\varphi}$, est définie récursivement par :*

$$\overline{\top} \stackrel{\text{def}}{=} \bot \qquad \overline{\psi} \stackrel{\text{def}}{=} \neg\psi \qquad \overline{\mathbf{X}\,\psi} \stackrel{\text{def}}{=} \mathbf{X}\,\overline{\psi}$$

$$\overline{\bot} \stackrel{\text{def}}{=} \top \qquad \overline{(\alpha \vee \beta)} \stackrel{\text{def}}{=} \overline{\alpha} \wedge \overline{\beta} \qquad \overline{(\alpha\,\mathbf{U}\,\beta)} \stackrel{\text{def}}{=} \overline{\alpha}\,\mathbf{U}\,\overline{\beta}$$

L'automate alternant correspondant à une formule $\varphi \in$ LTL est alors défini comme suit :
- l'alphabet est l'ensemble des parties de Prop ;
- les états sont les formules de $\mathrm{SubF}(\varphi)$;
- l'état initial est l'état φ ;
- les « transitions » sont données par la fonction ρ définie récursivement de la façon suivante, pour $A \subseteq \mathrm{Prop}$:
  - $\rho(p, A) = \top$ si, et seulement si, $p \in A$;
  - $\rho(p, A) = \bot$ si, et seulement si, $p \notin A$;
  - $\rho(\alpha \vee \beta, A) = \rho(\alpha, A) \vee \rho(\beta, A)$;
  - $\rho(\neg\alpha, A) = \overline{\rho(\alpha, A)}$;
  - $\rho(\mathbf{X}\,\alpha, A) = \alpha$;
  - $\rho(\alpha\,\mathbf{U}\,\beta, A) = \rho(\beta, A) \vee (\rho(\alpha, A) \wedge \alpha\,\mathbf{U}\,\beta)$;
- les états répétés sont les états de la forme $\neg(\alpha\,\mathbf{U}\,\beta)$.

[Var96] prouve que les chemins acceptés par cet automate sont exactement les chemins qui satisfont la propriété φ. On en déduit donc un algorithme en espace polynômial (puisqu'on peut tester la vacuité « à la volée ») pour la satisfaisabilité d'un formule de LTL.

2.2.1.2 Construction de l'automate

Nous allons maintenant voir comment construire l'automate alternant associé à une formule φ de NLTL. La « structure » de l'automate est semblable à celle de l'automate de Büchi que nous avons associé à une formule de PLTL à la page 87. Dans cet automate, lorsqu'on se trouve dans un état A après avoir lu les i premières lettres du « mot d'entrée » que représente le chemin π, cela signifie toutes les formules $\psi \in A$ sont vérifiées le long du chemin π à la position i. Les formules du passé que contient l'atome A ont donc été vérifiées au cours du début de l'exécution, et les autres formules ont été sélectionnées, de façon non déterministe, et devront être vérifiées dans la suite de l'exécution de l'automate.

L'alternation nous servira à vérifier les sous-formules de la forme $\mathbf{N}\,\psi$: en effet, quand un atome contient une telle sous-formule, nous devons bien entendu continuer l'exécution en cours, mais aussi en commencer une nouvelle, dans laquel l'« historique » sera vide (le premier état de cette exécution sera donc un état initial), et qui sera chargée de vérifier la propriété ψ.

Nous utiliserons les notions de clôture et d'atomes introduites aux définitions 2.13 et 2.14. Un atome sera dit *initial* s'il contient la formule $\neg\mathbf{X}^{-1}\top$. Nous noterons InitAt_φ l'ensemble des atomes initiaux de φ. Nous posons la définition suivante :

Définition 2.21
Soit φ une formule de NLTL. *Nous lui associons l'automate $\mathcal{A}_\varphi = (E, Q, Q_0, \rho, \mathcal{F})$ défini de la façon suivante :*

- $E = 2^{\mathrm{Prop}}$, *puisque le mot d'entrée sera un chemin utilisant les propositions atomiques de* Prop *;*
- $Q = \mathrm{At}_\varphi$ *;*
- $Q_0 = \{A \in \mathrm{InitAt}_\varphi \mid \varphi \in A\}$ *;*
- *la relation de transition est définie par*

$$\rho(A,e) = \bigvee_{A' \in \mathrm{Succ}(A,e)} (A' \wedge \bigvee_{A'' \in \mathrm{Now}(A')} A'')$$

avec :

$$\mathrm{Now}(A') \stackrel{\text{def}}{=} \{A'' \in \mathrm{InitAt}_\varphi \mid \forall\, \mathbf{N}\,\psi \in \mathrm{Cl}(\varphi).\ \mathbf{N}\,\psi \in A' \ \Leftrightarrow\ \mathbf{N}\,\psi \in A''\}$$

$$\mathrm{Succ}(A,e) \stackrel{\text{def}}{=} \begin{cases} \varnothing & \textit{si } e \neq A \cap \mathrm{Prop}^1 \\ \{A' \in \mathrm{At}_\varphi \mid \mathbf{X}^{-1}\top \in A' \textit{ et} & \textit{sinon.} \\ \qquad \forall\, \mathbf{X}\,\psi \in \mathrm{Cl}(\varphi).\ \mathbf{X}\,\psi \in A \ \Leftrightarrow\ \psi \in A' \textit{ et} & \\ \qquad \forall\, \mathbf{X}^{-1}\,\psi \in \mathrm{Cl}(\varphi).\ \psi \in A \ \Leftrightarrow\ \mathbf{X}^{-1}\,\psi \in A'\} & \end{cases}$$

- *l'ensemble $\mathcal{F}$ est défini par les conditions suivantes :*
  - *si* $\mathrm{Cl}(\varphi)$ *ne contient aucune formule de la forme $\alpha\,\mathbf{U}\,\beta$, alors $\mathcal{F}$ sera égal au singleton $\{Q\}$ (la condition d'acceptation étant qu'il faut que l'ensemble des états répétés intersecte tous les éléments de $\mathcal{F}$, tout chemin infini sera accepté) ;*
  - *sinon, on note $\{\psi_1\,\mathbf{U}\,\psi'_1, \ldots, \psi_k\,\mathbf{U}\,\psi'_k\}$ l'ensemble des formules de la forme $\alpha\,\mathbf{U}\,\beta$ contenues dans* $\mathrm{Cl}(\varphi)$*. Alors $\mathcal{F} = \{F_1, \ldots, F_k\}$ avec*

$$F_i \stackrel{\text{def}}{=} \{A \in \mathrm{At}_\varphi \mid \neg\mathbf{X}^{-1}\top \in A \textit{ ou } \psi'_i \in A \textit{ ou } \neg(\psi_i\,\mathbf{U}\,\psi'_i) \in A\}.$$

Cet automate est de taille exponentielle dans la taille de la formule. Il fournit alors un algorithme EXPSPACE pour le problème de satisfaisabilité de NLTL, grâce à la proposition suivante :

Proposition 2.22 ([LMS02b, Proposition 4.3])
Soit $\varphi \in$ NLTL. Alors φ est satisfaisable initialement si, et seulement si, l'automate $\mathcal{A}_\varphi$ a une exécution acceptante.

La preuve de ce résultat est l'objet de la section suivante.

2.2.1.3 Preuve de la correction de l'algorithme

Notons pour commencer que l'automate que nous avons construit admet un arbre acceptant si, et seulement si, il admet un arbre acceptant *binaire*, *i.e.* donc chaque nœud a exactement deux fils. En effet, la relation de transition peut se réécrire de la façon suivante :

$$\rho(A,e) = \bigvee_{A' \in \mathrm{Succ}(A,e)} \bigvee_{A'' \in \mathrm{Now}(A')} (A' \wedge A'').$$

[1] Dans ce cas, donc, $\rho(A,e) = \bot$.

Puisque les éléments de $\mathrm{Succ}(A, e)$ ne peuvent pas être initiaux, alors que tous ceux de $\mathrm{Now}(A')$ le sont, nous savons que A' et A'' sont nécessairement des atomes différents. Chaque nœud doit donc avoir au moins deux fils différents. De plus, deux fils suffisent pour satisfaire les formules de transitions, et on peut toujours « élaguer » l'arbre afin de se ramener à ce cas, tout en conservant la condition d'acceptation et le fait que c'est un arbre d'exécution de l'automate.

Dans toute la suite, donc, nous considérerons des arbres d'exécution binaires.

Dans un tel arbre T, chaque nœud possède au moins un prédécesseur étiqueté par un atome initial, puisque la racine de l'arbre est étiquetée par un atome initial. Nous noterons alors $l_r(x)$ la profondeur maximale des prédécesseurs du nœud x étiquetés par un atome initial.

Le lemme suivant nous permet de relier une exécution de l'automate de Büchi alternant au problème de satisfaisabilité de formules de NLTL :

Lemme 2.23
Soit φ une formule de NLTL*, ψ une formule de $\mathrm{Cl}(\varphi)$, et w un mot infini sur l'alphabet E (w représente donc un chemin infini). Si le mot w est accepté par l'automate alternant $\mathcal{A}_\varphi$,* i.e. *s'il existe un arbre d'exécution (binaire) T de $\mathcal{A}_\varphi$ sur l'entrée w, alors pour tout nœud x de cet arbre, si $i = |x|$ et $i_0 = l_r(x)$, si x est étiqueté par l'atome A, alors*

$$\psi \in A \quad \Leftrightarrow \quad w^{i_0}, i - i_0 \models \psi.$$

Démonstration :
Soit $T = ((N, \varepsilon, V), \sigma)$ un arbre d'exécution binaire de $\mathcal{A}_\varphi$. Nous remarquons tout d'abord que, cet arbre étant binaire, si i et i_0 sont deux entier donnés avec $i \geqslant i_0$, il existe toujours un nœud x de l'arbre tel que $|x| = i$ et $l_r(x) = i_0$.

Au vu de cette remarque, nous allons en fait montrer l'équivalence entre les trois propositions ci-dessous :

$$\begin{aligned} & \forall x \in N.\ (|x| = i\ \wedge\ l_r(x) = i_0) \Rightarrow \psi \in \sigma(x) && (2.1)\\ \Leftrightarrow\quad & \exists x \in N.\ (|x| = i\ \wedge\ l_r(x) = i_0\ \wedge\ \psi \in \sigma(x)) && (2.2)\\ \Leftrightarrow\quad & w^{i_0}, i - i_0 \models \psi && (2.3) \end{aligned}$$

La remarque ci-dessus nous donne l'implication (2.1) $\Rightarrow$ (2.2).

Nous allons montrer les implications (2.2) $\Rightarrow$ (2.3) et (2.3) $\Rightarrow$ (2.1) par induction sur la structure de la formule ψ :

- si $\psi = p$ est une proposition atomique : supposons dans un premier temps que (2.2) soit vraie. Alors le nœud x qui la rend vraie a deux successeurs qui satisfont $\rho(\sigma(x), w_i)$. En particulier $\rho(\sigma(x), w_i)$ n'est pas égal à $\perp$, donc w_i et $\sigma(x) \cap \mathrm{Prop}$ sont donc identiques, par construction de $\mathcal{A}_\varphi$. On en déduit $w, i \models p$, puis $w^{i_0}, i - i_0 \models \psi$, donc (2.3).
  Supposons maintenant que (2.3) soit vraie, et considérons un nœud x de l'arbre T de profondeur i et tel que $l_r(x) = i_0$. Comme précédemment, nous savons que x a deux successeurs vérifiant $\rho(\sigma(x), w_i)$, et que donc $\sigma(x) \cap \mathrm{Prop} = w_i$. Donc $p \in \sigma(x)$;

- si $\psi = \neg\psi_1$, supposons que (2.2) soit vraie pour $\neg\psi_1$. Cela entraîne alors que (2.1) est fausse pour ψ_1. D'après l'hypothèse d'induction, (2.3) n'est pas vraie pour ψ_1, et (2.3) est donc vraie pour $\neg\psi_1$.
  Supposons maintenant que (2.3) soit vraie pour la sous-formule $\neg\psi_1$. Alors (2.3) est fausse pour ψ_1, donc (2.2) également. Or la négation de (2.2) pour ψ_1 est exactement égale à (2.1) pour la sous-formule $\neg\psi_1$, qui est donc vraie ;
- si $\psi = \psi_1 \vee \psi_2$, supposons (2.2). Alors ψ_1 ou ψ_2 appartient à $\sigma(x)$, puisque c'est un atome. Sans perte de généralité, nous supposons que c'est ψ_1. Par induction, $w^{i_0}, i - i_0 \models \psi_1$, et donc $w^{i_0}, i - i_0 \models \psi$.
  Supposons maintenant que (2.3) soit vrai pour la formule ψ. Nous pouvons supposer alors que $w^{i_0}, i-i_0 \models \psi_1$. Par hypothèse d'induction, (2.1) est vraie pour ψ_1. Puisque $\sigma(x)$ est un atome et que $\psi_1 \vee \psi_2$ est une sous-formule de φ, le fait que ψ_1 appartienne à $\sigma(x)$ entraîne que ψ y appartient aussi ;
- si $\psi = \mathbf{X}\,\psi_1$, supposons que (2.2) soit vraie pour cette sous-formule, et soit x un nœud qui la rende vraie. Soient x_1 et x_2 les deux successeurs de x dans l'arbre T. Un de ces nœuds, et un seul, est étiqueté par un atome initial. Nous supposerons que c'est x_2. Alors $\sigma(x_1)$ appartient à $\mathrm{Succ}(\sigma(x))$. Puisque $\mathbf{X}\,\psi_1$ appartient à $\sigma(x)$, on en déduit que ψ_1 appartient à $\sigma(x_1)$. D'après l'hypothèse d'induction, appliquée au nœud x_1 (avec $|x_1| = i + 1$ et $l_r(x_1) = i_0$), on a $w^{i_0}, i + 1 - i_0 \models \psi_1$. Donc $w^{i_0}, i - i_0 \models \mathbf{X}\,\psi_1$.
  Supposons maintenant que (2.3) soit vraie pour la formule $\mathbf{X}\,\psi_1$, et considérons un nœud x tel que $|x| = i$ et $l_r(x) = i_0$. Comme précédemment, ce nœud a un unique successeur, que nous noterons x_1, qui ne soit pas étiqueté par un atome initial. Puisque $w^{i_0}, i - i_0 \models \mathbf{X}\,\psi_1$, et donc $w^{i_0}, i+1-i_0 \models \psi_1$, on sait que, pour tout nœud y vérifiant $|y| = i + 1$ et $l_r(y) = i_0$, $\sigma(y)$ contient ψ_1. Cela est vrai, en particulier, pour x_1. Puisque x_1 appartient à $\mathrm{Succ}(x, w_i)$, et puisque $\mathbf{X}\,\psi_1 \in \mathrm{Cl}(\varphi)$, on obtient que $\mathbf{X}\,\psi_1 \in \sigma(x)$;
- si $\psi = \mathbf{X}^{-1}\,\psi_1$, supposons que (2.2) soit vraie pour cette formule et considérons un nœud qui rende cette proposition vraie. Puisque $\mathbf{X}^{-1}\,\psi_1 \in \sigma(x)$, l'atome $\sigma(x)$ n'est pas initial, et il admet donc un prédécesseur y. On a donc $\sigma(x) \in \mathrm{Succ}(\sigma(y), w_{i-1})$. On en déduit que ψ_1 appartient à $\sigma(y)$, le nœud y vérifiant également les égalités $|y| = i - 1$ et $l_r(y) = i_0$. En appliquant l'hypothèse d'induction à ce nœud pour la formule ψ_1, on obtient que $w^{i_0}, i - 1 - i_0 \models \psi_1$. Donc $w^{i_0}, i - i_0 \models \mathbf{X}^{-1}\,\psi_1$.
  Supposons maintenant (2.3), et considérons un nœud x de T de profondeur i et tel que $l_r(x) = i_0$. Le prédécesseur y de ce nœud dans T vérifie que $\sigma(x) \in \mathrm{Succ}(\sigma(y), w_{i-1})$. Puisque ${}^{i_0}, i - 1 - i_0 \models \psi_1$, nous savons que tous les nœuds de profondeur $i - 1$ et tels que $l_r(y) = i_0$ sont étiquetés par un atome contenant ψ_1. Puisque $\sigma(x) \in \mathrm{Succ}(\sigma(y), w_{i-1})$, $\mathbf{X}^{-1}\,\psi_1$ appartient à l'atome $\sigma(x)$;
- si $\psi = \psi_1\,\mathbf{U}\,\psi_2$, supposons encore que (2.2) soit vraie, et considérons un nœud x qui la rende vraie. Par définition de ρ, et puisque T est binaire, chaque nœud a exactement un successeur étiqueté par un atome initial et un étiqueté par un atome non initial. Donc il existe une (et une seule) branche, que nous noterons b par la suite, partant du nœud x et qui ne passe que par des états non initiaux (sauf éventuellement x lui-même). Cette branche ne contient pas une infinité de nœuds étiquetés par des atomes initiaux, donc, pour satisfaire la condition d'acceptation, elle contient une infinité

de nœuds étiquetés par des atomes contenant ψ_2 ou $\neg(\psi_1 \mathbf{U} \psi_2)$. Or si l'étiquette d'un nœud de b contient $\psi_1 \mathbf{U} \psi_2$ mais ne contient pas ψ_2, alors elle contient ψ_1 et l'étiquette du successeur de ce nœud (le long de la branche b) contient $\psi_1 \mathbf{U} \psi_2$. Si aucun nœud de b n'est étiqueté par un atome contenant ψ_2, alors tous les nœuds (excepté le nœud x) sont étiquetés par $\neg(\psi_1 \mathbf{U} \psi_2)$. Cela contredit le fait que le branche b vérifie les condition d'acceptation de $\mathcal{A}_\varphi$. Il existe donc un nœud étiqueté par un atome contenant ψ_2 le long de la branche b. Il existe donc un descendant y de x, le long de b, étiqueté par ψ_2 et de profondeur minimale. Ce nœud vérifie alors que tous les nœuds intermédiaires (entre x inclus et y exclu) sont étiquetés par des atomes contenant ψ_1 (puisqu'un atome contenant $\psi_1 \mathbf{U} \psi_2$ contient nécessairement soit ψ_1, soit ψ_2). En appliquant l'hypothèse d'induction, on obtient que $w^{i_o}, |y| - i_0 \models \psi_2$ et, pour tout k tel que $i = |x| \leqslant k < |y|$, $w^{i_o}, k - i_0 \models \psi_1$. Donc $w^{i_o}, i - i_0 \models \psi_1 \mathbf{U} \psi_2$.
Montrons maintenant (2.3) $\Rightarrow$ (2.1) ; si $w^{i_0}, i - i_0 \models \psi_1 \mathbf{U} \psi_2$, alors il existe un entier k tel que $w^{i_0}, i + k - i_0 \models \psi_2$ et que pour tout entier $j < k$, $w^{i_0}, i + j - i_0 \models \psi_1$. Soit x un nœud de T de profondeur i tel que $l_r(x) = i_0$. Nous considérons, comme précédemment, la branche b issue de x ne contenant que des nœuds étiquetés par des états initiaux. Le long de cette branche, tout nœud y vérifie $l_r(y) = i_0$. D'après ce qui précède, et en appliquant l'hypothèse d'induction, le nœud de cette branche qui se situe à profondeur $i + k$ est étiqueté par un atome contenant ψ_2, et les nœuds situés à une profondeur comprise entre i et $i + k - 1$ sont étiquetés par des atomes contenant ψ_1. Puisque $\psi_1 \mathbf{U} \psi_2$ appartient à $\mathrm{Cl}(\varphi)$, l'étiquette du nœud de b contient $\psi_1 \mathbf{U} \psi_2$, puisqu'elle contient ψ_2. Les étiquettes des œuds de b dont la profondeur est comprise entre i et $i+k-1$ contenant ψ_1, elles contiendront donc également $\psi_1 \mathbf{U} \psi_2$. Donc l'étiquette de x contient bien $\psi_1 \mathbf{U} \psi_2$;

– si $\psi = \psi_1 \mathbf{S} \psi_2$: nous supposons que (2.2) est vraie, et considérons un nœud x qui la rend vraie. Si $\sigma(x)$ est un atome initial, puisque $\psi_1 \mathbf{S} \psi_2 \in \sigma(x)$, nécessairement $\psi_2 \in \sigma(x)$. Par induction, $w^{i_0}, i - i_0 \models \psi_2$, et donc $w^{i_0}, i - i_0 \models \psi_1 \mathbf{S} \psi_2$. Si $\sigma(x)$ n'est pas initial, il peut néanmoins contenir ψ_2, auquel cas la conclusion est la même. Maintenant, si $\sigma(x)$ contient $\psi_1 \mathbf{S} \psi_2$ mais ne contient pas ψ_2, il contient nécessairement $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$. On sait par ailleurs que, parmi les prédécesseurs de x, il y en a qui sont étiquetés par des atomes initiaux, donc ne contenant pas $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$. Nous pouvons donc considérer le nœud y, prédécesseur de x, dont l'étiquette ne contient pas $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$, et qui soit le plus proche de x parmi tous ceux qui vérifient ces conditions. Dans ce cas, le fils z de y qui est situé sur la branche reliant y à x est étiqueté par un atome (non initial) contenant $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$. Puisque z est un fils de y, l'atome $\sigma(y)$ contient $\psi_1 \mathbf{S} \psi_2$. Puisqu'il ne contient pas $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$, il contient ψ_2. Il existe donc un nœud t, prédécesseur de x, dont l'étiquette contient ψ_2, et qui soit le plus proche de x vérifiant ces propriétés. Alors les états situés entre t et x sont tous étiquetés par des atomes contenant $\psi_1 \mathbf{S} \psi_2$. Puisque parmi ces atomes, seul $\sigma(t)$ contient ψ_2, les autres contiennent ψ_1. Par induction, on en déduit que $w_{i_0}, |t| - i_0 \models \psi_2$ et que, pour tout entier k vérifiant $|t| < k \leqslant |x|$, on a $w^{i_0}, k - i_0 \models \psi_1$. Donc, $w^{i_0}, i - i_0 \models \psi_1 \mathbf{S} \psi_2$.
Supposons maintenant que (2.3) soit vraie, et considérons un nœud x de profondeur i et tel que $l_r(x) = i_0$. Puisque $w^{i_0}, i - i_0 \models \psi_1 \mathbf{S} \psi_2$, il existe un entier k, inférieur à $i - i_0$, tel que $w^{i_0}, i - k - i_0 \models \psi_2$, et tel que pour tout entier j tel que $k < j \leqslant 0$,

$w^{i_0}, i - j - i_0 \models \psi_1$. On en déduit que le prédécesseur y de x situé à la profondeur $i - k$ et tel que $l_r(y) = i_0$ est étiqueté par un atome contenant ψ_2, donc contenant également $\psi_1 \,\mathbf{S}\, \psi_2$. Tous les nœuds situés entre x et y, y non compris, sont quant à eux étiquetés par des atomes contenant ψ_1. On en déduit donc que leurs étiquettes contiennent également $\psi_1 \,\mathbf{S}\, \psi_2$, ce qui est vrai en particulier pour le nœud x ;

- enfin, si $\psi = \mathbf{N}\, \psi_1$: supposons que (2.2) est vraie, et montrons que (2.3) en découle. Soit x un nœud vérifiant les conditions de (2.2) pour la formule $\mathbf{N}\, \psi_1$. Si x est étiqueté par un atome initial, cet atome contient alors les formules $\mathbf{N}\, \psi_1$ et ψ_1. Puisque x est étiqueté par un atome initial, on a $i = i_0$. Par hypothèse d'induction, $w^{i_0}, 0 \models \psi_1$. Donc $w^{i_0}, i - i_0 \models \mathbf{N}\, \psi_1$. Si x n'est pas étiqueté par un atome initial, il a alors un et un seul frère (puisque l'arbre est binaire) et ce frère, que nous noterons y, appartient à $\mathrm{Now}(\sigma(x))$. On en déduit que $\psi_1 \in \sigma(y)$, et, par induction, $w^i, 0 \models \psi_1$. Donc pour tout entier k inférieur à i, $w^k, i - k \models \mathbf{N}\, \psi_1$, en particulier pour l'entier i_0.
  Montrons maintenant (2.3) $\Rightarrow$ (2.1). Supposons donc que $w^{i_0}, i - i_0 \models \mathbf{N}\, \psi_1$, et considérons un nœud x de profondeur i et tel que $l_r(x) = i_0$. Puisque $w^i, 0 \models \psi_1$, tout nœud z de profondeur i et tel que $l_r(z) = i$ est étiqueté par un atome contenant ψ_1, d'après l'hypothèse d'induction. C'est en particulier le cas du frère y du nœud x. On a donc $\psi_1 \in \sigma(y)$, $y \in \mathrm{Now}(\sigma(x))$, et $\mathbf{N}\, \psi_1 \in \mathrm{Cl}(\varphi)$. On en déduit que $\mathbf{N}\, \psi_1 \in \sigma(x)$. $\square$

Ce lemme nous fournit directement une implication de l'équivalence suivante :

Proposition 2.24
Soit ψ une formule de NLTL. *Alors l'automate $\mathcal{A}_\varphi$ admet une exécution acceptante si, et seulement si, la formule φ est initialement satisfaisable.*

Démonstration :
Si T est un arbre correspondant à une exécution acceptante de $\mathcal{A}_\varphi$ sur un mot d'entrée w, nous savons que la racine de cet arbre est étiquetée par un atome appartenant à Q_0, donc par un atome initial contenant φ. D'après le lemme précédent, il s'ensuit que $w, 0 \models \varphi$.

Inversement, en supposant que la formule φ est satisfaisable, nous allons prendre un chemin w (*i.e.* un mot sur l'alphabet de l'automate alternant $\mathcal{A}_\varphi$) qui la satisfait, construire une exécution de $\mathcal{A}_\varphi$ acceptant ce mot w.

Soit donc w un tel mot. Nous définissons alors l'atome $A_\varphi^{w^j}(i)$ par :

$$A_\varphi^{w^j}(i) \stackrel{\text{def}}{=} \{\psi \in \mathrm{Cl}(\varphi) \mid w^j, i \models \psi\}.$$

C'est bien entendu un atome, qui est initial si, et seulement si, $i = 0$. Soit maintenant $T = ((N, \varepsilon, S), \sigma)$ un arbre binaire, dont toutes les branches sont infinies, étiqueté par des atomes de la manière suivante :

- la racine ε est étiquetée par l'atome $A_\varphi^w(0)$;
- pour tout nœud x étiqueté par $A_\varphi^{w^j}(k)$, ses deux fils x_1 et x_2 sont étiquetés par $A_\varphi^{w^j}(k+1)$ et $A_\varphi^{w^{j+k+1}}(0)$ respectivement.

On vérifie alors facilement, par induction, que tout nœud tel que $|x| = i$ et $l_r(x) = i_0$ est étiqueté par $A_\psi^{w^{i_0}}(i - i_0)$.

Montrons maintenant que cet arbre ainsi construit est bien une exécution acceptante de $\mathcal{A}_\varphi$ sur le mot d'entrée w :

- la racine de l'arbre est bien étiquetée par un atome appartenant à l'ensemble des états initiaux de $\mathcal{A}_\varphi$;
- si x est un nœud de profondeur i tel que $l_r(x) = i_0$, alors ses successeurs x_1 et x_2 ont pour profondeur $i+1$, et on peut supposer, sans perte de généralité, que $l_r(x_1) = i_0$ et $l_r(x_2) = i+1$. Donc, $\sigma(x_1) = A_\psi^{w^{i_0}}(i+1-i_0)$, et $\sigma(x_2) = A_\psi^{w^{i+1}}(0)$. Alors
  - $\sigma(x_1) \in \mathrm{Succ}(\sigma(x), w_i)$, car d'une part, $\sigma(x) \cap \mathrm{Prop} = \{p \in \mathrm{Prop} \mid w^{i_0}, i - i_0 \models p\} = w_i$, et d'autre part, on a les trois propriétés suivantes :
    - $\sigma(x_1)$ n'est pas un atome initial ;
    - pour toute formule de la forme $\mathbf{X}\,\psi$ dans $\mathrm{Cl}(\varphi)$,

$$\begin{aligned} \mathbf{X}\,\psi \in \sigma(x) = A_\varphi^{w^{i_0}}(i-i_0) \quad &\Leftrightarrow \quad w^{i_0}, i - i_0 \models \mathbf{X}\,\psi \\ &\Leftrightarrow \quad w^{i_0}, i+1-i_0 \models \psi \\ &\Leftrightarrow \quad \psi \in A_\varphi^{w^{i_0}}(i+1-i_0) = \sigma(x_1)\,; \end{aligned}$$

    - pour toute formule $\mathbf{X}^{-1}\,\psi \in \mathrm{Cl}(\varphi)$, on a, de la même façon :

$$\begin{aligned} \mathbf{X}^{-1}\,\psi \in \sigma(x_1) = A_\varphi^{w^{i_0}}(i+1-i_0) \quad &\Leftrightarrow \quad w^{i_0}, i+1-i_0 \models \mathbf{X}^{-1}\,\psi \\ &\Leftrightarrow \quad w^{i_0}, i - i_0 \models \psi \\ &\Leftrightarrow \quad \psi \in A_\varphi^{w^{i_0}}(i-i_0) = \sigma(x)\,; \end{aligned}$$

  - $\sigma(x_2) \in \mathrm{Now}(\sigma(x_1))$, car $\sigma(x_2)$ est un atome initial, et pour toute formule $\mathbf{N}\,\psi$ dans $\mathrm{Cl}(\varphi)$,

$$\begin{aligned} \mathbf{N}\,\psi \in \sigma(x_1) = A_\varphi^{w^{i_0}}(i+1-i_0) \quad &\Leftrightarrow \quad w^{i_0}, i+1-i_0 \models \mathbf{N}\,\psi \\ &\Leftrightarrow \quad w^{i}, 0 \models \psi \\ &\Leftrightarrow \quad \psi \in A_\varphi^{w^{i}}(0) = \sigma(x_2)\,; \end{aligned}$$

- enfin, nous devons montrer que toutes les branches satisfont la condition d'acceptation.
  - c'est le cas pour toute branche dont une infinité de nœuds est étiquetée par un atome initial ;
  - dans les autres cas, les branches ne passent qu'un nombre fini de fois par un nœud étiqueté par un atome initial, et donc ne rencontrent plus de tels nœuds à partir d'une certaine profondeur. Soit donc b une telle branche, et x un nœud dont aucun des descendants n'est étiqueté par un atome initial. On pose $i = |x|$ et $i_0 = l_r(x)$. Alors tout descendant y de x est étiqueté par $A_\psi^{w^{i_0}}(|y| - i_0)$. Par définition de $A_\psi^{w^k}(j-k)$, si une formule du type $\psi_1\,\mathbf{U}\,\psi_2$ appartient à $A_\psi^{w^{i_0}}(|y|-i_0)$, c'est que $w^{i_0}, |y| - i_0 \models \psi_1\,\mathbf{U}\,\psi_2$, et donc qu'il existe un entier $l \geqslant |y|$ tel que $w^{i_0}, l - i_0 \models \psi_2$. Puisque $l \geqslant |y|$, on sait qu'il existe un successeur z de x, le long de la branche b, tel que $|z| = l$, et qui sera donc étiqueté par l'atome $A_\psi^{w^{i_0}}(l-i_0)$, qui contient ψ_2. S'il existe, le long de b, une infinité de descendants de x étiqueté par un atome contenant $\psi_1\,\mathbf{U}\,\psi_2$, il existera alors une infinité de nœuds de b étiquetés par un

atome contenant ψ_2. Dans le cas contraire, il y aura une infinité de nœuds étiquetés par un atome contenant $\neg(\psi_1 \mathbf{U} \psi_2)$. $\square$

Il en découle le résultat suivant :

Théorème 2.25
Il existe un algorithme utilisant un espace exponentiel pour décider de la satisfaisabilité d'une formule de NLTL.

Démonstration :
Étant donnée une formule de NLTL, l'automate $\mathcal{A}_{\mathbf{F}\varphi}$ est de taille exponentielle en la taille de la formule φ, et peut être construit en utilisant un espace polynomial. Une fois cet automate construit, il est possible de tester en espace polynomial (en la taille de l'automate, et donc en espace exponentiel en la taille de la formule de départ) si le langage qu'il reconnaît est vide [MH84, Var95]. D'après la proposition précédente, le langage est vide si, et seulement si, il n'existe aucun chemin qui satisfasse initialement la formule $\mathbf{F}\,\varphi$, donc aucun chemin qui satisfasse (globalement) la formule ψ. Cet algorithme est donc correct, et s'effectue en espace exponentiel. $\square$

Concernant le model checking, comme nous le faisions dans le chapitre concernant PLTL, nous pouvons traduire la structure de Kripke K sur laquelle nous travaillons en une formule de logique temporelle acceptant exactement les mêmes chemins, puis vérifier la satisfaisabilité de la conjonction de cette contrainte et de la formule initiale. Cet algorithme a un coût exponentiel.

Pour résumer, nous avons donc obtenu le résultat suivant :

Théorème 2.26
Le model checking et la satisfaisabilité de formules de NLTL *sont dans* EXPSPACE.

2.2.2 EXPSPACE-difficulté

Nous allons montrer dans cette section que l'algorithme que nous avons défini à la section précédente est optimal, c'est à dire que les problèmes de satisfaisabilité et de model checking sont EXPSPACE-durs pour la logique NLTL.

Nous allons pour ce faire écrire une réduction d'un problème de pavage à ces problèmes de vérification.

2.2.2.1 Un problème de pavage

Nous reprenons les notations introduites à la section 1.3.4.2, et nous définissons le problème suivant, semblable au problème Pavage2 :

Définition 2.27

Pavage6 : *entrée :* – *un ensemble T de types de dominos*
– *un entier m (en unaire)*
– *deux couleurs c_d et c_f*

sortie : *existe-t-il un entier n et une grille pavée correcte G, de largeur 2^m et de hauteur n, telle que* $\mathrm{down}(G(1,1)) = c_d$ *et* $\mathrm{up}(G(2^m, n)) = c_f$ *?*

La seule différence avec le problème Pavage2 est que la largeur du pavage est 2^m. La complexité du problème s'en ressent :

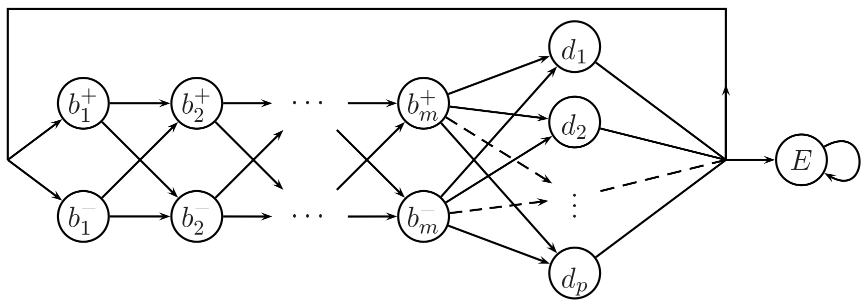
Théorème 2.28 ([Har92])
Le problème Pavage6 *est* EXPSPACE-*complet.*

2.2.2.2 Réduction du model checking de NLTL à ce problème de pavage

Lemme 2.29
Le problème Pavage6 *se réduit au problème du model checking de* NLTL.

Démonstration :
Soit (C, T, m, c_d, c_f) une instance du problème Pavage6. Nous lui associons la structure de Kripke K_4 représentée à la figure 2.2. Cette structure de Kripke utilise les propositions

Figure 2.2 La structure de Kripke associée au problème Pavage6



atomiques $\{b_1^+, \ldots, b_m^+, b_1^-, \ldots, b_m^-\} \cup \{d \mid d \in T\} \cup \{E\}$. Les bits b_i^+ et b_i^- servent à encoder un compteur, allant de 0 à $2^m - 1$, qui nous permettra de coder l'abscisse des cellules.

De la même manière que pour les preuves de la section 1.3.4, nous codons un pavage par un chemin dans cette structure, en énumérant les dominos ligne par ligne, de gauche à droite sur chaque ligne, et de bas en haut. Nous devons exprimer les conditions que doit satisfaire un chemin pour correspondre à un pavage correct satisfaisant les contraintes du problème Pavage6. Nous utiliserons les abréviations suivantes :

$$b_i \stackrel{\text{def}}{=} b_i^+ \vee b_i^- \qquad\qquad d_{init} \stackrel{\text{def}}{=} \bigvee_{\mathrm{down}(d)=c_d} d \qquad\qquad d_{fin} \stackrel{\text{def}}{=} \bigvee_{\mathrm{up}(d)=c_f} d$$

Nous écrivons donc :

– que le chemin commence bien avec un domino « initial » et à l'abscisse zéro :

$$\bigwedge_{i=1}^{m} \mathbf{X}^{i-1} b_i^- \wedge \mathbf{X}^m d_{init} \,;$$

– que le chemin atteindra l'état E, que la dernière case codée sera bien la case numérotée $2^m - 1$, et que le dernier domino sera bien un domino final :

$$\mathbf{F}\Big(\bigwedge_{i=1}^{m} \mathbf{X}^{i-1} b_i^+ \wedge \mathbf{X}^m d_{fin} \wedge \mathbf{X}^{m+1} E\Big) \,;$$

– les cellules sont listées dans l'ordre et de manière cyclique :

$$\begin{aligned}\mathbf{G}\Big[&(b_1 \wedge \neg\mathbf{X}^{m+1} E) \Rightarrow \big((b_1^+ \;\Leftrightarrow\; \mathbf{X}^{m+1} b_1^-)\wedge \\ &\bigwedge_{i=1}^{m-1} \big((\mathbf{X}^{i} b_{i+1}^+ \;\Leftrightarrow\; \mathbf{X}^{m+i+1} b_{i+1}^-) \;\Leftrightarrow\; (\mathbf{X}^{i-1} b_i^+ \wedge \mathbf{X}^{m+i} b_i^-)\big)\big)\Big]\end{aligned}$$

La première partie de cette formule spécifie que la valeur du bit b_1 change à chaque nouvelle cellule (b_1 est le bit faible du compteur). La seconde partie exprime que la valeur du bit b_{i+1} change si, et seulement si, « la valeur du bit b_i est passée de $+$ à $-$ ». De cette manière, le compteur est incrémenté de 0 jusqu'à $2^m - 1$, et remis à zéro lorsqu'il vaut $2^m - 1$.

– une cellule et sa voisine de droite doivent avoir la même couleur sur leur arête commune (excepté à la fin d'une ligne) :

$$\bigwedge_{d \in T} \mathbf{G}\,\big(d \Rightarrow \bigvee_{\substack{d' \in T \\ \mathrm{left}(d) = \mathrm{right}(d')}} \mathbf{X}^{m+1} d' \vee \mathbf{X}\, E \vee \bigwedge_{i=1}^{m} \mathbf{X}^{i} b_i^-\big) \,;$$

– de même, une cellule et la cellule au-dessus doivent avoir la même couleur sur l'arête qu'elles partagent. Nous allons utiliser ici des modalités du passé, ainsi que l'opérateur $\mathbf{N}$. Nous définissons l'abréviation suivante :

$$\varphi_{up} \;\stackrel{\text{def}}{=}\; b_1 \wedge \bigvee_{i=1}^{m} \big(\mathbf{X}^{i-1} b_i^+ \;\Leftrightarrow\; \mathbf{F}^{-1}\,(\neg\mathbf{X}^{-1}\,\top \wedge \mathbf{X}^{i-1} b_i^+)\big)$$

Cette formule caractérise les cellules ayant le même numéro (*i.e.* ayant les mêmes valeurs pour les bits $b_1, \ldots, b_m$) que la cellule « initiale » (*i.e.* correspondant au début du chemin). Le début du chemin sera modifié grâce à l'opérateur $\mathbf{N}$. Ainsi, $\mathbf{N}\,(\neg\varphi_{up}\,\mathbf{U}\,\varphi_{up})$ nous permet de trouver la cellule qui se trouve « au dessus » de la cellule courante. Nous écrirons donc

$$\mathbf{G}\Big(b_1 \Rightarrow \mathbf{N}\,\mathbf{X}^m \bigwedge_{d \in T} (d \Rightarrow \neg\varphi_{up}\,\mathbf{U}\,(E \vee (\varphi_{up} \wedge \bigvee_{\substack{d' \in T \\ \mathrm{up}(d) = \mathrm{down}(d')}} \mathbf{X}^m d')))\Big)$$

S'il existe un chemin dans K_4 satisfaisant toutes les conditions que nous venons d'énumérer, alors notre codage nous permet de faire correspondre à ce chemin une grille pavée de largeur 2^m, correcte, et vérifiant les conditions du problème Pavage6. Inversement, il est clair que s'il existe un pavage solution du problème Pavage6, alors ce pavage correspond à un chemin dans K_4 vérifiant les conditions ci-dessus. La réduction est donc correcte. □

Théorème 2.30
Le model checking et la satisfaisabilité de NLTL *sont des problèmes* EXPSPACE*-durs.*

Démonstration :
Pour le model checking, cela découle directement du lemme précédent. Pour la satisfaisabilité, cela vient directement du codage d'une structure de Kripke en une formule de logique temporelle, comme par exemple à la proposition 1.41. □

Il découle de cette section et de la précédente le résultat général suivant :

Théorème 2.31
La satisfaisabilité et le model checking de formules de NLTL *sont des problèmes* EXPSPACE*-complets.*

Démonstration :
Si φ est une formule de NLTL, nous pouvons construire l'automate $\mathcal{A}_\varphi$, qui sera de taille exponentielle dans la taille de φ. Cet automate admet une exécution acceptante si, et seulement si, la formule φ est (globalement) satisfaisable (d'après la proposition 1.35). Puisque le problème de la vacuité du langage reconnu par un automate de Büchi alternant admet un algorithme en espace polynomial, la satisfaisabilité initiale de NLTL admet un algorithme en espace exponentiel.

Tous les autres problèmes que nous avons définis à la section 1.3.1 ont exactement la même complexité, puisque les réductions que nous avons données dans cette section concernant PLTL sont valables *a fortiori* pour NLTL, et que EXPSPACE = co-EXPSPACE.

□

Notre algorithme à base d'automates nous permet de raffiner un peu ces résultats de complexité, en distinguant la complexité qui est liée à la formule de celle qui est liée à la structure de Kripke [LP85, VW86]. En effet, dans beaucoup de cas pratiques, la formule à vérifier est assez petite, alors que la structure est plutôt grande. Il est important de connaître la contribution de chacun de ces deux paramètres sur la complexité du problème. [LP85] a montré que la *complexité en formule* (*i.e.* lorsque la structure est fixée) du model checking de LTL est PSPACE-complet, et [VW86] a montré que sa *complexité en programme* (*i.e.* lorsque la formule est fixée) est dans NLOGSPACE, et donc NLOGSPACE-complet (le problème de l'accessibilité, qui est NLOGSPACE-complet, se réduit facilement à ce problème).

Concernant NLTL, on montre facilement que la complexité en programme est la même. En effet, si la formule de NLTL est fixée, il suffit de la convertir en une formule équivalente dans LTL pour obtenir un algorithme dans NLOGSPACE. Par contre, la complexité en formule est EXPSPACE-complète : la réduction du problème Pavage6 au model checking d'une formule de NLTL peut être modifiée de telle sorte que la structure de Kripke ne dépende pas de l'instance choisie pour faire la réduction.

2.2.3 Les fragments de NLTL

Les résultats de complexité que nous venons de voir s'étendent, assez facilement, aux principaux fragments de NLTL, comme nous allons le voir ici.

2.2.3.1 Le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1}, \mathbf{N})$

Les formules que nous avons écrites dans la démonstration du lemme 2.29, à l'exception de la dernière, n'utilisent que des modalités du fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1}, \mathbf{N})$.

La dernière formule utilise la modalité $\mathbf{U}$, mais on peut s'en passer en appliquant, par exemple, le théorème 2.7. La taille de la formule reste polynomiale dans la taille des données. On en déduit que les problème de model checking et de satisfaisabilité pour le fragment $\mathrm{L}(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1}, \mathbf{N})$ sont EXPSPACE-durs, et donc EXPSPACE-complets.

2.2.3.2 Les fragments $\mathrm{L}(\mathbf{U}, \mathbf{S}, \mathbf{N})$ et $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{N})$

Toutes les formules que nous avons écrites dans la preuve de la réduction du problème de pavage Pavage6 utilisent les modalités $\mathbf{X}$ ou $\mathbf{X}^{-1}$. Nous allons modifier les formules afin de ne plus utiliser ces modalités. La difficulté sera que l'on ne peut plus « compter » le nombre de transitions, et il sera donc un peu plus compliqué de savoir « où on se trouve ». Afin de simplifier les formules, nous ajoutons les deux abréviations suivantes[2] :

$$D \stackrel{\text{def}}{=} \bigvee_{d \in T} d \qquad \varphi_{\neg D} \stackrel{\text{def}}{=} \mathbf{G}^{-1}(\neg D) \qquad \varphi_{first} \stackrel{\text{def}}{=} \mathbf{G}^{-1}\Big(\bigvee_{i=1}^{m} b_i^-\Big) \qquad \varphi_{last} \stackrel{\text{def}}{=} \mathbf{G}^{-1}\Big(\bigvee_{i=1}^{m} b_i^+\Big)$$

Nous écrivons maintenant :

– pour exprimer que le chemin commence à l'abscisse zéro avec un domino initial :

$$b_1^- \wedge \mathbf{F}\,(d_{init} \wedge \mathbf{G}^{-1}\,(D \vee \varphi_{first})$$

– nous devons exprimer que le chemin aboutira dans l'état E juste après être passé dans une case du ruban numérotée $2^m - 1$, et un domino « final » :

$$\mathbf{F}\,\mathbf{N}\,(b_1^+ \wedge \mathbf{F}\,(E \wedge \mathbf{G}^{-1}\,(E \vee d_{fin} \vee \varphi_{last})))$$

– il faut contrôler que le « compteur » est correctement incrémenté à chaque étape :

$$\begin{aligned} \mathbf{G}\,\mathbf{N}\Big[(b_1 \wedge \mathbf{F}\,(b_2 \wedge \mathbf{F}\,b_1)) \Rightarrow \Big(\big(b_1^+ \;\Leftrightarrow\; \mathbf{F}\,\big(b_m \wedge \varphi_{\neg D} \wedge \mathbf{N}\,\mathbf{F}\,(b_1^- \wedge \mathbf{G}^{-1}\,\neg b_2)\big)\big) \wedge \\ \bigwedge_{i=1}^{m-1} \Big(\Big[\mathbf{F}\,(b_{i+1}^+ \wedge \varphi_{\neg D}) \;\Leftrightarrow\; \mathbf{F}\,\big(b_m \wedge \varphi_{\neg D} \wedge \mathbf{N}\,\mathbf{F}\,(b_1 \wedge \mathbf{G}^{-1}\,\neg b_2 \wedge \mathbf{N}\,\mathbf{F}\,(b_{i+1}^- \wedge \varphi_{\neg D}))\big)\Big] \\ \Leftrightarrow\; \Big[\mathbf{F}\,(b_i^+ \wedge \varphi_{\neg D}) \wedge \mathbf{F}\,\big(b_m \wedge \varphi_{\neg D} \wedge \mathbf{N}\,\mathbf{F}\,(b_i^- \wedge \mathbf{G}^{-1}\,\neg b_{i+1})\big)\Big]\Big)\Big)\Big] \end{aligned}$$

[2] Cette réduction suppose que m est supérieur ou égal à 3.

– une cellule et sa voisine de droite doivent avoir la même couleur sur leur arête commune :

$$\bigwedge_{d\in T} \mathbf{G}\left(d \Rightarrow \mathbf{N}\,\mathbf{F}\,(b_2 \wedge \mathbf{G}^{-1}\neg b_m \wedge \mathbf{N}\,\mathbf{F}\,(\mathbf{G}^{-1}\neg b_1 \wedge \bigvee_{\substack{d'\in T\\ \mathrm{left}(d)=\mathrm{right}(d')}} d'))\right)$$

– enfin, une cellule et sa voisine supérieure doivent également avoir la même couleur sur l'arête qu'elles ont en commun. Comme pour la preuve du lemme, nous définissons au préalable la formule φ_{up} de la façon suivante :

$$\varphi_{up} \stackrel{\text{def}}{=} b_1 \wedge \bigwedge_{i=1}^{m} \Big(\mathbf{N}\,\mathbf{F}\,(b_i^+ \wedge \varphi_{\neg D}) \;\Leftrightarrow\; \mathbf{I}\,\mathbf{F}\,(b_i^+ \wedge \varphi_{\neg D})\Big)$$

On peut donc écrire maintenant la formule voulue :

$$\begin{aligned}\mathbf{G}\Big(b_1 \Rightarrow \mathbf{N}\Big(\bigwedge_{d\in T}\Big(&\mathbf{F}\,(b_2 \wedge \varphi_{\neg D} \wedge \mathbf{N}\,\mathbf{F}\,(d \wedge \mathbf{G}^{-1}\neg b_1)) \Rightarrow\\ &\big[\mathbf{F}\,(E \wedge \mathbf{G}^{-1}\neg\varphi_{zero}) \vee \mathbf{F}\,\big(D \wedge \mathbf{F}\,(\varphi_{up} \wedge \mathbf{G}^{-1}\,(b_1 \vee \mathbf{G}^{-1}\neg\varphi_{up})\wedge\\ &\qquad \mathbf{F}\,(b_2 \wedge \varphi_{\neg D} \wedge \mathbf{N}\,\mathbf{F}\,(\bigvee_{\substack{d'\in T\\ \mathrm{up}(d)=\mathrm{down}(d')}} d' \wedge \mathbf{G}^{-1}\neg b_1))))\big]\Big)\Big)\Big)\end{aligned}$$

Cela nous donne donc une preuve que le model checking d'une formule de $\mathrm{L}(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{N})$ est EXPSPACE-complet. Il en est donc de même pour le fragment $\mathrm{L}(\mathbf{U}, \mathbf{S}, \mathbf{N})$. On peut également facilement étendre ce résultat à la satisfaisabilité, en procédant par exemple comme dans la démonstration du lemme 1.74, page 65.

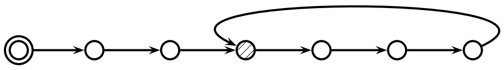
2.3 Model checking d'un chemin

Nous avons déjà évoqué le problème du model checking d'un chemin pour une formule de PLTL, à la section 1.3.2, à la page 43. Nous avions alors donné un algorithme en temps polynomial pour ce problème, en mentionnant que la complexité exacte n'est pas connue pour LTL et PLTL.

2.3.1 Un algorithme en temps polynomial

Considérons la boucle de la figure 2.3. L'algorithme de model checking de PLTL que

Figure 2.3 La boucle de départ



nous avons écrit dans le premier chapitre était un algorithme d'étiquetage. Nous avions établi, au théorème 1.48 de la page 45, qu'il fallait « dérouler » la boucle un certain nombre de fois afin que l'étiquetage ne soit pas ambigu. Nous allons établir le même résultat pour NLTL :

Théorème 2.32 ([LMS02b, lemme 5.2])
Soient φ une formule de NLTL, et L une boucle de type (m,p). Alors, pour tout entier k supérieur ou égal à $m + h_P(\varphi)\cdot p$ et pour toute sous-formule ψ de φ, on a

$$\pi_L, k \models \psi \;\Leftrightarrow\; \pi_L, k+p \models \psi.$$

Démonstration :
La preuve est simplement une extension de celle du théorème 1.48 pour l'opérateur $\mathbf{N}$. Nous devons donc montrer que

$$\pi_L, k \models \mathbf{N}\,\psi \;\Leftrightarrow\; \pi_L, k+p \models \mathbf{N}\,\psi,$$

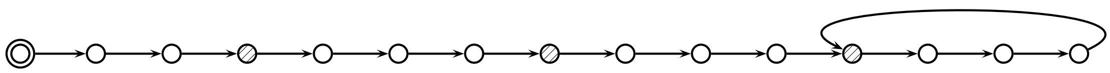
ce qui équivaut à montrer que

$$\pi_L^k, 0 \models \psi \;\Leftrightarrow\; \pi_L^{k+p}, 0 \models \psi.$$

Ce résultat est immédiat, puisque les chemins π_L^k et π_L^{k+p} sont isomorphes, puisque k est supérieur à m. □

Nous déroulons donc la boucle L de départ, pour obtenir la boucle de la figure 2.4, et

Figure 2.4 La boucle « déroulée »



nous utilisons maintenant un algorithme d'étiquetage pour résoudre le problème du model checking pour NLTL. La difficulté à laquelle nous sommes confronté est qu'il faut vérifier les sous-formules pour tous les suffixes du chemin, afin de prendre en compte les $\mathbf{N}$ éventuels.

Pour ce faire, nous allons en fait remplir un tableau booléen $V[i,j,\psi]$, de telle sorte que $V[i,j,\psi] = \top \;\Leftrightarrow\; \pi^j, i-j \models \psi$. L'indice j varie de 0 à $m+p$, puisque π^{m+p+1} est isomorphe à π^{m+1}. L'indice i varie de j à $m+(h_P(\varphi)+1)\cdot p$. Enfin, ψ parcourra la clôture de Fischer-Ladner de la formule φ de départ.

Remplir le tableau se fait par des techniques de programmation dynamique, en commençant par les plus petites formules de la clôture de Fischer-Ladner $\mathrm{FL}(\varphi)$. Pour les formules de la forme $\mathbf{N}\,\psi$, la valeur de $V[i,j,\mathbf{N}\,\psi]$ sera celle de $V[i,i',\psi]$ où

$$i' = \begin{cases} i & \text{si } i < m+p \\ i - k\cdot p & \text{si } i \in [\![m+kp, m+(k+1)\cdot p - 1]\!]. \end{cases}$$

L'utilisation de i' au lieu de i nous permet de ne considérer que $m+p$ suffixes de π, ce qu'il est possible de faire puisque les suffixes suivants sont isomorphes à ceux déjà considérés.

Cet algorithme s'exécute en temps polynomial : la taille du tableau V est $O(|\varphi|^2\cdot|L|)$, et la valeur de chaque case peut être calculée en temps polynomial.

2.3.2 P-complétude du model checking d'un chemin pour NLTL

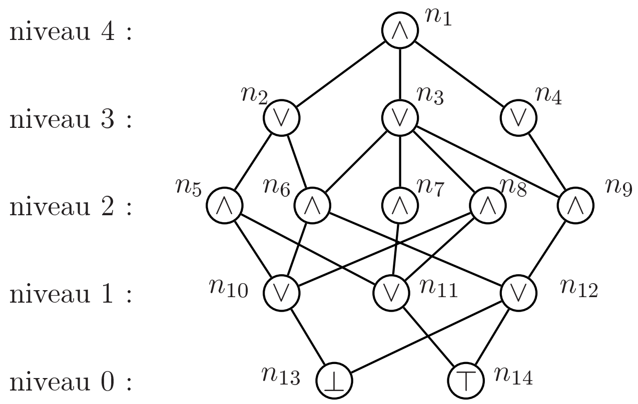
Nous allons voir que le problème du model checking d'un chemin pour NLTL est P-complet. Cela montre que l'algorithme précédent est « optimal ». C'est également un résultat intéressant par le fait que la complexité exacte du model checking d'un chemin pour PLTL ou LTL n'est pas connue.

Théorème 2.33 ([LMS02b, proposition 5.3])
Le model checking d'un chemin pour NLTL *est* P*-dur.*

Démonstration :

La preuve se fait par réduction d'une variante du problème Circuit-Value, nommée SAMCVP [GHR95, problème A.1.6]. Nous considérons donc un circuit *monotone* (*i.e.* qui n'utilise pas de négation), *alternant* (*i.e.* le long de tout chemin d'une entrée vers la sortie, les portes ET et OU alternent) et *synchrone* (*i.e.* les nœuds d'un niveau ne peuvent être reliés qu'au niveau précédent). Un exemple d'un tel circuit est représenté à la figure 2.5. Formellement, ce circuit est un graphe (N, V) où $N = \{n_1, n_2, \ldots, n_{14}\}$, et $V = \{(n_1, n_2), (n_1, n_3), \ldots, (n_{12}, n_{14})\}$, étiqueté par une fonction $\sigma \colon N \to \{\wedge, \vee, \top, \bot\}$. Nous supposons aussi donnée une fonction $niv \colon N \to \mathbb{N}$ qui donne le niveau d'un nœud.

Figure 2.5 Une instance du problème SAMCVP



De manière évidente, à ce circuit, nous pouvons associer une valuation v_C des nœuds de la façon suivante :

- le nœud n_{14} vérifie $v_C(n_{14}) = \top$;
- de la même façon, $v_C(n_{13}) = \bot$;
- un nœud étiqueté par $\wedge$ est évalué à vrai si, et seulement si, tous ses fils le sont ;
- un nœud étiqueté par $\vee$ est évalué à vrai si, et seulement si, un de s'est fils l'est.

Dans notre circuit de la figure 2.5, les nœuds n_{14}, n_{11}, n_{12}, n_7, n_9, n_3 et n_4 s'évaluent à vrai, les autres s'évaluent à faux.

À un tel circuit, nous associons la boucle de la figure 2.6. Nous allons maintenant écrire des formules φ_p telles que, pour tout nœud n_i de niveau p dans le circuit, φ_p sera vraie à

Figure 2.6 La boucle associée au problème SAMCVP

$$\pi_C : \begin{tikzcd} n_1 \arrow[r] & n_2 \arrow[r] & n_3 \arrow[r] & \cdots \arrow[r] & n_{13} \arrow[r] & n_{14} \arrow[r] & \bigcirc \arrow[loop right] \end{tikzcd}$$

la position $i-1$ (la position étiquetée par n_i) de la boucle si, et seulement si, le nœud n_i du circuit s'évalue à vrai.

Nous allons définir φ_p récursivement. Nous aurons besoin plusieurs fois dans cette définition d'une formule « testant » si la position courante correspond à un nœud qui est un « fils », dans le circuit, du nœud correpondant à la position initiale du chemin. Nous écrirons donc :

$$\varphi_{next} \stackrel{\text{def}}{=} \bigvee_{(n,n')\in V} \left(n' \wedge \mathbf{F}^{-1}\left(n \wedge \neg \mathbf{X}^{-1} \top\right)\right)$$

Nous définissons maintenant φ_p de la façon suivante :

$$\begin{aligned} \varphi_0 &\stackrel{\text{def}}{=} n_{14} \\ \varphi_{2k+1} &\stackrel{\text{def}}{=} \mathbf{N}\,\mathbf{F}\left(\varphi_{next} \wedge \varphi_{2k}\right) \\ \varphi_{2k+2} &\stackrel{\text{def}}{=} \mathbf{N}\,\mathbf{G}\left(\varphi_{next} \Rightarrow \varphi_{2k+1}\right) \end{aligned}$$

Il reste à établir l'équivalence suivante :

$$\forall p \in \mathbb{N}.\ \forall n_i \in N.\ \big((niv(n_i) = p) \Rightarrow (\pi_C, i \models \varphi_p \quad \Leftrightarrow \quad v_C(n_i) = \top)\big)$$

Cela se fait par induction sur p :
- si $p = 0$, c'est immédiat ;
- si le résultat est vrai au niveau p, considérons un nœud n_i au niveau $p+1$. Nous distinguons deux cas :
  - si $p+1$ est impair, alors le nœud n_i est étiqueté par $\vee$. Si $\pi_C, i-1 \models \varphi_p$, alors il existe un entier k tel que $\pi_C^{i-1}, k \models \varphi_{next} \wedge \varphi_p$. Par définition de φ_{next}, on sait que le nœud n_{k+1} est un fils du nœud n_i. Il est donc à la profondeur p. Puisqu'il vérifie φ_p, il s'évalue à vrai. Le nœud n_i également, donc. Inversement, si le nœud n_i s'évalue à vrai, un de ses fils s'évalue à vrai, et la position correspondante est un témoin du fait que $\pi_C, i-1 \models \mathbf{N}\,\mathbf{F}\left(\varphi_{next} \wedge \varphi_p\right)$;
  - si $p+1$ est pair, le nœud est étiqueté par $\wedge$. Si $\pi_C, i-1 \models \varphi_p$, alors toutes les positions correspondant à un fils du nœud n_i vérifient φ_p. Par induction, tous les fils du nœud n_i s'évaluent à vrai, donc le nœud n_i également. Inversement, si tous les fils de n_i s'évaluent à vrai, alors toutes les positions correspondant à ces nœuds vérifient φ_p, et donc $\pi_C, i-1 \models \mathbf{N}\,\mathbf{G}\left(\varphi_{next} \Rightarrow \varphi_p\right)$. □

Conclusion

Nous avons étudié dans ce chapitre l'ajout de l'opérateur $\mathbf{N}$, défini dans [LS95], à la logique temporelle du temps linéaire avec passé.

De la même manière que l'ajout des modalités du passé dans la logique LTL, l'ajout de cet opérateur permet de simplifier l'expression de certaines propriétés. En particulier, nous avons montré que certaines formules peuvent s'exprimer de manière exponentiellement plus concise avec $\mathbf{N}$. Cependant, la logique NLTL a le même pouvoir d'expression que la logique LTL, c'est à dire que toute formule de NLTL peut être traduite en une formule de LTL équivalente.

Par contre, en ce qui concerne les problèmes de vérification, nous avons vu que leur complexité est exponentiellement plus grande pour la logique NLTL que pour les logiques PLTL ou LTL étudiées au chapitre précédent. L'apport de concision a donc un prix dans le cas de l'opérateur $\mathbf{N}$, contrairement au cas des modalités du passé.

Pour en savoir plus...

Nous avons proposé plusieurs méthodes pour « oublier le passé », en particulier avec l'opérateur $\mathbf{C}$ (*chop*). Cet opérateur a été récemment étudié dans le cadre du μ-calcul de [Koz83] par [MO99]. La complexité du model checking reste élémentaire dans ce cas, puisque [LS02] montre que le problème est dans EXPTIME, et qu'il est PSPACE-complet si on borne le nombre d'alternations de points fixes. Par ailleurs, FLC est strictement plus expressif (*cf.* [MO99]) et exponentiellement plus concis (*cf.* [LS02]) que le μ-calcul.

Par ailleurs, [PZ93] étudie un opérateur de « concaténation » de chemins, semblable à l'opérateur $\mathbf{C}$. Cet opérateur permet de « concaténer » des chemins vérifiant certaines formules.

Concernant les logiques hybrides, de nombreux travaux ont été entrepris sur le sujet. La référence la plus complète se trouve à l'adresse `http://www.hylo.net/`.

CHAPITRE 3

Logiques temporelles du temps arborescent

Introduction

Au cours des deux premiers chapitres, nous nous sommes intéressé aux logiques temporelles du temps linéaire, c'est-à-dire à des logiques permettant d'exprimer des propriétés sur *une* exécution du système étudié. Cette exécution est « extraite » du système étudié, et on ne peut pas coder un comportement dépendant des différentes évolutions possibles à un moment donné. Dans la sémantique du temps linéaire, on ne considère qu'un seul successeur pour chaque état. Par exemple, il est impossible d'exprimer, en logique du temps linéaire, qu'un système *peut* toujours revenir à son état initial (*i.e.* que dans tout état, *il existe un chemin* retournant à l'état initial). On peut simplement écrire qu'il va *effectivement* toujours repasser par son état initial, ce qui est complètement différent.

La logique du temps arborescent offre cette possibilité, en ajoutant des quantificateurs de chemins. Il devient alors possible d'écrire qu'« il existe un chemin retournant dans l'état initial », par exemple. L'approche est alors complètement différente, que ce soit dans la spécification ou dans les algorithmes de vérification de propriétés avec ce type de logiques temporelles.

Tout comme pour le temps linéaire, il existe une grande variété de logiques temporelles du temps arborescent. La logique CTL [CE81], en particulier, a l'avantage d'être facilement vérifiable : le problème du model checking de cette logique est en effet P-complet [CES83]. Cela a valu à cette logique un succès incontestable dans les applications pratiques. Cette logique est cependant peu expressive, et de nombreuses extensions ont été définies à partir de CTL, par exemple CTL^+ [EH85], ECTL^+ [EH86] ou FCTL [EL87]. Étonnamment, la complexité du model checking de ces logiques n'est pas précisément connue, alors qu'elles ont été définies au milieu des années 1980. L'objet de ce chapitre est d'établir la complexité du model checking de ces logiques.

3.1 Le temps arborescent

Dans cette section, nous allons définir un certain nombre de logiques temporelles du temps arborescent. Ces logiques permettent d'exprimer des propriétés sur l'ensemble des exécutions possibles du système à partir d'un état donné, et non plus uniquement sur une seule exécution. Ces logiques ont été beaucoup étudiées depuis leur introduction en vérification, et il existe de nombreuses présentations générales dans la littérature [EH85, ES88, Eme90].

3.1.1 La logique CTL

Parmi les logiques temporelles du temps arborescent, la plus connue et la plus étudiée est CTL (Computational Tree Logic). Cette logique a été définie au début des années 1980, dans deux travaux indépendants [CE81, QS82].

3.1.1.1 Syntaxe de CTL

Comme aux chapitres précédents, nous considérons un ensemble Prop de propositions atomiques.

Définition 3.1
La syntaxe de CTL *est définie de la façon suivante :*

$$\begin{aligned} \mathsf{CTL} \ni \varphi, \psi \ &::= \ \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \dots \\ \mathsf{CTL}_c \ni \varphi_c \ &::= \ \varphi \, \mathbf{U} \, \psi \mid \mathbf{X} \, \varphi \end{aligned}$$

Les formules de CTL_c *sont appelées « formules de chemin », les formules de* CTL *étant des « formules d'état ».*

Intuitivement, **E** signifie « *il existe un chemin tel que* », et **A** signifie « *tout chemin vérifie* ». Les formules de CTL sont évaluées dans des états d'une structure arborescente, et les chemins en questions doivent donc partir de l'état où est évaluée la formule.

Les « modalités » **X** et **U** sont les mêmes modalités que dans la logique du temps linéaire. Ainsi, par exemple, $\mathbf{E}(\varphi \, \mathbf{U} \, \psi)$ signifie qu'il existe un chemin partant de l'état courant, et tel que ψ sera vraie dans un état futur, et que tous les états intermédiaires vérifieront φ. De la même façon , $\mathbf{AX} \, \varphi$ signifie que le long de tout chemin partant de l'état courant, le successeur de l'état courant satisfait φ. En d'autres termes, cela signifie que tous les successeurs de l'état courant satisfont φ.

Tout comme pour la logique du temps linéaire, nous pouvons définir d'autres modalités à partir des quatre modalités de départ :

$$\mathbf{EF}\,\varphi \overset{\text{def}}{=} \mathbf{E}(\top \, \mathbf{U} \, \varphi) \qquad \mathbf{AF}\,\varphi \overset{\text{def}}{=} \mathbf{A}(\top \, \mathbf{U} \, \varphi)$$

$$\mathbf{EG}\,\varphi \overset{\text{def}}{=} \neg\mathbf{AF}\,\neg\varphi \qquad \mathbf{AG}\,\varphi \overset{\text{def}}{=} \neg\mathbf{EF}\,\neg\varphi$$

$\mathbf{EF}\,\varphi$ signifie qu'il existe un chemin le long duquel φ sera vérifiée à une certaine position ; donc qu'il est possible d'arriver à un état vérifiant φ. $\mathbf{AF}\,\varphi$ signifie que φ sera vraie à une certaine position le long de tout chemin, *i.e.* que φ est inévitable. $\mathbf{AG}\,\varphi$ indique que φ est toujours vraie pour tout état accessible. Enfin, $\mathbf{EG}\,\varphi$ signifie qu'il existe un chemin le long duquel φ est toujours vraie.

Ainsi, par exemple, pour exprimer que, dans tout état, il est possible de retourner dans l'état initial, on écrira $\mathbf{AG}\,(\mathbf{EF}\ \texttt{init})$.

3.1.1.2 Sémantique de CTL

Nous venons de voir la signification intuitive des modalités de CTL ; donnons leur maintenant une sémantique précise : une formule de CTL est évaluée dans un état d'une structure de Kripke (la logique CTL est une « logique d'états », contrairement à LTL, qui est une « logique de chemin »).

Définition 3.2
Soit K une structure de Kripke, π un chemin dans K et s un état de K. Soient φ et ψ deux formules de CTL, *et φ_c une formule de* CTL_c. *On définit la relation $\models$ de manière inductive à partir des règles suivantes :*

$$
\begin{array}{rcll}
K,s & \models & \neg\varphi & \textit{si, et seulement si,} \quad K,s \not\models \varphi \\
K,s & \models & \varphi \vee \psi & \textit{si, et seulement si,} \quad K,s \models \varphi \textit{ ou } K,s \models \psi \\
K,s & \models & \mathbf{E}\varphi_c & \textit{si, et seulement si,} \quad \exists \pi \in \Pi_K.\ (\pi_0 = s \textit{ et } \pi \models \varphi_c) \\
K,s & \models & \mathbf{A}\varphi_c & \textit{si, et seulement si,} \quad \forall \pi \in \Pi_K.\ (\pi_0 = s \textit{ et } \pi \models \varphi_c) \\
K,s & \models & p & \textit{si, et seulement si,} \quad l(s)(p) = \top \qquad \forall p \in \mathrm{Prop} \\
\pi & \models & \mathbf{X}\,\varphi & \textit{si, et seulement si,} \quad K,\pi_1 \models \varphi \\
\pi & \models & \varphi\,\mathbf{U}\,\psi & \textit{si, et seulement si,} \quad \exists k \geqslant 0.\ ((K,\pi_k \models \psi) \wedge \\
 & & & \qquad\qquad (\forall 0 \leqslant j < k.\ K,\pi_j \models \varphi))
\end{array}
$$

Lorsqu'il n'y a pas d'ambiguïté, nous noterons simplement $s \models \varphi$ pour $K,s \models \varphi$.

Nous définissons une notion d'équivalence entre formules :

Définition 3.3
Deux formules φ et ψ de la logique temporelle du temps arborescent sont initialement équivalentes[1] *si, pour toute structure de Kripke K et pour tout état $s \in S$,*

$$K,s \models \varphi \textit{ si, et seulement si, } K,s \models \psi.$$

Nous définissons alors les problèmes de satisfaisabilité et de model checking de la façon suivante :

[1] Nous dirons simplement *équivalentes* dans la suite. Nous pourrions définir une notion d'équivalence globale, qui supposerait que nous évaluions une formule dans un état de la structure de Kripke avec un certain contexte, *i.e.* avec un certain passé.

Définition 3.4
Le problème de la satisfaisabilité pour CTL *est, étant donnée une formule de* CTL*, de vérifier s'il existe une structure de Kripke* $K = (S, S_0, l, R)$ *et un état initial* $s_0 \in S_0$*, tels que* $K, s_0 \models \varphi$.

Le problème du model checking consiste, étant données une structure de Kripke K *et une formule* φ*, à vérifier que, pour tout état initial* $s_0 \in S_0$*,* $K, s_0 \models \varphi$.

Pour CTL, nous avons les résultats de complexité suivants :

Théorème 3.5
Le problème du model checking pour CTL *est* P*-complet [CES86]. Le problème de la satisfaisabilité d'une formule de* CTL *est* EXPTIME*-complet [EH85].*

Le fait que l'algorithme de model checking soit optimal (*i.e.* le fait que le model checking de CTL soit P-dur) n'est pas un résultat très connu. Une démonstration en est donnée dans [Sch02, théorème 4.2].

3.1.1.3 Définition de nouvelles modalités

Tout comme nous l'avons fait pour la logique temporelle du temps linéaire, nous pouvons définir de nouvelles modalités à partir des quatre modalités de base de la logique CTL.

- les modalités **EF**, **EG**, **AF** et **AG**, dont nous avons parlé ci-dessus. Ces modalités s'appellent couramment « *potentially* », « *some* », « *inevitably* » et « *globally* », respectivement ;
- la modalité $\overset{\infty}{\mathbf{AF}}$ définie par $\overset{\infty}{\mathbf{AF}}\,\varphi \overset{\text{def}}{=} \mathbf{AG}\,(\mathbf{AF}\,\varphi)$. Cette formule signifie, comme on peut s'y attendre, que la propriété φ est vérifiée une infinité de fois le long de tout chemin de K ;
- la modalité $\overset{\infty}{\mathbf{EG}}$ peut être définie de la même manière : $\overset{\infty}{\mathbf{EG}}\,\varphi \overset{\text{def}}{=} \mathbf{EF}\,(\mathbf{EG}\,\varphi)$. Cette formule signifie qu'il existe un chemin le long duquel φ est vérifiée « presque partout » ;
- les modalités **EW** et **AW** : par exemple, **EW** peut être définie par $\mathbf{E}\varphi\,\mathbf{W}\,\psi \overset{\text{def}}{=} \neg(\mathbf{A}(\neg\psi)\,\mathbf{U}\,(\neg\varphi \wedge \neg\psi))$. Une telle formule signifie qu'il n'est pas vrai que tous les chemins vérifient $(\neg\psi)\,\mathbf{U}\,(\neg\varphi \wedge \neg\psi)$, et donc qu'il existe un chemin qui vérifie la formule $\varphi\,\mathbf{W}\,\psi$ telle que nous l'avons définie dans le cas de la logique du temps linéaire (*cf.* page 24) ;
- les modalités $\mathbf{EU}_s$ et $\mathbf{AU}_s$: on les définit respectivement par

$$\mathbf{E}\varphi\,\mathbf{U}_s\,\psi \overset{\text{def}}{=} \mathbf{EX}\,(\mathbf{E}\varphi\,\mathbf{U}\,\psi) \qquad\qquad \mathbf{A}\varphi\,\mathbf{U}_s\,\psi \overset{\text{def}}{=} \mathbf{AX}\,(\mathbf{A}\varphi\,\mathbf{U}\,\psi).$$

  Noter que, comme dans le cas de la logique PLTL, la modalité $\mathbf{EU}_s$ permet de redéfinir **EX** et **EU** :

$$\mathbf{EX}\,\varphi \equiv \mathbf{E}\bot\,\mathbf{U}_s\,\varphi \qquad\qquad \mathbf{E}\varphi\,\mathbf{U}\,\psi \equiv \psi \vee (\varphi \wedge \mathbf{E}\varphi\,\mathbf{U}_s\,\psi).$$

Remarquons que nous n'avons pas défini les modalités $\overset{\infty}{\mathbf{EF}}$ ou $\overset{\infty}{\mathbf{AG}}$. Intuitivement, la sémantique de $\overset{\infty}{\mathbf{EF}}\,\varphi$ serait « il existe un chemin le long duquel φ est vraie infiniment souvent », et $\overset{\infty}{\mathbf{AG}}$ aurait la sémantique duale. On a en fait le résultat suivant :

Proposition 3.6 ([EC80, EH86])
Il n'existe pas de formule CTL *exprimant le fait qu'il existe un chemin le long duquel une formule* φ *est satisfaite infiniment souvent (*i.e. *on ne peut pas exprimer* $\mathbf{E}\overset{\infty}{\mathbf{F}}\,\varphi$ *dans la logique* CTL*).*

Ce résultat est assez surprenant, et souligne un des problèmes de CTL : les formules de CTL sont assez difficiles à « comprendre ». Intuitivement, il eût été naturel d'exprimer $\mathbf{E}\overset{\infty}{\mathbf{F}}$ par $\mathbf{EG}\,\mathbf{EF}$. Cependant, la formule $\mathbf{EG}\,(\mathbf{EF}\,\varphi)$ indique simplement qu'il existe un chemin le long duquel, dans chaque état, on peut atteindre (éventuellement par un autre chemin) un état vérifiant φ. Dans la structure de la figure 3.1(a) par exemple, il n'existe

Figure 3.1 Structures de Kripke illustrant les difficultés d'utilisation de CTL

(a) Pour la formule $\mathbf{EG}\,(\mathbf{EF}\,\varphi)$

(b) Pour $\mathbf{AF}\,(\mathbf{AX}\,\varphi)$ et $\mathbf{AX}\,(\mathbf{AF}\,\varphi)$

pas de chemin le long duquel on rencontre infiniment souvent φ (donc $\mathbf{E}\overset{\infty}{\mathbf{F}}\,\varphi$ n'est pas vraie, quel que soit l'état de départ), alors que le chemin bouclant tout le temps dans l'état de gauche vérifie bien que, à toute les positions, il est possible d'atteindre un état vérifiant φ. [Var01] a souligné cette difficulté d'utilisation de CTL sur l'exemple suivant : « quelle est la différence entre $\mathbf{AX}\,(\mathbf{AF}\,\varphi)$ et $\mathbf{AF}\,(\mathbf{AX}\,\varphi)$? » En utilisant la structure de la figure 3.1(b), on peut mettre en évidence cette différence : l'état de gauche vérifie en effet la formule $\mathbf{AX}\,(\mathbf{AF}\,\varphi)$, puisque tous les chemins issus de tous ses successeurs passeront par l'état vérifiant φ. Par contre, le chemin partant de l'état de gauche et ne visitant jamais l'état central ne vérifie jamais $\mathbf{AX}\,\varphi$, puisque, pour l'état de gauche comme pour l'état de droite, il existe un successeur vérifiant $\neg\varphi$. L'état de gauche vérifie donc $\mathbf{EG}\,(\mathbf{EX}\,\neg\varphi)$, il ne vérifie donc pas $\mathbf{AF}\,(\mathbf{AX}\,\varphi)$.

D'autres problèmes d'expressivité de CTL apparaissent rapidement à l'usage. Ainsi, on ne peut pas (syntaxiquement, du moins) écrire la formule $\mathbf{E}(\mathbf{F}\,a \wedge \mathbf{F}\,b)$, exprimant qu'il existe un chemin le long duquel a et b seront satisfaits un jour (pas nécessairement simultanément). Pour toutes ces raisons, plusieurs extentions de CTL ont été définies, comme nous allons le voir dans la section suivante.

3.1.2 Quelques extensions de CTL

Nous allons voir ici plusieurs extensions de CTL, définies au cours des années 1980 afin de contourner les défauts d'expressivité de CTL.

3.1.2.1 La logique CTL*

La logique CTL*, proposée par [EH86], englobe à la fois CTL et LTL. Par rapport à CTL, elle autorise à écrire, dans la portée d'un quantificateur de chemin, une formule quelconque de LTL. Par rapport à LTL, elle permet de mettre des quantifications sur les chemins.

Définition 3.7
La syntaxe de CTL* *est définie par*

$$\mathsf{CTL}^\star \ni \varphi, \psi \ ::= \ \varphi \,\mathbf{U}\, \psi \mid \mathbf{X}\, \varphi \mid \mathbf{E}\varphi \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \ldots$$

La sémantique de CTL* se déduit directement de celle de CTL. Le fait de ne pas distinguer entre formules de chemin et formules d'état influe uniquement sur la syntaxe des formules.

Il est manifeste que cette logique contient CTL et LTL. La complexité des problèmes de vérification de CTL* sera donc au moins aussi importante que celles des problèmes de vérification de CTL et LTL. On a en fait :

Théorème 3.8
La satisfaisabilité de CTL* *est un problème* 2EXPTIME*-complet [VS85]. Le model checking de* CTL* *est* PSPACE*-complet [CES86].*

3.1.2.2 La logique CTL$^+$

La logique CTL* que nous venons de voir perd l'intérêt principal de CTL du point de vue de la vérification : la complexité du model checking devient PSPACE-complète, alors qu'elle était polynomiale pour CTL. Nous allons essayer maintenant de définir des extensions « intermédiaires » qui restent dans des classes de complexité assez basses.

Définition 3.9
La syntaxe de la logique CTL$^+$ *est définie comme suit :*

$$\begin{aligned}\mathsf{CTL}^+ \ni \varphi, \psi \ &::= \ \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \ldots \\ \mathsf{CTL}^+_c \ni \varphi_c \ &::= \ \varphi \,\mathbf{U}\, \psi \mid \mathbf{X}\, \varphi \mid \neg\varphi_c \mid \varphi_c \vee \psi_c \mid p \mid q \mid \ \ldots\end{aligned}$$

Par rapport à CTL, la logique CTL$^+$ autorise de faire des combinaisons booléennes des modalités $\mathbf{X}$ et $\mathbf{U}$ dans la portée d'un quantificateur de chemin. Ainsi, la formule $\mathbf{E}(\mathbf{F}\, a \wedge \mathbf{F}\, b)$ est une formule de CTL$^+$. Notons que l'on peut écrire une formule équivalente dans CTL, par exemple $\mathbf{EF}\,(a \wedge \mathbf{EF}\, b) \vee \mathbf{EF}\,(b \wedge \mathbf{EF}\, a)$. On a en fait le résultat suivant :

Théorème 3.10 ([EH85])
CTL *et* CTL$^+$ *ont la même expressivité.*

Dans [EH85], un algorithme effectif est donné, la formule d'arrivée étant exponentielle. Il existe en fait des cas où cette explosion est inévitable :

Théorème 3.11 ([Wil99b])
CTL$^+$ *peut être exponentiellement plus succinct que* CTL.

On a en fait un résultat plus précis :

Théorème 3.12 ([AI01])
La concision de CTL$^+$ *par rapport à* CTL *est exactement* $\Theta(n)!$.

La section 3.3 aborde la question de la complexité du model checking pour cette logique. Concernant la satisfaisabilité, la question a été résolue récemment :

Théorème 3.13 ([JL03])
La satisfaisabilité de CTL$^+$ *est un problème* 2EXPTIME*-complet.*

3.1.2.3 Les logiques ECTL et ECTL$^+$

Un autre défaut de CTL, que nous avons mentionné à la proposition 3.6, est que l'on ne peut pas exprimer qu'une propriété est satisfaite infiniment souvent le long d'un chemin.

Pour pallier ce problème, [EH86] a défini la logique ECTL (appelée $\mathrm{B}(\mathbf{F}, \mathbf{X}, \mathbf{U}, \overset{\infty}{\mathbf{F}})$ dans l'article original). Cette logique est définie de la même façon que CTL, mais avec une modalité ($\overset{\infty}{\mathbf{F}}$) en plus :

Définition 3.14
La logique ECTL *est définie par la syntaxe suivante :*

$$\begin{aligned} \mathsf{ECTL} \ni \varphi, \psi &::= \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \ldots \\ \mathsf{ECTL}_c \ni \varphi_c &::= \varphi \,\mathbf{U}\, \psi \mid \mathbf{X}\,\varphi \mid \overset{\infty}{\mathbf{F}}\,\varphi \end{aligned}$$

Nous savons déjà que cette logique est strictement plus expressive que CTL. Cependant, [EH86] prouve que cette logique n'est pas encore assez expressive : l'introduction de la modalité $\overset{\infty}{\mathbf{F}}$ a pour but, en pratique, de vérifier des propriétés le long de chemins vérifiant des propriétés « d'équité » (par exemple, le long de tous les chemins vérifiant une propriété de la forme $\overset{\infty}{\mathbf{F}}\,\varphi$). On voudrait donc pouvoir écrire des formules dela forme $\mathbf{E}(\psi \wedge \overset{\infty}{\mathbf{F}}\,\varphi)$). Dans cette optique, [EH86] définit également une extension[2] de ECTL, appelée ECTL$^+$, qui combine les avantages de ECTL et de CTL$^+$.

Définition 3.15
La logique ECTL$^+$ *est définie par la syntaxe suivante :*

$$\begin{aligned} \mathsf{ECTL}^+ \ni \varphi, \psi &::= \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \ldots \\ \mathsf{ECTL}^+_c \ni \varphi_c &::= \varphi \,\mathbf{U}\, \psi \mid \mathbf{X}\,\varphi \mid \overset{\infty}{\mathbf{F}}\,\varphi \mid \neg\varphi_c \mid \varphi_c \vee \psi_c \mid p \mid q \mid \ \ldots \end{aligned}$$

[EH86] prouve que cette logique est strictement plus expressive que les logiques ECTL et CTL$^+$ (en prouvant que $\mathbf{E}(\overset{\infty}{\mathbf{F}}\,a \wedge \overset{\infty}{\mathbf{F}}\,b)$ n'est pas exprimable dans ces deux logiques), et qu'elle reste strictement moins expressive que CTL* (car, par exemple, on ne peut pas exprimer $\mathbf{EG}\,(a \vee \mathbf{X}\,a)$ dans ECTL$^+$).

[2] Une logique très semblable a été étudiée auparavant, dans [EC80].

Concernant la complexité de la vérification de ces logiques, [CES86, théorème 6.2] prouve que le model checking de CTL^+ est NP-dur et co-NP-dur (et qu'il admet un algorithme dans Δ_2^P). Les théorèmes 3.8 et 3.13 impliquent que la satisfaisabilité de ECTL^+ est 2EXPTIME-complet.

3.1.2.4 Les logiques FCTL et GFCTL

Toujours dans le but d'imposer des conditions d'équité aux chemins considérés, [EL87] a défini les logiques FCTL et GFCTL.

Définition 3.16
Une spécification en FCTL *est un couple* (φ, Φ_F) *où* φ *est une formule de* CTL, *et* Φ_F *est une combinaison booléenne de formules de la forme* $\overset{\infty}{\mathbf{F}}\, p$ *et* $\overset{\infty}{\mathbf{G}}\, p$. *Dans la formule* φ, *les quantificateurs de chemin s'écrivent* $\mathbf{E}_F$ *et* $\mathbf{A}_F$, *et ne quantifient que sur les chemins vérifiant la formule d'équité* Φ_F.

Dans cette logique FCTL, une seule et même formule d'équité est utilisée pour tous les quantificateurs de chemins. Cela est généralement suffisant en pratique, mais pour éventuellement lever cette restriction, [EL87] définit également la logique GFCTL :

Définition 3.17
La logique GFCTL *est définie par la syntaxe suivante :*

$$\begin{aligned}
\mathsf{GFCTL} \ni \varphi, \psi \ &::= \ \mathbf{E}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ \ldots \\
\mathsf{GFCTL}_c \ni \varphi_c \ &::= \ \Phi_F \wedge (\varphi_s \,\mathbf{U}\, \psi_s) \mid \Phi_F \wedge \neg(\varphi_s \,\mathbf{U}\, \psi_s) \mid \Phi_F \wedge \mathbf{X}\, \varphi_s \\
\mathsf{GFCTL}_f \ni \Phi_F, \Psi_F \ &::= \ \neg\Phi_F \mid \Phi_F \vee \Psi_F \mid \overset{\infty}{\mathbf{F}}\, p \mid \overset{\infty}{\mathbf{G}}\, p \mid \overset{\infty}{\mathbf{F}}\, q \mid \overset{\infty}{\mathbf{G}}\, q \mid \ \ldots
\end{aligned}$$

Ces deux logiques sont incluses, syntaxiquement, dans la logique ECTL^+. Nous verrons à la section 3.3 que, du point de vue du model checking, elles ont la même complexité[3] que ECTL^+.

3.1.2.5 Autres extensions et restrictions de CTL

D'autres extensions de CTL ont été proposées.
- la logique CTL^F, définie dans [CES86], a la même syntaxe que CTL. Les formules s'appliquent a des structure de Kripke particulières, qui imposent une condition d'équité sur leurs chemins — cette condition est semblable à la condition d'acceptation généralisée des automates de Büchi (voir la définition 2.17) ;
- [EH86] définit les logiques B(. . .), de la même façon que les logiques linéaires L(. . .) que nous avions utilisées au premier chapitre. Cependant, ici, les logiques B(. . .) ne sont pas nécessairement des restrictions de CTL, comme nous le verrons plus bas, sur

[3] [EL87] affirme que le model checking de FCTL et de GFCTL est NP-complet, en réduisant ce problème au problème *Fair State Problem*, qui est lui-même NP-complet. La réduction fait cependant appel au problème *Fair State Problem* un nombre de fois proportionnel à la taille de la formule à vérifier. Ce genre de réduction est trop général pour la classe NP (*cf.* [Pap94, section 8.4]).

des exemples. La syntaxe de $\mathrm{B}(\mathbf{M}_1, \ldots, \mathbf{M}_p, \mathbf{O}_1, \ldots, \mathbf{O}_q)$, où $\mathbf{M}_i$ sont des modalités et $\mathbf{O}_j$ des opérateurs booléens, est définie comme suit :

$$\begin{aligned}
\mathrm{B}(\mathbf{M}_i, \mathbf{O}_j) \ni \varphi, \psi &::= \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid \neg\varphi \mid \varphi \vee \psi \mid p \mid q \mid \ldots \\
\mathrm{B}_c \ni \varphi_c, \psi_c &::= \mathbf{M}_i(\varphi, \psi, \ldots) \mid \mathbf{O}_j(\varphi_c, \psi_c, \ldots) \mid p \mid q \mid \ldots
\end{aligned}$$

Avec cette définition,

$$\begin{array}{ll}
\mathsf{CTL} = \mathrm{B}(\mathbf{U}, \mathbf{X}) & \mathsf{CTL}^+ = \mathrm{B}(\mathbf{U}, \mathbf{X}, \vee, \neg) \\
\mathsf{ECTL} = \mathrm{B}(\mathbf{U}, \mathbf{X}, \overset{\infty}{\mathbf{F}}) & \mathsf{ECTL}^+ = \mathrm{B}(\mathbf{U}, \mathbf{X}, \overset{\infty}{\mathbf{F}}, \vee, \neg)
\end{array}$$

Citons également $\mathsf{BTL} \overset{\text{def}}{=} \mathrm{B}(\mathbf{F})$ [Lam80] et $\mathsf{UB} \overset{\text{def}}{=} \mathrm{B}(\mathbf{F}, \mathbf{X})$ [BAPM83].

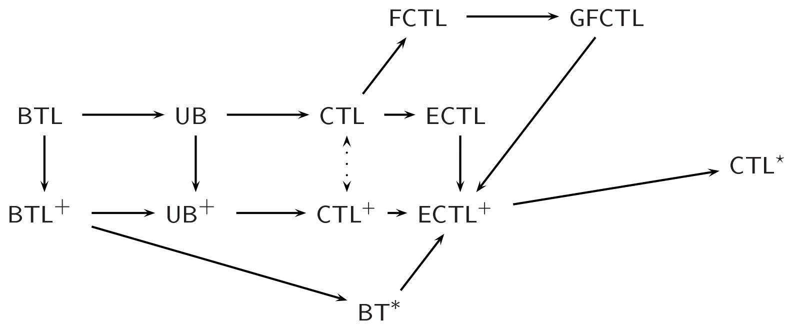
– de la même façon, nous définissons les fragments $\mathrm{B}^+(\ldots)$. La définition de ces fragments se fait simplement par $\mathrm{B}^+(\mathbf{M}_i) = \mathrm{B}(\mathbf{M}_i, \vee, \neg)$. Ainsi, $\mathsf{CTL}^+ = \mathrm{B}^+(\mathbf{U}, \mathbf{X})$, $\mathsf{UB}^+ \overset{\text{def}}{=} \mathrm{B}^+(\mathbf{F}, \mathbf{X})$ et $\mathsf{BTL}^+ \overset{\text{def}}{=} \mathrm{B}^+(\mathbf{F})$;
– les logiques de la forme $\mathrm{B}(\ldots)$ ne permettent pas d'atteindre $\mathsf{CTL}^\star$, puisqu'on ne peut toujours pas emboîter les modalités $\mathbf{U}$ et $\mathbf{X}$. Nous pouvons définir les logiques $\mathrm{B}^*(\mathbf{M}_1, \ldots, \mathbf{M}_p)$ de la façon suivante :

$$\mathrm{B}^*(\mathbf{M}_i) \ni \varphi, \psi ::= \mathbf{E}\varphi \mid \mathbf{A}\varphi \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{M}_i(\varphi, \psi, \ldots) \mid p \mid q \mid \ldots$$

Bien entendu, $\mathsf{CTL}^\star = \mathrm{B}^*(\mathbf{U}, \mathbf{X})$, et nous définissons $\mathsf{BT}^* \overset{\text{def}}{=} \mathrm{B}^*(\mathbf{F})$.

De nombreuses études ont été menées sur ces logiques. [EH85, EH86, ES88, Lar94] énoncent de nombreux résultats d'expressivité concernant ces logiques. Par contre, peu de résultats sont connus concernant la complexité du model checking pour ces différentes extensions de CTL. Nous résumons ces résultats dans les figures 3.2 et 3.3.

Figure 3.2 Expressivité comparée de quelques logiques du temps arborescent (La flèche en pointillé indique une différence de concision, les autres flèches indiquent une différence d'expressivité)



Dans la suite de ce chapitre, nous établissons les résultats manquant concernant la complexité du model checking des différentes extensions de CTL que nous avons définies dans cette section.

Figure 3.3 Quelques résultats de complexité des logiques que nous avons définies

	model checking	satisfaisabilité
BTL, UB, CTL, ECTL	P-complet [CES83, Sch02]	EXPTIME-complet [EH85, ES88]
CTL*	PSPACE-complet [CES86]	2EXPTIME-complet [ES84]

3.2 La hiérarchie polynomiale

Nous allons définir, dans cette section, une famille de classes de complexité se situant entre la classe NP et la classe PSPACE. Une présentation générale de ces classes peut être trouvée dans [GJ79, Sto76, Sto87, Pap94] par exemple.

3.2.1 Machines de Turing avec oracle

De manière informelle, une machine de Turing avec oracle est une machine de Turing $\mathcal{M}$ disposant d'un « oracle » auquel elle peut « poser des questions » et obtenir une réponse immédiatement.

Plus précisément [Wra76, HU79, Pap94] :

Définition 3.18
Soit Σ un alphabet, et A un langage sur cet alphabet. Une machine de Turing avec oracle *A est une machine de Turing, que nous noterons $\mathcal{M}^A$, sur l'alphabet $\Sigma \cup \Gamma$, à deux rubans : l'un sert à faire les calculs « usuels » de la machine de Turing, sur l'alphabet Γ ; l'autre, appelé « ruban de l'oracle », sert à la machine $\mathcal{M}^A$ à écrire un mot sur l'alphabet Σ. Si la machine $\mathcal{M}^A$ entre dans l'état particulier q_{oracle}, elle passe alors immédiatement dans l'état q_{oui} si le mot que contient le « ruban de l'oracle » appartient au langage A, et dans l'état q_{non} sinon. Dans les deux cas, le ruban de l'oracle est réinitialisé après une telle transition.*

Une telle machine est dite déterministe *si, à chaque étape, il existe au plus une transition possible.*

Considérons, par exemple, une machine de Turing $\mathcal{M}^A$ déterministe, avec oracle, et qui termine en temps polynomial. Si A est un langage tel que le problème de l'appartenance d'un mot à A est NP-complet (par exemple, si A est l'ensemble des instances satisfaisables de 3-SAT), alors la machine $\mathcal{M}^A$ peut décider si son mot d'entrée appartient à A : il suffit de copier ce mot sur le ruban de l'oracle, et de demander à l'oracle si ce mot appartient bien à A. La machine peut également décider si son mot d'entrée n'appartient pas à A (ce problème est alors co-NP). Comme nous le verrons plus loin, une telle machine peut en fait faire beaucoup plus (à condition, bien entendu, que $\mathsf{P} \neq \mathsf{NP}$).

3.2.2 La hiérarchie du temps polynomial

Nous venons de voir qu'une machine de Turing déterministe qui termine en temps polynomial, si elle a accès à un oracle NP, définit une classe de complexité plus grande que NP. Nous noterons P^{NP} la classe de complexité ainsi définie. De manière plus générale, si A est un langage appartenant à une classe de complexité $\mathcal{C}$, nous noterons $\mathsf{P}^{\mathcal{C}}$ la classe des problèmes que peut résoudre une machine de Turing déterministe, ayant accès à un oracle pour A, en temps polynomial. Nous noterons $\mathsf{NP}^{\mathcal{C}}$ la classe obtenue en levant la restriction sur le déterminisme.

On a alors $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$, de façon évidente. D'après ce que nous avons vu précédemment, co-$\mathsf{NP} \cup \mathsf{NP} \subseteq \mathsf{P}^{\mathsf{NP}}$.

Définition 3.19 ([MS72])
La hiérarchie du temps polynomial *est l'ensemble des classes de complexité définies par les relations suivantes :*

$$\Sigma_0^P \stackrel{\text{def}}{=} \mathsf{P} \qquad \Pi_0^P \stackrel{\text{def}}{=} \mathsf{P} \qquad \Delta_0^P \stackrel{\text{def}}{=} \mathsf{P}$$
$$\Sigma_{k+1}^P \stackrel{\text{def}}{=} \mathsf{NP}^{\Sigma_k^P} \qquad \Pi_{k+1}^P \stackrel{\text{def}}{=} co\text{-}\Sigma_{k+1}^P \qquad \Delta_{k+1}^P \stackrel{\text{def}}{=} \mathsf{P}^{\Sigma_k^P}$$

Ainsi, on vérifie que $\Sigma_1^P = \mathsf{NP}$, $\Pi_1^P =$ co-NP, et $\Delta_1^P = \mathsf{P}$ au premier niveau de la hiérarchie, et $\Sigma_2^P = \mathsf{NP}^{\mathsf{NP}}$, $\Pi_2^P =$ co-$\mathsf{NP}^{\mathsf{NP}}$, et $\Delta_2^P = \mathsf{P}^{\mathsf{NP}}$ au deuxième niveau. On se convainc facilement des inclusions suivantes :

$$\Delta_k^P \subseteq \Sigma_k^P \cap \Pi_k^P \qquad \Sigma_k^P \subseteq \Delta_{k+1}^P \qquad \Pi_k^P \subseteq \Delta_{k+1}^P$$

Par ailleurs, on montre assez facilement (mais ce n'est pas notre sujet) que toutes ces classes sont incluses dans la classe PSPACE [Pap94, proposition 17.1]. La hiérarchie peut donc être représentée comme à la figure 3.4. On ne sait pas si cette hiérarchie est stricte.

Figure 3.4 La hiérarchie du temps polynomial

$$\begin{tikzcd}[column sep=small]
 & \text{co-}\mathsf{NP} = \Pi_1^P \arrow[dr] & & \Pi_2^P \arrow[dr] & & \cdots \arrow[dr, dotted] & \\
\mathsf{P} \arrow[ur] \arrow[dr] & & \Delta_2^P \arrow[ur] \arrow[dr] & & \Delta_3^P \arrow[ur] \arrow[dr] & & \mathsf{PSPACE} \\
 & \mathsf{NP} = \Sigma_1^P \arrow[ur] & & \Sigma_2^P \arrow[ur] & & \cdots \arrow[ur, dotted] &
\end{tikzcd}$$

Cependant, on se convainc facilement que si deux niveaux sont égaux, alors toute la hiérarchie « s'effondre » jusqu'au premier niveau, et que tous les niveaux de la hiérarchie sont alors égaux. De même, si $\mathsf{P} = \mathsf{NP}$ ou si $\mathsf{NP} =$ co-NP, tous les niveaux (excepté le niveau zéro) de la hiérarchie sont alors égaux [Pap94].

3.2.3 Le problème SNSAT

Nous allons, dans cette section, définir un problème « de satisfaisabilité », semblable à 3-SAT, qui sera Δ_2^P-complet.

3.2.3.1 Quelques problèmes Δ_2^P-complet

[Pap84] a fournit la première preuve de Δ_2^P-complétude d'un problème « naturel ». Le problème en question est le problème de l'unicité du chemin optimal pour le voyageur de commerce :

UOTSP	entrée :	une instance du problème du voyageur de commerce, c'est à dire une matrice symétrique d'entiers positifs [GJ79, Pap94]
	sortie :	Le tour optimal est-il unique ?

D'autres problèmes Δ_2^P-complets ont été proposés dans [Wag87], [Miy91], ou [Got95]. Ce dernier article définit le problème DAGS(SAT) :

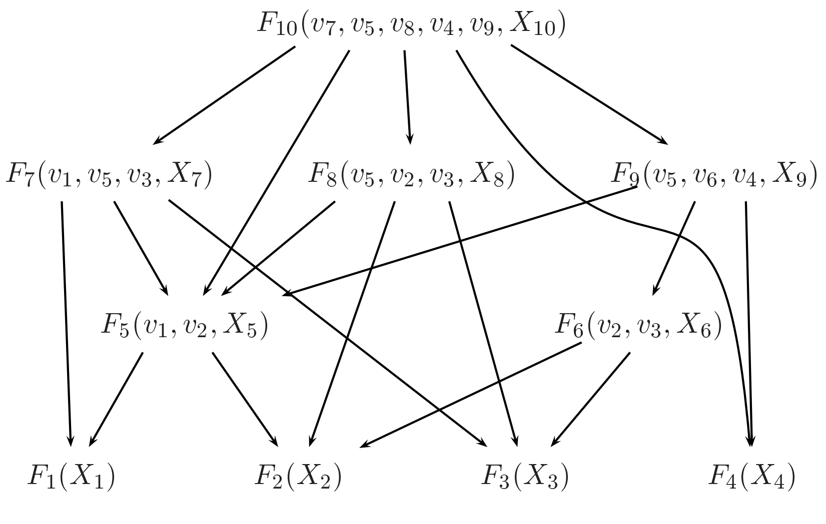
Définition 3.20 ([Got95])
Un NP-dag *est un graphe orienté acyclique G dont les nœuds sont des instances « paramétriques » de* 3-SAT *: si un nœud N a pour fils N_1, …, N_p, alors l'instance de N est un problème de satisfaisabilité d'une formule composée de propositions atomiques, mais aussi de variables v_1, …, v_p telles que $v_i = \top$ si l'instance N_i est satisfaisable. Bien entendu, un nœud n'ayant pas de fils est une instance « simple » de* 3-SAT*. Enfin, un des nœuds du* NP*-dag est appelé le « nœud résultat ».*

Le problème DAGS(SAT) *est le suivant :*

DAGS(SAT)	*entrée :*	*un* NP*-dag*
	sortie :	*le « nœud résultat » est-il satisfaisable ?*

La figure 3.5 montre une instance d'un tel problème. On peut supposer sans perte de généralité que les formules F_i sont sous forme normale conjonctive. On trouve alors facile-

Figure 3.5 Une instance du problème DAGS(SAT)



ment un algorithme dans Δ_2^P pour ce problème : sur l'instance de la figure 3.5, l'algorithme appellerait l'oracle NP sur les quatre instances $F_1(X_1)$, $F_2(X_2)$, $F_3(X_3)$ et $F_4(X_4)$. Il calculerait ensuite les instances F_5 et F_6 en fonction des résultats précédents, puis appellerait à nouveau l'oracle sur les deux instances de 3-SAT obtenues. De même ensuite pour F_7, F_8 et F_9, et enfin pour F_{10}.

On a en fait le résultat suivant :

Théorème 3.21 ([Got95])
Le problème DAGS(SAT) *est* Δ_2^P*-complet.*

Cela signifie que les graphes orientés acycliques permettent de combiner suffisamment les différents appels à l'oracle pour obtenir toute la complexité de la classe Δ_2^P. [Got95] montre que, si on considère des arbres au lieu des graphes orientés acycliques, le problème est plus facile : il est complet pour la classe $\mathsf{P}^{\mathsf{NP}}[O(\log n)]$, qui est une sous-classe de Δ_2^P dans laquelle on ne peut faire qu'un nombre logarithmique (et non plus polynomial) d'appels à l'oracle[4].

3.2.3.2 Définition du problème SNSAT

Le problème SNSAT est une version simplifiée du problème DAGS(SAT) :

Définition 3.22 ([LMS01])
SNSAT *entrée :* *un ensemble de variables booléennes $X = \{x_1, \ldots, x_n\}$, et une liste d'équivalences définissant les valeurs de ces variables :*

$$\mathcal{I} = \begin{cases} x_1 := \top \text{ si, et seulement si, } & \exists Z_1.\ F_1(Z_1) \\ x_2 := \top \text{ si, et seulement si, } & \exists Z_2.\ F_2(x_1, Z_2) \\ x_3 := \top \text{ si, et seulement si, } & \exists Z_3.\ F_3(x_1, x_2, Z_3) \\ \vdots & \\ x_n := \top \text{ si, et seulement si, } & \exists Z_n.\ F_n(x_1, \ldots, x_{n-1}, Z_n) \end{cases}$$

sortie : *Quelle est la valeur de x_n ?*

Ce problème est donc constitué de n instances du problème SAT, chacune dépendant de *toutes* les précédentes.

Soit $\mathrm{Var} = X \cup Z_1 \cup \cdots \cup Z_n$ l'ensemble des variables intervenant dans une instance de SNSAT. La liste $\mathcal{I}$ induit de manière évidente une valuation $v_{\mathcal{I}}$ des variables de X. Le problème SNSAT revient à décider si $v_{\mathcal{I}}(x_n) = \top$.

Nous dirons qu'une valuation $v\colon \mathrm{Var} \to \{\top, \bot\}$ est
- *sûre* lorsque, pour tout $i \leqslant n$, si $v(x_i) = \top$, alors $F_i(v(x_1), \ldots, v(x_{i-1}), v(Z_i)) = \top$;
- *correcte* si pour tout i, $v(x_i) = F_i(v(x_1), \ldots, v(x_{i-1}), v(Z_i))$;
- *admissible* si v est correcte et coïncide avec $v_{\mathcal{I}}$ sur X.

Cette distinction permet de mieux voir en quoi le problème SNSAT est « difficile » : il ne suffit pas de deviner les valeurs des variables des Z_i puis de calculer les valeurs des variables de X, comme pourrait le faire une machine de Turing non-déterministe en temps polynomial. En effet, toute valuation des variables des ensembles Z_i s'étend de façon unique en une valuation v sur Var, mais rien ne dit que cette valuation est *admissible*. On peut juste affirmer qu'elle est *correcte*, mais rien ne nous assure que les valeurs des variables des Z_i ont été choisies de façon « optimale », *i.e.* que si F_i était satisfaisable, alors les variables de l'ensemble Z_i la satisfont.

[4] La classe $\mathsf{P}^{\mathsf{NP}}[O(\log n)]$ contient les classes NP et co-NP, et est contenue dans la classe Δ_2^P. Bien entendu, il est possible que toutes ces classes coïncident, si la hiérarchie « s'effondre ».

Il existe toujours une valuation admissible pour une instance de SNSAT, mais vérifier son admissibilité nécessite bien de résoudre n problèmes de satisfaisabilité, chaque problème dépendant des précédents.

3.2.3.3 Complexité du problème SNSAT

D'après tout ce qui précède, nous obtenons facilement le résultat suivant :

Théorème 3.23 ([LMS01])
Le problème SNSAT *est* Δ_2^P*-complet.*

Démonstration :
Tout comme pour le problème DAGS(SAT), il est facile d'écrire un algorithme dans Δ_2^P résolvant ce problème : l'algorithme fera appel à l'oracle pour déterminer la valeur de x_1, remplacera x_1 par cette valeur dans F_2 puis appellera l'oracle pour calculer la valeur de x_2, et ainsi de suite.

De plus, ce problème est clairement Δ_2^P-difficile, puisque toute instance de DAGS(SAT) peut facilement se coder en une instance de SNSAT, la réduction utilisant un espace logarithmique. □

3.3 Complexité du model checking de CTL$^+$, ECTL$^+$, FCTL, ...

Nous allons voir dans cette section que les problèmes de model checking des logiques auxquelles nous nous intéressons sont Δ_2^P-complets. Nous montrons tout d'abord qu'ils sont Δ_2^P-difficiles pour les « petites » logiques (BTL$^+$ et FCTL), puis nous donnerons un algorithme dans Δ_2^P pour la logique ECTL$^+$.

3.3.1 Les bornes inférieures

Les « plus petites » logiques pour lesquelles nous voulons montrer que le model checking est Δ_2^P-complet sont BTL$^+$ et FCTL. Nous allons donc montrer, dans cette section, que le model checking est Δ_2^P-difficile pour ces deux logiques. Il s'ensuivra que le problème du model checking est Δ_2^P-difficile pour BTL$^+$, UB$^+$, CTL$^+$, ECTL$^+$, BT*, FCTL et GFCTL.

3.3.1.1 Le model checking de BTL$^+$ est Δ_2^P-difficile

Commençons par BTL$^+$. Pour simplifier cette réduction, nous allons la donner dans un premier temps pour la logique UB$^+$ (*i.e.* $\mathrm{B}^+(\mathbf{F}, \mathbf{X})$), puis nous modifierons les constructions afin de ne plus utiliser que la logique BTL$^+$ (*i.e.* $\mathrm{B}^+(\mathbf{F})$).

Théorème 3.24 ([LMS01])
Le problème SNSAT *se réduit, en espace logarithmique, au model checking de* UB$^+$*.*

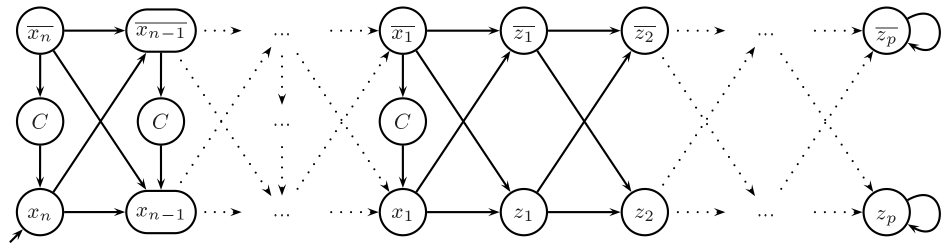
Démonstration :
Soit $\mathcal{I}$ une instance de SNSAT ; nous pouvons supposer, en toute généralité, que les

formules F_i sont sous forme normale conjonctive, et nous écrirons

$$F_i = \bigwedge_{j=1}^{l} \bigvee_{k=1}^{m} \alpha_{i,j,k}$$

où $\alpha_{i,j,k}$ est une variable $u \in Z_i \cup \{x_1, \ldots, x_{i-1}\}$ ou sa négation. À cette instance $\mathcal{I}$, nous associons la structure de Kripke représentée à la figure 3.6. Cette structure contient deux

Figure 3.6 La structure de Kripke $\mathcal{S}_\mathcal{I}$ associée à l'instance $\mathcal{I}$ de SNSAT



types d'états :
- les états u et $\overline{u}$, pour chaque variable $u \in \mathrm{Var}$;
- les états C, situés uniquement entre les nœuds $\overline{x_i}$ et x_i.

De même, il y a trois types de transitions dans cette structure :
- les transitions partant d'un état u ou $\overline{u}$ vers un état correspondant à la variable suivante, v ou $\overline{v}$, les variables étant énumérées dans l'ordre suivant :

$$x_n, x_{n-1}, \ldots, x_1, z_1, z_2, \ldots, z_p\,;$$

- les transitions allant d'un état $\overline{x_i}$ vers l'état x_i, en passant par le i-ème état C ;
- les boucles sur les états z_p et $\overline{z_p}$, dont le rôle est simplement de rendre les chemins infinis.

Soit π un chemin initial dans cette structure de Kripke. Si π ne visite jamais un état C, alors pour chaque variable $u \in \mathrm{Var}$, le chemin passe soit par u, soit par $\overline{u}$. On peut donc associer à un tel chemin une valuation des variables, en convenant que, si le chemin passe par u, la variable u vaut $\top$, et s'il passe par $\overline{u}$, la variable u vaut $\bot$. Nous noterons v_π cette valuation (en supposant, toujours, que π ne visite aucun état C). Inversement, à une valuation v des variables de Var, nous pouvons associer un chemin initial π_v ne passant jamais par un état C, et passant par l'état u si, et seulement si, $v(u) = \top$.

Nous allons maintenant définir une famille de formules de UB$^+$, qui nous permettra de relier la structure $S_\mathcal{I}$ à la liste des équivalences de l'instance de SNSAT que nous considérons.

Pour cela, nous définissons tout d'abord la formule ρ, de la façon suivante :

$$\rho \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \left[(\mathbf{F}\, x_i) \Rightarrow \bigwedge_{j=1}^{l} \bigvee_{k=1}^{m} \mathbf{F}\, \alpha_{i,j,k} \right].$$

Soit π un chemin initial dans $S_{\mathcal{I}}$ ne passant jamais par C, et v_π la valuation associée. Alors ρ est vraie le long de π si, et seulement si, la valuation v_π est sûre.

Nous définissons maintenant, récursivement, les formules suivantes :

$$\varphi_0 \stackrel{\text{def}}{=} \top$$
$$\varphi_k \stackrel{\text{def}}{=} \mathbf{E}\left[\mathbf{G}\,\neg C \wedge \rho \wedge \mathbf{G}\left(\left(\bigvee_{i=1}^{n} \overline{x_i}\right) \Rightarrow \mathbf{EX}\left(C \wedge \mathbf{EX}\,\neg\varphi_{k-1}\right)\right)\right]$$

Intuitivement, la formule φ_k signifie qu'il existe un chemin ne passant jamais par C, dont la valuation associée soit sûre, et pouvant « justifier », le cas échéant, qu'une variable x_i soit évaluée à $\perp$.

Pour prouver que cette réduction correspond effectivement à l'intuition que nous venons de nous en faire, nous montrons le lemme suivant :

Lemme 3.25
Soit k un entier, et $r \in [\![1, n]\!]$. Alors
- *si $k \geqslant 2r-1$, alors $v_{\mathcal{I}}(x_r) = \top$ si, et seulement si, $x_r \models \varphi_k$;*
- *si $k \geqslant 2r$, alors $v_{\mathcal{I}}(x_r) = \perp$ si, et seulement si, $\overline{x_r} \models \varphi_k$;*

Démonstration :

La preuve de ce lemme se fait par induction sur k. Lorsque $k = 0$, le résultat est évidemment vrai puisqu'il n'existe pas de valeur de r satisfaisant les inégalités entre k et r. Soit donc k un entier strictement positif. Nous supposons que le lemme est vrai pour $k-1$.

- montrons tout d'abord le sens direct pour chacune des deux équivalences. Pour ce faire, nous considérons une valuation admissible w, et π le suffixe de π_w commençant dans l'état x_r (ou dans l'état $\overline{x_r}$, suivant la valeur de $w(x_r)$). Nous allons montrer que, si $k \geqslant 2r-1$ (respectivement $k \geqslant 2r$), et si nous supposons que $v_{\mathcal{I}}(x_r) = \top$ (respectivement $v_{\mathcal{I}}(x_r) = \perp$), ce chemin π « témoigne » du fait que $x_r \models \varphi_k$ (respectivement $\overline{x_r} \models \varphi_k$).
  La valuation w étant admissible, donc sûre, son chemin associé satisfait $\mathbf{G}\,\neg C \wedge \rho$, donc π aussi. Montrons que π satisfait également la dernière partie de la formule φ_k.
  - si $k \geqslant 2r-1$ et $v_{\mathcal{I}}(x_r) = \top$, supposons que π visite un état $\overline{x_i}$. Alors $w(x_i) = \perp$, et donc $v_{\mathcal{I}}(x_i) = \perp$ puisque w est admissible. Par ailleurs, on sait que $i < r$, puisque $v_{\mathcal{I}}(x_i) = \perp$ et $v_{\mathcal{I}}(x_r) = \top$. Alors $k-1 \geqslant 2i-1$, et par hypothèse d'induction, puisque $v_{\mathcal{I}}(x_i) \neq \top$, on obtient que $x_i \not\models \varphi_{k-1}$. Donc $\overline{x_i} \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,(\neg\varphi_{k-1}))$;
  - si $k \geqslant 2r$ et $v_{\mathcal{I}}(x_r) = \perp$, supposons de la même façon que π visite un état $\overline{x_i}$. Comme précédemment, nous en déduisons que $v_{\mathcal{I}}(x_i) = \perp$. Si $i = r$, on a alors $k \geqslant 2r = 2i$, et sinon, $i < r$. Dans les deux cas, $k-1 \leqslant 2i-1$, et, par hypothèse d'induction, $x_i \not\models \varphi_{k-1}$, d'où l'on déduit $\overline{x_i} \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,(\neg\varphi_{k-1}))$;
- montrons maintenant la réciproque de ces deux équivalences : Soit r tel que $k \geqslant 2r-1$ et $x_r \models \varphi_k$ (respectivement $k \geqslant 2r$ et $\overline{x_r} \models \varphi_k$). Il existe un chemin π, partant de l'état x_r (respectivement $\overline{x_r}$), et qui témoigne du fait que φ_k est satisfaite dans l'état x_r (respectivement $\overline{x_r}$). Nous allons montrer, par récurrence, que la valuation v_π correspondante vérifie que $v_\pi(x_i) = v_{\mathcal{I}}(x_i)$ pour tout $i \in [\![1, r]\!]$.

– dans le cas où $i = 1$: si $v_\pi(x_1) = \top$, alors π passe par l'état x_1, et vérifie donc $\bigwedge_j \bigvee_k \mathbf{F}\,\alpha_{1,j,k}$. La formule F_1 est donc satisfaisable, et $v_\mathcal{I}(x_1) = \top$.
Inversement, si $v_\pi(x_1) = \bot$, alors le chemin π passe par l'état $\overline{x_1}$, et cet état satisfait la formule $\mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\varphi_{k-1})$. Donc $x_1 \not\models \varphi_{k-1}$. Si $r > i$, alors nécessairement $k-1 \geqslant 2i-1$ et $v_\mathcal{I}(x_1) \neq \top$. Si $r = i$, alors $\overline{x_r} \models \varphi_k$ et $k \geqslant 2r$ (d'après l'hypothèse de départ). Donc $k-1 \geqslant 2i-1$, et à nouveau $v_\mathcal{I}(x_i) \neq \top$.
– supposons le résultat vrai jusqu'au rang $i-1$, et montrons-le maintenant pour le rang i. Si $v_\pi(x_i) = \top$, l'argument est le même que pour le cas $i = 1$: le chemin π passe par l'état x_i, et vérifie ρ. La formule $\bigwedge_j \bigvee_k v_\pi(\alpha_{i,j,k})$ est donc vraie. Donc $F_i(v_\pi(x_1), \ldots, v_\pi(x_{i-1}), Z_i)$ est satisfaisable. Par hypothèse de récurrence (sur i), $v_\pi(x_t)$ et $v_\mathcal{I}(x_t)$ coïncident pour $t \leqslant i-1$. Donc $F_i(v_\mathcal{I}(x_1), \ldots, v_\mathcal{I}(x_{i-1}), Z_i)$ est satisfaisable, et $v_\mathcal{I}(x_i) = \top$;
Inversement, si $v_\pi(x_i) = \bot$, le chemin π passe par $\overline{x_i}$. Cet état satisfait donc $\mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\varphi_{k-1})$, et par suite, l'état x_i ne satisfait pas φ_{k-1}. Si $i < r$, on a alors $k-1 \geqslant 2i-1$, et donc $v_\mathcal{I}(x_i) \neq \top$ par induction. Si $i = r$, alors $\overline{x_r} \models \varphi_k$, et $k \geqslant 2r$ (d'après l'hypothèse de départ). Donc $k-1 \geqslant 2i-1$ et on a, toujours par induction, $v_\mathcal{I}(x_i) \neq \top$. □

On en déduit immédiatement le résultat que nous cherchions :

$$v_\mathcal{I}(x_n) = \top \text{ si, et seulement si, } \; x_n \models \varphi_{2n-1}.$$

Nous avons donc bien écrit une réduction, calculable en espace logarithmique, de SNSAT au problème du model checking d'une formule de UB$^+$. □

Étendons maintenant ce résultat à la logique BTL$^+$:

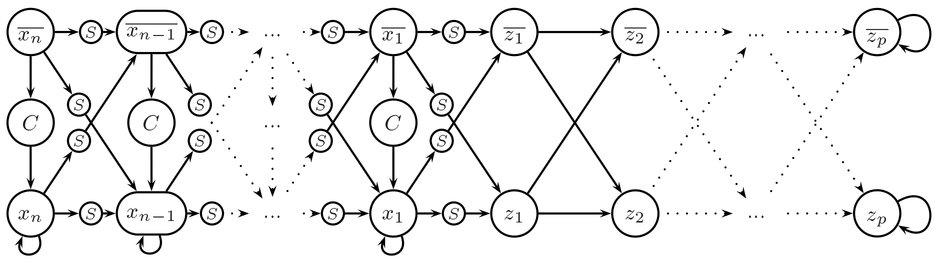
Théorème 3.26 ([LMS01])
Le problème SNSAT *se réduit, en espace logarithmique, au model checking de* BTL$^+$.

Démonstration :
Dans la démonstration précédente, nous utilisons la modalité $\mathbf{X}$ uniquement nous assurer que l'on passe bien d'un état $\overline{u}$ à l'état u correspondant. nous allons modifier la structure de telle sort que, entre un état x_i ou $\overline{x_i}$ et un état x_j ou $\overline{x_j}$, avec $j < i$, il y ait nécessairement un état « particulier », que nous étiquetterons pas la proposition S. L'absence d'un état S le long d'un chemin nous assurera alors que nous sommes bien passé d'un état $\overline{u}$ à un état u.

La structure de Kripke sera donc modifiée comme sur la figure 3.7. Nous modifions également la définition des formules φ_k en conséquence : au lieu d'écrire $\mathbf{EX}\,(C \wedge \mathbf{EX}\,(\neg\varphi_{k-1}))$, nous écrirons qu'il existe un chemin ne passant jamais par un état S, et passant par un état x_i dans lequel φ_{k-1} n'est pas satisfaite. Plus précisément :

$$\varphi'_0 \stackrel{\text{def}}{=} \top$$
$$\varphi'_k \stackrel{\text{def}}{=} \mathbf{E}\left[\mathbf{G}\,\neg C \wedge \rho \wedge \mathbf{G}\left(\Big(\bigvee_{i=1}^{n} \overline{x_i}\Big) \Rightarrow \mathbf{E}\Big[\neg\mathbf{F}\,S \wedge \mathbf{F}\,\Big(\big(\bigvee_{i=1}^{n} x_i\big) \wedge \neg\varphi'_{k-1}\Big)\Big]\right)\right]$$

Figure 3.7 La structure de Kripke $S'_{\mathcal{I}}$ associée à l'instance $\mathcal{I}$ de SNSAT



Cela nous fournit bien des formules de BTL^+, et on peut prouver le lemme 3.25 de la même façon que précédemment avec ces nouvelels formules. □

De ce résultat, nous déduisons le théorème suivant :

Théorème 3.27
Le model checking des logiques BTL^+, UB^+, CTL^+, ECTL^+ *et* BT^* *est* Δ_2^P*-difficile.*

3.3.1.2 Le model checking de FCTL est Δ_2^P-difficile

Nous allons établir le résultat suivant :

Théorème 3.28 ([LMS01])
Le problème SNSAT *se réduit, en espace logarithmique, au model checking de* FCTL.

Démonstration :
La réduction de SNSAT au problème du model checking de FCTL est basée sur les réductions précédentes. Néanmoins, la réduction précédente repose en grande partie sur le fait que nous pouvons écrire, dans BTL^+, des combinaisons booléennes de formules de chemin dans la portée d'un seul quantificateur de chemin. Cela n'est plus possible dans FCTL. Par contre, dans FCTL, il est possible d'associer, à une formule de chemin, une (et une seule) formule d'équité sur le chemin.

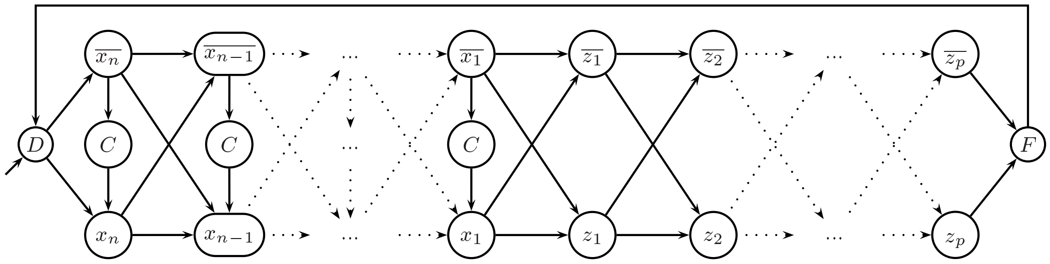
Une formule d'équité permet de s'assurer qu'un état est visité une infinité de fois le long d'un chemin. Afin d'utiliser au mieux cette possibilité, nous modifions la structure $S_{\mathcal{I}}$ comme montré sur la figure 3.8.

Avec cette nouvelle structure, bien entendu, un chemin ne correspondra pas à une valuation des variables de Var, même s'il ne passe par aucun état C. Nous allons alors introduire la condition d'équité suivante :

$$\Phi \stackrel{\text{def}}{=} \bigwedge_{u \in \text{Var}} \big[\overset{\infty}{\mathbf{G}} \neg u \vee \overset{\infty}{\mathbf{G}} \neg \overline{u}\big].$$

Si un chemin π vérifie cette contrainte d'équité, alors il existe un entier i tel que, pour tout $j \geqslant i$, le suffixe π^j de π ne visite jamais u (et donc infiniment souvent $\overline{u}$) ou jamais $\overline{u}$

Figure 3.8 La structure de Kripke $\mathcal{S}''_{\mathcal{I}}$ associée à l'instance $\mathcal{I}$ de SNSAT



(et infiniment souvent u), pour toute variable $u \in \mathrm{Var}$. Il est clair alors que ces suffixes ne passent jamais dans un état C. Nous pouvons donc associer au chemin π une valuation v'_π des variables de Var, de façon naturelle. Inversement, à toute valuation v des variables de Var, nous pouvons associer le chemin π'_v ne passant jamais par u si $v(u) = \bot$, et jamais par $\overline{u}$ si $v(u) = \top$, pour tout $u \in \mathrm{Var}$. Il est clair que ce chemin vérifie la contrainte Φ.

Par ailleurs, la formule ρ que nous avions écrite dans la démonstration du théorème 3.24 doit subir les mêmes transformations. Nous définissons donc :

$$\rho' \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \left[(\stackrel{\infty}{\mathbf{F}}\, x_i) \Rightarrow \bigwedge_{j=1}^{l} \bigvee_{k=1}^{m} \stackrel{\infty}{\mathbf{F}}\, \alpha_{i,j,k} \right].$$

Nous allons maintenant transformer les formules φ_k, afin d'y intégrer ces contraintes d'équité. Pour simplifier, nous commençons par écrire ces formules dans GFCTL :

$$\xi_0 \stackrel{\text{def}}{=} \top$$

$$\xi_k \stackrel{\text{def}}{=} \mathbf{E}\left[\mathbf{G}\left((\bigvee_{2i-1 \leqslant k} \overline{x_i}) \Rightarrow \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\xi_{k-1}) \right) \wedge \rho' \wedge \Phi \right]$$

De la même façon que dans les deux preuves précédentes, le lemme suivant permet de relier le problème SNSAT et la vérification des formules précédentes :

Lemme 3.29
Soit k un entier, et $r \in [\![1, n]\!]$. Alors
- *si $k \geqslant 2r-1$, alors $v_{\mathcal{I}}(x_r) = \top$ si, et seulement si, $x_r \models \xi_k$;*
- *si $k \geqslant 2r$, alors $v_{\mathcal{I}}(x_r) = \bot$ si, et seulement si, $\overline{x_r} \models \xi_k$;*

Démonstration :

La preuve se fait à nouveau par induction sur k, et pour $k = 0$, le résultat est toujours immédiat, puisqu'aucun r ne satisfait l'ensemble des hypothèses.

Nous prenons donc $k \geqslant 1$, et nous supposons que le lemme est vrai pour $k-1$. Soit r un entier entre 1 et n. Nous montrons d'abord le sens direct des deux équivalences, puis la réciproque.

– Soit w une valuation admissible, et π_w le chemin de $S''_{\mathcal{I}}$, commençant en x_n ou $\overline{x_n}$, et qui passe par

$$\begin{array}{l} \overline{x_i} \text{ si } i > r \text{ ou } w(x_i) = \bot, \\ x_i \text{ si } w(x_i) = \top \text{ et } i \leqslant r, \\ \overline{z_j} \text{ si } w(z_j) = \bot, \\ z_j \text{ si } w(z_j) = \top \end{array}$$

à chaque « tour » dans la structure $S''_{\mathcal{I}}$. Il est alors clair que ce chemin vérifie Φ. Il vérifie également ρ', puisque la valuation w est admissible, donc sûre. Il nous reste à montrer que le début de la formule est vérifié aussi :

  – si $k \geqslant 2r-1$ et $v_{\mathcal{I}}(x_r) = \top$, alors si π visite un état $\overline{x_i}$ avec $2i-1 \leqslant k$, on a alors $i \leqslant r$, et même $i < r$ puisque $v_{\mathcal{I}}(x_i) = \bot$ alors que $v_{\mathcal{I}}(x_r) = \top$. Donc $k-1 \geqslant 2i-1$, et puisque $v_{\mathcal{I}}(x_i) \neq \top$, l'hypothèse de récurrence nous donne que $x_i \not\models \varphi_{k-1}$. Donc $\overline{x_i} \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,(\neg\varphi_{k-1}))$;
  – si $k \geqslant 2r$ et $v_{\mathcal{I}}(x_r) = \bot$, et si π visite un état $\overline{x_i}$ avec $2i-1 \leqslant k$, on a alors encore $v_{\mathcal{I}}(x_i) = \bot$ et $i \leqslant r$. Donc $k \geqslant 2i$, et $k-1 \geqslant 2i-1$. Par hypothèse de récurrence, $x_i \not\models \varphi_{k-1}$, d'où le résultat recherché ;

– passons maintenant à la réciproque de ces deux implications. Supposons que $k \geqslant 2r-1$ et $x_r \models \varphi_k$ (respectivement $k \geqslant 2r$ et $\overline{x_r} \models \varphi_k$). Il existe alors un chemin partant de x_r (respectivement $\overline{x_r}$) vérifiant φ_k. En particulier, ce chemin vérifie Φ, et donc, à partir d'une certaine position p, le chemin satisfait

$$\bigwedge_{u \in \mathrm{Var}} \left[\mathbf{G}\,\neg u \vee \mathbf{G}\,\neg\overline{u}\right].$$

Nous considérons alors le suffixe de π, noté π', commençant à cette position p. Ce suffixe vérifie toujours, *a fortiori*, la formule

$$\mathbf{G}\left(\Big(\bigvee_{2i-1 \leqslant k} \overline{x_i}\Big) \Rightarrow \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\xi_{k-1})\right).$$

Nous associons à ce chemin π' la valuation définie comme suit :

$$v_{\pi'}(u) = \top \;\Leftrightarrow\; \pi' \models \mathbf{G}\,\neg\overline{u}.$$

Cette valuation n'est pas ambiguë, et est bien définie pour toute variable u, d'après ce qui précède. Nous allons montrer, par récurrence, que pour tout $i \in [\![1, r]\!]$, on a $v_{\mathcal{I}}(x_i) = v_{\pi'}(x_i)$.

  – pour $i = 1$: si $v_{\pi'}(x_1) = \top$, alors π' passe par l'état x_1, donc vérifie $\bigwedge_j \bigvee_k \mathbf{F}\,\alpha_{1,j,k}$. La formule F_1 est donc satisfaisable, d'où le résultat.
  Inversement, si $v_{\pi'}(x_1) = \bot$, alors π' passe par $\overline{x_1}$, et donc $\overline{x_1} \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\varphi_{k-1})$, puisque $2i-1 \leqslant k$. Donc $x_1 \not\models \varphi_{k-1}$. Si $r > i$, alors $k-1 \geqslant 2i-1$, et $v_{\mathcal{I}}(x_i) \neq \top$. Si $r = i = 1$, alors $\overline{x_r} \models \varphi_k$, et on est donc dans le cas ou $k \geqslant 2r$. Donc $k-1 \geqslant 2r-1$, et $v_{\mathcal{I}}(x_i) \neq \top$;
  – prenons maintenant $i > 1$, et supposons que le résultat soit vrai à tous les rangs strictement inférieurs à i. Si $v_{\pi'}(x_i) = \top$, alors le chemin π' passe par x_i, et vérifie ρ. La formule $\bigwedge_j \bigvee_k \mathbf{F}\,\alpha_{i,j,k}$ est donc vraie, et donc $F_i(v_{\pi'}(x_1), \ldots, v_{\pi'}(x_1), Z_i)$ est

satisfaisable. Par hypothèse de récurrence, $v_{\pi'}(x_t) = v_{\mathcal{I}}(x_t)$ pour $t < i$. Donc $F_i(v_{\mathcal{I}}(x_1), \ldots, v_{\mathcal{I}}(x_1), Z_i)$, et $v_{\mathcal{I}}(x_i) = \top$.
Réciproquement, si $v_{pi'}(x_i) = \top$, le chemin π passe par $\overline{x_i}$. Donc $x_i \not\models \varphi_{k-1}$, puisque $2i-1 \leqslant k$. Si $i < r$, alors $k-1 \geqslant 2i-1$, et donc $v_{\mathcal{I}}(x_i) \neq \top$, par hypothèse de récurrence sur k. Si $i = r$, on a alors $\overline{x_r} \models \varphi_k$, et on est donc dans le cas où $k \geqslant 2r$. Donc $k-1 \geqslant 2i-1$, et à nouveau, $v_{\mathcal{I}}(x_i) \neq \top$. □

On obtient à nouveau l'équivalence

$$v_{\mathcal{I}}(x_n) = \top \text{ si, et seulement si, } \ x_n \models \varphi_{2n-1}.$$

on a donc bien écrit une réduction de SNSAT au problème du model checking d'une formule de GFCTL. Afin de faire de nos formules ci-dessus des formules de FCTL, il faut que la même contrainte d'équité soit appliquée à tous les quantificateurs de chemins. Cependant, il est facile de remarquer que, de tout chemin de la structure, il existe un chemin vérifiant la condition d'équité $\Phi \wedge \rho'$. Les quantificateurs de chemin apparaissant dans $\mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\xi_{k-1})$ peuvent être restreints par cette contrainte d'équité, puisqu'ils ne s'intéressent qu'au deuxième état du chemin sur lequel ils quantifient, et pas du tout à son comportement global. Le résultat précédent est donc toujours vrai pour les formules suivantes :

$$\begin{aligned} \xi_0 &\stackrel{\text{def}}{=} \top \\ \xi_k &\stackrel{\text{def}}{=} \mathbf{E}\left[\mathbf{G}\left(\Big(\bigvee_{2i-1 \leqslant k} \overline{x_i}\Big) \Rightarrow \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\xi_{k-1})\right)\right] \end{aligned}$$

avec, comme contrainte d'équité pour tous les chemins $\Phi \wedge \rho'$. □

3.3.2 Les bornes supérieures

Nous allons maintenant montrer que les problèmes de model checking que nous considérons sont tous dans Δ_2^P. Ce résultat est déjà connu pour les logiques CTL$^+$ et BT* [CES86]. Nous allons étendre ces résultats ici à la logique ECTL$^+$.

Théorème 3.30 ([LMS01])
Le model checking de la logique ECTL$^+$ *est dans* Δ_2^P.

Démonstration :
Soit φ une formule de ECTL$^+$, de la forme $\mathbf{E}\varphi_P$ où φ_P ne contient pas de quantificateur de chemin. Nous appellerons *sous-formules principales* de φ_P les sous-formules de la forme $\psi_1\,\mathbf{U}\,\psi_2$, $\overset{\infty}{\mathbf{F}}\,\psi_1$ ou $\mathbf{X}\,\psi_1$, c'est-à-dire les formules ayant une modalité à leur racine.

Soit $\pi = q_0 \cdots q_{m-1}(q_m \cdots q_{m+p-1})^\omega$ un chemin ultimement périodique. Nous rappelons (*cf.* chapitre 1) qu'un tel chemin correspond à une structure de Kripke particulière, que nous avons appelée boucle. Une sous-boucle acceptable est une boucle correspondant à un sous-chemin ultimement périodique du chemin correspondant. La sous-boucle est dite acceptable si sa partie initiale (respectivement périodique) est incluse dans la partie initiale (respectivement périodique) de la boucle de départ. Un sous-chemin est acceptable si la sous-boucle qui lui correspond est acceptable. Toutes ces définitions sont données plus précisément à la page 43.

Nous définissons l'ensemble $w(\pi, \varphi_P) \subseteq \{0, 1, \ldots, m+p-1\}$ des « positions témoins » de la façon suivante : pour chaque sous-formule principale de φ_P, $w(\pi, \varphi_P)$ contient un (ou parfois zéro) position, suivant les conditions suivantes :

- pour $\mathbf{X}\,\psi_1$, la position témoin est 1 ;
- pour $\mathbf{F}^{-1}\,\psi_1$, alors il y a une position témoin si, et seulement si, $\pi \models \mathbf{F}^{-1}\,\psi_1$. Cette position témoin est alors le plus petit entier supérieur ou égal à m tel que $q_i \models \psi_1$;
- pour $\psi_1\,\mathbf{U}\,\psi_2$, trois cas sont possibles :
  - si $\pi \models \psi_1\,\mathbf{U}\,\psi_2$, alors la position témoin est la plus petite position i telle que $q_i \models \psi_2$;
  - sinon, s'il existe un état qui vérifie ψ_2, alors la position témoin est la plus petite position satisfaisant $\neg\psi_1 \vee \neg\psi_2$;
  - sinon, il n'y a pas de position témoin.

Alors si $\pi' = q_0 \cdot q_{i_1} \cdot q_{i_2} \cdots$ est un sous-chemin acceptable de π, ayant le même état initial, on a l'équivalence suivante :

$$\pi \models \varphi_P \quad \Leftrightarrow \quad \pi' \models \varphi_P.$$

Il est clair en effet que π et π' vérifient les mêmes sous-formules principales. On en déduit qu'ils vérifient donc exactement les mêmes sous-formules de φ_P. Donc φ_P a la même valeur le long de ces deux chemins.

On en déduit le lemme suivant :

Lemme 3.31
Soit K une structure de Kripke ayant n états, et $\mathbf{E}\varphi_P$ une formule de ECTL$^+$ *telle que φ_P ne contienne pas de quantificateur de chemin. Si $K \models \mathbf{E}\varphi_P$, alors il existe un chemin ultimement périodique $\pi = q_0 \cdots q_{m-1}(q_m \cdots q_{m+p-1})^\omega \in \Pi(S)$ qui vérifie φ_P et tel que $m + p$ soit dans $O(n \times |\varphi_P|)$.*

Démonstration :

La preuve est simple. Nous avons vu au chapitre 1 qu'une formule telle que φ_P admet un témoin ultimement périodique. On peut en extraire un sous-chemin ultimement périodique acceptable π', en ne gardant que les positions de $w(\pi, \varphi_P)$ ainsi que le plus petit nombre possible d'état intermédiaire permettant d'en faire un chemin de K. De cette façon, φ_P s'évalue à la même valeur le long de π et de π', et il y aura au plus $n - 1$ état entre deux états de $w(\pi, \varphi_P)$. Puisqu'il y a $O(|\varphi_P|)$ éléments dans $w(\pi, \varphi_P)$, on obtient le résultat voulu. □

On en déduit immédiatement qu'il existe un algorithme NP pour le model checking d'une formule $\mathbf{E}\varphi_P \in$ ECTL$^+$, si φ_P ne contient pas de quantificateur de chemin : il suffit de « deviner » un chemin ultimement périodique de taille polynomiale, puis de vérifier qu'il satisfait la formule φ_P. Cette vérification peut se faire en temps polynomial dans la taille de la formule et du chemin ultimement périodique, en appliquant par exemple le théorème 1.49 et l'algorithme de model checking de PLTL détaillé à la page 49.

Afin d'obtenir un algorithme pour vérifier n'importe quelle formule de ECTL$^+$, nous appliquons la méthode de la preuve de [CES86, théorème 6.1]. Soient φ une formule de

ECTL$^+$, et K une structure de Kripke dans laquelle on souhaite vérifier φ. Soit $\psi = \mathbf{E}\varphi_P$ une sous-formule de φ telle que φ_P ne contienne aucun quantificateur de chemin. Soit P_ψ une nouvelle proposition atomique, et φ' la formule obtenue en remplaçant, dans φ, toutes les occurences de ψ par la proposition P_ψ. Soit K' la structure obtenue en étiquetant les états vérifiant ψ par la proposition P_ψ. La construction de K' peut se faire par un algorithme NP, d'après ce qui précède. Alors clairement, $K \models \varphi$ si, et seulement si, $K' \models \varphi'$. En appliquant cette transformation récursivement, on élimine tous les quantificateurs de chemins en au plus $|\varphi|$ étapes. Cet algorithme est donc dans P^{NP}. □

Nous déduisons de cette étude le résultat suivant :

Théorème 3.32
Le problème du model checking est Δ_2^P-complet pour les logiques BTL$^+$, UB$^+$, CTL$^+$, BT*, FCTL, GFCTL *et* ECTL$^+$.

Conclusion

Cette étude a fourni, à notre connaissance, les premiers problèmes issus du domaine de la vérification qui soient complets pour la classe Δ_2^P. Elle clôt également des questions de complexité de problèmes de model checking, en particulier pour la logique FCTL, qui étaient ouvertes depuis le milieu des années 1980.

Des travaux récents ont complétés l'étude de la complexité de différentes logiques du temps arborescent [Sch03, JL03].

Pour en savoir plus...

La logique arborescente a été beaucoup étudiée au cours des années 1980, du fait que la vérification de propriétés de CTL pouvait être traitée en temps polynomial, alors que la vérification de propriétés de la logique du temps linéaire nécessite un temps exponentiel (et est PSPACE-complète). Ces deux types de logiques temporelles ont été comparés dans plusieurs travaux, notamment [Var98a, Var98c, Var01], et selon plusieurs points de vue (complexité, expressivité, ...) La logique CTL* a été définie dans [EH86] afin d'« unifier » les cadres linéaire et arborescent, puisqu'elle permet d'exprimer des propriétés selon les deux points de vue. Cependant, cette extension perd le principal intérêt de CTL, qui est la faible complexité de son problème de model checking.

La définition de la syntaxe de CTL que nous avons utilisée fait la distinction entre « formules d'état » et « formules de chemin ». Il est possible de définir la logique CTL en tant que logique *modale*, *i.e.* à partir d'un nombre (fini) de modalités :

$$\mathsf{CTL} \ni \varphi, \varphi \ ::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{EX}\,\varphi \mid \mathbf{AX}\,\varphi \mid \mathbf{E}(\varphi\,\mathbf{U}\,\psi) \mid \mathbf{A}(\varphi\,\mathbf{U}\,\psi) \mid p \mid q \mid \ \ldots$$

Les modalités sont donc **E U**, **A U**, **E X** et **A X** (les deux dernières sont « duales » l'une de l'autre, dès lors qu'on considère des chemins infinis). La logique ECTL peut également se définir de cette façon, puisque ce n'est qu'une extension de CTL obtenue en ajoutant les modalités $\mathbf{E}\overset{\infty}{\mathbf{F}}$ et $\mathbf{A}\overset{\infty}{\mathbf{F}}$. Par contre, [RS00] montre que c'est impossible pour ECTL^+, *i.e.* que ECTL^+ ne peut pas être définie à partir d'un nombre fini de modalités.

Comme nous le signalions à la fin du premier chapitre, l'ajout de modalités du passé dans la logique arborescente a été étudiée dans [KP95, LS95, LS00]. Une des difficultés qui se posent est le choix de la sémantique du temps à donner à ces modalités : doivent-elles faire référence au passé de l'exécution considérée (on parle alors de passé linéaire), ou à tous les passés permettant d'arriver dans l'état courant (le passé est alors arborescent) ? L'opérateur **N** que nous avons étudié au chapitre 2 a également été étudié dans le cadre de la logique du temps arborescent [LS95, LS00].

CHAPITRE 4

Logiques temporelles quantitatives

Introduction

Toutes les logiques temporelles que nous avons étudiées jusqu'à maintenant permettent d'exprimer des contraintes sur les successions des évènements, mais elles ne permettent pas de « mesurer le temps » qui s'écoule entre deux évènements. En pratique cependant, il est souvent utile de pouvoir « contrôler » le temps de réaction des systèmes étudiés.

Ce problème a été évoqué au début des années 1980, dans [BH81, KdR83], mais n'a été réellement traité qu'à partir de la fin des années 1980, en particulier dans [AH89, AH90, EMSS92]. De très nombreuses approches ont été proposées, que ce soit du point de vue de la logique temporelle utilisée ou du point de vue du type de modèle dans lequel est codé le système. Le modèle le plus étudié a été sans conteste le modèle des automates temporisés de [AD94]. Ces modèles sont très expressifs, mais en contrepartie, leur vérification est PSPACE-dure.

Il existe des modèles plus simples dans lesquels la vérification de logique temporelle « temporisée » peut être traitée en temps polynomial. Par exemple, [EMSS92] étudie la logique TCTL (une logique temporelle du temps arborescent avec contraintes quantitatives sur les modalités) sur des structures de Kripke simples, les contraintes quantitatives portant sur le nombre de transitions suivies le long du chemin.

Dans ce chapitre, nous étudions principalement une extension de ce modèle, que nous nommons « *structure de Kripke avec durées* ». Dans ces modèles, chaque transition est étiquetée par un intervalle, indiquant le temps qui s'écoule au moment de suivre cette transition. Cela constitue une extension naturelle des structures de Kripke. Nous verrons qu'il est toujours possible d'avoir des procédures de model checking en temps polynomial pour de tels modèles, à condition d'utiliser une logique arborescente sans contraintes d'égalité. Dans le cas où les contraintes d'égalité sont autorisées, le problème du model checking devient alors complet pour la classe Δ_2^P. Nous verrons que ce motif est général, *i.e.* que la vérification de logiques avec contraintes d'égalité est généralement plus complexe que lorsque les contraintes d'égalité sont interdites.

Ce chapitre correspond à [LMS02a].

4.1 Extensions temporisées des logiques et des modèles

Nous allons exposer, dans cette partie, plusieurs façons d'ajouter une notion « temporisée », dans les logiques temporelles d'une part, et dans les modèles d'autre part.

4.1.1 Logiques temporelles temporisées

Afin d'ajouter une notion « quantitative » (qui sert souvent à mesurer le temps qui s'écoule entre les transitions) aux différentes logiques temporelles, plusieurs méthodes ont été proposées. Nous détaillons deux méthodes ci-dessous : les logiques avec « *freeze quantifier* », et les logiques avec contraintes. [AH91] donne une présentation générale de plusieurs logiques et modèles permettant de traiter les systèmes « temps réel ».

Plutôt que « temps réel », nous devrions parler de « temps discret », puisque le «temps » que nous considérons dans toute la suite de ce chapitre prend ses valeurs dans $\mathbb{N}$, et non dans $\mathbb{R}$.

4.1.1.1 Logiques avec « freeze quantifier »

Afin de mesurer le « temps » écoulé entre deux évènements, [AH89, AH94] proposent de comparer des variables numériques dans les formules de la logique (on pourra donc écrire « $x \leqslant y$ » dans les formules), et d'utiliser un *freeze quantifier*, c'est à dire un opérateur permettant d'initialiser ces variables. La différence entre deux variables indiquera alors le délai entre deux évènements.

Définition 4.1 ([AH94])
Soient $V = \{x, y, z, \ldots\}$ un ensemble de variables, et Prop *un ensemble de propositions atomiques. La logique* FQLTL *est définie de la façon suivante :*

$$\begin{aligned} \pi &::= x + c \mid c \\ \mathsf{FQLTL} \ni \varphi, \psi &::= \pi_1 \leqslant \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\psi \mid x \cdot \varphi \mid p \mid q \mid \ \ldots \end{aligned}$$

pour $x \in V$, $p, q, \ldots \in$ Prop *et $c, d \in \mathbb{N} \times \mathbb{N}^*$.*

Bien entendu, les abréviations $\wedge$, $=$, $<$, ... sont définies de manière évidente à partir des opérateurs de FQLTL, de même que les modalités $\mathbf{F}$, $\mathbf{G}$, ... L'opérateur *freeze*, que l'on note $x \cdot \varphi$, signifie, intuitivement, que l'on « initialise » la variable x avant d'évaluer la formule φ.

Ces formules ne sont plus évaluées le long de chemins simples, mais le long de chemins temporisés :

Définition 4.2
Une suite de dates *est une suite infinie $\tau_0, \tau_1, \ldots$ d'éléments de $\mathbb{N}$ croissante et divergente. Un* chemin temporisé *est un couple $\rho = (\pi, \tau)$ où π est un chemin infini et τ une suite de dates. Le i-ème suffixe du chemin ρ est le chemin temporisé $\rho^i = (\pi^i, \tau^i - \tau_i)$.*

Ainsi, par exemple, $\rho \models x \cdot \mathbf{F}\, y \cdot (p \wedge (y - x \leqslant +10))$ signifie que, moins de dix unités de temps après l'état courant, p sera vraie le long de ρ. En effet, x est initialisée à l'instant de départ, et y l'est au moment où p est vérifiée. Si $y - x$ est inférieur à 10, alors l'évènement p considéré a bien eu lieu dans un délai de 10 unités de temps.

Cette extension de la logique temporelle est très expressive, mais aussi, en général, très complexe à vérifier. [AH94] prouve ainsi que le problème de satisfaisabilité est EXPSPACE-complet , tout comme le model checking d'un « graphe d'états temporisé », c'est-à-dire une structure de Kripke dont les états sont étiquetés, en plus des propositions atomiques, par un entier indiquant le « temps qui s'écoule » dans cet état.

4.1.1.2 Logiques avec contraintes temporelles

Une autre méthode couramment utilisée pour ajouter cette notion de « temporisation » dans le domaine de la vérification a été proposée dans [EMSS92]. Cette extension, qui consiste à ajouter une contrainte temporelle à la modalité **U**, permettant de spécifier le délai séparant l'état courant de celui où la partie droite du *until* sera satisfaite. Ainsi, pour exprimer qu'une proposition atomique p doit être satisfaite en moins de dix unités de temps, nous écrirons $\mathbf{F}_{\leqslant 10}\, p$. [EMSS92] applique cette extension à CTL uniquement, et [ET97] l'étend à LTL.

Définition 4.3
La logique TLTL *est définie par la syntaxe suivante :*

$$\mathsf{TLTL} \ni \varphi, \psi \ ::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}_{\sim k}\,\psi \mid p \mid q \mid \ \ldots$$

où $\sim \in \{\leqslant, <, =, >, \geqslant\}$, *et* $k \in \mathbb{N}$.
La logique TCTL *est définie par la syntaxe suivante :*

$$\begin{aligned} \mathsf{TCTL} \ni \varphi, \psi \ &::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid p \mid q \mid \ \ldots \\ \mathsf{TCTL}_c \ni \varphi_c, \psi_c \ &::= \ \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}_{\sim k}\,\psi \end{aligned}$$

où $\sim$, k *et* $p, q, \ldots$ *satisfont les mêmes conditions que précédemment.*
De la même façon que nous avions défini des extensions de CTL *au chapitre 3, nous allons définir les logiques* TCTL^+ *et* $\mathsf{TCTL}^\star$, *extensions de* TCTL, *que nous étudierons par la suite.*
Les formules de TCTL^+ *sont les formules de* TCTL^+_s *suivant la syntaxe ci-dessous :*

$$\begin{aligned} \mathsf{TCTL}^+ \ni \varphi, \psi \ &::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{E}\varphi_c \mid \mathbf{A}\varphi_c \mid p \mid q \mid \ \ldots \\ \mathsf{TCTL}^+_c \ni \varphi_c, \psi_c \ &::= \ \neg\varphi_c \mid \varphi_c \vee \psi_c \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}_{\sim k}\,\psi \end{aligned}$$

La logique $\mathsf{TCTL}^\star$ *est définie par la syntaxe suivante :*

$$\mathsf{TCTL}^\star \ni \varphi, \psi \ ::= \ \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}_{\sim k}\,\psi \mid \mathbf{E}\varphi \mid \mathbf{A}\varphi \mid p \mid q \mid \ \ldots$$

Pour chacune de ces logiques, nous définissons le fragment « sans égalité », dans lequel $\sim \in \{\leqslant, <, >, \geqslant\}$. *Dans ce cas, une contrainte sur une durée ne peut pas être « exacte ». Nous noterons* $\mathsf{TLTL}_{\leqslant,\geqslant}$, $\mathsf{TCTL}_{\leqslant,\geqslant}$, $\mathsf{TCTL}^+_{\leqslant,\geqslant}$ *et* $\mathsf{TCTL}^\star_{\leqslant,\geqslant}$ *les fragments ainsi obtenus.*

La sémantique de ces logiques découle de celle des logiques non temporisées équivalentes, mais appliquée à des chemins temporisés. Seule la sémantique de la modalité $\mathbf{U}_{\sim k}$ doit être définie :

$$\rho \models \varphi\,\mathbf{U}_{\sim k}\,\psi \quad\Leftrightarrow\quad \exists j.\ (\tau_j \sim k \wedge \rho^j \models \psi \wedge \forall 0 \leqslant i < j.\ \rho^i \models \varphi).$$

Bien entendu, l'équivalence entre formules est définie de la même manière que pour les logiques non temporisées, mais cette fois, dans les structures de Kripke avec durées ou le long de chemins temporisés.

[LST00] énonce plusieurs résultats d'expressivité sur ces logiques, dans le cas où le temps augmente « par petits pas », *i.e.* lorsque le temps est incrémenté dans les états étiquetés par une proposition atomique spéciale appelée *tick*. Dans ce cas, il existe une traduction de TCTL dans CTL, et il est possible d'établir plusieurs résultats de concision sur cette traduction et sur la traduction entre TCTL et $\mathsf{TCTL}_{\leqslant,\geqslant}$ [Sch01].

Dans le cas général, cependant, ces résultats ne sont pas valables, puisque l'écoulement du temps n'est plus mesurable au moyen des propositions atomiques. On notera par ailleurs que $\mathsf{TCTL}^{+}_{\leqslant,\geqslant}$ (et donc $\mathsf{TCTL}_{\leqslant,\geqslant}$) et $\mathsf{TLTL}_{\leqslant,\geqslant}$ ne permettent pas d'exprimer une contrainte « exacte » : par exemple, la formule $\mathbf{E}(\mathbf{F}_{\leqslant 10}\,p \wedge \mathbf{F}_{\geqslant 10}\,p)$ exprime simplement que p est vrai une première fois en moins de dix unités de temps, et une (autre) fois après plus de dix unités de temps. De la même façon, il n'est pas possible d'exprimer des « intervalles de contraintes », comme par exemple $\mathbf{E}(\varphi\,\mathbf{U}_{[3,5]}\,\psi)$ (dont la sémantique est évidente) sans les contraintes d'égalité.

Du point de vue de la vérification, toutes ces logiques ont déjà été étudiées dans de nombreux travaux :

- [EMSS92] étudie la satisfaisabilité et le model checking de $\mathsf{TCTL}_{\leqslant,\geqslant}$ et TCTL sur des structures de Kripke simples (les contraintes portant alors sur le nombre de transitions franchies). Concernant TCTL, la complexité est la même que pour CTL : la satisfaisabilité est EXPTIME-complet, et le model checking est P-complet. Lorsque l'on autorise les contraintes d'égalité, la complexité du model checking reste inchangée, mais la satisfaisabilité devient 2EXPTIME-complet ;
- [LST00] a prouvé que, pour des structures de Kripke dans lesquelles une transition dure zéro ou une unité de temps, le model checking de TCTL (et $\mathsf{TCTL}_{\leqslant,\geqslant}$) est P-complet ;
- [Hen98] fournit un algorithme de model checking de $\mathsf{TLTL}_{\leqslant,\geqslant}$ et TLTL sur des automates temporisés classiques, dans le cas du temps discret. Cet algorithme utilise des automates de Büchi, il s'exécute en espace polynomial pour la logique $\mathsf{TLTL}_{\leqslant,\geqslant}$ et en espace exponentiel pour la logique TLTL ;
- ces logiques ont également été étudiées sur le modèle des automates temporisés de [AD94], dans lequel le temps est « dense ». La satisfaisabilité de la logique TCTL devient alors indécidable, et le model checking est PSPACE-complet [ACD93].

4.1.2 Structures de Kripke avec durées

Les derniers résultats de la section précédente montrent que, dans le cas des logiques quantitatives, la complexité dépend beaucoup du modèle utilisé. Il y a une grande différence entre le model checking « quantitatif » sur les structures de Kripke et le model checking d'automates temporisés. Nous allons définir dans cette section une autre classe de modèles intermédiaires, afin d'avoir un modèle plus expressif que les structures de Kripke simples, mais dont la vérification serait moins complexe que pour les automates temporisés.

4.1.2.1 Les structures de Kripke avec durées

Dans toute la suite, nous notons $\mathcal{I}_{\mathbb{N}}$ l'ensemble des intervalles de $\mathbb{N}$, qui peuvent être soit finis (*i.e.* de la forme $[\![a, b]\!]$), soit infinis (*i.e.* de la forme $[\![a, \infty[\![$).

Définition 4.4
Une structure de Kripke avec durées *est un 4-uplet $K = (S, S_0, l, R)$ où*
- $S = \{s_1, s_2, \ldots, s_k\}$ *est un ensemble fini d'états ;*
- $S_0 \subseteq S$ *est l'ensemble des états initiaux ;*
- $l \in (\{\top, \bot\}^{\text{Prop}})^S$ *associe, à chaque état de S, une fonction indiquant quelles propositions atomiques sont vérifiées dans cet état ;*
- $R \in S \times \mathcal{I}_{\mathbb{N}} \times S$ *est la relation de transition avec durées. Cette relation est totale (*i.e. *pour tout état s, il existe un état s' et un intervalle I tels que $(s, I, s') \in R$). Nous supposerons également que cette relation est finie. Par ailleurs, les bornes des l'intervalles sont codées en binaire.*

Le fait de supposer que la relation de transition est finie est une restriction importante. On ne pourra pas coder, par exemple, le fait qu'une transition a une durée paire.

Par ailleurs, le fait de coder *en binaire* des *intervalles* constitue une double extension des structures de Kripke classiques. La différence de concision entre les structures de Kripke avec durées et les structures de Kripke classiques est exponentielle, puisque une transition de durée 2^n peut être codée en espace linéaire en n dans une structure de Kripke avec durées, mais représente 2^n transitions (unitaires) dans une structure de Kripke classique.

Un exemple de structure de Kripke avec durées est représenté à la figure 4.1. Elle représente l'évolution des publications d'un chercheur (les durées représentent le nombre de jours écoulés).

La sémantique que nous allons prendre pour ces structures est très différente de la sémantique habituellement utilisée pour les automates temporisés. Dans une structure de Kripke avec durées, une transition (s, I, s') signifie qu'il existe un entier $d \in I$ tel qu'on peut passer de s à s' en d unités de temps.

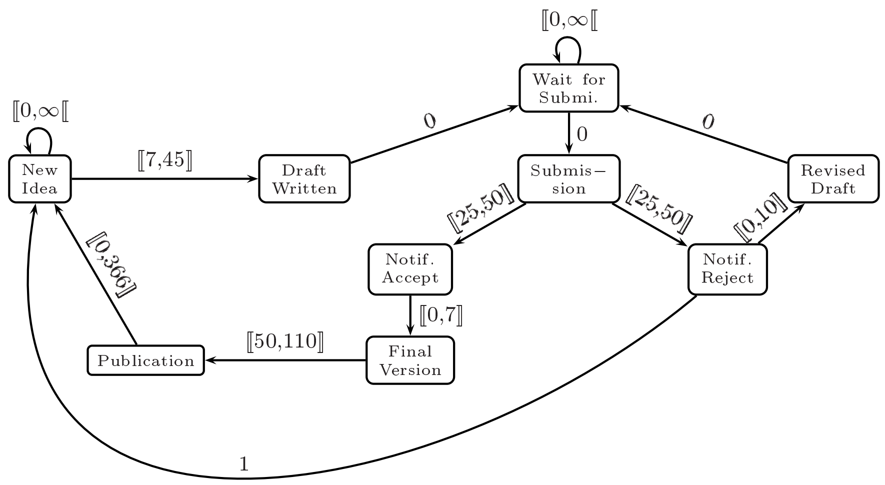
De cette façon, une suite de transition $\sigma = s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} s_2 \cdots$ induit un chemin temporisé $\rho = (\pi, \tau)$ défini par

$$\forall i \in \mathbb{N}.\ \pi_i = s_i \text{ et } \tau_i = \sum_{j=0}^{i-1} d_j.$$

Un tel chemin peut être fini ou infini, et un chemin fini peut toujours être complété en un chemin infini. Pour un chemin fini $\rho = s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} s_2 \cdots \xrightarrow{d_{n-1}} s_n$, nous définissons sa longueur $|\rho| = n$ (c'est le nombre de transitions), et sa durée

$$\text{Time}(\rho) = \tau_{|\rho|} = \sum_{i=0}^{|\rho|-1} d_i.$$

Figure 4.1 Une structure de Kripke avec durées



4.1.2.2 Restrictions sur ces modèles

Les structures que nous venons de définir sont une généralisation de structures « temporisées » déjà étudiées dans d'autres travaux par le passé. En voici trois exemples, que nous étudierons par la suite et auxquels nous comparerons aux structures de Kripke avec durées.

Les structures de Kripke avec durées strictes. Dans ce modèle, on impose aux intervalles qui « étiquettent » les transitions de la structure de Kripke avec durées d'être des singletons. Ce modèle est celui utilisé dans [AH94] pour le model checking de la logique FQLTL.

Les structures de Kripke avec durées unitaires. Ce sont les structures de Kripke avec durées dont les durées valent zéro ou un. Dans ce cas, il est possible de représenter l'écoulement du temps par une proposition atomique *tick*, comme dans [LST00]. On peut alors traduire les formules de logique temporelle quantitative en formule de la logique temporelle classique.

Les structures de Kripke simples. Dans ces structures, on considère qu'une unité de temps s'écoule à chaque transition. C'est donc un cas particulier du modèle précédent dans lequel tous les états seraient étiquetés par la proposition *tick*. C'est le modèle étudié dans [EMSS92].

4.2 Complexité de la vérification des logiques quantitatives

La plus grosse partie de notre étude portera sur le model checking de la logique TCTL : nous allons prouver que ce problème est complet pour la classe Δ_2^P, ce qui nous fournit un deuxième exemple de problème issu de la vérification qui soit complet pour cette classe. Nous allons voir également que, lorsque l'on utilise pas l'égalité dans les contraintes temporelles, le problème peut se résoudre en temps polynomial. Nous étudirons ensuite la complexité du même problème pour d'autres logiques, afin d'étayer l'affirmation selon laquelle les contraintes d'égalité augmentent sensiblement la complexité des problèmes de model checking.

4.2.1 Pour la logique TCTL

Nous commençons par le cas le plus simple, dans lequel les contraintes d'égalité ne sont pas autorisées. Nous traitons ensuite le cas général.

4.2.1.1 Sans contrainte d'égalité

Il est connu que le model checking de $\mathsf{TCTL}_{\leqslant,\geqslant}$ est P-complet pour les structures de Kripke simples [EMSS92] et pour les structures de Kripke avec durées unitaires [LST00]. Le problème reste P-complet pour les structures de Kripke avec durées, en vertu du résultat suivant :

Théorème 4.5 ([LMS02a, théorème 4.2])
Le model checking d'une formule $\varphi \in \mathsf{TCTL}_{\leqslant,\geqslant}$ sur une structure de Kripke avec durées K peut se résoudre en temps $O(|K|^2 \cdot |\varphi|)$.

Démonstration :
Nous allons étendre l'algorithme d'étiquetage de CTL, en lui ajoutant des procédures d'étiquetage de la structure de Kripke avec durées pour les formules $\mathbf{E}(P_1\,\mathbf{U}_{\sim c}\,P_2)$ et $\mathbf{A}(P_1\,\mathbf{U}_{\sim c}\,P_2)$:

- pour $\mathbf{E}(P_1\,\mathbf{U}_{\leqslant c}\,P_2)$: en appliquant la procédure d'étiquetage de CTL pour la formule $\mathbf{E}(P_1\,\mathbf{U}\,P_2)$, nous nous restreignons tout d'abord au sous-graphe des états satisfaisant cette formule. Nous calculons ensuite, pour chaque état q restant, la plus petite durée c_q telle que $q \models \mathbf{E}(P_1\,\mathbf{U}_{\leqslant c_q}\,P_2)$. Ce calcul peut être fait en temps $O(|K|^2)$ par un algorithme de recherche de plus court chemin [CLR99, chapitre 25], en prenant comme durée sur chaque arête la borne inférieure de l'intervalle qui l'étiquette. Nous étiquetons ensuite les états satisfaisant la condition $c_q \leqslant c$;
- pour $\mathbf{E}(P_1\,\mathbf{U}_{\geqslant c}\,P_2)$: de la même manière que précédemment, nous remplaçont les intervalles qui étiquettent les arêtes par leur borne supérieure. en effet, cette formule peut être satisfaite soit par un chemin « simple », *i.e.* sans boucle, qui aura alors une durée supérieure à c, soit par un chemin avec une boucle de durée non nulle, auquel cas il sera possible de faire plusieurs « tours » dans la boucle afin de s'assurer que le chemin soit assez long.
  Nous allons donc commencer par définir une nouvelle proposition atomique, notée $P_{\mathrm{SCC}^+(P_1)}$, qui sera vraie dans les états appartenant à une composante fortement

connexe dont tous les états vérifieront P_1, et dont au moins une transition aura une durée non nulle. En d'autres termes, $q \models P_{\mathrm{SCC}^+(P_1)}$ si, et seulement si, il existe un chemin partant de q, revenant dans q, dont tous les états sont étiquetés par P_1, et de durée totale non nulle. L'étiquetage de la structure par cette proposition atomique peut se traiter en temps $O(|K|)$ si les états sont déjà étiquetés par P_1.
Il nous reste maintenant à traiter le problème initial, sur le graphe étiqueté par la proposition $P_{\mathrm{SCC}^+(P_1)}$. On ne considère que le sous-graphe dont tous les états vérifient $\mathbf{E}(P_1 \, \mathbf{U} \, P_2)$. Soit q un état dans ce sous-graphe. Dans le cas où il y a des états étiquetés par $P_{\mathrm{SCC}^+(P_1)}$, alors on commence par faire le model checking de la formule $\mathbf{E}(P_1 \, \mathbf{U} \, (P_{\mathrm{SCC}^+(P_1)} \wedge \mathbf{E}(P_1 \, \mathbf{U} \, P_2)))$. Cela se traite en temps polynomial, puisque c'est une formule de CTL. Si cette formule est satisfaite dans q, alors $q \models \mathbf{E}(P_1 \, \mathbf{U}_{\geqslant c} \, P_2)$. Dans le cas contraire, il n'existe pas de chemin satisfaisant $P_1 \, \mathbf{U} \, P_2$ et passant par un état étiqueté par $P_{\mathrm{SCC}^+(P_1)}$. Nous pouvons supprimer ces états du sous-graphe, et nous ramener au cas où aucun état n'est étiqueté par $P_{\mathrm{SCC}^+(P_1)}$. Dans ce cas, la recherche du plus long chemin est équivalente à la recherche du plus court chemin dans le graphe dont les poids sont les opposés des poids initiaux. L'algorithme de recherche du plus court chemin de Bellman-Ford s'applique dans ce cas [CLR99, AHU74], et nous fournit en temps polynomial le plus court chemin dans le graphe « négatif », *i.e.* le plus long chemin dans le graphe initial.

- pour $\mathbf{A}(P_1 \, \mathbf{U}_{\leqslant c} \, P_2)$: nous nous ramenons aux cas précédents grâce aux deux équivalences suivantes :

$$\mathbf{A}(P_1 \, \mathbf{U}_{\leqslant c} \, P_2) \equiv \mathbf{AF}_{\leqslant c} \, P_2 \wedge \neg \mathbf{E}((\neg P_2) \, \mathbf{U} \, (\neg P_1 \wedge \neg P_2)) \tag{4.1}$$

$$\mathbf{AF}_{\leqslant c} \, P_2 \equiv \neg \mathbf{E}((\neg P_2) \, \mathbf{U}_{>c} \, \top) \wedge \neg \mathbf{E}((\neg P_2) \, \mathbf{U} \, P_{\mathrm{SCC}^0(\neg P_2)}) \tag{4.2}$$

  où $P_{\mathrm{SCC}^0(\neg P_2)}$ étiquette les composantes fortement connexes de durée totale nulle et dont tous les états vérifient $\neg P_2$.
  L'équation (4.1) exprime qu'il est équivalent de dire « *tous les chemins satisfont* $P_1 \, \mathbf{U}_{\leqslant c} \, P_2$ » et « *tous les chemins rencontrent un état étiqueté par* P_2 *en moins de* c *unités de temps, et de plus, il n'existe pas de chemin vérifiant* $\neg P_1$ *avant d'avoir vérifié* P_2 ». L'équation (4.2) exprime l'équivalence entre « *tous les chemins passent par un état vérifiant* P_2 *en moins de* c *unités de temps* » et « *il n'existe pas de chemin vérifiant* $\neg P_2$ *durant strictement plus de* c *unités de temps, et il n'existe pas non plus de chemin de durée finie vérifiant toujours* $\neg P_2$ ».

- pour $\mathbf{A}(P_1 \, \mathbf{U}_{\geqslant c} \, P_2)$: nous nous ramenons également aux cas précédents, grâce aux équivalences suivantes :

$$\mathbf{A}(P_1 \, \mathbf{U}_{\geqslant c} \, P_2) \equiv \mathbf{AG}_{<c} \left(P_1 \wedge \mathbf{A}(P_1 \, \mathbf{U}_{>0} \, P_2)\right) \tag{4.3}$$

$$\mathbf{A}(P_1 \, \mathbf{U}_{>0} \, P_2) \equiv \mathbf{AG}_{\leqslant 0} \, [P_1 \wedge \mathbf{AX} \, \mathbf{A}(P_1 \, \mathbf{U} \, P_2)] \wedge \mathbf{AF}_{\geqslant 1} \, \top \tag{4.4}$$

$$\mathbf{AG}_{<c} \, \varphi \equiv \neg \mathbf{EF}_{<c} \, \neg \varphi \tag{4.5}$$

  Nous avons donc ramené la vérificaiton d'une formule $\mathbf{A}(P_1 \, \mathbf{U}_{\geqslant c} \, P_2)$ à celle d'une formule $\mathbf{AF}_{\geqslant 1} \, \top$. Cette formule est équivalente à $\neg \mathbf{EG}_{>0} \, \bot$; nous devons donc écrire un algorithme déterministe en temps polynomial vérifiant une formule de la forme $\mathbf{EG}_{>0} \, \bot$, qui signifie qu'il existe un chemin (infini) de durée nulle issu de l'état courant. Cette propriété s'exprime par $\mathbf{EF}_{\leqslant 0} \, P_{\mathrm{SCC}^0(\top)}$, et se vérifie donc en temps polynomial.

Ces procédures d'étiquetage s'exécutent toutes en temps $O(|K|^2 \cdot \log c)$. En les ajoutant aux procédures d'étiquetage de CTL, nous obtenons un algorithme d'étiquetage polynomial pour la logique $\mathsf{TCTL}_{\leqslant,\geqslant}$. □

Bien entendu, le côté P-dur de ce problème provient de la P-difficulté du model checking de CTL. Donc :

Théorème 4.6
Le model checking de $\mathsf{TCTL}_{\leqslant,\geqslant}$ *sur les structures de Kripke avec durées est* P-*complet.*

4.2.1.2 Avec contraintes d'égalité

Pour la logique TCTL (avec contraintes d'égalité), plusieurs résultats sont déjà connus sur les modèles « simples » : [EMSS92] montre que, sur des structures de Kripke simples, le model checking de TCTL peut se faire en temps polynomial (et il est donc P-complet, puisque c'est une extension du model checking de CTL). [LST00] prouve que le model checking de TCTL sur des structures de Kripke avec durées unitaires est P-complet également.

Dans le cas du model checking sur des structures de Kripke avec durées, la complexité augmente, comme le remarque [LST00] :

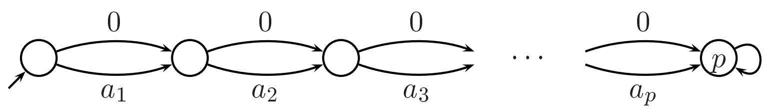
Proposition 4.7 ([LST00, proposition 5.1])
Le model checking de $\mathbf{EF}_{=k}\,p$ *sur une structure de Kripke avec durées strictes est* NP-*dur.*

Démonstration :
La preuve se fait par une réduction au problème SUBSET-SUM [GJ79, section A3.2, problème SP13]. Une instance de ce problème est composée d'un ensemble d'entiers naturels $A = \{a_1, a_2, \ldots, a_p\}$, et d'un entier « cible » B. Le problème consiste à décider de l'existence d'un sous-ensemble $A' \subseteq A$ tel que

$$B = \sum_{a \in A'} a.$$

Ce problème peut facilement être réduit à un problème de model checking d'une propriété de la forme $\mathbf{EF}_{=k}\,p$ sur une structure de Kripke avec durées strictes : nous considérons la structure de Kripke avec durées de la figure 4.2. Dans cette structure, passer d'un état

Figure 4.2 La structure de Kripke avec durées strictes pour la réduction de SUBSET-SUM



à son successeur peut se faire par deux transitions différentes : une de durée nulle, et une autre de durée a_i. Il est alors évident que notre instance du problème SUBSET-SUM admet une solution si, et seulement si, la formule $\mathbf{EF}_{=B}\, p$ est satisfaite dans l'état initial de cette structure. $\square$

Ce résultat affirme donc que le model checking d'une structure de Kripke avec durées sera strictement[1] plus complexe que dans le cas d'une structure de Kripke avec durées unitaire. Bien entendu, de la même façon que nous avons montré que ce problème est NP-dur, nous pouvons montrer qu'il est co-NP-dur, puisque la logique considérée est close par négation. Cela nous amène à penser que le problème sera Δ_2^P-dur.

Théorème 4.8 ([LMS02a, proposition 4.8])
Le model checking de TCTL *sur les structures de Kripke avec durées strictes est Δ_2^P-difficile.*

Démonstration :
Nous utilisons à nouveau le problème SNSAT (définition 3.22) que nous avons utilisé au chapitre précédent. Nous conservons les même notations et les mêmes définitions. Nous allons réduire ce problème, qui est Δ_2^P-complet, au problème du model checking de TCTL dans une structure de Kripke avec durées strictes.

Soit donc $\mathcal{I}$ une instance de SNSAT :

$$\mathcal{I} = \begin{cases} x_1 := \top \text{ si, et seulement si, } & \exists Z_1.\ F_1(Z_1) \\ x_2 := \top \text{ si, et seulement si, } & \exists Z_2.\ F_2(x_1, Z_2) \\ x_3 := \top \text{ si, et seulement si, } & \exists Z_3.\ F_3(x_1, x_2, Z_3) \\ \qquad\vdots & \\ x_n := \top \text{ si, et seulement si, } & \exists Z_n.\ F_n(x_1, \ldots, x_{n-1}, Z_n). \end{cases}$$

Nous supposons, en toute généralité, que les formules F_i sont sous forme normale conjonctive à trois littéraux, et nous écrivons

$$F_i = \bigwedge_{j=1}^{l} \bigvee_{k=1}^{3} \alpha_{i,j,k}$$

où $\alpha_{i,j,k}$ est une variable $u \in Z_i \cup \{x_1, \ldots, x_{i-1}\}$ ou sa négation. À chaque clause $\bigvee_{k=1}^3 \alpha_{i,j,k}$, nous associons la clause $C_{i,j} = \overline{x_i} \vee \bigvee_{k=1}^3 \alpha_{i,j,k}$. Nous notons $\mathrm{Cl} = \{C_1, C_2, \ldots\}$ l'ensemble des clauses ainsi obtenues, après renumérotation (l'ordre n'importe pas). De la même façon, nous notons $Z = Z_1 \cup \cdots \cup Z_n = \{z_1, z_2, \ldots, z_p\}$.

Soit K un entier naturel. Nous allons associer à chaque variable et à chaque clause un « poids » s, de la façon suivante :

$$s(x_i) \stackrel{\text{def}}{=} K^i \qquad\qquad s(z_i) \stackrel{\text{def}}{=} K^{n+i} \qquad\qquad s(C_i) \stackrel{\text{def}}{=} K^{n+p+i}$$

Nous étendons cette définition à tout multi-ensemble $\mathcal{M}$ de variables et de clauses ($\mathcal{M} \in \mathbb{N}^{\mathrm{Var}\cup\mathrm{Cl}}$), en écrivant

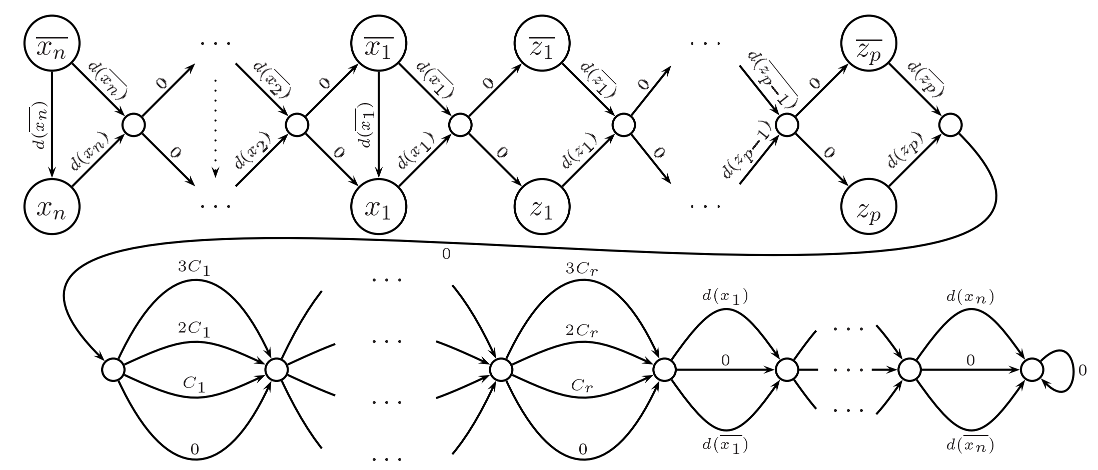
$$s(\mathcal{M}) \stackrel{\text{def}}{=} \sum_{x \in \mathrm{Var}\cup\mathrm{Cl}} s(x) \times \mathcal{M}(x).$$

[1]En supposant que $\mathsf{P} \neq \mathsf{NP}$.

De cette façon, si $\mathcal{M}$ et $\mathcal{M}'$ sont deux multi-ensembles suffisamment petits (*i.e.* si, pour tout x, $\mathcal{M}(x) < K$ et $\mathcal{M}'(x) < K$), l'égalité de ces deux multi-ensembles est équivalente à l'égalité de $s(\mathcal{M})$ et $s(\mathcal{M}')$.

Nous définissons maintenant la structure de Kripke avec durées strictes $K_{\mathcal{I}}$, comme sur la figure 4.3. Dans cette structure, il y a deux sortes de nœuds : les nœuds étiquetés par des

Figure 4.3 La structure de Kripke avec durées $K_{\mathcal{I}}$ associée à l'instance $\mathcal{I}$ de SNSAT



littéraux (ceux du haut de la figure, que nous appellerons « nœuds littéraux » par la suite) et les nœuds « complémentaires » (ceux de la partie inférieure de la figure). À un chemin ne passant pas par les arêtes $\overline{x_i} \xrightarrow{d(\overline{x_i})} x_i$, nous pouvons associer, de manière évidente, une valuation des variables de Var, avec la partie supérieure du chemin. La partie inférieure sert à ajuster la valeur du « compteur », comme nous le verrons plus loin.

Pour chaque littéral α de la forme u ou $\overline{u}$, la valeur de $d(\alpha)$ est définie par la formule suivante :

$$d(\alpha) \stackrel{\text{def}}{=} s(u) + \sum_{\substack{C \in \text{Cl} \\ \alpha \Rightarrow C}} s(C).$$

Ainsi, lorsque le chemin passe dans les différents nœuds littéraux, il « emmagasine » le poids des variables, ainsi que celui des clauses qui sont satisfaites par ces variables (chaque clause est comptée au plus quatre fois, puisque les clauses sont de la forme $\overline{x_i} \vee \bigvee_{k=1}^{3} \alpha_{i,j,k}$). Lorsqu'il passe dans les nœuds complémentaires, le chemin peut alors « compléter » sa durée.

Nous définissons alors la valeur « cible » :

$$K' \stackrel{\text{def}}{=} \sum_{u \in \text{Var}} s(u) + 4 \times \sum_{C \in \text{Cl}} s(C).$$

Si K est assez grand, alors pour toute variable u, un chemin π de durée K' ne peut « contenir » $d(u)$ ou $d(\overline{u})$ qu'une fois et une seule.

En effet, nous pouvons associer à tout chemin π un multi-ensemble $\mathcal{M}_\pi$ contenant le nombre de fois que chaque variable ou clause est « ajoutée » à la durée totale du chemin.

Au vu du chemin, il est clair que chaque variable est présente au plus trois fois dans un multi-ensemble correspondant à un chemin, et chaque clause au plus onze[2] fois. Si $K \geqslant 12$, alors la valeur de $s(\mathcal{M}_\pi)$ détermine de façon unique le multi-ensemble $\mathcal{M}_\pi$ pour un chemin issu de la structure $K_\mathcal{I}$. Ainsi, donc, un chemin de durée exactement K' contient une et une seule occurrence de chaque variable dans son multi-ensemble, et donc passe une et une seule fois par une arête de durée $d(u)$ ou $d(\overline{u})$. On en déduit immédiatement qu'un chemin de durée K' ne passe pas par les arêtes « transversales » de $\overline{x_i}$ vers x_i, et qu'il correspond donc à une valuation des variables de Var. Par ailleurs, les nœuds complémentaires ne peuvent apporter que $3 \times s(C_l)$ pour chaque l, et donc chaque clause de Cl doit être satisfaite au moins une fois lors du passage dans les nœuds littéraux. La valuation associée à un chemin de longueur K' satisfait donc toutes les clauses de Cl.

Nous définissons maintenant, de manière récursive, les formules qui devront être vraies dans la structure $K_\mathcal{I}$:

$$\varphi_0 \stackrel{\text{def}}{=} \top$$
$$\forall k \geqslant 1.\ \varphi_k \stackrel{\text{def}}{=} \mathbf{E}\Big[P_{\overline{x}} \Rightarrow \mathbf{EX}\,(P_x \wedge \neg\varphi_{k-1})\Big]\,\mathbf{U}_{=K'}\,\top$$

où P_x (respectivement $P_{\overline{x}}$) est une proposition atomique étiquetant les n états x_i (respectivement $\overline{x_i}$) de la structure $K_\mathcal{I}$.

Nous allons maintenant établir le lemme suivant, similaire au lemme 3.25 du chapitre précédent, et qui relie la valuation $v_\mathcal{I}$, correspondant à l'instance $\mathcal{I}$ de SNSAT, à la structure $K_\mathcal{I}$ et aux formules φ_k :

Lemme 4.9
Soit k un entier naturel, et $r \in [\![1, n]\!]$. On a alors :
- *si $k \geqslant 2r-1$, alors $v_\mathcal{I}(x_r) = \top$ si, et seulement si, $K_\mathcal{I}, x_r \models \varphi_K$;*
- *si $k \geqslant 2r$, alors $v_\mathcal{I}(x_r) = \bot$ si, et seulement si, $K_\mathcal{I}, \overline{x_r} \models \varphi_K$;*

Démonstration :

La preuve se fait par récurrence sur la valeur de k. Lorsque k est nul, il n'existe pas de valeur de r vérifiant les contraintes du lemme, et le résultat est donc vrai.

Soit k un entier strictement positif. Supposons que le lemme soit vrai pour l'entier $k-1$.

- Nous prouvons tout d'abord le sens direct de chacune des deux équivalences. Pour ce faire, nous considérons une valuation admissible w, et nous lui associons un chemin π_w, dont le premier état est x_r ou $\overline{x_r}$, selon que $w(x_r)$ vaut $\top$ ou $\bot$, qui passe dans l'état u si $w(u) = \top$ et dans l'état $\overline{u}$ si $w(u) = \bot$, et dont la durée totale est K'. Un tel chemin existe, puisque w est admissible, et donc toutes les clauses de Cl sont satisfaites : en effet, soit la clause $\bigvee_{k=1}^{3} \alpha_{i,j,k}$ de F_i est satisfaite, soit x_i s'évalue à $\bot$. Dans les deux cas, $\overline{x_i} \vee \bigvee_{k=1}^{3} \alpha_{i,j,k}$ est satisfaite.
  Ce chemin π est alors un témoin du fait que $x_r \models \varphi_k$ (respectivement que $\overline{x_r} \models \varphi_k$). Supposons en effet que le chemin π traverse un état $\overline{x_i}$. On a alors $i \leqslant r$, et $w(x_i) = \bot$. Si $i < r$, alors $k-1 \geqslant 2i$ et $k-1 \geqslant 2i-1$. Si $i = r$, alors nous sommes

[2] Une clause C_k peut être comptée quatre fois lors du passage dans la partie supérieure de la structure de Kripke, puis trois fois si le chemin passe par l'arête de poids $3 \times C_k$, puis à nouveau quatre fois sur les n derniers états de la structure.

dans le cas où $w(x_r) = \bot$, donc on a $k \geqslant 2i$, et par suite, $k-1 \geqslant 2i-1$. Dans tous les cas, nous pouvons appliquer l'hypothèse de récurrence pour l'état x_i, ce qui nous assure que le formule φ_k est bien satisfaite dans l'état x_i. On en déduit que $\pi \models [P_{\overline{x}} \Rightarrow \mathbf{EX}\,(P_x \wedge \neg\varphi_{k-1})]\,\mathbf{U}_{=K'}\,\top$;

– montrons maintenant la réciproque de ces deux équivalences : supposons que $k \geqslant 2r-1$ et $x_r \models \varphi_k$ (respectivement $k \geqslant 2r$ et $\overline{x_r} \models \varphi_k$). Soit alors π un chemin témoin, commençant dans l'état x_r (respectivement $\overline{x_r}$), de durée totale K, et dont tous les états satisfont $P_{\overline{x}} \Rightarrow \mathbf{EX}\,(P_x \wedge \neg\varphi_{k-1})$. D'après les remarques que nous avons faites avant l'énoncé du lemme, le fait que le chemin ait une durée K' nous assure que la valuation w associée satisfait toutes les clauses de Cl. Il nous suffit, pour conclure, de montrer que pour tout $i \leqslant r$, $w(x_i) = v_{\mathcal{I}}(x_i)$. Nous prouvons cette égalité par récurrence sur i :
  – si $w(x_i) = \top$, alors $\bigvee_{k=1}^{3} w(\alpha_{i,j,k}) = \top$, puisque la clause $\overline{x_i} \vee \bigvee_{k=1}^{3} \alpha_{i,j,k}$ est satisfaite. Donc $F_i(w(x_1), \ldots, w(x_{i-1}), Z_i)$ est satisfaisable. Par hypothèse d'induction, $w(x_s) = v_{\mathcal{I}}(x_s)$ pour $s < i$. Donc $F_i(v_{\mathcal{I}}(x_1), \ldots, v_{\mathcal{I}}(x_{i-1}), Z_i)$ est satisfaisable, et donc $v_{\mathcal{I}}(x_i) = \top$;
  – si $w(x_i) = \bot$, alors le chemin π passe par $\overline{x_i}$, et donc $\overline{x_i} \models \mathbf{EX}\,(P_x \wedge \neg\varphi_{k-1})$. Donc $x_i \models \varphi_{k-1}$. Si $i < r$, alors $k-1 \geqslant 2i-1$, et l'hypothèse de récurrence nous assure que $v_{\mathcal{I}}(x_i) = \bot$. Si $i = r$, alors nous sommes dans le cas où $k \geqslant 2r$ et $\overline{x_i} \models \varphi_k$. On en déduit que $k-1 \geqslant 2i-1$, et par récurrence, $v_{\mathcal{I}}(x_i) = \bot$. □

On en déduit immédiatement que l'instance $\mathcal{I}$ de SNSAT est « positive », *i.e.* $v_{\mathcal{I}}(x_n) = \top$, si, et seulement si, $K_{\mathcal{I}}, x_n \models \varphi_{2n-1}$. Les constructions que nous avons faites dans cette démonstration pouvant être réalisée avec un espace logarithmique, le problème SNSAT se réduit bien au model checking de TCTL sur des structures de Kripke avec durées strictes. Ce dernier problème est donc Δ_2^P-dur. □

Théorème 4.10 ([LMS02a, Proposition 4.4])
Il existe un algorithme dans Δ_2^P pour le model checking de TCTL *sur les structures de Kripke avec durées.*

Démonstration :
Cette preuve est semblable à celle du théorème 3.30 : nous allons ajouter à l'algorithme de $\mathsf{TCTL}_{\leqslant,\geqslant}$ des procédures d'étiquetage non déterministes, s'exécutant en temps polynomial, pour l'étiquetage par des formules de la forme $\mathbf{E}(\varphi\,\mathbf{U}_{=c}\,\psi)$ et $\mathbf{A}(\varphi\,\mathbf{U}_{=c}\,\psi)$.

Lemme 4.11
Le model checking de formules du type $\mathbf{E}(\varphi\,\mathbf{U}_{=c}\,\psi)$ sur des structures de Kripke avec durées est dans NP.

Démonstration :
La difficulté du problème vient du fait que, la constante c étant codée en binaire, un chemin témoin peut être de longueur exponentielle. On ne peut donc pas simplement « deviner » un chemin puis vérifier qu'il convient. Pour pallier cette difficulté, nous allons représenter les chemins par leur *image de Parikh*.

Soit $K = (S, S_0, l, R)$ une structure de Kripke avec durées, et q_0 un état de S vérifiant $\mathbf{E}(\varphi\, \mathbf{U}_{=c}\, \psi)$. Nous considérons, dans un premier temps, que les durées sont strictes, *i.e.* que les intervalles étiquetant les transitions sont des singletons. Supposons que

$$\pi = q_0 \xrightarrow{d_0} q_1 \xrightarrow{d_1} q_2 \xrightarrow{d_2} \cdots \xrightarrow{d_{n-1}} q_n$$

soit un chemin dans K qui témoigne du fait que $q \models \mathbf{E}(\varphi\, \mathbf{U}_{=c}\, \psi)$. Nous pouvons supprimer de ce chemin les éventuelles boucles de durée nulle, et supposer, en conséquence, que $n < c \times |S|$. Puisque c est codée en binaire, la valeur de n peut être exponentielle dans la taille des données.

Afin d'obtenir un algorithme dans NP, nous n'allons pas deviner le chemin π lui-même, mais plutôt l'image de Parikh de sa suite de transitions, *i.e.* la fonction $\Phi_\pi \colon R \mapsto \mathbb{N}$ qui, à chaque transition de la structure de Kripke avec durées K, associe le nombre de fois que cette arête apparaît dans le chemin π. Il est clair que, contrairement au chemin π, une telle fonction peut être représentée en espace polynomial, puisque les entiers seront codés en binaire. Ce gain en espace entraine la perte de l'information sur l'ordre exact dans lequel ces transitions apparaissent le long de π, mais nous allons voir que l'image de Parikh est suffisante pour déterminer si le chemin correspondant vérifie bien $\varphi\, \mathbf{U}_{=c}\, \psi$.

Soit donc Φ une fonction de R dans $\mathbb{N}$. Nous définissons

$$\varphi^e(q) \stackrel{\text{def}}{=} \sum_{\substack{q' \in S \\ (q',q) \in R}} \Phi(q', q) \qquad\qquad \varphi^s(q) \stackrel{\text{def}}{=} \sum_{\substack{q' \in S \\ (q,q') \in R}} \Phi(q, q')$$

La première fonction compte le nombre de fois qu'une transition entrant dans l'état q apparaît le long d'un (éventuel) chemin correspondant à Φ. La seconde compte combien de fois une transition de cet éventuel chemin sort de l'état q.

Afin de vérifier que Φ correspond à un chemin partant d'un état q_0, se terminant dans un état q_n, et vérifiant $\varphi\, \mathbf{U}_{=c}\, \psi$, il nous suffit de vérifier les conditions suivantes :

– soit tous les états $q \in S$ vérifient $\Phi^e(q) = \Phi^s(q)$ (dans ce cas, nous noterons $q_n = q_0$), soit il existe un état q_n (différent de q_0) tel que

$$\begin{aligned} \forall q \in S \smallsetminus \{q_0, q_n\}.\quad \Phi^e(q) &= \Phi^s(q) \\ \Phi^e(q_0) &= \Phi^s(q_0) - 1 \\ \Phi^e(q_n) &= \Phi^s(q_n) + 1. \end{aligned}$$

– le sous-graphe de K induit par les états dont l'image par Φ est strictement positive est connexe ;

– la durée totale des chemins éventuels représent'es par Φ vaut bien exactement c :

$$\sum_{t=(q,[d,d],q') \in R} d \times \Phi(t) = c\,;$$

– enfin, il faut que tous les états, sauf q_n, vérifient φ, et que q_n vérifie ψ.

L'ensemble de ces conditions est nécessaire et suffisant pour que Φ corresponde bien à un chemin dans K (cela vient des deux premières conditions, d'après le théorème du circuit Eulérien [Bon95]), vérifiant la formule $\varphi\, \mathbf{U}_{=c}\, \psi$ (d'après les deux derniers points). Par ailleurs, tout cela se vérifie bien en temps polynomial, et on en déduit aisément un algorithme non déterministe en temps polynomial pour notre problème.

Dans le cas des structures de Kripke avec durées générales, il suffit de remplacer la troisième condition par la suivante :

$$\sum_{t=(q,[d,d'],q')\in R} d \times \Phi(t) \leqslant c \leqslant \sum_{t=(q,[d,d'],q')\in R} d' \times \Phi(t).$$

Si cette condition est satisfaite, il est alors possible de choisir les durées des transitions afin d'obtenir un chemin de durée totale exactement égale à c. $\square$

Reste à faire l'autre cas :

Lemme 4.12
*Le model checking de formules du type $\mathbf{A}(\varphi\, \mathbf{U}_{=c}\, \psi)$ sur des structures de Kripke avec durées est dans co-*NP*.*

Démonstration :
En utilisant l'équivalence

$$\mathbf{A}(P_1\, \mathbf{U}_{=c}\, P_2) \equiv \mathbf{A}(P_1\, \mathbf{U}_{\geqslant c}\, P_2) \wedge \neg \mathbf{EG}_{=c} \neg P_2$$

nous voyons qu'il suffit de prouver que le model checking de $\mathbf{EG}_{=c}\, P_2$ est dans NP. Pour montrer ce résultat, nous utilisons les mêmes techniques que pour la preuve du lemme précédent. L'algorithme devra cette fois « deviner » deux fonctions $\Phi_d \colon R \to \mathbb{N}$ et $\Phi_f \colon R \to \mathbb{N}$, puis de vérifier que :

- la fonction Φ_d correspond à un chemin fini, de durée totale strictement inférieure à c, partant de l'état q_0 et arrivant dans un état q_p. Pour cela :
  - soit tous les états $q \in S$ vérifient $\Phi^e_d(q) = \Phi^s_d(q)$ (auquel cas $q_p = q_0$), soit il existe un état q_p (différent de q_0) tel que

    $$\begin{aligned} \forall q \in S \smallsetminus \{q_0, q_p\}.\ \ \Phi^e_d(q) &= \Phi^s_d(q) \\ \Phi^e_d(q_0) &= \Phi^s_d(q_0) - 1 \\ \Phi^e_d(q_p) &= \Phi^s_d(q_p) + 1\,; \end{aligned}$$

  - le sous-graphe de K induit par les états dont l'image par Φ_d est strictement positive est connexe ;
  - la durée totale des chemins correspondant à Φ_d est strictement inférieure à c :

    $$\sum_{t=(q,[d,d],q')\in R} d \times \Phi(t) = c.$$

  Nous notons d cette durée, qui nous servira par la suite.
- la fonction Φ_f correspond à la « fin » du chemin. Deux cas peuvent se produire :
  - soit cette fonction correspond à un chemin constitué d'une seule transition, dont l'état initial est q_p, et dont la durée est strictement plus grande que $c - d$. Dans ce cas, si on met « bout à bout » ce chemin et un chemin represénté par Φ_d, on obtient un chemin de durée strictement supérieure à c, et ne vérifiant jamais la contrainte « $= c$ ». Puisque la relation de transition est totale, il est possible de compléter ce chemin en un chemin infini vérifiant $\mathbf{EG}_{=c}\, P_2$;

– soit la fonction Φ_f correspond à un chemin commençant dans l'état q_p, se terminant dans un état q_n (avec éventuellement $q_n = q_p$), de durée totale strictement plus grande que $c - d$, et dont tous les états, sauf éventuellement q_p et q_n, vérifient P_2.

Le premier cas est juste un cas particulier du premier, nous pouvons en fait vérifier les deux cas de la même façon :

– soit tous les états $q \in S$ vérifient $\Phi^e_f(q) = \Phi^s_f(q)$ (dans ce cas, nous noterons $q_n = q_p$), soit il existe un état q_n (différent de q_p) tel que

$$\forall q \in S \smallsetminus \{q_p, q_n\}.\quad \Phi^e_f(q) = \Phi^s_f(q)$$
$$\Phi^e_f(q_p) = \Phi^s_f(q_p) - 1$$
$$\Phi^e_f(q_n) = \Phi^s_f(q_n) + 1.$$

– le sous-graphe de K induit par les états dont l'image par Φ_f est strictement positive est connexe ;

– la durée totale du chemin est (ou du moins « peut être ») strictement supérieure à $c - d$:

$$c - d < \sum_{t=(q,[d,d'],q')\in R} d' \times \Phi_f(t).$$

– tous les états, sauf q_p et q_n éventuellement, vérifient P_2.

Il est clair que s'il existe un chemin partant de q_0 et vérifiant $\mathbf{G}_{=c}\, P_2$, alors il existe deux fonctions Φ_d et Φ_f vérifiant les conditions ci-dessus. Et inversement, si deux telles fonctions existent, elles correspondent a des chemins qui, mis bout-à-bout et complétés en un chemin infini, fournissent un témoin du fait que $q_0 \models \mathbf{EG}_{=c}\, P_2$. Comme précédemment, les fonctions Φ_d et Φ_f sont représentées en binaire, et sont donc de taille polynomiale. Et la vérification des différentes conditions qu'elles doivent satisfaire se fait en temps polynomial. On a donc bien obtenu un algorithme dans NP pour le problème du model checking d'une formule $\mathbf{EG}_{=c}\, P_2$, ce qui nous fournit un algorithme dans co-NP pour le model checking d'une formule $\mathbf{A} P_1 \,\mathbf{U}_{=c}\, P_2$. □

En ajoutant ces procédures à l'algorithme de model checking de $\mathsf{TCTL}_{\leqslant,\geqslant}$ du théorème 3.30, on obtient un algorithme qui fait appel au plus $O(|\varphi| \cdot |K|)$ fois à un oracle NP, afin d'étiqueter les états de la structure par les sous-formules avec contraintes d'égalité, le reste des calculs se faisant en temps polynomial. Cet algorithme est donc bien dans Δ^P_2. □

À partir des théorèmes 4.8 et 4.10, on déduit :

Théorème 4.13 ([LMS02a, théorème 4.9])
Le model checking de TCTL *sur des structures de Kripke avec durées est* Δ^P_2*-complet.*

Par ailleurs, la réduction de la preuve du théorème 4.8 utilisant des durées strictes, ce résultat reste valable pour les structures de Kripke avec durées strictes.

4.2.2 Les autres logiques

Nous étudions ici la complexité du problème du model checking pour trois autres logiques : TCTL^+, TLTL et $\mathsf{TCTL}^\star$.

4.2.2.1 La logique TCTL^+

Puisque le model checking de CTL^+ est Δ_2^P-complet (*cf.* chapitre 3), nous savons que le model checking de $\mathsf{TCTL}^+_{\leqslant,\geqslant}$, même sur les structures de Kripke simples, sera au moins Δ_2^P-difficile. Le résultat suivant nous donne la borne supérieure :

Théorème 4.14
Le model checking de TCTL^+ sur des structures de Kripke avec durées est Δ_2^P-facile.

Démonstration :
Cette preuve repose sur le lemme suivant :

Lemme 4.15
Le model checking de formules de la forme

$$\mathbf{E}\Big(\bigwedge_{i=1}^{m} P_i \,\mathbf{U}_{\sim c_i}\, P'_i \wedge \bigwedge_{j=1}^{n} \neg(Q_j \,\mathbf{U}_{\sim d_j}\, Q'_j)\Big)$$

(où $\sim \in \{\leqslant, =, \geqslant\}$) peut être traité par un algorithme non déterministe polynomial.

Démonstration :
Nous remplaçons, dans un premier temps, les négations de formules avec $\mathbf{U}$ par des formules équivalentes, d'après les équivalences suivantes :

$$\begin{aligned}
\neg(Q_j \,\mathbf{U}_{\leqslant d_j}\, Q'_j) &\equiv \neg(Q_j \,\mathbf{U}\, Q'_j) \vee \mathbf{G}_{\leqslant c}\, \neg Q'_j \\
\neg(Q_j \,\mathbf{U}_{= d_j}\, Q'_j) &\equiv \mathbf{F}_{< d_j}\,(\neg Q_j) \vee \mathbf{F}_{< d_j}\,(\mathbf{X}\, \neg(Q_j \,\mathbf{U}\, Q'_j) \vee \mathbf{G}_{=c}\, \neg Q'_j \\
\neg(Q_j \,\mathbf{U}_{\geqslant d_j}\, Q'_j) &\equiv \mathbf{F}_{< d_j}\,(\neg Q_j) \vee \mathbf{F}_{< d_j}\,(\mathbf{X}\, \neg(Q_j \,\mathbf{U}\, Q'_j)) \\
\neg(Q_j \,\mathbf{U}\, Q'_j) &\equiv \mathbf{G}\, \neg Q'_j \vee (\neg Q'_j) \,\mathbf{U}\, (\neg Q_j \wedge \neg Q'_j)
\end{aligned}$$

Notre algorithme non déterministe va donc s'appliquer de la façon suivante :
- il va tout d'abord « deviner » de quelle façon les formules $\neg(Q_j \,\mathbf{U}_{\sim d_j}\, Q'_j)$ seront vérifiées : d'après les équivalences ci-dessus, il existe en effet plusieurs façons de les vérifier (les formules de droites sont des disjonctions, et en vertu de l'équivalence $\mathbf{E}(\varphi \vee \psi) \equiv \mathbf{E}\varphi \vee \mathbf{E}\psi$, il suffit de n'en vérifier qu'une) ;
- il va ensuite « deviner » les « transitions importantes » du chemin à construire, à savoir :
  - pour chaque constante c qui apparaît dans la formule, la transition qui fera passer la longueur du chemin de « $< c$ » à « $= c$ » et de « $= c$ » à « $> c$ », ou directement de « $< c$ » à « $> c$ » ;
  - pour chaque formule de la forme $\mathbf{F}_{< d_j}\,(\mathbf{X}\,\varphi)$, la transition passant de l'état vérifiant $\mathbf{X}\,\varphi$ à l'état vérifiant φ ;
  - pour chaque formule de la forme $a \,\mathbf{U}\, b$, la « dernière » transition permettant la vérification de cette sous-formule, *i.e.* une transition menant d'un état vérifiant a à un état vérifiant b ;

– une fois posés tous ces « jalons », l'algorithme « devine » les images de Parikh de toutes les portions de chemin entre ces différents points, puis vérifie que l'ensemble des conditions, sur la durée de ces chemins et sur leur étiquetage.

Le nombre de « transitions importantes » est polynomial, de façon évidente. Le nombre de fonctions de R dans $\mathbb{N}$ aussi, et chacune de ces images est de taille polynomiale, comme dans la preuve du lemme 4.11. La vérification de la « correction » du chemin ainsi deviné peut également se faire en temps polynomial. L'algorithme est donc bien dans la classe NP.

□

Le cas général (*i.e.* lorsque la combinaison booléenne peuvent contenir des disjonctions, ou lorsque le quantificateur de chemin est universel) se déduit facilement de ce lemme, et du fait que

$$\mathbf{E}(\varphi \vee \psi) \equiv \mathbf{E}\varphi \vee \mathbf{E}\psi.$$

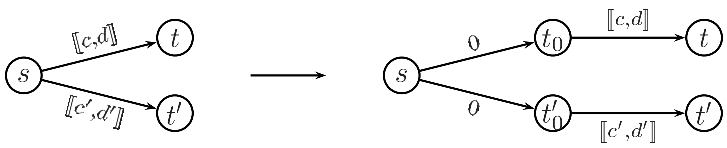
Ces résultats nous assurent de l'existence d'un algorithme dans Δ_2^P pour le problème du model checking de TCTL^+ sur les structures de Kripke avec durées. □

On en déduit immédiatement le théorème suivant :

Théorème 4.16 ([LMS02a, théorème 5.3])
Le model checking de TCTL^+ et $\mathsf{TCTL}^+_{\leqslant,\geqslant}$ est Δ_2^P-complet sur les structures de Kripke simples et sur les structures de Kripke avec durées.

4.2.2.2 La logique TLTL

Sans contrainte d'égalité. Nous savons que ce problème est PSPACE-dur, puisque le model checking de LTL l'est. Ce problème est en fait également PSPACE-facile. Ce résultat est prouvé dans [Hen98] lorsque le modèle considéré est les structures de Kripke simples ou les automates temporisés de [AD94]. Or les automates temporisés sont beaucoup plus expressifs que les structures de Kripke avec durées que nous considérons ici. Une structure de Kripke avec durées peut en effet être représentée par un automate temporisé à une seule horloge, cette horloge étant remise à zéro à l'entrée dans chaque état, et « gardant » les transitions. Afin de garder la sémantique des structures de Kripke avec durées, nous devrions cependant ajouter un état intermédiaire pour chaque transition, comme indiqué sur la figure suivante :



Afin que les états intermédiaires que nous venons d'ajouter n'interfèrent pas dans l'évaluation de la formule, nous les étiquetons par la proposition atomique $P_\varnothing$, et remplaçons la formule φ à vérifier par la formule obtenue en appliquant récursivement les transformations suivantes :

$$\overline{p} \overset{\text{def}}{=} \neg P_\varnothing \wedge p \qquad \overline{\varphi \wedge \psi} \overset{\text{def}}{=} \neg P_\varnothing \wedge \overline{\varphi} \wedge \overline{\varphi}$$
$$\overline{\neg\varphi} \overset{\text{def}}{=} \neg P_\varnothing \wedge \neg\overline{\varphi} \qquad \overline{\mathbf{X}\,\varphi} \overset{\text{def}}{=} \neg P_\varnothing \wedge \mathbf{X}\,\overline{\varphi}$$
$$\overline{\varphi\,\mathbf{U}\,\psi} \overset{\text{def}}{=} \neg P_\varnothing \wedge (\overline{\varphi}\,\mathbf{U}\,\overline{\psi})$$

On a ainsi réduit le problème du model checking des structures de Kripke avec durées au model checking des automates temporisés, pour la logique $\mathsf{TLTL}_{\leqslant,\geqslant}$. Ce dernier problème étant PSPACE-dur, le premier l'est aussi. D'où le théorème suivant :

Théorème 4.17
Le model checking de $\mathsf{TLTL}_{\leqslant,\geqslant}$ *est* PSPACE*-complet sur les structures de Kripke simples et sur les structures de Kripke avec durées.*

Avec contraintes d'égalité. Lorsqu'on a des contraintes d'égalité, nous allons montrer que, quel que soit le type de structure utilisé, le problème est EXPSPACE-complet.

[AH94] a prouvé que le model checking de la logique FQLTL sur des « *timed state graphs* » est EXPSPACE-complet. Les *timed state graphs* ne permettent pas, cependant, de coder des structures de Kripke avec durées (ce sont, *grosso modo*, des structures de Kripke avec durées strictes). Cependant, [AH94] montre que la validité de FQLTL est également EXPSPACE-complet, et c'est sur ce résultat que nous allons nous baser : comme dans [Hen91, chapitre 4] pour les *timed state graphs*, nous allons coder nos structures de Kripke avec durées en des formules de FQLTL.

Soit K une structure de Kripke avec durées, et φ une formule de TLTL. La formule φ peut être transformée en une formule équivalente $\overline{\varphi}$ de FQLTL. Concernant le codage de K, il est semblable au codage des structures de Kripke dans LTL que nous avons vu dans la démonstration de la proposition 1.41, page 42. La seule modification à apporter est, bien entendu, la gestion de la durée des transitions. En conservant les mêmes notations que dans la proposition 1.41, nous écrivons :

$$\varphi_K^{trans} \overset{\text{def}}{=} \mathbf{G}\Big(\bigvee_{(s,\llbracket d,d'\rrbracket,s')\in R} x.(P_s \wedge \mathbf{X}\,y.(P_{s'} \wedge d \leqslant x - y \leqslant d'))\Big).$$

Si nous notons φ_K la formule complète codant la structure de Kripke avec durées K, alors tous les chemins de K issus d'un état s_0 vérifient φ si, et seulement si, la formule $(s_0 \wedge \varphi_K) \Rightarrow \overline{\varphi}$ est valide.

Inversement, il est facile d'adapter la réduction du lemme 2.29, à la page 98, à la logique TLTL, en gardant le modèle des structures de Kripke simples. Nous gardons exactement la même structure que dans le lemme 2.29, et les mêmes notations[3]. Nous n'avons qu'une formule à modifier (la dernière), puisque les autres formules de la preuve du lemme sont des formules de LTL.

[3]En fait, étant donné que l'on peut « compter » jusqu'à 2^n avec la logique TLTL, les bits b_1, ..., b_m ne sont plus utiles ici. Cependant, afin de ne pas introduire de nouvelles notations, nous calquons cette réduction sur celle du lemme 2.29.

Pour exprimer qu'une cellule et sa cellule « supérieure » éventuelle portent bien la même couleur sur l'arête qu'elles ont en commun, il nous suffira d'écrire :

$$\mathbf{G}\Big[\bigwedge_{d\in T}\Big(d \Rightarrow \Big(\mathbf{F}_{<2^m(m+1)}\,E \vee \mathbf{F}_{=2^m(m+1)}\Big(\bigvee_{\substack{d'\in T\\ \mathrm{up}(d)=\mathrm{down}(d')}} d'\Big)\Big)\Big)\Big].$$

Les constantes étant codées en binaire, cette formule est bien de taille polynomiale dans la taille des données. On a donc réduit un problème EXPSPACE-complet au problème de model checking de TLTL sur des structures de Kripke simples.

Théorème 4.18
Le model checking de TLTL *est* EXPSPACE-*complet sur les structures de Kripke simples et sur les structures de Kripke avec durées.*

4.2.2.3 La logique TCTL*

Pour TCTL* et TCTL$^\star_{\leqslant,\geqslant}$, en appliquant les techniques développées dans [EL87], nous obtenons des algorithmes de la même complexité que pour les logiques du temps linéaire correspondantes : il est en effet facile d'écrire un algorithme en temps polynomial faisant appel à un oracle afin d'étiqueter les états par les sous-formules de chemin.

Dans le cas de la logique TCTL$^\star_{\leqslant,\geqslant}$, l'oracle devra vérifier des propriétés de TLTL$_{\leqslant,\geqslant}$, et sera donc dans PSPACE. Dans le cas de TCTL*, il vérifiera des propriétés de TCTL* et sera donc dans EXPSPACE. Les algorithmes obtenus sont donc de complexité $\mathsf{P}^{\mathsf{PSPACE}}$ et $\mathsf{P}^{\mathsf{EXPSPACE}}$ respectivement. On se convainc facilement (*cf.* [Pap94]) que ces classes sont égales à PSPACE et EXPSPACE, respectivement.

Inversement, puisque TCTL$^\star_{\leqslant,\geqslant}$ et TCTL* « contiennent » TLTL$_{\leqslant,\geqslant}$ et TLTL respectivement, leurs problèmes de model checking sont respectivement PSPACE-dur et EXPSPACE-dur.

Le tableau 4.1 récapitule les résultats que nous avons établis tout au long de ce chapitre.

		$\longrightarrow, \xrightarrow{0/1}$	$\xrightarrow{d}, \xrightarrow{[\![c,d]\!]}$
TCTL	$\leqslant,\geqslant$	P-complet	P-complet
	$\leqslant,=,\geqslant$		Δ_2^P-complet
TCTL$^+$	$\leqslant,\geqslant$	Δ_2^P-complet	
	$\leqslant,=,\geqslant$		
TLTL	$\leqslant,\geqslant$	PSPACE	
	$\leqslant,=,\geqslant$	EXPSPACE	
TCTL*	$\leqslant,\geqslant$	PSPACE	
	$\leqslant,=,\geqslant$	EXPSPACE	

TAB. 4.1 – Récapitulatif des résultats

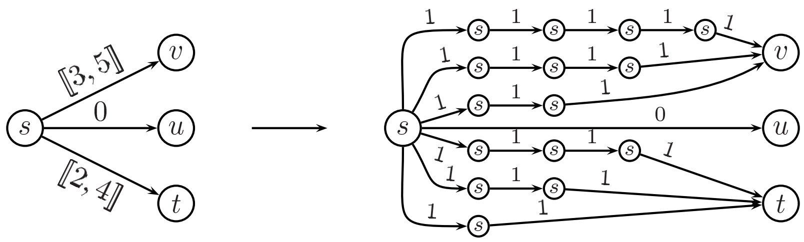
4.3 Autres sémantiques

La sémantique que nous avons utilisée sur nos structures de Kripke est assez particulière, et il est, bien entendu, possible d'en définir d'autres. Nous allons en présenter trois ici, et voir que le choix de la sémantique influe sur la complexité des problèmes de vérification.

4.3.1 Le sémantique « continue »

La sémantique que nous avons utilisée dans la section précédente est quelque peu particulière : entre deux états, et pour la durée de la transition, il n'y a pas d'état intermédiaire. Il est possible de définir une sémantique « avec états intermédiaires », afin de rendre plus « lisses », plus « continus », les chemins issus de ces structures. Dans la sémantique « continue », les transitions se comportent comme sur la figure 4.4 On ne peut donc pas attendre

Figure 4.4 La sémantique « continue »

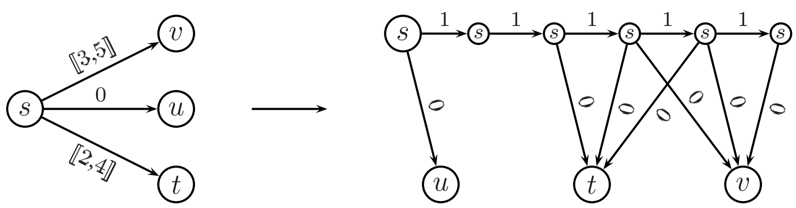


dans un état, dans le sens où il faut décider immédiatement quelle transition (et quelle durée, le cas échéant) on va prendre. Néanmoins, il y a des états intermédiaires, c'est à dire que les propositions atomiques de s restent vérifiées du temps 0 au temps $d-1$, si une transition a pour durée d.

4.3.2 La sémantique « late choice »

Dans la sémantique des structures de Kripke avec durées que nous avons utilisée jusqu'à maintenant, le choix de la transition a prendre se faisait dès l'arrivée dans un état. La sémantique « late choice », comme son nom l'indique, permet de retarder ce choix, en restant dans un état jusqu'à ce que la transition suivante soit effective, et en ayant à tout moment la possibilité de faire toutes les transitions encore possibles. La figure 4.5 montre le « comportement » des transitions avec une telle sémantique. Cette sémantique

Figure 4.5 La sémantique « late choice »

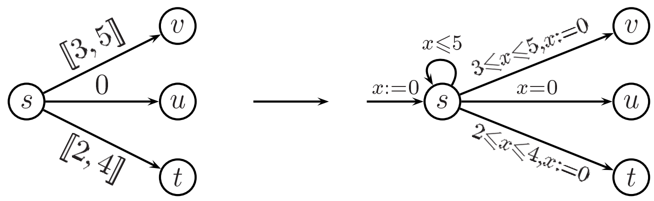


est semblable à celles des automates temporisés (avec temps discret), mais, tout comme la sémantique continue, elle n'est pas définie lorsqu'un intervalle est non borné à droite.

4.3.3 La sémantique « automates temporisés »

Cette sémantique donne aux structures de Kripke avec durées un comportement comparable à celui des automates temporisés de [AD94][4]. Cette sémantique est schématisée à la figure 4.6. Cette sémantique permet de traiter le cas des intervalles non bornés.

Figure 4.6 La sémantique « automate temporisé »



4.3.4 La complexité dépend beaucoup de la sémantique

Ces différentes sémantiques sont plus « naturelles » et plus proches de celles des automates temporisés. Les sémantiques continue et *late choice* posent cependant un problème dans le cas des intervalles non bornés, alors que la sémantique « automates temporisés » peut traiter ce cas.

Nos travaux sur ces différentes sémantiques sont en cours. Nous avons réussi à montrer que, pour les sémantiques continue et *late choice*, même avec des structures de Kripke avec durées strictes, *i.e.* avec des singletons sur les transitions, le problème du model checking de TCTL devient PSPACE-dur. Dans le cas de $\mathsf{TCTL}_{\leqslant,\geqslant}$, le problème semble rester P-complet, même avec la sémantique des automates temporisés.

Conclusion

Dans ce chapitre, nous avons ajouté une notion quantitative aux logiques temporelles, et étudié la différence de complexité par rapport aux logiques non quantitatives. Nous pouvons tirer deux enseignements de cette étude :

- nous avons trouvé de nouveaux problèmes complets pour la classe Δ_2^P dans le domaine de la vérification. La complexité de ces problèmes est très liée à la sémantique que nous avons utilisée sur les modèles auxquels ils s'appliquent, puisque ces problèmes deviennent PSPACE-dur si on utilise la sémantique (plus naturelle) des automates temporisés ;

[4] Il y a cependant une différence fondamentale entre les structures de Kripke avec durées et les automates temporisés, qui tient au fait que ces derniers peuvent être « synchronisés », et qu'il est ainsi possible de définir des systèmes complexes (du point de vue de leurs comportements) de manière succincte.

– pour les modèles que nous avons étudiés, et à l'exception du cas particulier de la logique TCTL$^+$, nous avons confirmé que, comme déjà exprimée dans [EMSS92, AFH96, Hen98], l'ajout de contraintes d'égalité dans ces logiques quantitatives augmente la complexité du problème.

Pour en savoir plus...

De nombreux travaux ont été faits sur les automates temporisés et sur les automates hybrides (qui sont des automates « temporisés » dont les horloges peuvent varier de différentes manières) depuis leurs introductions au début des années 1990 [ACHH93, HNSY94, ACH$^+$95, HKPV98].

Concernant les logiques temporisées, notons que plusieurs logiques permettant d'exprimer les contraintes temporelles sous la forme d'intervalles de $\mathbb{N}$ ont été proposées (MTL [AH90], MITL [AFH91]). D'autres possibilités sont également citées dans [AH91].

Par ailleurs, et pour faire un lien avec le premier chapitre de cette thèse, [AH93] a étudié l'ajout du passé dans la logique TLTL. La logique qui en résulte est nommée TPTL$_P$. À la différence du cas non temporisé, l'ajout du passé dans la logique TLTL n'est pas « gratuit » puisque le problème de la validité de TPTL$_P$ est non-élémentaire.

Conclusion et perspectives

Bilan de cette étude

Nous avons abordé, au cours de cette thèse, les problèmes d'expressivité et de complexité de différents types de logiques temporelles.

L'étude de l'impact du passé dans la logique temporelle du temps linéaire a confirmé les conjectures qui étaient formulées depuis une vingtaine d'années par les chercheurs en informatique :

– du point de vue de l'expressivité, nous avons établi l'existence d'une différence de concision exponentielle entre la logique avec modalités du passé et la logique « pure futur ». Nous avons observé également que la logique PLTL a la même expressivité que la logique LTL, mais que l'ajout, dans des fragments de LTL, des modalités « symétriques » aux modalités déjà autorisées pouvait apporter de l'expressivité ;
– du point de vue de la vérification, notre étude a confirmé que l'utilisation de modalités du passé n'augmente pas la complexité des problèmes de vérification, même dans les fragments de LTL.

Bien que cela confirme les résultats conjecturés dans le domaine depuis de nombreuses années, nos conclusions sont différentes de celle communément admises ; en effet, de nombreuses études et implémentations n'ont considéré que le fragment « pur futur » de la logique temporelle du temps linéaire, puisque l'expressivité était la même. Notre conclusion sur ce point est à l'opposé : puisque la complexité est identique, mais que le passé est plus concis, il est important d'intégrer les modalités du passé dans les recherches sur les logiques temporelles, et dans les implémentations d'algorithmes de model checking.

Toujours dans ce cadre, nous avons étudié l'impact, toujours sous le double point de vue de l'expressivité et de la complexité, de l'ajout de la modalité $\mathbf{N}$ définie dans [LS95]. Cette modalité n'avait été étudiée jusqu'alors que dans le cadre de la logique arborescente. L'ajout de cette modalité dans la logique PLTL est comparable, du point de vue de l'expressivité, à l'ajout des modalités du passé dans la logique LTL : plus concis, plus pratique, mais pas plus expressif. Au niveau de la complexité cependant, cette modalité fait passer le problème du model checking de PSPACE à EXPSPACE, puisque les problèmes de vérification de NLTL sont EXPSPACE-complets.

Nous avons ensuite étudié plusieurs logiques arborescentes : CTL^+, ECTL, ECTL^+, FCTL, ... Ces logiques ont été définies au milieu des années 1980, afin d'enrichir la logique CTL, principalement pour lui permettre d'exprimer des propriétés d'équité.

Du point de vue de la vérification, seule la complexité exacte de ECTL avait été caractérisée. Pour les autres logiques, il était connu que leur model checking était NP-dur et co-NP-dur. Nous avons développé dans cette thèse une technique permettant de caractériser des problèmes complets pour la classe Δ_2^P, une classe intermédiaire entre NP et PSPACE. Cette classe contient en fait tous les problèmes pouvant être traités par un algorithme déterministe en temps polynomial, utilisant un nombre polynomial de requêtes à un oracle NP.

Nous prouvons donc que les problèmes de model checking de CTL^+, FCTL et BT^* sont Δ_2^P-complets, ce qui constituait également un important problème ouvert depuis le milieu des années 1980.

La technique que nous avons développée pour montrer la complétude de problèmes pour la classe Δ_2^P a pu être appliquée à d'autres logiques : nous avons en effet défini, dans le dernier chapitre de cette thèse, des modèles « quantifiés » permettant de donner une durée aux transitions des systèmes étudiés. Sur ces systèmes, il est alors possible d'exprimer des propriétés quantitatives comme, par exemple, le fait qu'un évènement se produira « au plus tard dans 10 unités de temps ». La complexité de ces logiques avaient déjà été étudiée dans des cas simples, en particulier lorsque les modèles étaient des structures de Kripke « classiques » dont chaque transition a une durée égale à 1. Dans ce cas, le model checking de la logique TCTL (une version temporisée de CTL dans laquelle la modalité $\mathbf{U}$ est « quantifiée » par une contrainte de la forme $\sim c$ où $\sim \in \{<, \leqslant, =, \geqslant, >\}$ et $c \in \mathbb{N}$) reste P-complet. Dans notre modèle, dans lequel les transitions peuvent avoir une durée arbitraire, ce problème devient Δ_2^P-complet, mais reste polynomial si les formules n'utilisent pas de contraintes d'égalité.

Perspectives

De nombreux travaux sont envisageables pour étendre et continuer les études des logiques temporelles.

Pour commencer, nous avons prouvé que la logique PLTL est exponentiellement plus concise que LTL. Inversement, le meilleur algorithme connu semble aboutir à des formules de taille triple exponentielle dans la taille de la formule initiale. Il reste donc soit à améliorer l'algorithme de traduction, soit à prouver que la différence de concision est doublement ou triplement exponentielle.

Cependant, du point de vue de la vérification et puisque la traduction va nécessairement faire « exploser » la taille de la formule, il serait relativement inefficace d'utiliser un algorithme de traduction afin de faire du model checking de propriétés de PLTL. Puisque le model checking de PLTL a la même complexité que celui de LTL, il nous semble opportun d'écrire un outil de model checking de la logique PLTL, puisqu'aucun outil n'existe à l'heure actuelle pour cette logique (les model checkers comme SPIN ou SMV ne permettent de vérifier que des propriétés de LTL).

Concernant toujours la logique temporelle du temps linéaire, nous avons observé que le model checking d'un chemin ultimement périodique est P-complet pour la logique NLTL (*i.e.* la logique PLTL augmentée de l'opérateur $\mathbf{N}$). La complexité du model checking d'un chemin ultimement périodique pour une formule de LTL ou de PLTL n'est pas connue. Le meilleur algorithme connu à l'heure actuelle s'exécute en temps polynomial (voir, par exemple, l'algorithme que nous avons développé dans le premier chapitre). Fournir un algorithme plus simple permettrait, par exemple, d'améliorer la qualité et la rapidité d'algorithmes d'analyses de logs et de traces d'exécutions de logiciels.

Concernant les logiques quantitatives, comme nous l'avons évoqué à la fin du dernier chapitre, il serait intéressant d'étudier l'évolution de la complexité de la vérification de ces logiques pour différentes sémantiques des structures temporisées que nous avons définies (les structures de Kripke avec durées). Nous espérons que cette étude débouchera également sur la caractérisation de la classe de complexité des problèmes de vérification des logiques temporisées sur des automates temporisées ayant une seule horloge.

Index

Bibliographie

[ACD93] Rajeev Alur, Costas Courcoubetis, et David L. Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1), pages 2–34, Academic Press, Mai 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), pages 3–34, Elsevier Science, 1995.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, et Pei-Hsin Ho. Hybrid Automata : An Algorithmic Approach to the Specification and Verification of Hybrid Systems. Dans Robert L. Grossman, Anil Nerode, Anders P. Ravn, et Hans Rischel, éditeurs, *Hybrid Systems*, volume 736 de *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.

[AD94] Rajeev Alur et David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2), pages 183–235, Elsevier Science, Avril 1994.

[AFH91] Rajeev Alur, Tómas Feder, et Thomas A. Henzinger. The Benefits of Relaxing Punctuality. Dans *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC'91)*, Août 1991, pages 139–152. ACM Press, Août 1991.

[AFH96] Rajeev Alur, Tómas Feder, et Thomas A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1), pages 116–146, ACM Press, Janvier 1996.

[AH89] Rajeev Alur et Thomas A. Henzinger. A Really Temporal Logic. Dans *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, Octobre 1989, pages 164–169. IEEE Comp. Soc. Press, Octobre 1989.

[AH90] Rajeev Alur et Thomas A. Henzinger. Real-Time Logics : Complexity and Expressiveness. Dans *Proceedings of the 5th Annual Symposium on Logic in Computer Science (LICS'90)*, 1990, pages 390–401. IEEE Comp. Soc. Press, 1990.

[AH91] Rajeev Alur et Thomas A. Henzinger. Logics and Models of Real Time : A Survey. Dans Jaco W. de Bakker, Cornelis Huizing, Willem-Paul de Roever, et Grzegorz Rozenberg, éditeurs, *Proceedings of REX Workshop 1991*, Juin 1991, volume 600 de *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, Juin 1991.

[AH93] Rajeev Alur et Thomas A. Henzinger. Real-time Logics : Complexity and Expressiveness. *Information and Computation*, 104(1), pages 35–77, Academic Press, Mai 1993.

[AH94] Rajeev Alur et Thomas A. Henzinger. A Really Temporal Logic. *Journal of the ACM*, 41(1), pages 181–203, ACM Press, Janvier 1994.

[AHU74] Alfred V. Aho, John Hopcroft, et Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AI01] Micah Adler et Neil Immerman. An n! Lower Bound On Formula Size. Dans *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS 2001)*, Juin 2001, pages 197–206. IEEE Comp. Soc. Press, Juin 2001.

[BAPM83] Mordechai Ben-Ari, Amir Pnueli, et Zohar Manna. The Temporal Logic of Branching Time. *Acta Informatica*, 20, pages 207–226, Springer-Verlag, 1983.

[BdRV01] Patrick Blackburn, Marteen de Rijke, et Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[BH81] Arthur J. Bernstein et Jr. Harter, Paul K. Proving Real-Time Properties of Programs with Temporal Logic. Dans *Proceedings of the 8th ACM Symposium on Operating Systems Principles (SOSP'81)*, Décembre 1981, volume 15(5) de *Operating System Review*, pages 1–11. ACM Press, Décembre 1981.

[Bla00] Patrick Blackburn. Representations, Reasoning and Relational Structures : a Hybrid Logic Manifesto. *Logic Journal of the IGPL*, 8(3), pages 339–365, Oxford University Press, Mai 2000.

[Bon95] John Adrian Bondy. Basic Graph Theory : Paths and Circuits. Dans Ronald L. Graham, Martin Grötschel, et László Lovász, éditeurs, *Handbook of Combinatorics*, volume 1, pages 3–110. North-Holland, 1995.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transaction on Computers*, 35(8), pages 677–691, IEEE Comp. Soc. Press, Août 1986.

[BS95] Patrick Blackburn et Jerry Seligman. Hybrid Languages. *Journal of Logic, Language and Information*, 4(3), pages 251–272, Kluwer Academic, 1995.

[BT99] Patrick Blackburn et Miroslava Tzakova. Hybrid Languages and Temporal Logic. *Logic Journal of the IGPL*, 7(1), pages 27–54, Oxford University Press, Janvier 1999.

[Büc62] Julius R. Büchi. On a decision method in restricted second order arithmetic. Dans Ernest Nagel, Patrick Suppes, et Alfred Tarski, éditeurs, *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science*, 1960, pages 1–11. Stanford University Press, 1962.

[Bus93] Samuel R. Buss. Algorithms for Boolean Formula Evaluation and for Tree Contraction. Dans Peter Clote et Jan Krajícek, éditeurs, *Arithmetic, Proof Theory and Computational Complexity*, pages 95–115. Oxford University Press, 1993.

[CCG+02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, et Armando Tacchella. nuSMV2 : An OpenSource Tool for Symbolic Model Checking. Dans *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, Juillet 2002, Lecture Notes in Computer Science, pages 359–364. Springer-Verlag, Juillet 2002.

[CCGR99] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, et Marco Roveri. NuSMV : A New Symbolic Model Checker. Dans Nicolas Halbwachs et Doron A. Peled, éditeurs, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, Juillet 1999, volume 1633 de *Lecture Notes in Computer Science*, pages 495–499. Springer-Verlag, Juillet 1999.

[CE81] Edmund M. Clarke et E. Allen Emerson. Design and Synthesis of Synchronous Skeletons using Branching-Time Temporal Logic. Dans Dexter Kozen, éditeur, *Proceedings of the 3rd Workshop on Logics of Programs (LOP'81)*, Mai 1981, volume 131 de *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, Mai 1981.

[CES83] Edmund M. Clarke, E. Allen Emerson, et A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications : A Practical Approach. Dans *Conference Record of the 10th ACM Symposium on Principles of Programming Languages (POPL'83)*, Janvier 1983, pages 117–126. ACM Press, Janvier 1983.

[CES86] Edmund M. Clarke, E. Allen Emerson, et A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2), pages 244–263, ACM Press, Avril 1986.

[CGH94] Edmund M. Clarke, Orna Grumberg, et Kiyoharu Hamaguchi. Another Look at LTL Model Checking. Dans David L. Dill, éditeur, *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, Juin 1994, volume 818 de *Lecture Notes in Computer Science*, pages 415–427. Springer-Verlag, Juin 1994.

[CGP99] Edmund M. Clarke, Orna Grumberg, et Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CLR99] Thomas H. Cormen, Charles E. Leiserson, et Ronald L. Rivest. *Introduction à l'algorithmique*. Dunod, 1999.

[CS01] Edmund M. Clarke et Bernd-Holger Schlingloff. Model Checking. Dans Alan Robinson et Andrei Voronkov, éditeurs, *Handbook of Automated Reasoning*, pages 1635–1790. Elsevier Science, 2001.

[CVWY91] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, et Mihalis Yannakakis. Memory-Efficient Algorithms for the Verification of Temporal Properties. Dans Edmund M. Clarke et Robert P. Kurshan, éditeurs, *Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV'90)*, Juin 1990, volume 531 de *Lecture Notes in Computer Science*, pages 233–242. Springer-Verlag, 1991.

[Der00] Nachum Dershowitz. Software Horror Stories. http://www.cs.tau.ac.il/~nachumd/verify/horror.html, 2000.

[Dre00] Rolf Drechsler. *Formal Verification of Circuits*. Kluwer Academic, 2000.

[DS02] Stéphane Demri et Philippe Schnoebelen. The Complexity of Propositional Linear Temporal Logics in Simple Cases. *Information and Computation*, 174(1), pages 84–103, Academic Press, Avril 2002.

[EC80] E. Allen Emerson et Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. Dans Jaco W. de Bakker et Jan van Leeuwen, éditeurs, *Proceedings of the 7th International Colloquium on Automata, Languages and Programming (ICALP'80)*, Juillet 1980, volume 85 de *Lecture Notes in Computer Science*, pages 169–181. Springer-Verlag, Juillet 1980.

[EC96] European Space Agency et Centre National d'Études Spatiales. Rapport de la Commission d'enquête Ariane 501. http://www.cnes.fr/espace_pro/communiques/cp96/rapport_501/rapport_501.html, Juillet 1996.

[EH85] E. Allen Emerson et Joseph Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences*, 30(1), pages 1–24, Academic Press, Février 1985.

[EH86] E. Allen Emerson et Joseph Y. Halpern. "Sometimes" and "Not Never" Revisited : On Branching versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1), pages 151–178, ACM Press, Janvier 1986.

[EL87] E. Allen Emerson et Chin-Laung Lei. Modalities for Model Checking : Branching Time Logic Strikes Back. *Science of Computer Programming*, 8(3), pages 275–306, Elsevier Science, Juin 1987.

[Eme90] E. Allen Emerson. Temporal and Modal Logic. Dans Jan van Leeuwen, éditeur, *Handbook of Theoretical Computer Science*, volume B, chapitre 16, pages 995–1072. Elsevier Science, 1990.

[EMSS92] E. Allen Emerson, Aloysius Ka-Lau Mok, A. Prasad Sistla, et Jayashankar Srinivasan. Quantitative Temporal Reasoning. *Real-Time Systems*, 4, pages 331–352, Kluwer Academic, 1992.

[ES84] E. Allen Emerson et A. Prasad Sistla. Deciding Full Branching Time Logic. *Information and Control*, 61(3), pages 175–201, Academic Press, Juin 1984.

[ES88] E. Allen Emerson et Jayashankar Srinivasan. Branching Time Temporal Logic. Dans Jaco W. de Bakker, Willem-Paul de Roever, et Grzegorz Rozenberg, éditeurs, *Proceedings of REX School/Workshop 1988*, Mai-Juin 1988, volume 354 de *Lecture Notes in Computer Science*, pages 123–172. Springer-Verlag, Mai 1988.

[ET97] E. Allen Emerson et Richard J. Trefler. Generalized Quantitative Temporal Reasoning. Dans Michel Bidoit et Max Dauchet, éditeurs, *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development (TAPSOFT'97)*, Avril 1997, volume 1214 de *Lecture Notes in Computer Science*, pages 189–200. Springer-Verlag, Avril 1997.

[Ete99] Kousha Etessami. Stutter-Invariant Languages, ω-Automata, and Temporal Logic. Dans Nicolas Halbwachs et Doron A. Peled, éditeurs, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, Juillet 1999, volume 1633 de *Lecture Notes in Computer Science*, pages 236–248. Springer-Verlag, Juillet 1999.

[Ete00] Kousha Etessami. A Note on a Question of Peled and Wilke Regarding Stutter-Invariant LTL. *Information Processing Letters*, 75(6), pages 261–263, Elsevier Science, Novembre 2000.

[EVW97] Kousha Etessami, Moshe Y. Vardi, et Thomas Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. Dans *Proceedings of the 12th Annual Symposium on Logic in Computer Science (LICS'97)*, Juin-Juillet 1997, pages 228–235. IEEE Comp. Soc. Press, Juin 1997.

[EVW02] Kousha Etessami, Moshe Y. Vardi, et Thomas Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. *Information and Computation*, 179(2), pages 279–295, Academic Press, Décembre 2002.

[EW00] Kousha Etessami et Thomas Wilke. An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraïssé Game for Temporal Logic. *Information and Computation*, 160(1-2), pages 88–108, Academic Press, Juillet 2000.

[Gab89] Dov M. Gabbay. The Declarative Past and Imperative Future : Executable Temporal Logic for Interactive Systems. Dans Behnam Banieqbal, Howard Barringer, et Amir Pnueli, éditeurs, *Proceedings of the 1st Conference on Temporal Logic in Specification*, Avril 1987, volume 398 de *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

[Gal99] Antony Galton. Stanford Encyclopedia of Philosophy : Temporal Logic. `http://plato.stanford.edu/entries/logic-temporal/`, 1999.

[Gar01] James W. Garson. Stanford Encyclopedia of Philosophy : Modal Logic. `http://plato.stanford.edu/entries/logic-modal/`, 2001.

[GHR95] Raymond Greenlaw, H. James Hoover, et Walter L. Ruzzo. *Limits to Parallel Computation*. Oxford University Press, 1995.

[GJ79] Michael R. Garey et David S. Johnson. *Computers and Intractability*. W. H. Freeman & Co., 1979.

[GO01] Paul Gastin et Denis Oddoux. Fast LTL to Büchi Automata Translation. Dans Gérard Berry, Hubert Comon, et Alain Finkel, éditeurs, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, Juillet 2001, volume 2102 de *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, Juillet 2001.

[GO03] Paul Gastin et Denis Oddoux. LTL with Past and Two-Way Very-Weak Alternating Automata. Rapport technique, LIAFA, Université Paris 7, Paris, France, 2003.

[Gor96] Valentin Goranko. Hierarchies of Modal and Temporal Logics with Reference Pointers. *Journal of Logic, Language and Information*, 5(1), pages 1–24, Kluwer Academic, Avril 1996.

[Got95] Georg Gottlob. NP Trees and Carnap's Modal Logic. *Journal of the ACM*, 42(2), pages 421–457, ACM Press, Mars 1995.

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, et Jonathan Stavi. On the Temporal Analysis of Fairness. Dans *Conference Record of the 7th ACM Symposium on Principles of Programming Languages (POPL'80)*, Janvier 1980, pages 163–173. ACM Press, Janvier 1980.

[GPVW96] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, et Pierre Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. Dans Piotr Dembinski et Marek Sredniawa, éditeurs, *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification (PSTV'95)*, Juin 1995, volume 38 de *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1996.

[GRF00] Dov M. Gabbay, Mark Reynolds, et Marcello Finger, éditeurs. *Temporal Logic : Mathematical Foundations and Computational Aspects*, volume 2. Clarendon Press, 2000.

[Har83] David Harel. Recurring Dominoes : Making the Highly Undecidable Highly Understandable (preliminary report). Dans Marek Karpinski, éditeur, *Proceedings of the 4th International Symposium on Fundamentals of Computation Theory (FCT'83)*, Août 1983, volume 158 de *Lecture Notes in Computer Science*, pages 177–194. Springer-Verlag, Août 1983.

[Har85] David Harel. Recurring Dominoes : Making the Highly Undecidable Highly Understandable. *Annals of Discrete Mathematics*, 24, pages 51–72, North-Holland, 1985.

[Har92] David Harel. *Algorithmics : The Spirit of Computing.* Addison-Wesley, seconde édition, 1992.

[Hen91] Thomas A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* Thèse de doctorat, Stanford University, Palo Alto, California, USA, 1991.

[Hen98] Thomas A. Henzinger. It's About Time : Real-Time Logics Reviewed. Dans Davide Sangiorgi et Robert de Simone, éditeurs, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, Septembre 1998, volume 1466 de *Lecture Notes in Computer Science*, pages 439–454. Springer-Verlag, Septembre 1998.

[HKP82] David Harel, Dexter Kozen, et Rohit Parikh. Process Logic : Expressiveness, Decidability and Completeness. *Journal of Computer and System Sciences*, 25(2), pages 144–170, Academic Press, Octobre 1982.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, et Pravin Varaiya. What is decidable about Hybrid Automata ? *Journal of Computer and System Sciences*, 57(1), pages 94–124, Academic Press, Août 1998.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, et Sergio Yovine. Symbolic Model Checking for Real Time Systems. *Information and Computation*, 111(2), pages 193–244, Academic Press, Juin 1994.

[Hol91] Gerard J. Holzmann. *Design and Verification of Computer Protocols*. Prentice Hall Int., 1991. `http://spinroot.com/spin/Doc/Book91.html`.

[Hol97] Gerard J. Holzmann. The Model Checker SPIN. *IEEE Transaction on Software Engineering*, 23(5), pages 279–295, IEEE Comp. Soc. Press, Mai 1997.

[HP85] David Harel et David Peleg. Process Logic with Regular Formulas. *Theoretical Computer Science*, 38, pages 307–322, Elsevier Science, 1985.

[HSLL97] Klaus Havelund, Arne Skou, Kim Gulstrand Larsen, et Kristian Lund. Formal Modelling and Analysis of an Audio/Video Protocol : An Industrial Case Study Using UPPAAL. Dans *Proceedings of the 18th Symposium on Real-Time Systems (RTS'97)*, Décembre 1997, pages 2–13. IEEE Comp. Soc. Press, Décembre 1997.

[HU79] John Hopcroft et Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[IK89] Neil Immerman et Dexter Kozen. Definability with Bounded Number of Bound Variables. *Information and Computation*, 83(2), pages 121–139, Academic Press, Novembre 1989.

[JL03] Jan Johannsen et Martin Lange. CTL^+ is Complete for Double Exponential Time. Dans *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, Juin-Juillet 2003, Lecture Notes in Computer Science. Springer-Verlag, Juin 2003. À paraître.

[Joh90] David S. Johnson. A Catalog of Complexity Classes. Dans Jan van Leeuwen, éditeur, *Handbook of Theoretical Computer Science*, volume A, chapitre 2, pages 67–161. Elsevier Science, 1990.

[Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. Thèse de doctorat, UCLA, Los Angeles, California, USA, 1968.

[KdR83] Ron Koymans et Willem-Paul de Roever. Examples of a Real-Time Temporal Logic Specification. Dans B. Tim Denvir, W. T. Harwood, M. I. Jackson, et M. J. Wray, éditeurs, *The Analysis of Concurrent Systems*, volume 207 de *Lecture Notes in Computer Science*, pages 231–251. Springer-Verlag, 1983.

[KMMP93] Yonit Kesten, Zohar Manna, Hugh McGuire, et Amir Pnueli. A Decision Algorithm for Full Propositional Temporal Logic. Dans Costas Courcoubetis, éditeur, *Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93)*, Juin-Juillet 1993, volume 697 de *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, Juin 1993.

[Koz83] Dexter Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27, pages 333–354, Elsevier Science, 1983.

[KP95] Orna Kupferman et Amir Pnueli. *Once* and *For All*. Dans *Proceedings of the 10th Annual Symposium on Logic in Computer Science (LICS'95)*, Juin 1995, pages 25–35. IEEE Comp. Soc. Press, Juin 1995.

[KPR98] Yonit Kesten, Amir Pnueli, et Li-on Raviv. Algorithmic Verification of Linear Temporal Logic Specifications. Dans Kim Gulstrand Larsen, Sven Skyum, et Glynn Swinkel, éditeurs, *Proceedings of the 25th International Colloquium on*

Automata, Languages and Programming (ICALP'98), Juillet 1998, volume 1443 de *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Juillet 1998.

[KPV01] Orna Kupferman, Nir Piterman, et Moshe Y. Vardi. Extended Temporal Logic Revisited. Dans Kim Gulstrand Larsen et Mogens Nielsen, éditeurs, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR 2001)*, Août 2001, volume 2154 de *Lecture Notes in Computer Science*, pages 519–535. Springer-Verlag, Août 2001.

[Kri63] Saul A. Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16, pages 83–94, Philosophical Society of Finland, 1963.

[Lam80] Leslie Lamport. "Sometimes" is sometimes "Not Never". Dans *Conference Record of the 7th ACM Symposium on Principles of Programming Languages (POPL'80)*, Janvier 1980, pages 174–185. ACM Press, Janvier 1980.

[Lam83] Leslie Lamport. What good is Temporal Logic. Dans R. E. A. Mason, éditeur, *Proceedings of the 9th IFIP World Computer Congress (IFIPWCS'83)*, Septembre 1983, IFIP Conference Proceedings, pages 657–668. North Holland/IFIP, Septembre 1983.

[Lar94] François Laroussinie. *Logiques temporelles avec passé pour la spécification et la vérification des systèmes réactifs*. Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 1994.

[LMS01] François Laroussinie, Nicolas Markey, et Philippe Schnoebelen. Model Checking CTL^+ and FCTL is hard. Dans Furio Honsell et Marino Miculan, éditeurs, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS 2001)*, Avril 2001, volume 2030 de *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, Avril 2001.

[LMS02a] François Laroussinie, Nicolas Markey, et Philippe Schnoebelen. On Model Checking Durational Kripke Structures (Extended Abstract). Dans Mogens Nielsen et Uffe Engberg, éditeurs, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS 2002)*, Avril 2002, volume 2303 de *Lecture Notes in Computer Science*, pages 264–279. Springer-Verlag, Avril 2002.

[LMS02b] François Laroussinie, Nicolas Markey, et Philippe Schnoebelen. Temporal Logic with Forgettable Past. Dans *Proceedings of the 17th Annual Symposium on Logic in Computer Science (LICS 2002)*, Juillet 2002, pages 383–392. IEEE Comp. Soc. Press, Juillet 2002.

[LP85] Orna Lichtenstein et Amir Pnueli. Checking that Finite State Concurrent Programs Satisfy their Linear Specifiction. Dans *Conference Record of the 12th ACM Symposium on Principles of Programming Languages (POPL'85)*, Janvier 1985, pages 97–107. ACM Press, Janvier 1985.

[LPZ85] Orna Lichtenstein, Amir Pnueli, et Lenore D. Zuck. The Glory of the Past. Dans Rohit Parikh, éditeur, *Proceedings of the Conference on Logics of Programs*, Juin 1985, volume 193 de *Lecture Notes in Computer Science*, pages 413–424. Springer-Verlag, Juin 1985.

[LS95] François Laroussinie et Philippe Schnoebelen. A Hierarchy of Temporal Logics with Past. *Theoretical Computer Science*, 148(2), pages 303–324, Elsevier Science, Septembre 1995.

[LS00] François Laroussinie et Philippe Schnoebelen. Specification in CTL+Past for Verification in CTL. *Information and Computation*, 156(1-2), pages 236–263, Academic Press, Janvier 2000.

[LS02] Martin Lange et Colin Stirling. Model Checking Fixed Point Logic with Chop. Dans Mogens Nielsen et Uffe Engberg, éditeurs, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS 2002)*, Avril 2002, volume 2303 de *Lecture Notes in Computer Science*, pages 250–263. Springer-Verlag, Avril 2002.

[LST00] François Laroussinie, Philippe Schnoebelen, et Mathieu Turuani. On the Expressive and Complexity of Quantitative Branching-Time Temporal Logics. Dans Gastn H. Gonnet, Daniel Panario, et Alfredo Viola, éditeurs, *Proceedings of the 4th Latin American Symposium on Theoretical INformatics (LATIN 2000)*, Avril 2000, volume 1776 de *Lecture Notes in Computer Science*, pages 437–446. Springer-Verlag, Avril 2000.

[Mar02] Nicolas Markey. Past is for Free : On the Complexity of Verifying Linear Temporal Properties with Past. Dans Uwe Nestmann et Prakash Panangaden, éditeurs, *Proceedings of the 9th International Workshop on Expressiveness in Concurrency (EXPRESS 2002)*, Août 2002, volume 68(2) de *Electronic Notes in Theoretical Computer Science*, pages 89–106. Elsevier Science, Août 2002.

[McM93a] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[McM93b] Kenneth L. McMillan. *Symbolic Model Checking – an Approach to the State Explosion Problem*. Thèse de doctorat, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1993.

[McN66] Robert McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *Information and Control*, 9(6), pages 521–530, Academic Press, Décembre 1966.

[McN71] Robert McNaughton. *Counter Free Automata*. MIT Press, 1971.

[MCO99] Mars Climate Orbiter Mishap Inversigation Board : Phase I Report. `ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf`, Novembre 1999.

[MH84] Satoru Miyano et Takeshi Hayashi. Alternating Finite Automata on ω-words. *Theoretical Computer Science*, 32, pages 321–330, Elsevier Science, 1984.

[Miy91] Satoru Miyano. Δ_2^P-complete lexicographically first maximal subgraph problems. *Theoretical Computer Science*, 88(1), pages 33–57, Elsevier Science, 1991.

[MMSKR94] Louise E. Moser, P. Michael Melliar-Smith, George Kutty, et Y. Srinivas Ramakrishna. Completeness and Soundness of Axiomatizations for Temporal Logics without Next. *Fundamenta Informaticae*, 21(4), pages 257–305, IOS Press, Octobre 1994.

[MO99] Markus Müller-Olm. A Modal Fixpoint Logic with Chop. Dans Christoph Meinel et Sophie Tison, éditeurs, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, Mars 1999, volume 1563 de *Lecture Notes in Computer Science*, pages 510–520. Springer-Verlag, Mars 1999.

[MP88] Zohar Manna et Amir Pnueli. The Anchored Version of the Temporal Framework. Dans Jaco W. de Bakker, Willem-Paul de Roever, et Grzegorz Rozenberg, éditeurs, *Proceedings of REX School/Workshop 1988*, Mai-Juin 1988, volume 354 de *Lecture Notes in Computer Science*, pages 201–284. Springer-Verlag, Mai 1988.

[MP90a] Oded Maler et Amir Pnueli. Tight Bounds on the Complexity of Cascaded Decomposition of Automata. Dans *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'90)*, Octobre 1990, pages 672–682. IEEE Comp. Soc. Press, Octobre 1990.

[MP90b] Zohar Manna et Amir Pnueli. A Hierarchy of Temporal Properties. Dans *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing (PODC'90)*, Août 1990, pages 377–410. ACM Press, Août 1990.

[MP91] Zohar Manna et Amir Pnueli. Completing the Temporal Picture. *Theoretical Computer Science*, 83(1), pages 91–130, Elsevier Science, 1991.

[MP94] Oded Maler et Amir Pnueli. On the Cascade Decomposition of Automata, its Complexity and its Application to Logic. `http://www-verimag.imag.fr/PEOPLE/Oded.Maler/Papers/decomp.ps`, 1994.

[MP95] Zohar Manna et Amir Pnueli. *Temporal Verification of Reactive Systems : Safety*. Springer-Verlag, 1995.

[MS72] Albert R. Meyer et Larry J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. Dans *Proceedings of the 13th Annual Symposium on Foundations of Computer Science (FOCS'72)*, Octobre 1972, pages 125–129. IEEE Comp. Soc. Press, Octobre 1972.

[MS00] Zohar Manna et Henny Sipma. Alternating the Temporal Picture for Safety. Dans Ugo Montanari, José D. P. Rolim, et Emo Welzl, éditeurs, *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*, Juillet 2000, volume 1853 de *Lecture Notes in Computer Science*, pages 429–450. Springer-Verlag, Juillet 2000. À paraître.

[MS02] Paolo Maggi et Riccardo Sisto. Using SPIN to verify security properties of cryptographic protocols. Dans Dragan Bosnacki et Stefan Leue, éditeurs, *Proceedings of the 9th International SPIN Workshop (SPIN 2002)*, Avril 2002, volume 2318 de *Lecture Notes in Computer Science*, pages 187–204. Springer-Verlag, Avril 2002.

[Pap84] Christos H. Papadimitriou. On the Complexity of Unique Solutions. *Journal of the ACM*, 31(2), pages 392–400, ACM Press, Avril 1984.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. Dans *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, Octobre-Novembre 1977, pages 46–57. IEEE Comp. Soc. Press, Octobre 1977.

[Pnu81] Amir Pnueli. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science*, 13, pages 45–60, Elsevier Science, 1981.

[Pri57] Arthur N. Prior. *Time and Modality*. Clarendon Press, 1957.

[Pri67] Arthur N. Prior. *Past, Present and Future*. Clarendon Press, 1967.

[Pri69] Arthur N. Prior. *Papers on Time and Tense*. Clarendon Press, 1969.

[PW97] Doron A. Peled et Thomas Wilke. Stutter-Invariant Temporal Properties are Expressible Without the Next-time Operator. *Information Processing Letters*, 63(5), pages 243–246, Elsevier Science, 1997.

[PZ93] Amir Pnueli et Lenore D. Zuck. In and Out of Temporal Logic. Dans *Proceedings of the 8th Annual Symposium on Logic in Computer Science (LICS'93)*, Juin 1993, pages 124–135. IEEE Comp. Soc. Press, Juin 1993.

[QS82] Jean-Pierre Queille et Joseph Sifakis. Specification and verification of concurrent systems in CESAR. Dans Marangiola Dezani-Ciancaglini et Ugo Montanari, éditeurs, *Proceedings of the 5th International Symposium on Programming (SOP'82)*, Avril 1982, volume 137 de *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, Avril 1982.

[RGL01] Muriel Roger et Jean Goubault-Larrecq. Log Auditing through Model-Checking. Dans *Proceedings of the 14th IEEE Computer Science Foundations Workshop (CSFW 2001)*, Juin 2001, pages 220–236. IEEE Comp. Soc. Press, Juin 2001.

[RMD+92] Y. Srinivas Ramakrishna, Louise E. Moser, Laura K. Dillon, P. Michael Melliar-Smith, et George Kutty. An Automata Theoretic Decision Procedure for Propositional Temporal Logic with Since and Until. *Fundamenta Informaticae*, 17(3), pages 272–282, IOS Press, Novembre 1992.

[RP86] Roni Rosner et Amir Pnueli. A Choppy Logic. Dans *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, Juin 1986, pages 306–313. IEEE Comp. Soc. Press, Juin 1986.

[RS00] Alexander Moshe Rabinovich et Philippe Schnoebelen. BTL_2 and expressive completeness for $ECTL^+$. Rapport technique, Laboratoire Spécification et Vérification, ENS Cachan, Cachan, France, Octobre 2000.

[SB01] Viktor Schuppan et Armin Biere. A Simple Verification of the Tree Identify Protocol with SMV. Dans Savi Maharaj, Judi Romijn, et Carron Shankland, éditeurs, *Proceedings of the IEEE 1394 (FireWire) Workshop*. Departement of Computing Science and Mathematics, University of Stirling, 2001.

[SBB+99] Philippe Schnoebelen, Béatrice Bérard, Michel Bidoit, François Laroussinie, et Antoine Petit. *Vérification de logiciels : techniques et outils du model-checking*. Vuibert, 1999.

[SC85] A. Prasad Sistla et Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3), pages 733–749, ACM Press, Juillet 1985.

[Sch01] Philippe Schnoebelen. *Spécification et Vérification des Systèmes Concurrents*. Thèse d'habilitation, Université Paris VII, Octobre 2001.

[Sch02] Philippe Schnoebelen. The Complexity of Temporal Logic Model Checking. Dans *Proceedings of the 4th Workshop on Advances in Modal Logic (AIML 2002)*, Septembre-Octobre 2002, pages 481–517, Septembre 2002.

[Sch03] Philippe Schnoebelen. Oracle Circuits for Branching-Time Model Checking. Dans *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, Juin-Juillet 2003, Lecture Notes in Computer Science. Springer-Verlag, Juin 2003. À paraître.

[SG98] Abraham Silberschatz et Peter Baer Galvin. *Operating Systems Concepts*. Addison-Wesley, fifth édition, 1998.

[Sis83] A. Prasad Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems*. Thèse de doctorat, Harvard University, Cambridge, Massachussets, USA, 1983.

[SR99] Pierre-Yves Schobbens et Jean-François Raskin. The Logic of "Initially" and "Next". *Information Processing Letters*, 69(5), pages 221–225, Elsevier Science, 1999.

[Sti92] Colin Stirling. Modal and Temporal Logic. Dans Samson Abramsky, Dov M. Gabbay, et Thomas S. E. Maibaum, éditeurs, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.

[Sto74] Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. Thèse de doctorat, Massachusets Institute of Technology, Cambridge, Massasuchets, USA, 1974.

[Sto76] Larry J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1), pages 1–22, Elsevier Science, 1976.

[Sto87] Larry J. Stockmeyer. Classifying the Computational Complexity of Problems. *Journal of Symbolic Logic*, 52(1), pages 1–43, Association for Symbolic Logic, Mars 1987.

[SVW87] A. Prasad Sistla, Moshe Y. Vardi, et Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logics. *Theoretical Computer Science*, 49, pages 217–237, Elsevier Science, 1987.

[Tho96] Wolfgang Thomas. Languages, Automata and Logic. Rapport technique, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität, Kiel, Germany, Mai 1996.

[TW96] Denis Thérien et Thomas Wilke. Temporal Logic and Semidirect Products : an Effective Characterization of the Until Hierarchy. Dans *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, Octobre 1996, pages 256–263. IEEE Comp. Soc. Press, Octobre 1996.

[TW98] Denis Thérien et Thomas Wilke. Over Words, Two Variables Are as Powerful as One Quantifier Alternation. Dans *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC'98)*, Mai 1998, pages 234–240. ACM Press, Mai 1998.

[TW02] Denis Thérien et Thomas Wilke. Nesting Until and Since in Linear Temporal Logic. Dans Helmut Alt et Afonso Ferreira, éditeurs, *Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, Mars 2002, volume 2285 de *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, Mars 2002.

[Var95] Moshe Y. Vardi. Alternating Automata and Program Verification. Dans Jan van Leeuwen, éditeur, *Computer Science Today : Recent Trends and Developments*, volume 1000 de *Lecture Notes in Computer Science*, pages 471–485. Springer-Verlag, 1995.

[Var96] Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. Dans Faron Moller et Graham M. Birtwistle, éditeurs, *Logics for Concurrency - Structure versus Automata : Proceedings 8th Banff Higher Order Workshop (Banff'94)*, Août-Septembre 1994, volume 1043 de *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

[Var98a] Moshe Y. Vardi. Linear versus Branching Time : a Complexity-Theoretic Approach. Dans *Proceedings of the 13th Annual Symposium on Logic in Computer Science (LICS'98)*, Juin 1998, pages 394–405. IEEE Comp. Soc. Press, Juin 1998.

[Var98b] Moshe Y. Vardi. Reasoning about the Past with Two-Way Automata. Dans Kim Gulstrand Larsen, Sven Skyum, et Glynn Swinkel, éditeurs, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, Juillet 1998, volume 1443 de *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Juillet 1998.

[Var98c] Moshe Y. Vardi. Sometimes and Not Never Revisited : on Branching versus Linear Time. Dans Davide Sangiorgi et Robert de Simone, éditeurs, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, Septembre 1998, volume 1466 de *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, Septembre 1998.

[Var01] Moshe Y. Vardi. Branching versus Linear Time : the Final Showdown. Dans Tiziana Margaria et Wang Yi, éditeurs, *Proceedings of the 7th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2001)*, Avril 2001, volume 2031 de *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Avril 2001.

[vEB83] Peter van Emde Boas. Dominoes are Forever. Dans Lutz Priese, éditeur, *Report on the 1st GTI-workshop in Foundations of Theoretical Computer Science*, 1982, pages 75–95. Fachbereich Mathematik-Informatik, Universität Gesamthochschule, 1983.

[vEB97] Peter van Emde Boas. The convenience of tiling. Dans Andrea Sorbi, éditeur, *Complexity, Logic and Recursion Theory*, volume 187 de *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

[VS85] Moshe Y. Vardi et Larry J. Stockmeyer. Improved Upper and Lower Bounds for Modal Logics of Programs. Dans *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing (STOC'85)*, Mai 1985, pages 240–251. ACM Press, Mai 1985.

[VW86] Moshe Y. Vardi et Pierre Wolper. An Automata-Theoretic Approach to Automatic Program Verification. Dans *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, Juin 1986, pages 332–344. IEEE Comp. Soc. Press, Juin 1986.

[VW94] Moshe Y. Vardi et Pierre Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115(1), pages 1–37, Academic Press, Novembre 1994.

[Wag87] Klaus W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51, pages 53–80, Elsevier Science, 1987.

[Wil99a] Thomas Wilke. Classifying Discrete Temporal Properties. Dans Christoph Meinel et Sophie Tison, éditeurs, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, Mars 1999, volume 1563 de *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, Mars 1999.

[Wil99b] Thomas Wilke. CTL^+ is exponentially more succinct than CTL. Dans C. Pandu Rangan, Vijayshankar Raman, et R. Ramanujam, éditeurs, *Proceedings of the 19th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, Décembre 1999, volume 1738 de *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, Décembre 1999.

[Wra76] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1), pages 23–33, Elsevier Science, 1976.

[WVS83] Pierre Wolper, Moshe Y. Vardi, et A. Prasad Sistla. Reasoning about Infinite Computation Paths. Dans *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, Novembre 1983, pages 185–194. IEEE Comp. Soc. Press, Novembre 1983.

Résumé

Ce travail s'inscrit dans le cadre de la vérification formelle de programmes : le model checking est une technique qui permet de s'assurer qu'une propriété, exprimée en logique temporelle, est vérifiée par le modèle d'un système.

Cette thèse étudie plusieurs logiques temporelles, du point de vue de l'expressivité et de la complexité. Nous étudions trois grands types de logiques temporelles :

– concernant la logique temporelle du temps linéaire, nous prouvons que l'ajout de modalités du passé permet de simplifier l'écriture des formules sans augmenter la complexité des problèmes de model checking. Nous étudions également l'impact de l'ajout de la modalité Now ;
– concernant la logique temporelle du temps arborescent, nous établissons la complexité du model checking pour plusieurs extensions classiques de CTL ;
– enfin, nous montrons qu'il est possible, sous certaines conditions, de vérifier efficacement des propriétés quantitatives sur la durée séparant deux évènements.

Summary

This thesis deals with formal verification of reactive systems: model checking is a technique for verifying that the model of a system under study fulfills a given property, expressed in some temporal logic.

This PhD compares the expressive power and the complexity of verification problems for three sorts of temporal logics:

- concerning linear-time temporal logic, we prove that adding past-time modalities simplifies the specification of properties, and does not increase the complexity of model checking problems. We also study the impact of adding another operator, called Now;

- as regards branching-time temporal logics, we establish the complexity of model checking problems for several classical extensions of CTL;

- we also prove that temporal properties with quantitative constraints can be efficiently model checked under certain restrictions.

Mots-clefs

– spécification et vérification formelle de systèmes réactifs
– model checking
– logiques temporelles
– expressivité
– complexité