INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Logiques temporelles basées sur actions pour la vérification des systèmes asynchrones

Radu Mateescu

N° 5032

Décembre 2003

THÈME 1

Rapport de recherche

ISSN 0249-6399

Logiques temporelles basées sur actions pour la vérification des systèmes asynchrones

Radu Mateescu

Thème 1 — Réseaux et systèmes
Projet VASY

Rapport de recherche n° 5032 — Décembre 2003 — 39 pages

Résumé : La vérification formelle est indispensable pour assurer la fiabilité des applications complexes et critiques comme les protocoles de télécommunication et les systèmes répartis. L'approche basée sur les modèles (*model checking*) consiste à traduire l'application vers un modèle système de transitions étiquetées, sur lequel les propriétés de bon fonctionnement, exprimées comme formules de logique temporelle, sont vérifiées au moyen d'algorithmes spécifiques. Ce rapport présente de manière unifiée les logiques temporelles basées sur actions qui sont actuellement les plus utilisées dans le contexte des systèmes parallèles asynchrones comportant du non-déterminisme. Les différentes logiques traitées (modales, arborescentes, régulières et de point fixe) sont illustrées à travers des exemples de propriétés typiques des systèmes parallèles asynchrones (sûreté, vivacité, équité) et sont comparées suivant l'expressivité, la facilité d'utilisation et l'efficacité des algorithmes de vérification associés.

Mots-clés : bisimulation, logique temporelle, non-déterminisme, programme parallèle, système de transitions étiquetées, spécification, vérification

Une version abrégée de ce rapport est également disponible comme “Logiques basées sur actions pour la vérification des systèmes asynchrones”, *Technique et science informatiques*, 2003. Les travaux de recherche sous-jacents ont été financés par le projet européen IST-2001-32360 “ArchWare”.

* Radu.Mateescu@inria.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Action-Based Temporal Logics for the Verification of Asynchronous Systems

Abstract: Formal verification is essential for ensuring the reliability of complex, critical applications such as communication protocols and distributed systems. The model checking approach consists in translating the application into a labelled transition system model, on which the correctness properties, expressed as temporal logic formulas, are verified by means of specific algorithms. This report presents in a unified manner the action based temporal logics that are currently the most used in the framework of parallel asynchronous systems involving nondeterminism. The different logics described (modal, branching-time, regular, and fixed point based) are illustrated through various examples of typical properties of parallel asynchronous systems (safety, liveness, fairness) and are compared with respect to expressiveness, user-friendliness, and efficiency of the corresponding verification algorithms.

Key-words: bisimulation, temporal logic, nondeterminism, parallel program, labelled transition system, specification, verification

1 Introduction

Durant la dernière décennie, les méthodes formelles sont devenues une composante indispensable intégrée au processus de conception des applications complexes à caractère critique, telles que les protocoles de télécommunication, les systèmes répartis et les architectures multiprocesseurs. En effet, la complexité de ces applications rendant leur analyse manuelle extrêmement ardue, leur fiabilité ne saurait être garantie autrement qu'en employant des méthodes de spécification et vérification formelle, assistées par des outils informatiques performants. Une technique de vérification offrant un bon compromis coût-performances est la vérification énumérative (*model-checking*) [CGP00]. Cette technique consiste à traduire l'application – préalablement décrite dans un langage approprié – vers un *modèle* (généralement un graphe d'états), sur lequel les propriétés de correction attendues sont vérifiées au moyen d'algorithmes spécifiques. Bien que limitée aux applications ayant un nombre fini d'états, la vérification énumérative est particulièrement utile dans les premières phases du processus de conception, permettant une détection rapide et économique des erreurs.

Pour décrire les propriétés de bon fonctionnement des applications, les *logiques temporelles* sont des formalismes bien adaptés, notamment par leur capacité à exprimer l'ordonnancement des actions (événements) dans le temps. Les descriptions de propriétés en logique temporelle présentent deux qualités importantes [MP90] : elles sont *abstraites*, c'est-à-dire indépendantes des détails d'implémentation de l'application (par exemple, la propriété qu'au plus un processus peut utiliser une ressource à un instant donné doit être satisfaite par tous les protocoles d'exclusion mutuelle) et elles sont *modulaires*, c'est-à-dire modifiables localement (le rajout, le changement ou la suppression d'une propriété ne remet pas en cause la validité des autres).

Un large éventail de logiques temporelles ont été définies et étudiées dans la littérature. Schématiquement, elles peuvent être classifiées suivant deux critères : logiques *linéaires* (comme LTL [MP92] et le μ-calcul linéaire [Var88]) ou *arborescentes* (comme CTL [CES86] et CTL* [EH86]), selon qu'elles expriment des propriétés sur les séquences ou les arbres d'exécution du modèle ; et logiques basées sur *états* (comme LTL, CTL et CTL*) ou sur *actions* (comme TLA [Lam94], ACTL [NV90] et le μ-calcul modal [Koz83]), selon qu'elles font référence aux états ou aux actions du modèle. Les logiques arborescentes sont mieux adaptées pour les applications parallèles asynchrones comportant du non-déterminisme, puisqu'elles permettent de prendre en compte le branchement de l'exécution. Cette caractéristique est illustrée par l'exemple classique du distributeur de boissons, qui accepte une pièce de monnaie (action *money*) et ensuite doit délivrer du café (action *coffee*) ou du thé (action *tea*) suivant le choix de l'utilisateur. Considérons deux modélisations du distributeur en CCS [Mil89] : $D_1 = money.(coffee.0 + tea.0)$ et $D_2 = money.coffee.0 + money.tea.0$. Bien que les termes D_1 et D_2 présentent les mêmes séquences d'exécution $\{money.coffee, money.tea\}$ – ce qui les rend équivalents du point de vue d'une logique temporelle linéaire – leur interaction avec l'utilisateur est radicalement différente : D_1 permet à l'utilisateur de choisir la boisson après avoir inséré une pièce, alors que D_2 la choisit lui-même de façon non déterministe lors de l'insertion de la pièce. Cette différence essentielle de comportement ne peut être décelée que par une logique arborescente, capable d'exprimer le choix non déterministe de D_2 sur l'action *money*.

Lors du choix d'une logique temporelle, plusieurs aspects doivent être considérés, parmi lesquels : l'*expressivité* (la capacité de la logique à exprimer les classes de propriétés intéressantes, telles que la sûreté, la vivacité ou l'équité) ; la *complexité d'évaluation* (la complexité des algorithmes permettant de vérifier qu'un modèle satisfait une propriété) ; et la *facilité d'utilisation* (la capacité à exprimer les propriétés de manière concise et naturelle). L'optimisation de l'un ou l'autre de ces aspects ne pouvant généralement se faire qu'au détriment des autres, le choix doit passer par un compromis judicieux (par exemple, si l'efficacité d'évaluation est l'aspect le plus important, alors l'expressivité de la logique devra être limitée). En outre, en raison de la diversité des logiques temporelles existantes et des résultats présents dans la littérature, il n'est pas toujours aisé de réunir les éléments pertinents pour choisir une logique temporelle adaptée à un certain contexte.

Dans cet article, nous présentons de manière unifiée plusieurs logiques temporelles arborescentes basées sur actions, adaptées à la vérification énumérative des programmes parallèles décrits au moyen des algèbres de processus (comme Ccs [Mil89], Csp [BHR84] et Acp [BK84]) ou des langages de description formelle normalisés qui en sont inspirés (comme Lotos [ISO89] et E-Lotos [ISO01]). Dans ce contexte, nous n'abordons pas les logiques linéaires (comme Ltl et ses extensions Etl [Wol83] et Dltl [HT99] avec des opérateurs réguliers), qui n'autorisent pas la prise en compte du non-déterminisme, ni les logiques basées sur états (comme Ctl et Ctl*), qui ne sont pas adéquates avec les relations d'équivalence (*bisimulations*) utilisées pour les algèbres de processus.

Nous introduisons les différentes logiques considérées par ordre croissant d'expressivité : logiques modales telles que Hml [HM85] (section 3), logiques arborescentes telles que Actl (section 4), logiques régulières telles que Pdl [FL79] (section 5) et logiques de point fixe telles que le μ-calcul modal [Koz83] (section 6). Par souci d'uniformité, nous définissons la sémantique de ces logiques de manière dénotationnelle en interprétant les formules sur des modèles et, dans certains cas, nous étendons leur syntaxe par rapport à leur définition originelle afin de souligner leurs aspects communs et de comparer plus facilement leur expressivité. En particulier, bien que Pdl ait été définie originellement comme logique dynamique pour prouver la correction des programmes séquentiels réguliers dont la syntaxe fait partie intégrante de celle des formules logiques (approche *exogène*), nous avons adopté ici l'approche *endogène* consistant à interpréter Pdl comme logique temporelle sur les modèles des programmes parallèles en vue de la vérification énumérative [Bra92].

Pour chaque logique considérée, nous illustrons les différentes classes de propriétés qu'elle peut exprimer, nous précisons son adéquation avec les relations d'équivalence entre modèles et nous indiquons également ses diverses extensions et variantes proposées dans la littérature. Nous décrivons également les principaux types d'algorithmes de vérification existants (sections 7.1 et 7.2) et nous offrons une image synthétique de l'expressivité et de la complexité d'évaluation des différentes logiques présentées (section 7.3). Finalement, nous présentons brièvement quelques outils de vérification dédiés à certaines de ces logiques (section 7.4) et nous indiquons des références supplémentaires permettant d'approfondir le sujet (section 8).

2 Systèmes de transitions étiquetées

Les logiques temporelles basées sur actions que nous allons considérer sont interprétées sur des Systèmes de Transitions Etiquetées (STE), qui sont les modèles naturellement associés aux langages de type algèbre de processus. Formellement, un STE est un quadruplet $M = (S, A, T, s_0)$, où S est l'ensemble d'états, A est l'ensemble d'actions (contenant l'action *invisible* τ), $T \subseteq S \times A \times S$ est la relation de transition et $s_0 \in S$ est l'état initial. Une transition $(s_1, a, s_2) \in T$, notée également $s_1 \xrightarrow{a} s_2$, signifie que le système peut passer de l'état s_1 à l'état s_2 en exécutant l'action a (la transition est appelée visible si $a \neq \tau$ et invisible sinon). Cette notation est étendue aux séquences de transitions : $s_1 \xrightarrow{l} s_2$ (où $l \subseteq A^*$) signifie qu'à partir de l'état s_1 le système peut effectuer une séquence de transitions qui mène à s_2 et dont les actions concaténées forment un mot du langage l. Tous les états de S sont supposés être accessibles à partir de l'état initial s_0 par des séquences de transitions de T.

Pour comparer les STE, différentes relations d'équivalence ont été proposées. Nous rappelons ici les définitions de deux d'entre elles : l'équivalence *forte* [Par81] et l'équivalence *observationnelle* [Mil89]. Soit deux STE $M_i = (S_i, A_i, T_i, s_{0i})$, où $i \in \{1, 2\}$. L'équivalence forte est la plus grande relation $\sim \ \subseteq S_1 \times S_2$ telle que pour tout couple d'états $s_1 \in S_1$ et $s_2 \in S_2$, $s_1 \sim s_2$ si et seulement si à chaque transition $s_1 \xrightarrow{a} s_1'$ (resp. $s_2 \xrightarrow{a} s_2'$) correspond une transition $s_2 \xrightarrow{a} s_2'$ (resp. $s_1 \xrightarrow{a} s_1'$) telle que $s_1' \sim s_2'$. Intuitivement, deux états sont fortement équivalents si chacun peut "simuler" les transitions de l'autre. M_1 est fortement équivalent à M_2 (notation $M_1 \sim M_2$) si et seulement si $s_{01} \sim s_{02}$. L'équivalence observationnelle est définie de manière similaire, excepté qu'elle fait abstraction des séquences de transitions invisibles entrelacées avec les transitions visibles. Il s'agit de la plus grande relation $\approx \ \subseteq S_1 \times S_2$ telle que pour tout couple d'états $s_1 \in S_1$ et $s_2 \in S_2$, $s_1 \approx s_2$ si et seulement si à chaque transition $s_1 \xrightarrow{a} s_1'$ (resp. $s_2 \xrightarrow{a} s_2'$) correspond un état $s_2' \in S_2$ (resp. $s_1' \in S_1$) tel que $s_1' \approx s_2'$ et soit $s_2 \xrightarrow{\tau^*} s_2'$ si $a = \tau$, soit $s_2 \xrightarrow{\tau^*.a.\tau^*} s_2'$ si $a \neq \tau$ (resp. soit $s_1 \xrightarrow{\tau^*} s_1'$ si $a = \tau$, soit $s_1 \xrightarrow{\tau^*.a.\tau^*} s_1'$ si $a \neq \tau$). M_1 est observationnellement équivalent à M_2 (notation $M_1 \approx M_2$) si et seulement si $s_{01} \approx s_{02}$.

Une logique temporelle L est *adéquate* par rapport à une relation d'équivalence R si deux STE équivalents modulo R satisfont le même ensemble de propriétés spécifiées en L. La notion d'adéquation est importante en pratique, car elle permet d'améliorer les performances de la vérification : si la propriété est exprimée dans une logique L adéquate par rapport à R, alors au lieu d'être vérifiée directement sur un STE, elle peut être vérifiée – avec le même résultat – sur le STE minimisé modulo R. Lorsque R est une équivalence faible, telle que l'équivalence observationnelle, de fortes réductions en taille du STE (parfois, plusieurs ordres de grandeur) peuvent être obtenues après minimisation modulo R.

Pour illustrer l'interprétation des propriétés sur un STE, nous considérons le protocole d'exclusion mutuelle de Peterson [Pet83], qui présente l'avantage d'être suffisamment simple pour que son modèle STE puisse être inspecté visuellement, tout en étant suffisamment complexe pour faire l'objet de propriétés temporelles intéressantes. La figure 1 montre une spécification du protocole en LOTOS [ISO89]. Le protocole implémente l'exclusion mutuelle entre deux processus parallèles P_0 et P_1 (modélisés par des appels du processus P) qui sont en compétition pour accéder à une ressource partagée. L'utilisation de la ressource par le processus P_j (section critique) est modélisée par les actions `BCS`j et `ECS`j, marquant le début

```
specification Peterson [NCS0, BCS0, ECS0, NCS1, BCS1, ECS1,
                        F0, F1, T] : noexit
library BOOLEAN, NATURAL endlib

type COMMAND is
   sorts Cmd
   opns  Read (*!  constructor *), Write (*!  constructor *) :  -> Cmd
endtype

behaviour
  hide F0, F1, T in
  (  P [NCS0, BCS0, ECS0, F0, F1, T] (0)
     |||
     P [NCS1, BCS1, ECS1, F1, F0, T] (1)  )
  |[F0, F1, T]|
  (  Flag [F0] (false) ||| Flag [F1] (false) ||| Turn [T] (0)  )
 where
  process P [NCS, BCS, ECS, Fj, Fi, T] (j:Nat) :  noexit :=
     NCS;  Fj !Write !true;  T !Write !j;
     (  Fi !Read ?v_fi:Bool [not (v_fi)];
           BCS;  ECS;  Fj !Write !false;
              P [NCS, BCS, ECS, Fj, Fi, T] (j)
        []
        T !Read ?v_t:Nat [v_t == ((j + 1) mod 2)];
           BCS;  ECS;  Fj !Write !false;
              P [NCS, BCS, ECS, Fj, Fi, T] (j)  )
  endproc

  process Flag [F] (v_f:Bool) :  noexit :=
     F !Read !v_f;
        Flag [F] (v_f)
     []
     F !Write ?v_f2:Bool;
        Flag [F] (v_f2)
  endproc

  process Turn [T] (v_t:Nat) :  noexit :=
     T !Read !v_t;
        Turn [T] (v_t)
     []
     T !Write ?v_t2:Nat;
        Turn [T] (v_t2)
  endproc
endspec
```

Figure 1: Spécification en LOTOS du protocole d'exclusion mutuelle de Peterson

et la fin de la section critique. L'exécution du processus P_j sans utiliser la ressource (section non critique) est modélisée par l'action `NCS`j. L'exclusion mutuelle est mise en œuvre au moyen de trois variables partagées : les variables booléennes $Flag_0$ et $Flag_1$ (modélisées par des appels du processus Flag), initialisées à faux, et la variable entière *Turn* (modélisée par l'appel du processus Turn), initialisée à 0. Les deux processus P_0 et P_1 peuvent lire et écrire les trois variables par des actions de synchronisation sur les portes F0, F1 et T, accompagnées de l'envoi de commandes Read et Write appropriées. Lorsque le processus P_j désire accéder à la ressource, il commence par mettre la variable $Flag_j$ à vrai et la variable *Turn* à j, et ensuite il attend jusqu'à ce que la variable $Flag_i$ passe à faux ou la variable *Turn* passe à i (où $i = (j+1)\ mod\ 2$). A ce moment, le processus P_j peut exécuter sa section critique, après laquelle il libère la ressource en mettant la variable $Flag_j$ à faux et continue son fonctionnement. Les actions de lecture et d'écriture des variables partagées $Flag_0$, $Flag_1$ et *Turn*, qui ne sont pas pertinentes du point de vue des propriétés du protocole, ont été cachées (rendues invisibles) au moyen de l'opérateur "hide" de Lotos.

La figure 2 montre le modèle Ste correspondant à cette spécification Lotos, minimisé modulo l'équivalence observationnelle. Le Ste comporte 25 états et 46 transitions, l'état initial ayant l'index 0 (les actions invisibles, notées "i" en Lotos, ont été renommées en "tau" pour plus de lisibilité). Il a été généré automatiquement, minimisé et mis en forme graphique au moyen de la boîte à outils Cadp [GLM02] pour l'ingénierie des protocoles et des systèmes distribués. La plupart des propriétés temporelles illustrées par la suite seront interprétées sur ce Ste.

3 Logiques modales

Les logiques *modales* sont parmi les logiques les plus simples qui permettent de spécifier des propriétés arborescentes utiles sur les modèles Ste. Elles peuvent être vues comme des extensions des logiques propositionnelles avec certains opérateurs (*modalités*) exprimant l'exécution possible ou nécessaire de certaines actions.

3.1 Définition de HML

Hml (*Hennessy-Milner Logic*) [HM85] peut être considérée comme le représentant standard des logiques modales pour les Ste. La syntaxe et la sémantique de la variante de Hml que nous considérons ici sont définies dans le tableau 1. La logique contient deux types d'entités : les formules sur actions (notées α) et les formules sur états (notées φ), qui permettent respectivement de caractériser des sous-ensembles d'actions et d'états d'un Ste $M = (S, A, T, s_0)$. Les formules sur actions sont construites au-dessus du vocabulaire des actions $a \in A$ au moyen des opérateurs booléens standard[1]. Des opérateurs booléens dérivés sont définis de la manière habituelle : $\alpha_1 \Rightarrow \alpha_2 = \neg\alpha_1 \vee \alpha_2$ et $\alpha_1 \Leftrightarrow \alpha_2 = (\alpha_1 \Rightarrow \alpha_2) \wedge (\alpha_2 \Rightarrow \alpha_1)$. Les formules sur états sont construites au moyen des opérateurs booléens et des opérateurs modaux de *possibilité* ($\langle\alpha\rangle\,\varphi$) et de *nécessité* ($[\alpha]\,\varphi$).

[1] Cela constitue une extension par rapport à la définition originelle de Hml [HM85], où les formules sur actions étaient simplement des actions a.

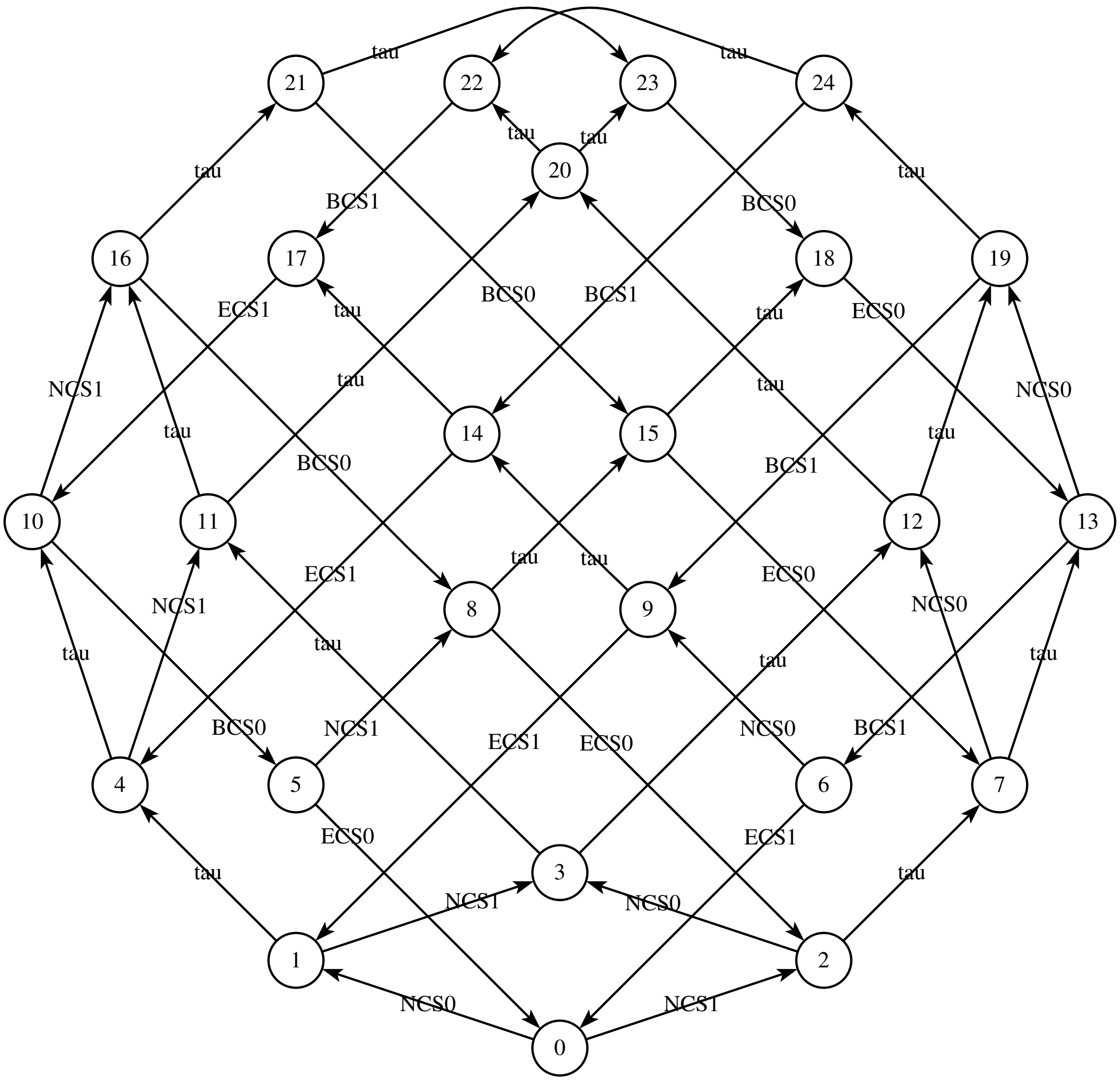

Figure 2: Modèle Ste du protocole de Peterson

Formules sur actions :

$$
\begin{array}{lll}
\alpha ::= a & & [\![a]\!] = \{a\} \\
\quad \mid \ \mathsf{ff} & & [\![\mathsf{ff}]\!] = \emptyset \\
\quad \mid \ \mathsf{tt} & & [\![\mathsf{tt}]\!] = A \\
\quad \mid \ \neg\alpha_1 & & [\![\neg\alpha_1]\!] = A \setminus [\![\alpha_1]\!] \\
\quad \mid \ \alpha_1 \vee \alpha_2 & & [\![\alpha_1 \vee \alpha_2]\!] = [\![\alpha_1]\!] \cup [\![\alpha_2]\!] \\
\quad \mid \ \alpha_1 \wedge \alpha_2 & & [\![\alpha_1 \wedge \alpha_2]\!] = [\![\alpha_1]\!] \cap [\![\alpha_2]\!]
\end{array}
$$

Formules sur états :

$$
\begin{array}{lll}
\varphi ::= \mathsf{ff} & & [\![\mathsf{ff}]\!] = \emptyset \\
\quad \mid \ \mathsf{tt} & & [\![\mathsf{tt}]\!] = S \\
\quad \mid \ \neg\varphi_1 & & [\![\neg\varphi_1]\!] = S \setminus [\![\varphi_1]\!] \\
\quad \mid \ \varphi_1 \vee \varphi_2 & & [\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
\quad \mid \ \varphi_1 \wedge \varphi_2 & & [\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\
\quad \mid \ \langle\alpha\rangle\,\varphi_1 & & [\![\langle\alpha\rangle\,\varphi_1]\!] = \{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\} \\
\quad \mid \ [\alpha]\,\varphi_1 & & [\![[\alpha]\,\varphi_1]\!] = \{s \in S \mid \forall s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \Rightarrow s' \in [\![\varphi_1]\!]\}
\end{array}
$$

Table 1: Syntaxe et sémantique de Hml

La sémantique d'une formule α sur un Ste $M = (S, A, T, s_0)$ est définie par l'interprétation $[\![\alpha]\!] \subseteq A$, qui dénote le sous-ensemble d'actions du Ste satisfaisant α. Les opérateurs booléens ont la sémantique habituelle définie en termes d'opérations ensemblistes. La sémantique d'une formule φ est définie par l'interprétation $[\![\varphi]\!] \subseteq S$, qui dénote le sous-ensemble d'états du Ste satisfaisant φ. La modalité de possibilité $\langle\alpha\rangle\,\varphi$ dénote les états ayant au moins une transition successeur étiquetée par une action satisfaisant α et menant à un état satisfaisant φ. La modalité de nécessité $[\alpha]\,\varphi$ dénote les états dont toutes les transitions successeurs étiquetées par des actions satisfaisant α mènent à des états satisfaisant φ. Les deux opérateurs modaux sont duaux ($[\alpha]\,\varphi = \neg\,\langle\alpha\rangle\,\neg\varphi$) et monotones (si $\varphi_1 \Rightarrow \varphi_2$ alors $\langle\alpha\rangle\,\varphi_1 \Rightarrow \langle\alpha\rangle\,\varphi_2$ et $[\alpha]\,\varphi_1 \Rightarrow [\alpha]\,\varphi_2$).

Un état s satisfait une formule φ si et seulement si $s \in [\![\varphi]\!]$. Un modèle Ste $M = (S, A, T, s_0)$ satisfait φ si et seulement si son état initial satisfait φ.

3.2 Spécification de propriétés

Hml permet d'exprimer des propriétés simples sur la succession et le branchement des transitions dans un Ste. La plus simple propriété concerne l'absence de blocages, c'est-à-dire d'états n'ayant aucun successeur (également appelés *états puits*). Ces états peuvent être caractérisés en Hml par la formule suivante :

$$[\mathsf{tt}]\,\mathsf{ff}$$

spécifiant que toutes les transitions issues de l'état courant doivent mener à des états satisfaisant faux (comme de tels états n'existent pas, les transitions correspondantes n'existent pas non plus). Il est aisé de s'assurer que le STE montré en figure 2 ne comporte pas d'états puits et donc qu'il est exempt de blocages[2].

Des propriétés plus complexes peuvent être exprimées en imbriquant les opérateurs modaux et en les combinant avec les connecteurs booléens. Ainsi, la formule HML suivante permet de distinguer les deux implémentations D_1 et D_2 du distributeur de boissons illustrées en section 1 :

$$[money]\,(\langle coffee\rangle\,\mathsf{tt} \wedge \langle tea\rangle\,\mathsf{tt})$$

Cette formule spécifie qu'après une action *money*, les deux actions *coffee* et *tea* sont possibles. Elle est satisfaite par D_1, mais pas par D_2, qui – après une action *money* – empêche d'exécuter soit l'action *coffee*, soit l'action *tea*. HML est donc capable de caractériser le branchement et le non-déterminisme ; de ce point de vue, elle est mieux adaptée aux systèmes asynchrones qu'une logique temporelle linéaire.

3.3 Expressivité et adéquation

Bien qu'elle autorise la description de propriétés utiles, HML présente une expressivité limitée. Cela vient du fait que les opérateurs modaux permettent de caractériser uniquement des arbres d'exécution de profondeur bornée, égale au niveau maximal d'imbrication des modalités dans la formule. Un exemple de propriété impossible à exprimer en HML est l'existence d'une séquence de longueur quelconque menant à une transition étiquetée par a (ce qui nécessiterait une formule HML de la forme $\langle\mathsf{tt}\rangle\,\langle\mathsf{tt}\rangle \cdots \langle\mathsf{tt}\rangle\,\langle a\rangle\,\mathsf{tt}$ contenant un nombre inconnu de modalités imbriquées).

Cependant, HML présente un intérêt théorique, par son adéquation avec l'équivalence forte des STE [HM85] : deux STE sont fortement équivalents si et seulement s'ils satisfont le même ensemble de formules HML. En outre, étant donné deux STE qui ne le sont pas (comme les deux implémentations D_1 et D_2 du distributeur de boissons), il est toujours possible de construire une formule de HML qui est satisfaite par un STE et pas par l'autre (comme la formule indiquée en section 3.2). Cela permet de construire automatiquement des contre-exemples (représentés comme formules HML) pour la comparaison de STE modulo l'équivalence forte [Cle90a].

3.4 Autres logiques modales

Différentes variantes et extensions de HML ont été proposées dans le but d'assurer l'adéquation avec des relations d'équivalence faibles entre STE, comme l'équivalence observationnelle ou l'équivalence de branchement [vGW89]. La variante *observationnelle* de HML [HM85, Sti01] – adéquate avec l'équivalence observationnelle – remplace les modalités de possibilité et de nécessité de HML, appelées modalités *fortes*, par des modalités *faibles*,

[2]Pour exprimer des propriétés *invariantes* (c'est-à-dire satisfaites par tous les états du STE) telles que l'absence de blocage, la définition de la satisfaction donnée en section 3.1 n'est pas suffisante dans le contexte de HML, cette logique ne permettant pas d'écrire une formule qui – évaluée sur l'état initial du STE – assure que tous les états accessibles satisfont une propriété. Cet inconvénient disparaîtra pour les logiques présentées aux sections 4, 5 et 6.

notées respectivement $\langle\langle\alpha\rangle\rangle\,\varphi$ et $[[\alpha]]\,\varphi$, où les formules α sont interprétées sur $A \setminus \{\tau\}$ (c'est-à-dire qu'elles doivent dénoter des actions visibles). L'opérateur $\langle\langle\alpha\rangle\rangle\,\varphi$ caractérise les états s_1 à partir desquels il existe une séquence $s_1 \stackrel{\tau^*.a.\tau^*}{\longrightarrow} s_2$ telle que l'action a satisfait α et l'état s_2 satisfait φ. L'opérateur $[[\alpha]]\,\varphi$ est défini de manière duale : $[[\alpha]]\,\varphi = \neg\,\langle\langle\alpha\rangle\rangle\,\neg\varphi$. Deux opérateurs supplémentaires $\langle\langle\rangle\rangle\,\varphi$ et $[[]]\,\varphi$ expriment respectivement l'accessibilité possible et nécessaire d'états satisfaisant φ après des séquences de zéro ou plusieurs transitions invisibles.

La logique HmlU [dNV90] – adéquate avec l'équivalence de branchement – étend Hml avec un opérateur $\varphi_1\,\langle\alpha\rangle\,\varphi_2$, appelé *Until*. Cet opérateur caractérise les états s_1 à partir desquels il existe une séquence $s_1 \stackrel{\tau^*.a}{\longrightarrow} s_2$ telle que l'action a satisfait α, l'état s_2 satisfait φ_2 et tous les états intermédiaires (y compris s_1 lui-même) satisfont φ_1. L'opérateur supplémentaire $\varphi_1\,\langle\varepsilon\rangle\,\varphi_2$ permet de spécifier l'accessibilité d'un état satisfaisant φ_2 après une séquence de zéro ou plusieurs transitions invisibles qui passent par des états satisfaisant φ_1. HmlU est strictement plus expressive que la variante observationnelle de Hml (ce qui dérive du fait que l'équivalence de branchement est plus forte que l'équivalence observationnelle), puisqu'elle permet d'exprimer les modalités faibles de possibilité comme $\langle\langle\alpha\rangle\rangle\,\varphi = \mathsf{tt}\,\langle\alpha\rangle\,\langle\varepsilon\rangle\,\varphi$ et $\langle\langle\rangle\rangle\,\varphi = \mathsf{tt}\,\langle\varepsilon\rangle\,\varphi$.

4 Logiques arborescentes

Les logiques *arborescentes* permettent de spécifier des propriétés sur les arbres d'exécution (de profondeur non bornée) issus des états d'un Ste. Elles peuvent être vues comme des extensions des logiques modales avec des opérateurs temporels exprimant l'accessibilité potentielle ou inévitable de certains états et/ou actions.

4.1 Définition d'ACTL

Actl (*Action Computation Tree Logic*) [NV90] peut être considérée comme le représentant standard des logiques arborescentes pour les Ste. La syntaxe et la sémantique d'Actl sont définies dans le tableau 2. La logique contient trois types d'entités : les formules sur actions (notées α), les formules sur chemins (notées ψ) et les formules sur états (notées φ), qui permettent respectivement de caractériser des sous-ensembles d'actions, de chemins et d'états d'un Ste $M = (S, A, T, s_0)$. Les formules sur actions sont identiques à celles de la logique Hml (voir le tableau 1), excepté le fait qu'elles doivent dénoter uniquement des actions visibles. Les formules sur chemins sont construites au moyen des opérateurs de succession *neXt* ($\mathsf{X}_\alpha\varphi$ et $\mathsf{X}_\tau\varphi$) et des opérateurs temporels *Until* ($\varphi_{1_\alpha}\mathsf{U}\ \varphi_2$ et $\varphi_{1_{\alpha_1}}\mathsf{U}_{\alpha_2}\varphi_2$). Les formules sur états sont construites au moyen des opérateurs booléens standard et des quantificateurs existentiel ($\mathsf{E}\psi$) et universel ($\mathsf{A}\psi$) appliqués aux formules sur chemins.

Etant donné un Ste $M = (S, A, T, s_0)$ et un état $s \in S$, nous notons $Path(s) = \{s \stackrel{a_1}{\to} s_1 \stackrel{a_2}{\to} s_2 \cdots\}$ l'ensemble des séquences d'exécution *maximales* (c'est-à-dire les séquences infinies ou se terminant par un état puits) issues de s. L'ensemble de toutes les séquences maximales issues des états du Ste est noté *Path*. La sémantique d'une formule ψ sur M est définie par l'interprétation $[\![\psi]\!] \subseteq Path$, qui dénote le sous-ensemble de séquences satisfaisant ψ. Les opérateurs $\mathsf{X}_\alpha\varphi$ et $\mathsf{X}_\tau\varphi$ dénotent les séquences dont la première transition

Formules sur chemins :

$$
\begin{array}{lrl}
\psi ::= \mathsf{X}_\alpha \varphi & [\![\mathsf{X}_\alpha \varphi]\!] = & \{s_1 \xrightarrow{a_1} s_2 \cdots \mid a_1 \in [\![\alpha]\!] \wedge s_2 \in [\![\varphi]\!]\} \\
\quad \mid \ \mathsf{X}_\tau \varphi & [\![\mathsf{X}_\tau \varphi]\!] = & \{s_1 \xrightarrow{\tau} s_2 \cdots \mid s_2 \in [\![\varphi]\!]\} \\
\quad \mid \ \varphi_1{}_\alpha\mathsf{U}\ \varphi_2 & [\![\varphi_1{}_\alpha\mathsf{U}\ \varphi_2]\!] = & \{s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} s_i \cdots \mid i \geq 1 \wedge s_i \in [\![\varphi_2]\!] \wedge \\
 & & \forall j \in [1, i-1].a_j \in [\![\alpha \vee \tau]\!] \wedge s_j \in [\![\varphi_1]\!]\} \\
\quad \mid \ \varphi_1{}_{\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2 & [\![\varphi_1{}_{\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2]\!] = & \{s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} s_i \cdots \mid i \geq 2 \wedge s_i \in [\![\varphi_2]\!] \wedge \\
 & & a_{i-1} \in [\![\alpha_2]\!] \wedge s_{i-1} \in [\![\varphi_1]\!] \wedge \\
 & & \forall j \in [1, i-2].a_j \in [\![\alpha_1 \vee \tau]\!] \wedge s_j \in [\![\varphi_1]\!]\}
\end{array}
$$

Formules sur états :

$$
\begin{array}{lrl}
\varphi ::= \mathsf{ff} & [\![\mathsf{ff}]\!] = & \emptyset \\
\quad \mid \ \mathsf{tt} & [\![\mathsf{tt}]\!] = & S \\
\quad \mid \ \neg\varphi_1 & [\![\neg\varphi_1]\!] = & S \setminus [\![\varphi_1]\!] \\
\quad \mid \ \varphi_1 \vee \varphi_2 & [\![\varphi_1 \vee \varphi_2]\!] = & [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
\quad \mid \ \varphi_1 \wedge \varphi_2 & [\![\varphi_1 \wedge \varphi_2]\!] = & [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\
\quad \mid \ \mathsf{E}\psi & [\![\mathsf{E}\psi]\!] = & \{s \in S \mid \exists p \in \mathit{Path}(s).p \in [\![\psi]\!]\} \\
\quad \mid \ \mathsf{A}\psi & [\![\mathsf{A}\psi]\!] = & \{s \in S \mid \forall p \in \mathit{Path}(s).p \in [\![\psi]\!]\}
\end{array}
$$

Table 2: Syntaxe et sémantique d'ACTL

est étiquetée par une action satisfaisant α (respectivement par une action invisible) et mène à un état satisfaisant φ. L'opérateur $\varphi_1{}_\alpha\mathsf{U}\ \varphi_2$ dénote les séquences qui mènent – après zéro ou plusieurs transitions étiquetées par des actions satisfaisant α (ou par des actions invisibles) et passant par des états intermédiaires satisfaisant φ_1 – à un état satisfaisant φ_2. L'opérateur $\varphi_1{}_{\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2$ dénote les séquences qui mènent – après zéro ou plusieurs transitions étiquetées par des actions satisfaisant α_1 (ou par des actions invisibles) et passant par des états intermédiaires satisfaisant φ_1 – à une transition étiquetée par une action satisfaisant α_2 et menant à un état satisfaisant φ_2. La sémantique d'une formule φ est définie par l'interprétation $[\![\varphi]\!] \subseteq S$, qui dénote le sous-ensemble d'états satisfaisant φ. Les opérateurs $\mathsf{E}\psi$ et $\mathsf{A}\psi$ dénotent les états à partir desquels certaines séquences (respectivement, toutes les séquences) satisfont ψ.

Plusieurs opérateurs dérivés peuvent être définis. Les modalités de HML s'expriment en ACTL comme $\langle\alpha\rangle\,\varphi = \mathsf{EX}_\alpha\varphi$ et $[\alpha]\,\varphi = \neg\,\langle\alpha\rangle\,\neg\varphi$. Les opérateurs de *potentialité* ($\mathsf{EF}_\alpha\varphi$) et d'*invariance* ($\mathsf{AG}_\alpha\varphi$) – exprimant que certaines séquences (respectivement, toutes les séquences) de transitions étiquetées par des actions satisfaisant α mènent à des états satisfaisant φ – sont définis respectivement comme $\mathsf{EF}_\alpha\varphi = \mathsf{E}(\mathsf{tt}_\alpha\mathsf{U}\ \varphi)$ et $\mathsf{AG}_\alpha\varphi = \neg\mathsf{EF}_\alpha\neg\varphi$. Par souci de clarté, nous utiliserons ces opérateurs dérivés autant que possible dans les formules ACTL.

4.2 Spécification de propriétés

ACTL permet d'exprimer des propriétés temporelles utiles sur des STE, notamment des propriétés de *sûreté* et de *vivacité*, qui intuitivement stipulent que "rien de mal n'arrivera", respectivement "quelque chose de bien arrivera" pendant l'exécution du programme. Une propriété de sûreté du protocole de Peterson est l'exclusion mutuelle, exprimée en ACTL par la formule suivante (où $i, j \in \{0, 1\}$ et $i \neq j$) :

$$\neg \mathsf{EF}_{\mathsf{tt}} \, \langle \mathtt{BCS}i \rangle \, \mathsf{EF}_{\neg \mathtt{ECS}i} \, \langle \mathtt{BCS}j \rangle \, \mathsf{tt}$$

Cette formule interdit les séquences qui comportent une entrée en section critique du processus i, suivie – avant la sortie de section critique du processus i – d'une entrée en section critique du processus j. Elle peut être également exprimée de la manière suivante, utilisant la dualité des modalités et des opérateurs EF et AG :

$$\mathsf{AG}_{\mathsf{tt}} \, [\mathtt{BCS}i] \, \mathsf{AG}_{\neg \mathtt{ECS}i} \, [\mathtt{BCS}j] \, \mathsf{ff}$$

Ce schéma de formule (imbrication d'opérateurs AG et de modalités de nécessité, terminée par ff) est caractéristique pour les propriétés de sûreté exprimables en ACTL. Il est aisé de s'assurer que le STE montré en figure 2 respecte l'exclusion mutuelle, en examinant les séquences de transitions issues de son état initial.

Une propriété de vivacité désirable pour le protocole de Peterson est le fait qu'aucun processus n'est bloqué au cours de l'exécution (absence de blocage local), c'est-à-dire que toutes les actions visibles du processus i sont en permanence *potentiellement* exécutables. Cela peut être exprimé en ACTL par la formule suivante (où $i \in \{0, 1\}$) :

$$\mathsf{AG}_{\mathsf{tt}}(\mathsf{EF}_{\mathsf{tt}} \, \langle \mathtt{NCS}i \rangle \, \mathsf{tt} \wedge \mathsf{EF}_{\mathsf{tt}} \, \langle \mathtt{BCS}i \rangle \, \mathsf{tt} \wedge \mathsf{EF}_{\mathsf{tt}} \, \langle \mathtt{ECS}i \rangle \, \mathsf{tt})$$

Cette formule, satisfaite par le STE de la figure 2, exprime qu'à partir de chaque état accessible, il existe des séquences menant aux actions visibles du processus i. Une propriété de vivacité plus forte concerne l'accès *inévitable* de chaque processus à la ressource partagée, exprimée par la formule ACTL suivante (où $i \in \{0, 1\}$) :

$$\mathsf{AG}_{\mathsf{tt}} \, [\mathtt{NCS}i] \, \mathsf{A}(\mathsf{tt}_{\mathsf{tt}} \mathsf{U}_{\mathtt{BCS}i} \mathsf{tt})$$

Cette formule impose que toutes les séquences suivant l'exécution de la section non critique du processus i conduisent, après un nombre fini de transitions, à l'entrée du processus i en section critique. Malheureusement, elle n'est pas satisfaite par le STE de la figure 2, puisque celui-ci présente des séquences d'exécution cycliques telles que $s_0 \xrightarrow{\mathtt{NCS0}} s_1 \xrightarrow{\mathtt{NCS1}} s_3 \xrightarrow{\mathtt{i}} s_{12} \xrightarrow{\mathtt{i}} s_{19} \xrightarrow{\mathtt{BCS1}} s_9 \xrightarrow{\mathtt{ECS1}} s_1 \cdots$ qui dénotent la monopolisation indéfinie de la ressource par le processus 1 au détriment du processus 0. Cette situation de *famine* signifie que le protocole de Peterson n'est pas équitable envers l'accessibilité des processus à la ressource partagée. D'autres protocoles d'exclusion mutuelle plus élaborés, tels que le protocole de Szymanski [Szy88], garantissent qu'aucun processus ne peut "doubler" l'autre lors de l'accès à la ressource plus d'un nombre fini de fois (*bounded overtaking*).

Ce problème peut être évité en supposant l'existence d'un ordonnanceur chargé de planifier l'exécution équitable des deux processus, en interdisant les séquences d'exécution inéquitables, c'est-à-dire les séquences infinies au cours desquelles un certain processus n'accède pas à la ressource. En pratique, il s'avère que pour exprimer l'accessibilité inévitable d'une action, la présence d'un ordonnanceur peut être modélisée implicitement dans les propriétés temporelles elles-mêmes. Ainsi, la propriété de vivacité précédente – devenue une propriété d'*équité* sous l'hypothèse d'un ordonnanceur – peut être exprimée en ACTL au moyen de la formule suivante :

$$\mathsf{AG}_{\mathsf{tt}}\,[\mathtt{NCS}i]\,\mathsf{AG}_{\neg\mathtt{BCS}i}\mathsf{EF}_{\mathsf{tt}}\,\langle\mathtt{BCS}i\rangle\,\mathsf{tt}$$

Cette formule spécifie qu'après une section non critique du processus i, tant que l'entrée du processus i en section critique n'est pas atteinte, il est possible de l'atteindre après un nombre fini de transitions (ce qui évite les séquences d'exécution inéquitables). Elle est satisfaite par le STE de la figure 2, puisqu'en imposant que les circuits du STE soient parcourus un nombre fini de fois, on atteint toujours l'entrée d'un processus en section critique. Le schéma de formule $\mathsf{AG}_{\neg a}\mathsf{EF}_{\mathsf{tt}}\,\langle a\rangle\,\mathsf{tt}$, exprimant l'accessibilité d'une action a sous l'hypothèse d'un ordonnanceur qui élimine les séquences inéquitables par rapport à a, constitue l'équivalent en termes d'actions de l'opérateur d'*inévitabilité équitable* défini pour une variante de la logique CTL [QS83].

4.3 Expressivité et adéquation

ACTL est strictement plus expressive que HML, puisqu'elle permet de décrire les modalités $\langle\alpha\rangle\,\varphi$ et $[\alpha]\,\varphi$ (voir section 4.1), ainsi que des propriétés inexprimables en HML, comme l'accessibilité d'une action a après une séquence de transitions quelconques, décrite en ACTL par la formule $\mathsf{EF}_{\mathsf{tt}}\,\langle a\rangle\,\mathsf{tt}$. ACTL est également plus expressive que la variante observationnelle de HML, puisque la modalité faible de possibilité est exprimée en ACTL comme $\langle\langle\alpha\rangle\rangle\,\varphi = \mathsf{EF}_{\mathsf{ff}}\,\langle\alpha\rangle\,\mathsf{EF}_{\mathsf{ff}}\varphi$. Enfin, la logique HMLU peut être considérée, elle aussi, comme un fragment d'ACTL, car les deux opérateurs *Until* peuvent être traduits en ACTL respectivement comme $\varphi_1\,\langle\alpha\rangle\,\varphi_2 = \mathsf{E}(\varphi_{1\,\mathsf{ff}}\mathsf{U}_\alpha\varphi_2)$ et $\varphi_1\,\langle\varepsilon\rangle\,\varphi_2 = \mathsf{E}(\varphi_{1\,\mathsf{ff}}\mathsf{U}\,\varphi_2)$ [NV90].

La logique ACTL complète n'est adéquate avec aucune relation d'équivalence faible entre STE. Par contre, le fragment d'ACTL obtenu en éliminant les opérateurs de succession $\mathsf{X}_\alpha\varphi$ et $\mathsf{X}_\tau\varphi$, appelé ACTL – X, est adéquat avec l'équivalence de branchement [NV90]. En outre, ACTL – X est strictement plus expressif que HMLU, puisque les traductions des opérateurs *Until* ci-dessus n'utilisent pas les opérateurs de succession. Cela justifie l'intérêt pratique d'ACTL, puisqu'il s'agit – du moins, historiquement – d'une des premières logiques arborescentes basées sur actions qui autorise l'expression de propriétés de sûreté et de vivacité relativement élaborées, tout en ayant un fragment significatif adéquat avec une équivalence faible entre STE.

4.4 Autres logiques arborescentes

Diverses extensions d'ACTL ont été proposées dans le but d'augmenter son expressivité. ACTL* [NV90] constitue une extension naturelle d'ACTL (semblable à CTL*, l'extension de CTL), obtenue en autorisant l'application des opérateurs X et U aux formules sur chemins au lieu des formules sur états uniquement. Par conséquent, les quantificateurs E et A peuvent être appliqués en ACTL* à des formules sur chemins plus complexes, comportant non pas un seul opérateur temporel U, F ou G comme en ACTL, mais aussi des imbrications quelconques de ces opérateurs. Cela permet de spécifier des propriétés d'équité plus fortes, inexprimables en ACTL, comme l'occurrence infiniment fréquente d'une action a sur chaque séquence d'exécution, décrite en ACTL* au moyen de la formule $\mathsf{AG}_{\mathsf{tt}}(\mathsf{tt}_{\mathsf{tt}}\mathsf{U}_a\mathsf{tt})$.

La logique μ-ACTL [FGR92] étend ACTL avec un opérateur de plus petit point fixe $\mu X.\varphi$ semblable à celui du μ-calcul modal (voir la section 6), représentant la plus petite solution de l'équation fonctionnelle $X = \varphi$, où X est une variable propositionnelle qui dénote un prédicat sur états (X est censée apparaître dans φ). Cet opérateur permet, entre autres, d'exprimer la répétition d'une certaine séquence d'actions un nombre fini de fois, ce qui rend possible la description de propriétés plus complexes, comme l'existence d'une séquence d'actions de la forme $(a^*.b)^*.c$, décrite en μ-ACTL par la formule $\mu X. \langle c \rangle\, \mathsf{tt} \vee \mathsf{E}(\mathsf{tt}_a \mathsf{U}_b X)$. En fait, il s'avère que μ-ACTL est tout aussi expressive que le μ-calcul modal, puisque les deux logiques peuvent se traduire l'une en termes de l'autre [FGR92].

5 Logiques régulières

Les logiques *régulières* permettent d'exprimer de manière concise et intuitive des propriétés temporelles arborescentes sur des STE. Elles peuvent être vues comme des extensions des logiques modales avec des opérateurs réguliers spécifiant la succession, le choix et la répétition de certaines séquences d'actions.

5.1 Définition de PDL

PDL (*Propositional Dynamic Logic*) [FL79] peut être considérée comme le représentant standard des logiques régulières pour les STE. La syntaxe et la sémantique de la variante de PDL que nous considérons ici sont définies dans le tableau 3. La logique contient trois types d'entités : les formules sur actions (notées α), les formules régulières (notées β) et les formules sur états (notées φ), qui permettent respectivement de caractériser des sous-ensembles d'actions, de séquences et d'états d'un STE $M = (S, A, T, s_0)$. Les formules sur actions[3] sont identiques à celles de la logique HML (voir le tableau 1). Les formules régulières sont construites au moyen des opérateurs de test ($\varphi?$), de concaténation ($\beta_1; \beta_2$), de choix ($\beta_1 \cup \beta_2$) et de répétition (β^*). Les formules sur états sont construites au moyen des opérateurs booléens et des opérateurs modaux de possibilité ($\langle \beta \rangle\, \varphi$) et de nécessité ($[\beta]\, \varphi$), semblables à ceux de HML, mais comportant des formules régulières à l'intérieur des modalités.

Etant donné un STE $M = (S, A, T, s_0)$, la sémantique d'une formule sur actions α est définie par l'interprétation $[\![\alpha]\!] \subseteq A$, qui dénote le sous-ensemble d'actions du STE satisfaisant α. La sémantique d'une formule β est définie par l'interprétation $\|\beta\| \subseteq S \times S$, qui dénote les couples d'états qui sont aux extrémités des séquences de transitions satisfaisant β. Les formules régulières atomiques α dénotent les séquences comportant une seule transition

[3]Cela constitue une extension par rapport à la définition originelle de PDL [FL79], où les formules sur actions, appelées *instructions*, étaient simplement des actions a.

Formules régulières :	
$\beta ::= \alpha$	$\\|\alpha\\| = \{(s, s') \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!]\}$
$\mid \varphi?$	$\\|\varphi?\\| = \{(s, s) \mid s \in [\![\varphi]\!]\}$
$\mid \beta_1; \beta_2$	$\\|\beta_1; \beta_2\\| = \\|\beta_1\\| \circ \\|\beta_2\\|$
$\mid \beta_1 \cup \beta_2$	$\\|\beta_1 \cup \beta_2\\| = \\|\beta_1\\| \cup \\|\beta_2\\|$
$\mid \beta_1^*$	$\\|\beta_1^*\\| = \\|\beta_1\\|^*$
Formules sur états :	
$\varphi ::= \mathsf{ff}$	$[\![\mathsf{ff}]\!] = \emptyset$
$\mid \mathsf{tt}$	$[\![\mathsf{tt}]\!] = S$
$\mid \neg\varphi_1$	$[\![\neg\varphi_1]\!] = S \setminus [\![\varphi_1]\!]$
$\mid \varphi_1 \vee \varphi_2$	$[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$
$\mid \varphi_1 \wedge \varphi_2$	$[\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$
$\mid \langle\beta\rangle\, \varphi_1$	$[\![\langle\beta\rangle\, \varphi_1]\!] = \{s \in S \mid \exists s' \in S.(s, s') \in \\|\beta\\| \wedge s' \in [\![\varphi_1]\!]\}$
$\mid [\beta]\, \varphi_1$	$[\![[\beta]\, \varphi_1]\!] = \{s \in S \mid \forall s' \in S.(s, s') \in \\|\beta\\| \Rightarrow s' \in [\![\varphi_1]\!]\}$

Table 3: Syntaxe et sémantique de PDL

étiquetée par une action satisfaisant α. L'opérateur de test $\varphi?$ dénote les séquences vides constituées d'états satisfaisant φ. Les opérateurs réguliers $\beta_1; \beta_2$, $\beta_1 \cup \beta_2$ et β^* dénotent respectivement la concaténation, le choix et la répétition (zéro ou plusieurs fois) de séquences de transitions satisfaisant β_1, β_2 et β ($\circ$, $\cup$ et $*$ dénotent la composition, l'union et la fermeture transitive et réflexive d'une relation). La sémantique d'une formule φ est définie par l'interprétation $[\![\varphi]\!] \subseteq S$, qui dénote le sous-ensemble d'états du STE satisfaisant φ. Les opérateurs booléens ont la sémantique habituelle définie en termes d'opérations ensemblistes. La modalité de possibilité $\langle\beta\rangle\, \varphi$ dénote les états à partir desquels il existe une séquence de transitions satisfaisant β qui mène à un état satisfaisant φ. La modalité de nécessité $[\beta]\, \varphi$ dénote les états à partir desquels toutes les séquences de transitions satisfaisant β mènent à des états satisfaisant φ. Comme dans le cas de HML, les deux opérateurs modaux sont duaux ($[\beta]\, \varphi = \neg \langle\beta\rangle \neg\varphi$).

Plusieurs opérateurs dérivés peuvent être définis. L'opérateur de séquence vide (ε) et l'opérateur de répétition une ou plusieurs fois (β^+) sont définis respectivement comme $\varepsilon = \mathsf{ff}^*$ et $\beta^+ = \beta; \beta^*$. Des opérateurs semblables aux constructions des langages de programmation séquentiels sont définis comme suit : les opérateurs de *branchement* **if** φ **then** β_1 **else** $\beta_2 = \varphi?; \beta_1 \cup \neg\varphi?; \beta_2$ et **if** φ **then** β = **if** φ **then** β **else** ε et l'opérateur d'*itération* **while** φ **do** $\beta = (\varphi?; \beta)^*; \neg\varphi?$. Par souci de clarté, nous utiliserons ces opérateurs dérivés autant que possible dans les formules PDL.

5.2 Spécification de propriétés

PDL permet de spécifier de manière succincte et intuitive des propriétés utiles sur les STE. Ainsi, la propriété d'exclusion mutuelle du protocole de Peterson, exprimée en ACTL avec deux modalités de nécessité et deux opérateurs AG imbriqués (voir la section 4.2) peut être décrite en PDL avec une seule modalité de nécessité contenant une formule régulière (où $i, j \in \{0, 1\}$ et $i \neq j$) :

$$[\mathsf{tt}^*; \mathtt{BCS}i; (\neg\mathtt{ECS}i)^*; \mathtt{BCS}j]\,\mathsf{ff}$$

Cette formule interdit les séquences d'exécution qui ne respectent pas l'exclusion mutuelle. Nous remarquons que les opérateurs temporels AG de la formule ACTL et leur imbrication avec les modalités de nécessité ont été remplacés respectivement par des opérateurs de répétition et de concaténation dans la formule régulière. Par ailleurs, le schéma de formule $[\beta]\,\mathsf{ff}$ – où la formule régulière β décrit des séquences d'exécution "mauvaises" qui ne doivent pas se produire – est caractéristique pour les propriétés de sûreté exprimables en PDL.

Le gain en clarté des formules PDL par rapport aux formules ACTL devient plus important au fur et à mesure que la complexité des propriétés augmente. Considérons une propriété de sûreté plus élaborée du protocole de Peterson, concernant le séquencement des actions d'entrée et sortie de section critique du processus i : à partir de l'état initial, ces actions doivent s'effectuer en stricte alternance, en commençant par une entrée en section critique. Une première spécification de cette propriété en PDL peut être faite au moyen de la formule suivante (où $i \in \{0, 1\}$) :

$$[(\neg\mathtt{BCS}i)^*; \mathtt{ECS}i]\,\mathsf{ff} \wedge [\mathsf{tt}^*; \mathtt{ECS}i; (\neg\mathtt{BCS}i)^*; \mathtt{ECS}i]\,\mathsf{ff} \wedge [\mathsf{tt}^*; \mathtt{BCS}i; (\neg\mathtt{ECS}i)^*; \mathtt{BCS}i]\,\mathsf{ff}$$

Cette formule – satisfaite par le STE en figure 2 – interdit les séquences d'exécution qui ne respectent pas la propriété d'alternance désirée : celles conduisant à une sortie de section critique avant qu'une entrée en section critique se soit produite et celles comportant deux sorties (ou deux entrées) consécutives de section critique sans qu'aucune entrée (ou aucune sortie) n'ait eu lieu entre temps. Elle peut être transformée – en appliquant quelques identités modales et régulières, comme $[\beta_1]\,\varphi \wedge [\beta_2]\,\varphi = [\beta_1 \cup \beta_2]\,\varphi$, $(\beta_1; \beta_3) \cup (\beta_2; \beta_3) = (\beta_1 \cup \beta_2); \beta_3$ et $\beta = \varepsilon; \beta$ – de manière à utiliser une seule modalité de nécessité :

$$[(\varepsilon \cup (\mathsf{tt}^*; \mathtt{ECS}i)); (\neg\mathtt{BCS}i)^*; \mathtt{ECS}i \cup \mathsf{tt}^*; \mathtt{BCS}i; (\neg\mathtt{ECS}i)^*; \mathtt{BCS}i]\,\mathsf{ff}$$

Une spécification en ACTL de la propriété ci-dessus – que le lecteur intéressé peut s'exercer à écrire – nécessiterait cinq opérateurs AG et cinq modalités de nécessité.

Des propriétés plus complexes peuvent être exprimées en imbriquant les modalités de nécessité et de possibilité. Ainsi, la propriété d'accessibilité équitable de l'entrée en section critique, exprimée en ACTL avec deux modalités de nécessité, un opérateur EF et deux opérateurs AG imbriqués (voir la section 4.2) peut être décrite en PDL – en utilisant l'identité $[\beta_1; \beta_2]\,\varphi = [\beta_1]\,[\beta_2]\,\varphi$ – avec seulement deux modalités imbriquées (où $i \in \{0, 1\}$) :

$$[\mathsf{tt}^*; \mathtt{NCS}i; (\neg\mathtt{BCS}i)^*]\,\langle\mathsf{tt}^*; \mathtt{BCS}i\rangle\,\mathsf{tt}$$

Le schéma de formule caractérisant l'accessibilité équitable d'une action a (voir la section 4.2) peut être décrit en PDL comme $[(\neg a)^*]\,\langle\mathsf{tt}^*; a\rangle\,\mathsf{tt}$.

Les formules PDL illustrées jusqu'ici ne prennent pas en compte les états intermédiaires des séquences d'exécution dénotées par les formules régulières présentes à l'intérieur des modalités. Cependant, PDL permet de décrire des propriétés sur les états intermédiaires grâce à l'opérateur de test (à partir duquel sont définis les opérateurs de branchement et d'itération). Ainsi, les opérateurs d'accessibilité potentielle $\mathsf{E}(\_\mathsf{U}\_)$ d'ACTL peuvent être traduits en PDL de la manière suivante :

$$\begin{aligned}\mathsf{E}(\varphi_{1\,\alpha}\mathsf{U}\ \varphi_2) &= \langle \textbf{while}\ \neg\varphi_2\ \textbf{do}\ (\textbf{if}\ \varphi_1\ \textbf{then}\ \alpha \vee \tau)\rangle\,\mathsf{tt}\\ \mathsf{E}(\varphi_{1\,\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2) &= \langle \textbf{while}\ [\textbf{if}\ \varphi_1\ \textbf{then}\ \alpha_2]\,\neg\varphi_2\ \textbf{do}\ (\textbf{if}\ \varphi_1\ \textbf{then}\ \alpha_1 \vee \tau)\rangle\,\mathsf{tt}\end{aligned}$$

Du fait qu'il permet la description de propriétés arborescentes plus fines, l'opérateur de test apporte un gain d'expressivité par rapport aux autres opérateurs réguliers de PDL : la variante de PDL obtenue en éliminant cet opérateur (appelée *test-free* PDL) est strictement moins expressive que PDL [FL79].

5.3 Expressivité et adéquation

PDL est plus expressive que HML, les modalités $\langle\alpha\rangle\,\varphi$ et $[\alpha]\,\varphi$ étant des cas particuliers des modalités régulières. PDL est aussi plus expressive que la variante observationnelle de HML, puisque la modalité faible de possibilité est exprimée en PDL comme $\langle\langle\alpha\rangle\rangle\,\varphi = \langle\tau^*;\alpha;\tau^*\rangle\,\varphi$. La logique HMLU peut également être vue comme une sous-logique de PDL, car les deux opérateurs *Until* peuvent être traduits en PDL respectivement comme $\varphi_1\,\langle\alpha\rangle\,\varphi_2 = \langle(\textbf{if}\ \varphi_1\ \textbf{then}\ \tau)^*;\textbf{if}\ \varphi_1\ \textbf{then}\ \alpha\rangle\,\varphi_2$ et $\varphi_1\,\langle\varepsilon\rangle\,\varphi_2 = \langle(\textbf{if}\ \varphi_1\ \textbf{then}\ \tau)^*\rangle\,\varphi_2$. En outre, PDL permet de décrire des propriétés inexprimables en ACTL, comme l'existence d'une séquence d'actions de la forme $(a^*.b)^*.c$ (voir la section 4.4), décrite en PDL par la formule $\langle(a^*;b)^*;c\rangle\,\mathsf{tt}$. En revanche, PDL ne permet pas d'exprimer l'absence de séquences infinies [Koz83], propriété qui peut être décrite en ACTL par la formule $\mathsf{A}(\mathsf{tt}_{\mathsf{tt}}\mathsf{U}\ [\mathsf{tt}]\,\mathsf{ff})$. Par conséquent, PDL et ACTL ne sont pas comparables du point de vue de leur expressivité, chaque logique autorisant la description de propriétés qui sont inexprimables dans l'autre.

A notre connaissance, l'adéquation de PDL avec des relations d'équivalence entre STE n'a fait l'objet d'aucune étude dans la littérature. Cependant, nous pouvons argumenter l'adéquation de PDL avec l'équivalence forte en exploitant le fait qu'elle a une expressivité comprise entre celle de HML et celle du μ-calcul modal, deux logiques adéquates avec l'équivalence forte (voir les sections 3.3 et 6.3). En effet, considérons deux STE fortement équivalents $M_1 = (S_1, A_1, T_1, s_{01})$ et $M_2 = (S_2, A_2, T_2, s_{02})$ et supposons qu'il existe une formule φ de PDL satisfaite par M_1 et pas par M_2. Puisque φ peut être traduite vers une formule φ' du μ-calcul modal, il s'ensuivrait que φ' n'est pas satisfaite par les deux STE, ce qui contredirait l'adéquation du μ-calcul modal avec l'équivalence forte. Inversement, supposons que toutes les formules de PDL ont la même interprétation sur les deux STE. Puisque les formules PDL englobent la totalité des formules HML, il s'ensuit – par l'adéquation de HML avec l'équivalence forte – que les deux STE doivent être fortement équivalents. Il serait également possible de concevoir une variante observationnelle de PDL (de manière semblable à celle de HML) en remplaçant les modalités fortes par des modalités faibles $\langle\langle\beta\rangle\rangle\,\varphi$ et $[[\beta]]\,\varphi$, où les formules sur actions α contenues dans β doivent dénoter des actions visibles. Cette variante de PDL aurait une expressivité située entre la variante observationnelle de HML et celle du μ-calcul modal ; en suivant un raisonnement similaire à celui ci-dessus, elle serait donc adéquate avec l'équivalence observationnelle.

5.4 Autres logiques régulières

PDL-Δ (PDL *with looping*) [Str82] étend PDL avec un opérateur de répétition infinie $\Delta\beta$ qui exprime l'existence d'une séquence d'exécution infinie formée de sous-séquences satisfaisant β. Cet opérateur augmente l'expressivité de PDL-Δ par rapport à PDL, puisqu'il autorise la description de propriétés inexprimables en PDL, comme l'absence de séquences infinies, décrite par la formule $\neg\Delta\mathsf{tt}$, ou la répétition indéfinie de deux actions a et b sur une séquence d'exécution, décrite par la formule $\Delta(\mathsf{tt}^*; a; \mathsf{tt}^*; b)$. En fait, PDL-$\Delta$ subsume ACTL, puisqu'elle est capable de décrire les propriétés d'inévitabilité de type $\mathsf{A}(.\mathsf{U}.)$; une traduction d'ACTL vers PDL-Δ est présentée en annexe A.

PDL$_{\mathrm{CF}}$ (*Context-Free* PDL) [HPS83] remplace les formules régulières de PDL par des formules décrites au moyen de grammaires hors-contexte, permettant de raisonner sur la correction des programmes non-réguliers. CDL (*Concurrent Dynamic Logic*) [Pel87] est une extension de PDL dédiée à l'analyse des programmes concurrents, grâce à un nouvel opérateur $\langle\beta_1 \cap \beta_2\rangle\,\varphi$ exprimant le fait que l'exécution concurrente des deux séquences dénotées par β_1 et β_2 peut conduire à un état satisfaisant φ. D'autres extensions de PDL, notamment avec des *nominaux* (propositions atomiques qui sont satisfaites par un seul état du STE) ont été proposées dans [PT91]. Par exemple, un nominal appelé init, caractérisant l'état initial du STE, permettrait de décrire – au moyen de la formule $[\mathsf{tt}^*]\,\langle\mathsf{tt}^*\rangle\,\mathsf{init}$ – le fait que l'état initial est toujours potentiellement accessible (le système à vérifier est *réinitialisable*). Un traitement approfondi de PDL et de ses différentes extensions est disponible dans [HKT00].

6 Logiques de point fixe

Les logiques *de point fixe* sont des logiques très expressives permettant la description de propriétés arborescentes sur les STE. Elles peuvent être vues comme des extensions des logiques modales avec des opérateurs de plus petit et de plus grand point fixe spécifiant des arbres d'exécution finis, respectivement infinis.

6.1 Définition du μ-calcul modal

Le μ-calcul modal (Lμ) [Koz83] peut être considéré comme le représentant standard des logiques de point fixe sur les STE. La syntaxe et la sémantique de la variante du μ-calcul modal que nous considérons ici sont définies dans le tableau 4. La logique contient deux types d'entités : les formules sur actions (notées α) et les formules sur états (notées φ), qui permettent respectivement de caractériser des sous-ensembles d'actions et d'états d'un STE $M = (S, A, T, s_0)$. Les formules sur actions[4] sont identiques à celles de la logique HML (voir le tableau 1). Les formules sur états sont construites sur un ensemble $\mathcal{X}$ de variables propositionnelles (notées X, X_1, X_2, etc.) – dénotant des sous-ensembles d'états du STE – au moyen des opérateurs booléens, modaux ($\langle\alpha\rangle\,\varphi$ et $[\alpha]\,\varphi$), de plus petit ($\mu X.\varphi$) et de plus grand point fixe ($\nu X.\varphi$).

[4] Cela constitue une extension par rapport à la définition originelle du μ-calcul modal [Koz83], où les formules sur actions étaient simplement des actions a.

Formules sur états :		
$\varphi ::=$	ff	$[\![\mathsf{ff}]\!]\,\rho = \emptyset$
$\mid$	tt	$[\![\mathsf{tt}]\!]\,\rho = S$
$\mid$	$\neg\varphi_1$	$[\![\neg\varphi_1]\!]\,\rho = S \setminus [\![\varphi_1]\!]\,\rho$
$\mid$	$\varphi_1 \vee \varphi_2$	$[\![\varphi_1 \vee \varphi_2]\!]\,\rho = [\![\varphi_1]\!]\,\rho \cup [\![\varphi_2]\!]\,\rho$
$\mid$	$\varphi_1 \wedge \varphi_2$	$[\![\varphi_1 \wedge \varphi_2]\!]\,\rho = [\![\varphi_1]\!]\,\rho \cap [\![\varphi_2]\!]\,\rho$
$\mid$	$\langle\alpha\rangle\,\varphi_1$	$[\![\langle\alpha\rangle\,\varphi_1]\!]\,\rho = \{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\,\rho\}$
$\mid$	$[\alpha]\,\varphi_1$	$[\![[\alpha]\,\varphi_1]\!]\,\rho = \{s \in S \mid \forall s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \Rightarrow s' \in [\![\varphi_1]\!]\,\rho\}$
$\mid$	X	$[\![X]\!]\,\rho = \rho(X)$
$\mid$	$\mu X.\varphi_1$	$[\![\mu X.\varphi_1]\!]\,\rho = \bigcap\{U \subseteq S \mid [\![\varphi_1]\!]\,\rho[U/X] \subseteq U\}$
$\mid$	$\nu X.\varphi_1$	$[\![\nu X.\varphi_1]\!]\,\rho = \bigcup\{U \subseteq S \mid U \subseteq [\![\varphi_1]\!]\,\rho[U/X]\}$

Table 4: Syntaxe et sémantique du μ-calcul modal

Les opérateurs de point fixe μ et ν constituent des mécanismes de définition (ou de liaison) pour les variables propositionnelles, semblables aux quantificateurs $\exists$ et $\forall$ en logique du premier ordre. Une formule sur états qui ne contient pas de variables libres est dite *fermée* et une formule n'ayant pas de variable qui soit à la fois libre et liée est dite en *forme normale*. Par la suite, nous allons considérer uniquement des formules sur états fermées et en forme normale. En outre, afin de pouvoir définir la sémantique des opérateurs de point fixe, les formules sur états sont supposées *syntaxiquement monotones* [Koz83] : pour chaque formule $\mu X.\varphi$ et $\nu X.\varphi$, toutes les occurrences de X dans φ sont dans la portée d'un nombre pair de négations.

La sémantique d'une formule α sur un Ste $M = (S, A, T, s_0)$ est définie par l'interprétation $[\![\alpha]\!] \subseteq A$, qui dénote le sous-ensemble d'actions du Ste satisfaisant α. La sémantique d'une formule φ est définie par l'interprétation $[\![\varphi]\!]\,\rho \subseteq S$ (où $\rho : \mathcal{X} \to 2^S$ est un environnement associant des sous-ensembles d'états à toutes les variables propositionnelles libres dans φ) qui dénote le sous-ensemble d'états du Ste satisfaisant φ dans l'environnement ρ. La notation $\rho[U/X]$ représente un environnement identique à ρ, excepté pour la variable X, à laquelle est associé l'ensemble d'états U. Les opérateurs booléens ont la sémantique habituelle définie en termes d'opérations ensemblistes et les opérateurs modaux ont la même sémantique que ceux de Hml. Les opérateurs $\mu X.\varphi$ et $\nu X.\varphi$ dénotent respectivement la plus petite et la plus grande solution de l'équation $X = \varphi$ dans un environnement ρ, c'est-à-dire les points fixes correspondants de la fonctionnelle $\Phi_\rho : 2^S \to 2^S$ définie par $\Phi_\rho(U) = [\![\varphi]\!]\,\rho[U/X]$. La correction de leur interprétation définie dans le tableau 4 est assurée par la monotonie syntaxique des formules φ (ce qui implique la monotonie des fonctionnelles Φ_ρ, c'est-à-dire $U_1 \subseteq U_2 \Rightarrow \Phi_\rho(U_1) \subseteq \Phi_\rho(U_2)$ pour tout environnement ρ et ensembles d'états $U_1, U_2 \subseteq S$) et par le théorème de Tarski [Tar55].

6.2 Spécification de propriétés

Le μ-calcul modal permet d'exprimer des propriétés arborescentes complexes sur les STE. Intuitivement, les opérateurs de plus petit et de plus grand point fixe caractérisent des arbres (ou des séquences) d'exécution de profondeur finie, respectivement infinie. La propriété d'exclusion mutuelle du protocole de Peterson, exprimée précédemment en ACTL et en PDL (voir les sections 4.2 et 5.2) peut également être décrite en μ-calcul modal par la formule suivante (où $i, j \in \{0, 1\}$ et $i \neq j$) :

$$\nu X_1.([\mathtt{BCS}i]\, \nu X_2.([\mathtt{BCS}j]\, \mathsf{ff} \wedge [\neg \mathtt{ECS}i]\, X_2) \wedge [\mathtt{tt}]\, X_1)$$

Cette formule interdit l'entrée en section critique du processus j après toute séquence de transitions comportant une entrée en section critique du processus i, suivie d'actions différentes de la sortie de section critique du processus i. Les deux opérateurs de plus grand point fixe définissant les variables X_1 et X_2 remplacent respectivement les opérateurs d'invariance $\mathsf{AG}_{\mathtt{tt}}$ et $\mathsf{AG}_{\neg \mathtt{ECS}i}$ de la formule ACTL et les opérateurs de répétition $\mathtt{tt}^*$ et $(\neg \mathtt{ECS}i)^*$ de la formule PDL.

La propriété d'accès inévitable à la ressource, exprimée auparavant en ACTL (voir la section 4.2) est décrite en μ-calcul modal par la formule suivante (où $i \in \{0, 1\}$) :

$$\nu X_1.([\mathtt{NCS}i]\, \mu X_2.(\langle \mathtt{tt} \rangle\, \mathtt{tt} \wedge [\neg \mathtt{BCS}i]\, X_2) \wedge [\mathtt{tt}]\, X_1)$$

L'opérateur de plus grand point fixe définissant X_1 remplace l'opérateur $\mathsf{AG}_{\mathtt{tt}}$ de la formule ACTL : il permet de parcourir tous les états accessibles à travers des séquences de transitions quelconques. L'opérateur de plus petit point fixe définissant X_2 remplace l'opérateur d'accessibilité inévitable $\mathsf{A}(\mathtt{tt}_{\mathtt{tt}} \mathsf{U}_{\mathtt{BCS}i} \mathtt{tt})$ de la formule ACTL : il exprime que l'état courant possède au moins un successeur (modalité $\langle \mathtt{tt} \rangle\, \mathtt{tt}$) et que toutes ses transitions successeurs étiquetées par des actions différentes de l'entrée du processus i en section critique mènent inévitablement, après un nombre fini de transitions – ce qui est assuré par la sémantique de l'opérateur μ – à l'entrée du processus i en section critique (modalité $[\neg \mathtt{BCS}i]\, X_2$).

Les deux formules précédentes ne comportent pas de récursion mutuelle entre les opérateurs de point fixe (les sous-formules définissant les variables X_2 sont fermées). Cependant, l'utilisation d'opérateurs de plus petit et de plus grand point fixe mutuellement récursifs permet d'exprimer des propriétés d'équité plus complexes (voir aussi la section 6.3). Par exemple, l'absence de séquences d'exécution infinies au cours desquelles uniquement le processus i accède à la ressource – séquences qui seraient inéquitables par rapport à l'autre processus j – peut être décrite en μ-calcul modal au moyen de la formule suivante (où $i, j \in \{0, 1\}$ et $i \neq j$) :

$$\mu X_1.(\nu X_2.([\mathtt{BCS}i]\, \nu X_3.([\mathtt{ECS}i]\, X_1 \wedge [\neg \mathtt{BCS}j]\, X_3) \wedge [\neg \mathtt{BCS}j]\, X_2))$$

Cette formule impose (par l'opérateur définissant X_1) que toutes les séquences d'exécution comprenant la répétition des entrées et sorties du processus i de section critique, séparées par des actions autres que l'entrée du processus j en section critique (séquences caractérisées par les opérateurs définissant X_2 et X_3) soient finies. Elle n'est pas satisfaite par le STE de la figure 2, qui contient précisément des séquences infinies ayant cette forme (voir la section 4.2). Cette propriété d'équité est inexprimable en ACTL ou en PDL ; par contre, elle peut être décrite en PDL-Δ au moyen de la formule $\neg \Delta((\neg \mathtt{BCS}j)^*; \mathtt{BCS}i; (\neg \mathtt{BCS}j)^*; \mathtt{ECS}i)$.

6.3 Expressivité et adéquation

Le μ-calcul modal est une logique très expressive, qui subsume la plupart des logiques temporelles définies dans la littérature. L'expressivité des formules φ du μ-calcul modal augmente avec leur *alternance* [EL86], qui mesure le degré de récursion mutuelle entre les opérateurs de plus petit et de plus grand point fixe à l'intérieur de φ. Intuitivement, les formules d'alternance 1 n'admettent pas de récursion mutuelle entre les opérateurs μ et ν, celles d'alternance 2 admettent deux opérateurs μ et ν mutuellement récursifs, etc. Il est important de noter que la notion d'alternance est indépendante du degré d'imbrication des opérateurs de point fixe. Par exemple, la formule $\nu X.([\mathtt{NCS}i]\, \mu Y.(\langle \mathtt{BCS}i \rangle\, \mathtt{tt} \vee \langle \mathtt{tt} \rangle\, Y) \wedge [\mathtt{tt}]\, X)$ – exprimant le fait que chaque passage en section non critique est potentiellement suivi d'une entrée en section critique – est d'alternance 1, puisque la variable de plus grand point fixe X n'apparaît pas dans la sous-formule $\langle \mathtt{BCS}i \rangle\, \mathtt{tt} \vee \langle \mathtt{tt} \rangle\, Y$ définissant la variable de plus petit point fixe Y ("Y n'appelle pas X"). En revanche, la formule $\nu X.\mu Y.(\langle \mathtt{BCS}i \rangle\, X \vee \langle \neg \mathtt{BCS}j \rangle\, Y)$ – exprimant l'existence d'une séquence infinie où le processus i monopolise la ressource partagée au détriment du processus j – bien qu'elle comporte également deux opérateurs de point fixe imbriqués, est d'alternance 2, puisque la variable de plus grand point fixe X et celle de plus petit point fixe Y s'appellent mutuellement.

La notion d'alternance induit une hiérarchie de fragments du μ-calcul modal, notés $\mathrm{L}\mu_1$, $\mathrm{L}\mu_2$, ... , chaque fragment $\mathrm{L}\mu_i$ contenant les formules d'alternance i. Le fragment $\mathrm{L}\mu_1$ (également appelé *sans alternance*) est le moins expressif de la hiérarchie, dont l'autre extrême est le μ-calcul modal complet $\mathrm{L}\mu$, qui réunit tous les fragments $\mathrm{L}\mu_i$ pour $i \geq 0$. Cependant, les fragments $\mathrm{L}\mu_1$ et $\mathrm{L}\mu_2$ sont suffisamment expressifs pour permettre la traduction de logiques arborescentes comme Actl ou régulières comme Pdl et Pdl-Δ. A titre d'exemple, les traductions en μ-calcul modal des logiques Actl [FGR92] et Pdl-Δ [EL86] sont illustrées dans le tableau 5.

Les formules[5] de μ-calcul modal correspondant aux opérateurs d'Actl et de Pdl appartiennent au fragment $\mathrm{L}\mu_1$: les opérateurs temporels d'Actl sont traduits vers des formules fermées ayant un seul opérateur de point fixe (ce qui ne produit pas de récursion entre les opérateurs de point fixe) et les modalités de possibilité et de nécessité de Pdl sont traduites vers des formules fermées ayant uniquement des opérateurs de plus petit, respectivement plus grand point fixe (ce qui ne produit de la récursion mutuelle qu'entre des opérateurs de point fixe de même signe). En outre, $\mathrm{L}\mu_1$ est strictement plus expressif que ces deux logiques, puisqu'il permet de décrire les formules $\mu X.(\langle c \rangle\, \mathtt{tt} \vee \langle a \rangle\, \langle b \rangle\, X)$ et $\mu X.\, [a]\, X$, qui sont respectivement inexprimables en Actl et en Pdl. En revanche, les formules de μ-calcul modal obtenues par traduction de l'opérateur de répétition infinie de Pdl-Δ appartiennent à $\mathrm{L}\mu_2$: les opérateurs $\Delta\beta$ conduisent – lorsque β contient des opérateurs de répétition – à des formules ayant des opérateurs ν et μ mutuellement récursifs (par exemple, la formule $\Delta(a^*; b)$ se traduit vers la formule $\nu X_1.\mu X_2.(\langle b \rangle\, X_1 \vee \langle a \rangle\, X_2)$ d'alternance 2). En fait, $\mathrm{L}\mu_2$ est stricte-

[5] Les traductions des opérateurs $\mathsf{A}(\varphi_{1\,\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$ et $\langle \beta_1 \cup \beta_2 \rangle\, \varphi$ dupliquent les sous-formules φ_2 et φ, ce qui peut conduire à des formules de μ-calcul modal de taille exponentielle par rapport aux formules initiales d'Actl ou de Pdl. Des traductions succinctes peuvent être obtenues en représentant les formules comme des graphes orientés acycliques (Dags), puisque le nombre de sous-formules distinctes reste linéaire par rapport à celui de la formule initiale [EL86].

Opérateur		Traduction
ACTL	$\mathsf{EX}_\alpha\varphi$	$\langle\alpha\rangle\,\varphi$
	$\mathsf{EX}_\tau\varphi$	$\langle\tau\rangle\,\varphi$
	$\mathsf{AX}_\alpha\varphi$	$\langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg\alpha]\,\mathsf{ff} \wedge [\alpha]\,\varphi$
	$\mathsf{AX}_\tau\varphi$	$\langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg\tau]\,\mathsf{ff} \wedge [\tau]\,\varphi$
	$\mathsf{E}(\varphi_{1_\alpha}\mathsf{U}\ \varphi_2)$	$\mu X.\varphi_2 \vee (\varphi_1 \wedge \langle\alpha \vee \tau\rangle\, X)$
	$\mathsf{E}(\varphi_{1_{\alpha_1}}\mathsf{U}_{\alpha_2}\varphi_2)$	$\mu X.\varphi_1 \wedge (\langle\alpha_2\rangle\,\varphi_2 \vee \langle\alpha_1 \vee \tau\rangle\, X)$
	$\mathsf{A}(\varphi_{1_\alpha}\mathsf{U}\ \varphi_2)$	$\mu X.\varphi_2 \vee (\varphi_1 \wedge \langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\,\mathsf{ff} \wedge [\alpha \vee \tau]\, X)$
	$\mathsf{A}(\varphi_{1_{\alpha_1}}\mathsf{U}_{\alpha_2}\varphi_2)$	$\mu X.\varphi_1 \wedge [\neg(\alpha_1 \vee \alpha_2 \vee \tau)]\,\mathsf{ff} \wedge [\alpha_2 \wedge \neg\alpha_1]\,\varphi_2 \wedge$ $\langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg\alpha_2]\, X \wedge [\alpha_1 \wedge \alpha_2]\,(\varphi_2 \vee X)$
PDL-Δ	$\langle\alpha\rangle\,\varphi$	$\langle\alpha\rangle\,\varphi$
	$\langle\varphi_1?\rangle\,\varphi_2$	$\varphi_1 \wedge \varphi_2$
	$\langle\beta_1;\beta_2\rangle\,\varphi$	$\langle\beta_1\rangle\,\langle\beta_2\rangle\,\varphi$
	$\langle\beta_1 \cup \beta_2\rangle\,\varphi$	$\langle\beta_1\rangle\,\varphi \vee \langle\beta_2\rangle\,\varphi$
	$\langle\beta^*\rangle\,\varphi$	$\mu X.\varphi \vee \langle\beta\rangle\, X$
	$\Delta\beta$	$\nu X.\,\langle\beta\rangle\, X$

Table 5: Traductions d'ACTL et de PDL-Δ en μ-calcul modal

ment plus expressif que PDL-Δ, puisqu'il permet de décrire la formule $\nu X.(\langle a\rangle\, X \wedge \langle b\rangle\, X)$, qui est inexprimable en PDL-Δ [EL86].

Le μ-calcul modal est adéquat avec l'équivalence forte entre les STE, ce qui constitue une généralisation du résultat concernant l'adéquation de HML [Sti01].

6.4 Autres logiques de point fixe

Diverses variantes du μ-calcul modal ont été proposées, dans le but d'assurer l'adéquation avec des équivalences faibles entre STE ou d'obtenir une spécification plus succincte de certaines propriétés. La variante observationnelle du μ-calcul modal [Sti01] – adéquate avec l'équivalence observationnelle – remplace les modalités fortes $\langle\alpha\rangle\,\varphi$ et $[\alpha]\,\varphi$ par des modalités faibles $\langle\langle\alpha\rangle\rangle\,\varphi$, $\langle\langle\rangle\rangle\,\varphi$ et $[[\alpha]]\,\varphi$, $[[]]\,\varphi$, ayant la même sémantique que celles de la variante observationnelle de HML.

Le μ-calcul *régulier* [MS03] remplace les formules sur actions du μ-calcul modal avec les formules régulières de PDL. Bien que cette extension n'augmente pas l'expressivité de la logique par rapport au μ-calcul modal standard – les modalités de PDL pouvant être traduites en μ-calcul modal (voir la section 6.3) – la description des propriétés en μ-calcul régulier est rendue plus simple et plus conviviale grâce à l'utilisation des opérateurs réguliers à la place des opérateurs de point fixe. Par exemple, la propriété d'accessibilité équitable de la section critique, exprimable en μ-calcul régulier par la formule $[\mathsf{tt}^*; \mathtt{NCS}i; (\neg\mathtt{BCS}i)^*]\,\langle\mathsf{tt}^*; \mathtt{BCS}i\rangle\,\mathsf{tt}$ (voir la section 5.2), serait décrite en μ-calcul modal par la formule de point fixe nettement moins lisible $\nu X_1.([\mathtt{NCS}i]\,\nu X_2.(\mu X_3.(\langle\mathtt{BCS}i\rangle\,\mathsf{tt} \vee \langle\mathsf{tt}\rangle\, X_3) \wedge [\neg\mathtt{BCS}i]\, X_2) \wedge [\mathsf{tt}]\, X_1)$. L'amélioration de la clarté devient plus conséquente avec la complexité des propriétés : le lecteur intéressé peut le constater en écrivant une formule de μ-calcul modal pour la propriété d'alternance des entrées et sorties de section critique, exprimée avec une seule modalité PDL dans la section 5.2 (une façon d'obtenir la formule est d'appliquer la traduction de PDL en μ-calcul modal définie dans le tableau 5).

La logique HMLR (HML *with recursion*) [Lar88] remplace les opérateurs de point fixe du μ-calcul modal par des systèmes d'équations récursives (de signe μ ou ν) ayant des variables propositionnelles sur la partie gauche et des formules de HML sur la partie droite. Une description de propriété en HMLR est constituée d'un système d'équations – comportant un ou plusieurs *blocs* d'équations de même signe – et d'une variable *principale* représentant la propriété. Par exemple, la propriété d'accessibilité inévitable de la section critique, exprimée auparavant en μ-calcul modal (voir la section 6.2), peut être décrite en HMLR au moyen d'un système d'équations comportant deux blocs $\{X_1 \stackrel{\nu}{=} [\mathtt{NCS}i]\, X_2 \wedge [\mathtt{tt}]\, X_1\}$ et $\{X_2 \stackrel{\mu}{=} \langle \mathtt{tt} \rangle\, \mathtt{tt} \wedge [\neg \mathtt{BCS}i]\, X_2\}$ avec la variable principale X_1. HMLR est aussi expressive que le μ-calcul modal, les deux logiques pouvant être traduites l'une en termes de l'autre [CKS92]. En outre, grâce à la possibilité de factoriser les sous-formules communes au moyen d'équations, HMLR autorise une description des propriétés plus succincte que le μ-calcul modal. Ainsi, la modalité $\langle \beta_1 \cup \beta_2 \rangle\, \varphi$ de PDL peut être traduite en HMLR par le système d'équations $\{X_1 \stackrel{\mu}{=} \langle \beta_1 \rangle\, X_2 \vee \langle \beta_2 \rangle\, X_2, X_2 \stackrel{\mu}{=} \varphi\}$ ayant la variable principale X_1, ce qui évite la duplication de φ. HMLR est également utilisée comme formalisme intermédiaire pour la vérification des formules du μ-calcul modal sur des STE [MS03].

La logique de Dicky [Dic86] est un formalisme équationnel – défini historiquement avant HMLR – conçu pour exprimer des propriétés temporelles sur les STE. Cette logique comporte des primitives permettant de manipuler les états aussi bien que les transitions d'un STE. Les variables X et Y présentes sur la partie gauche des équations dénotent respectivement des ensembles d'états et de transitions du STE (les variables de plus petit et de plus grand point fixe sont indiquées respectivement par des exposants '+' et '−'). Les formules sur la partie droite des équations sont construites au moyen des opérateurs booléens standard, des opérateurs T_a, qui dénotent les transitions étiquetées par a, des opérateurs $\mathrm{Source}(Y)$ et $\mathrm{Target}(Y)$, qui dénotent les états source et but des transitions satisfaisant Y, et des opérateurs $\mathrm{Tsource}(X)$ et $\mathrm{Ttarget}(X)$, qui dénotent les transitions sortantes et entrantes des états satisfaisant X. Divers opérateurs dérivés peuvent être définis. Les opérateurs $\mathrm{Pred}(X) = \mathrm{Source}(\mathrm{Ttarget}(X))$ et $\mathrm{Succ}(X) = \mathrm{Target}(\mathrm{Tsource}(X))$ dénotent les états prédécesseurs, respectivement successeurs des états satisfaisant X. Leurs duaux $\widetilde{\mathrm{Pred}}(X) = \neg \mathrm{Pred}(\neg X)$ et $\widetilde{\mathrm{Succ}}(X) = \neg \mathrm{Succ}(\neg X)$ dénotent les états dont tous les états successeurs, respectivement tous les états prédécesseurs, satisfont X.

Des opérateurs plus complexes peuvent être définis par des équations de point fixe. L'opérateur $\mathrm{Acc}(X)$, dénotant les états accessibles depuis les états satisfaisant X, est la solution de l'équation $Z^+ = X \vee \mathrm{Succ}(Z)$. L'opérateur $\mathrm{Coacc}(X)$, dénotant les états depuis lesquels sont accessibles des états satisfaisant X, est la solution de l'équation $Z^+ = X \vee \mathrm{Pred}(Z)$. L'opérateur Inf, dénotant les états à partir desquels il existe des chemins infinis, est la solution de l'équation $Z^- = \mathrm{Pred}(Z)$. Des propriétés de potentialité et d'inévitabilité peuvent également être exprimées : par exemple, l'accessibilité potentielle d'une action a, exprimée en ACTL par la formule $\mathsf{E}(\mathtt{tt}_{\mathtt{tt}}\mathsf{U}_a\mathtt{tt})$, peut être décrite comme solution de l'équation $X^+ = \mathrm{Source}(T_a) \vee \mathrm{Pred}(X)$.

La logique de Dicky munie des opérateurs ci-dessus a une expressivité comparable à celle du μ-calcul modal ; en outre, elle autorise la description symétrique de propriétés portant sur le passé et sur le futur (un exemple d'opérateur du passé est $\mathrm{Acc}(X)$, qui caractérise les états ayant les états de X comme "ancêtres" dans le STE). Enfin, l'opérateur $\mathrm{Loop}(X)$, dénotant les états présents sur les circuits du STE qui passent par des états satisfaisant X, est inexprimable en μ-calcul modal [Dic86].

7 Vérification énumérative

Lors du choix d'une logique temporelle en tant que formalisme de spécification de propriétés, un aspect important à prendre en compte est la complexité de la vérification (évaluation d'une formule sur un modèle STE), mesurée habituellement par rapport à la taille de la formule (nombre d'opérateurs) et du STE (nombre d'états et de transitions). En effet, même si la plupart des formules ne dépassent pas une dizaine d'opérateurs, les STE des applications réelles vont souvent au-delà d'un million d'états, ce qui impose l'utilisation de logiques ayant une complexité de vérification polynomiale afin d'obtenir des temps de réponse utiles en pratique. Dans cette section, nous illustrons brièvement les principaux types d'algorithmes permettant d'évaluer des formules temporelles sur des STE et nous précisons les complexités d'évaluation des différentes logiques présentées aux sections 3, 4, 5 et 6. Nous indiquons également plusieurs outils de vérification associés qui sont actuellement disponibles.

7.1 Algorithmes globaux

Considérons un STE $M = (S, A, T, s_0)$ et une formule sur états φ. Conformément à la définition donnée en section 3.1, la vérification de φ sur M revient à évaluer la valeur de vérité de φ sur s_0, c'est-à-dire à déterminer si $s_0 \in [\![\varphi]\!]$. Une première méthode de vérification, dite *globale*, consiste à calculer d'abord l'interprétation $[\![\varphi]\!]$ et à tester ensuite si l'état initial s_0 lui appartient. Nous présentons cette méthode pour le μ-calcul modal, les logiques arborescentes et régulières présentées aux sections 4 et 5 pouvant être traitées par traduction vers le μ-calcul modal (voir le tableau 5).

L'interprétation des opérateurs booléens et des modalités $\langle\alpha\rangle\,\varphi$ et $[\alpha]\,\varphi$ peut être calculée directement à partir de leur définition donnée dans le tableau 4. L'interprétation des opérateurs de point fixe $\mu X.\varphi$ et $\nu X.\varphi$ dans un environnement ρ peut être calculée en utilisant la variante "constructive" du théorème de Tarski. Dans le cas des STE finis, les fonctionnelles $\Phi_\rho : 2^S \to 2^S$ sont non seulement monotones, mais également *continues* :

$$\Phi_\rho(\textstyle\bigcup_{k\geq 0} U_k) = \bigcup_{k\geq 0} \Phi_\rho(U_k) \text{ pour toute suite croissante } U_k \subseteq U_{k+1}$$

$$\Phi_\rho(\textstyle\bigcap_{k\geq 0} W_k) = \bigcap_{k\geq 0} \Phi_\rho(W_k) \text{ pour toute suite décroissante } W_k \supseteq W_{k+1}$$

Cela permet de calculer le plus petit et le plus grand point fixe de Φ_ρ de manière itérative, comme $\bigcup_{k\geq 0} \Phi_\rho^k(\emptyset)$ et respectivement $\bigcap_{k\geq 0} \Phi_\rho^k(S)$.

Nous illustrons l'application de cette méthode pour calculer l'interprétation de la formule $\mu X.(\langle \mathtt{tt}\rangle\, \mathtt{tt} \wedge [\neg \mathtt{BCS0}]\, X)$, exprimant l'accessibilité inévitable de l'entrée du processus 0 en section critique, sur le STE du protocole de Peterson montré en figure 2. La fonctionnelle $\Phi : 2^S \to 2^S$ associée à cette formule (nous avons omis l'environnement ρ puisque la formule est fermée) est définie par $\Phi(U) = [\![\langle \mathtt{tt} \rangle\, \mathtt{tt} \wedge [\neg \mathtt{BCS0}]\, X]\!]\, [U/X]$. En utilisant la sémantique des opérateurs booléens et en tenant compte du fait que tous les états du STE ont au moins un successeur (voir la section 3.2), la fonctionnelle peut être réduite à $\Phi(U) = [\![[\neg \mathtt{BCS0}]\, X]\!]\, [U/X]$. Enfin, en utilisant la sémantique des opérateurs modaux, elle peut être exprimée comme $\Phi(U) = \{s \in S \mid \forall s \xrightarrow{a} s'.a \neq \mathtt{BCS0} \Rightarrow s' \in U\}$. La première itération produit l'ensemble $\Phi^1(\emptyset) = \{s_{23}\}$ contenant le seul état du STE n'ayant que des transitions successeurs étiquetées par BCS0. La deuxième itération produit $\Phi^2(\emptyset) = \Phi(\Phi^1(\emptyset)) = \{s_{21}, s_{23}\}$, la troisième $\Phi^3(\emptyset) = \Phi(\Phi^2(\emptyset)) = \{s_{16}, s_{21}, s_{23}\}$ et ainsi de suite ; le calcul converge – comme le lecteur intéressé peut s'en assurer – après onze itérations, produisant l'ensemble $\{s_4, s_{10}, s_{11}, s_{14}, s_{16}, s_{17}, s_{20}, s_{21}, s_{22}, s_{23}, s_{24}\}$ qui représente le plus petit point fixe recherché. L'état initial s_0 ne se trouvant pas dans cet ensemble, le STE ne satisfait pas la formule ci-dessus, ce qui dénote l'absence d'équité du protocole de Peterson vis-à-vis des accès à la ressource (voir la section 4.2).

Un des premiers algorithmes globaux pour un fragment de la logique de Dicky semblable à $L\mu_1$, ayant une complexité en temps de calcul $O(|\varphi|^2 \cdot (|S| + |T|))$, a été proposé en [AC88]. Plus tard, des algorithmes globaux pour $L\mu_1$, ayant une complexité $O(|\varphi| \cdot (|S| + |T|))$, ont été proposés en [CS93, And94]. Enfin, des algorithmes globaux pour le μ-calcul modal complet, ayant une complexité $O((|\varphi| \cdot (|S| + |T|))^{ad(\varphi)+1})$ et $O((|\varphi| \cdot (|S| + |T|))^{ad(\varphi)})$, où $ad(\varphi)$ dénote l'alternance de φ, ont été proposés respectivement en [EL86] et [And94].

7.2 Algorithmes locaux

Les méthodes de vérification globales nécessitent la construction du STE dans sa totalité *avant* de commencer l'évaluation d'une formule, ce qui peut s'avérer prohibitif en termes de mémoire pour les STE de grande taille. Afin de pallier cet inconvénient, des méthodes de vérification *locales* ont été développées, qui permettent de construire le STE *pendant* l'évaluation de la formule, en le générant dynamiquement au fur et à mesure des besoins. De ce fait, les méthodes locales peuvent être considérées plus sophistiquées que celles globales, puisqu'elles permettent de calculer le même résultat sans avoir une connaissance *a priori* de la totalité du STE.

Soit un STE $M = (S, A, T, s_0)$ et une formule de point fixe $\sigma X.\varphi$ ($\sigma \in \{\mu, \nu\}$). Une méthode naturelle de vérification locale consiste à reformuler le problème $s_0 \in [\![\sigma X.\varphi]\!]$ comme la résolution de la variable X_{s_0} d'un système d'équations booléennes $\{X_s \stackrel{\sigma}{=} (\varphi)_s\}_{s \in S}$ où chaque variable booléenne X_s indique si l'état s satisfait la variable propositionnelle X. Les formules booléennes $(\varphi)_s$ sur la partie droite des équations sont obtenues en "projetant" φ sur l'état s au moyen des règles suivantes : $(\mathsf{ff})_s = \mathsf{ff}$, $(\mathsf{tt})_s = \mathsf{tt}$, $(X)_s = X_s$, $(\varphi_1 \vee \varphi_2)_s = (\varphi_1)_s \vee (\varphi_2)_s$, $(\varphi_1 \wedge \varphi_2)_s = (\varphi_1)_s \wedge (\varphi_2)_s$, $(\langle \alpha \rangle\, \varphi)_s = \bigvee_{s \xrightarrow{a} s'} (a \in [\![\alpha]\!] \wedge (\varphi)_{s'})$ et $([\alpha]\, \varphi)_s = \bigwedge_{s \xrightarrow{a} s'} (a \in [\![\alpha]\!] \Rightarrow (\varphi)_{s'})$. La résolution de la variable X_{s_0} est effectuée en évaluant la formule $(\varphi)_{s_0}$ sur la partie droite de l'équation, ce qui peut nécessiter les valeurs d'autres variables X_s, obtenues par évaluation des formules $(\varphi)_s$ sur la partie droite de leurs équations, et ainsi de suite ; le calcul s'arrête lorsque la valeur de vérité de la variable d'intérêt X_{s_0} a pu être déterminée. Il est important de noter que l'évaluation des formules sur la partie droite des équations permet de parcourir les transitions du STE "en avant", ce qui autorise la construction du STE "à la volée" pendant la vérification.

Nous illustrons l'application de cette méthode pour vérifier la formule $\mu X.(\langle \texttt{BCS0} \rangle \, \texttt{tt} \vee \langle \texttt{tt} \rangle \, X)$, exprimant l'accessibilité potentielle de l'entrée du processus 0 en section critique, sur le STE du protocole de Peterson montré en figure 2. Le système d'équations booléennes correspondant est $\{X_s \stackrel{\mu}{=} \bigvee_{s \stackrel{\texttt{BCS0}}{\rightarrow} s'} \texttt{tt} \vee \bigvee_{s \stackrel{a}{\rightarrow} s'} X_{s'}\}$ ayant une équation pour chaque état $s \in S$. Le calcul de la variable X_{s_0} nécessite l'évaluation de $\bigvee_{s_0 \stackrel{\texttt{BCS0}}{\rightarrow} s'} \texttt{tt} \vee \bigvee_{s_0 \stackrel{a}{\rightarrow} s'} X_{s'} = X_{s_1} \vee X_{s_2}$, ce qui requiert les valeurs de X_{s_1} et X_{s_2} ; la variable X_{s_1} nécessite l'évaluation de $X_{s_3} \vee X_{s_4}$; X_{s_3} nécessite $X_{s_{11}} \vee X_{s_{12}}$; $X_{s_{11}}$ nécessite $X_{s_{16}} \vee X_{s_{20}}$; et finalement $X_{s_{16}}$ conduit à l'évaluation de la formule $\bigvee_{s_{16} \stackrel{\texttt{BCS0}}{\rightarrow} s'} \texttt{tt} \vee \bigvee_{s_{16} \stackrel{a}{\rightarrow} s'} X_{s'} = \texttt{tt}$. En substituant la variable $X_{s_{16}}$ dans la formule $X_{s_{16}} \vee X_{s_{20}}$, la variable $X_{s_{11}}$ devient vraie également ; la propagation des substitutions rend vraies toutes les formules intermédiaires (puisqu'il s'agit de disjonctions) et par conséquent la variable d'intérêt X_{s_0} est vraie, signifiant que la formule est satisfaite par le STE. Le calcul a nécessité l'exploration d'une partie seulement du STE (les états s_0, s_1, s_2, s_3, s_4, s_8, s_{11}, s_{12}, s_{16} et s_{20}), totalisant 40 % de l'ensemble d'états.

Des algorithmes locaux pour le μ-calcul modal basés sur les systèmes d'équations booléennes ont été proposés dans [VL94, And94, Mad97, MS03]. D'autres algorithmes locaux, basés sur le calcul des séquents et sur la théorie des jeux, ont été proposés respectivement en [Cle90b] et [SS98].

7.3 Expressivité versus complexité

Afin d'offrir une image synthétique des principales logiques modales et temporelles présentées aux sections 3, 4, 5 et 6, nous récapitulons dans cette section les résultats concernant leurs expressivités relatives et leurs complexités d'évaluation. La synthèse est illustrée dans la figure 3 (où LD, LD_1, $\text{L}\mu_1^{reg}$, HML_{obs} et $\text{L}\mu_{obs}$ dénotent respectivement la logique de Dicky avec et sans alternance, le μ-calcul régulier d'alternance 1 et les variantes observationnelles de HML et du μ-calcul modal). Une flèche $L_1 \rightarrow L_2$ signifie que la logique L_1 est strictement plus expressive que L_2 et une flèche $L_1 \leftrightarrow L_2$ signifie que L_1 et L_2 ont la même expressivité (elles peuvent être traduites l'une en termes de l'autre).

En haut de la hiérarchie se situent les logiques temporelles les plus expressives (logique de Dicky avec alternance quelconque, μ-calcul modal, HMLR et μ-ACTL), qui autorisent la description de propriétés très complexes sur les STE, mais qui ont également la complexité d'évaluation la plus élevée (exponentielle en taille de la formule et du STE). Au niveau intermédiaire se trouvent le fragment $\text{L}\mu_2$ du μ-calcul modal (formules d'alternance 2) et la logique PDL-Δ, qui permettent de spécifier des propriétés d'équité relativement élaborées, tout en ayant une complexité d'évaluation quadratique. En bas de la hiérarchie se trouvent les logiques "proches" du fragment $\text{L}\mu_1$ du μ-calcul modal (LD_1, $\text{L}\mu_1^{reg}$, ACTL, PDL et les variantes de HML), qui sont les plus attractives en pratique puisqu'elles permettent d'exprimer des propriétés utiles de sûreté, de vivacité et (dans une certaine mesure) d'équité, tout en ayant une complexité d'évaluation linéaire en taille de la formule et du STE.

Pour certaines des logiques illustrées sur la figure 3, les classes de complexité du problème d'évaluation sur un modèle STE ont été identifiées. Ainsi, le problème est P-complet pour

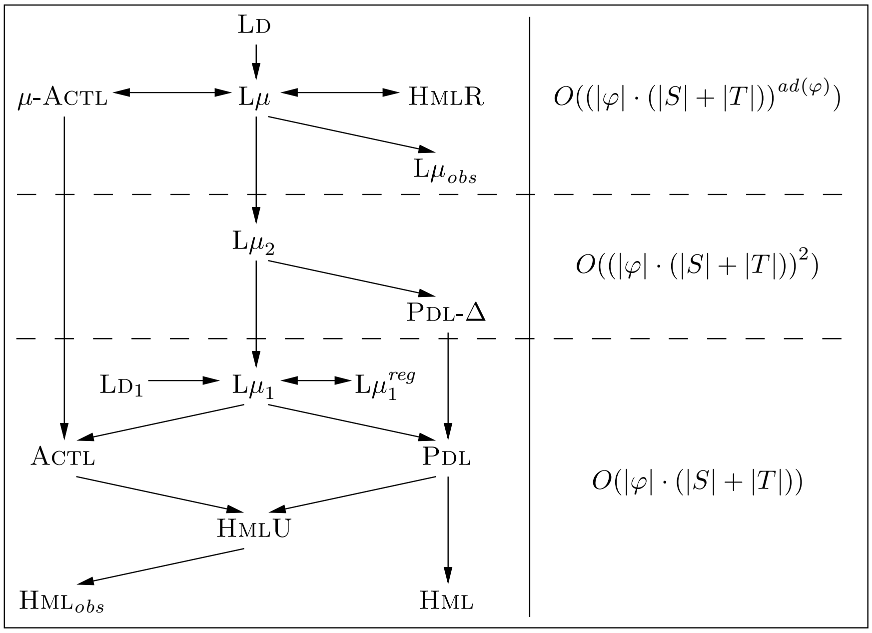

Figure 3: Expressivité comparée et complexité d'évaluation

ACTL (en transposant le résultat correspondant pour CTL [Sch02]) et pour Lμ_1 [ZSS94]. Pour Lμ (et, par conséquent, pour μ-ACTL et LD), le problème est dans NP $\cap$ co-NP [HKM03].

Le concepteur d'applications distribuées critiques désireux de choisir une logique temporelle est amené à faire un compromis entre la gamme des propriétés susceptibles d'être employées et la complexité de leur évaluation (qui est en relation directe avec la taille des modèles STE générés à partir des applications). Toutefois, l'expérience montre que les fragments Lμ_1 et Lμ_2 du μ-calcul modal sont habituellement suffisants pour exprimer la plupart des propriétés temporelles rencontrées en pratique.

7.4 Outils de vérification

Les algorithmes d'évaluation globaux et locaux mentionnés aux sections 7.1 et 7.2 ont fait l'objet d'instanciations dans différents outils de vérification. Nous décrivons brièvement quelques-uns de ces outils ci-dessous, en particulier ceux qui permettent de traiter certaines des logiques temporelles présentées aux sections 3, 4, 5 et 6.

- JACK [BGL94] est un évaluateur développé initialement pour la logique ACTL et étendu ultérieurement pour traiter μ-ACTL. Il fournit des algorithmes de vérification locaux et globaux, ainsi que des facilités de génération de diagnostics (sous forme textuelle) expliquant la valeur de vérité des formules. JACK permet de vérifier des applications décrites en LOTOS/MEIJE et en algèbre de processus CCS.

- Concurrency Factory [CLSS96] est un environnement d'analyse des systèmes asynchrones comprenant, entre autres, des évaluateurs pour le μ-calcul modal qui mettent en œuvre des algorithmes globaux et locaux basés sur les systèmes d'équations booléennes et le calcul des séquents, et qui génèrent des diagnostics (sous forme de formules HML). Concurrency Factory permet de traiter des applications décrites en algèbre de processus (CCS, CSP ou ACP) ou en VPL (*Value-Passing Language*).

- CWB (Concurrency Workbench) [CPS89] est un autre environnement pour l'ingénierie des systèmes distribués qui contient également un évaluateur dédié au μ-calcul modal. Celui-ci implémente des algorithmes locaux basés sur la théorie des jeux et autorise la génération interactive de diagnostics pour les formules. CWB permet de traiter des applications décrites en algèbre de processus (CCS, CSP ou ACP).

- MEC [ABC94] est un environnement dédié à la logique de Dicky, qui met en œuvre des algorithmes de vérification globaux basés sur les systèmes d'équations booléennes et fournit également des mécanismes pour définir des macro-opérateurs temporels au moyen d'équations de point fixe. MEC permet de vérifier des applications décrites au moyen du modèle Arnold-Nivat, basé sur les automates communicants et les vecteurs de synchronisation [Arn92].

- EVALUATOR [MS03] est un évaluateur dédié au μ-calcul régulier d'alternance 1, intégré dans la boîte à outils CADP [GLM02]. Il met en œuvre des algorithmes de vérification locaux basés sur des systèmes d'équations booléennes et fournit des mécanismes pour créer des bibliothèques de macro-opérateurs et pour générer des diagnostics (sous forme de fragments du STE). EVALUATOR a été développé au moyen de l'environnement générique OPEN/CÆSAR [Gar98] pour la manipulation de STE et de ce fait il est indépendant du langage de description des applications.

Ces outils de vérification ont été utilisés avec succès pour analyser des applications industrielles critiques dans des domaines très divers[6] (télécommunications, bases de données distribuées, systèmes embarqués, protocoles de sécurité, interfaces graphiques, électronique grand public, architectures multiprocesseurs, etc.).

8 Conclusion

Dans cet article, nous avons tenté d'offrir un panorama des principales logiques temporelles arborescentes basées sur actions, qui sont adaptées à la vérification des applications asynchrones décrites au moyen d'algèbres de processus (comme CCS, CSP ou ACP) ou de langages normalisés qui s'en inspirent (comme LOTOS ou E-LOTOS). Pour chacune de ces logiques – modales, arborescentes, régulières ou de point fixe – nous avons illustré son utilisation par des exemples de propriétés typiques (sûreté, vivacité, équité) et nous avons précisé son expressivité relative et son adéquation avec des relations d'équivalence entre les modèles STE.

Les logiques de point fixe, telles que le μ-calcul modal ou la logique de Dicky, sont les plus expressives, mais également les plus complexes à utiliser, pouvant être considérées comme

[6]Un catalogue d'études de cas industrielles effectuées avec la boîte à outils CADP peut être trouvé à l'adresse http://www.inrialpes.fr/vasy/cadp/case-studies

des “langages assembleurs” pour la description des propriétés temporelles. En revanche, les logiques arborescentes, comme ACTL, ou régulières, comme PDL, bien que moins expressives que les logiques de point fixe, fournissent une gamme d'opérateurs plus intuitifs, qui rendent la description de propriétés plus aisée. L'expérience pratique tend à montrer que la meilleure solution consiste à utiliser une logique temporelle suffisamment expressive – donc comportant des opérateurs de point fixe – et offrant également des opérateurs arborescents et réguliers (des exemples de telles logiques sont μ-ACTL et le μ-calcul régulier).

Nous avons également décrit les principales classes d'algorithmes de vérification (globaux et locaux) utilisés pour ces logiques et nous avons indiqué quelques outils de vérification qui les mettent en œuvre. En pratique, un compromis est nécessaire entre l'expressivité de la logique et sa complexité d'évaluation ; les fragments $L\mu_1$ et $L\mu_2$ du μ-calcul modal (formules d'alternance 1 et 2) constituent de bons candidats, car ils permettent la description de propriétés relativement élaborées et ont une complexité d'évaluation linéaire, respectivement quadratique, en taille du STE et de la formule.

La synthèse présentée ici est nécessairement partielle ; de nombreuses références permettent au lecteur intéressé d'approfondir le sujet. Deux ouvrages récents décrivant l'utilisation des logiques temporelles telles que CTL, LTL, CTL* et $L\mu$, les techniques de vérification associées (vérification énumérative ou symbolique, basée sur des diagrammes de décision binaires) et plusieurs outils qui les mettent en œuvre, sont [CGP00, BBL$^+$99]. Des algorithmes de vérification pour les logiques basées sur actions sont présentés dans [FGR94]. Une description détaillée de la logique de Dicky et des algorithmes de vérification associés, ainsi qu'une présentation synthétique de plusieurs logiques temporelles, peuvent être trouvées dans [Arn92]. Un ouvrage récent contenant un traitement moderne des logiques modales est [BdRV01]. Des présentations détaillées du μ-calcul modal, de ses applications à la vérification des programmes parallèles décrits en CCS, ainsi que des résultats récents sur son expressivité, sont disponibles dans [BS01, Sti01]. Enfin, un catalogue contenant des classes de propriétés génériques (telles que l'absence, l'existence, l'universalité, la précédence et la réponse), destinées à faciliter l'utilisation des logiques temporelles en pratique, est proposé dans [DAC99].

Références

[ABC94] A. Arnold, D. Bégay, and P. Crubillé. *Construction and Analysis of Transition Systems with MEC.* World Scientific, 1994.

[AC88] A. Arnold and P. Crubillé. A Linear Algorithm to Solve Fixed-Point Equations on Transition Systems. *Information Processing Letters*, 29:57–66, 1988.

[And94] H. R. Andersen. Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126(1):3–30, avril 1994.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Masson, 1992.

[BBL+99] B. Bérard, M. Bidoit, F. Laroussinie, A. Petit, and Ph. Schnœbelen. *Vérification de logiciels – Techniques et outils du model-checking*. Vuibert, 1999.

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[BGL94] A. Bouali, S. Gnesi, and S. Larosa. The Integration Project for the JACK Environment. *Bulletin of the EATCS*, 54:207–223, octobre 1994.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, juillet 1984.

[BK84] J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Computation*, 60:109–137, 1984.

[Bra92] J. C. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, Berlin, 1992.

[BS01] J. C. Bradfield and C. Stirling. Modal Logics and Mu-Calculi: An Introduction. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier, 2001.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, avril 1986.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[CKS92] R. Cleaveland, M. Klein, and B. Steffen. Faster Model Checking for the Modal Mu-Calculus. In G. v. Bochmann and D. K. Probst, editors, *Proceedings of the 4th International Conference on Computer Aided Verification CAV'92,* Montréal, Canada, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer Verlag, juin-juillet 1992.

[Cle90a] R. Cleaveland. On Automatically Explaining Bisimulation Inequivalence. In E. M. Clarke and R. P. Kurshan, editors, *Proceedings of the 2nd International Conference on Computer Aided Verification CAV'90,* New Brunswick, New Jersey, USA, volume 531 of *Lecture Notes in Computer Science*, pages 364–372. Springer Verlag, juin 1990.

[Cle90b] R. Cleaveland. Tableau-Based Model Checking in the Propositional Mu-Calculus. *Acta Informatica*, 27(8):725–747, 1990.

[CLSS96] R. Cleaveland, P. M. Lewis, S. A. Smolka, and O. Sokolsky. The Concurrency Factory: a Development Environment for Concurrent Systems. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification CAV'96,* New Brunswick, New Jersey, USA, volume 1102 of *Lecture Notes in Computer Science*, pages 398–401. Springer Verlag, août 1996.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench. In J. Sifakis, editor, *Automatic Verification of Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 24–37. Springer Verlag, 1989.

[CS93] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[DAC99] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Proceedings of the 21st International Conference on Software Engineering ICSE'99,* Los Angeles, CA, USA, mai 1999. disponible également à http://www.cis.ksu.edu/santos/spec-patterns.

[Dic86] A. Dicky. An Algebraic and Algorithmic Method for Analysing Transition Systems. *Theoretical Computer Science*, 46(2-3):285–303, 1986.

[dNV90] R. de Nicola and F. Vaandrager. Three Logics for Branching Bisimulation. In *Proceedings of the 5th Symposium on Logic in Computer Science LICS'90 (Philadelphia, USA)*, pages 118–129. IEEE Computer Society Press, juin 1990.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EL86] E. A. Emerson and C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of the 1st International Conference on Logic in Computer Science LICS'86*, pages 267–278. IEEE Computer Society Press, 1986.

[FGR92] A. Fantechi, S. Gnesi, and G. Ristori. From ACTL to Mu-Calculus. In IEI-CNR, editor, *Proceedings of the ERCIM Workshop on Theory and Practice in Verification (Pisa, Italy)*, pages 3–10, décembre 1992.

[FGR94] A. Fantechi, S. Gnesi, and G. Ristori. Model Checking for Action-Based Logics. *Formal Methods in System Design*, 4:187–203, 1994.

[FL79] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[Gar98] Hubert Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In B. Steffen, editor, *Proceedings of the 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98,* Lisbon, Portugal, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84. Springer Verlag, mars 1998. version étendue disponible comme rapport de recherche INRIA RR-3352.

[GLM02] Hubert Garavel, Frédéric Lang, and Radu Mateescu. An Overview of CADP 2001. *European Association for Software Science and Technology (EASST) Newsletter*, 4:13–24, août 2002. également disponible comme rapport technique INRIA RT-0254.

[HKM03] T. A. Henzinger, O. Kupferman, and R. Majumdar. On the Universal and Existential Fragments of the Mu-Calculus. In H. Garavel and J. Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'03,* Warsaw, Poland, volume 2619 of *Lecture Notes in Computer Science*, pages 49–64. Springer Verlag, avril 2003.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA, 2000.

[HM85] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, 32:137–161, 1985.

[HPS83] D. Harel, A. Pnueli, and J. Stavi. Propositional Dynamic Logic of Nonregular Programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[HT99] J. Henriksen and P. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied Logic*, 96(1-3):187–207, 1999.

[ISO89] ISO/IEC. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Genève, septembre 1989.

[ISO01] ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization – Information Technology, Genève, septembre 2001.

[Koz83] D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, mai 1994.

[Lar88] K. G. Larsen. Proof Systems for Hennessy-Milner logic with Recursion. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming CAAP'88,* Nancy, France, volume 299 of *Lecture Notes in Computer Science*, pages 215–230. Springer Verlag, mars 1988.

[Mad97] Angelika Mader. *Verification of Modal Properties Using Boolean Equation Systems*. VERSAL 8, Bertz Verlag, Berlin, 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[MP90] Z. Manna and A. Pnueli. A Hierarchy of Temporal Properties. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing PODC'90,* Québec, pages 377–408, août 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Volume I: Specification.* Springer-Verlag, 1992.

[MS03] R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Science of Computer Programming*, 46(3):255–281, mars 2003.

[NV90] R. De Nicola and F. W. Vaandrager. Action versus State Based Logics for Transition Systems. In *Semantics of Concurrency*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer Verlag, 1990.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, mars 1981.

[Pel87] D. Peleg. Concurrent Dynamic Logic. *Journal of the ACM*, 34(2):450–479, 1987.

[Pet83] G. L. Peterson. A New Solution to Lamport's Concurrent Programming Problem. *ACM Transactions on Programming Languages and Systems*, 5(1):121–134, 1983.

[PT91] S. Passy and T. Tinchev. An Essay in Combinatory Dynamic Logic. *Information and Computation*, 93(2):263–332, 1991.

[QS83] J-P. Queille and J. Sifakis. Fairness and Related Properties in Transition Systems – A Temporal Logic to Deal with Fairness. *Acta Informatica*, 19:195–220, 1983.

[Sch02] Ph. Schnœbelen. The Complexity of Temporal Logic Model Checking. In *Proceedings of the 4th International Workshop on Advances in Modal Logic AiML'02,* Toulouse, France, septembre-octobre 2002. à paraître dans *Advances in Modal Logic*, World Scientific, 2003.

[SS98] P. Stevens and C. Stirling. Practical Model-Checking using Games. In B. Steffen, editor, *Proceedings of the 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98,* Lisbon, Portugal, volume 1384 of *Lecture Notes in Computer Science*, pages 85–101. Springer Verlag, mars 1998.

[Sti01] C. Stirling. *Modal and Temporal Properties of Processes.* Springer Verlag, 2001.

[Str82] R. Streett. Propositional Dynamic Logic of Looping and Converse. *Information and Control*, 54:121–141, 1982.

[Szy88] B. K. Szymanski. A Simple Solution to Lamport's Concurrent Programming Problem with Linear Wait. In *Proceedings of the International Conference on Supercomputing Systems,* St Malo, France, pages 621–626, 1988.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Var88] M. Vardi. A Temporal Fixpoint Calculus. In *Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages POPL'88,* San Diego, California, pages 250–259, janvier 1988.

[vGW89] R. J. van Glabbeek and W. P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics (extended abstract). CS Report R8911, CWI, Amsterdam, 1989. également disponible dans *Proceedings of the IFIP 11th World Computer Congress, (San Francisco, USA)*, 1989.

[VL94] B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In S. Abiteboul and E. Shamir, editors, *Proceedings of the 21st International Colloquium on Automata, Languages and Programming ICALP'94,* Vienna, Austria, volume 820 of *Lecture Notes in Computer Science*, pages 304–315. Springer Verlag, juillet 1994.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1-2):72–99, 1983.

[ZSS94] S. Zhang, S. A. Smolka, and O. Sokolsky. On the Parallel Complexity of Model Checking in the Modal Mu-Calculus. In *Proceedings of 9th Annual IEEE Symposium on Logic in Computer Science*, pages 154–163. IEEE Computer Society Press, juillet 1994.

A Traduction d'ACTL vers PDL-Δ

L'intuition qui sert de base à la traduction d'ACTL vers PDL-Δ est la suivante. Les opérateurs de potentialité $\mathsf{E}(.\mathsf{U}.)$ d'ACTL décrivent l'existence de certains arbres d'exécution finis issus d'un état, qui peuvent également être décrits au moyen des modalités de possibilité ($\langle\beta\rangle\,\varphi$) et de l'opérateur de test ($\varphi?$) de PDL. Par exemple, la propriété $\mathsf{EF}_\alpha\varphi$ peut être exprimée en PDL comme $\langle\alpha^*\rangle\,\varphi$. Les opérateurs d'inévitabilité $\mathsf{A}(.\mathsf{U}.)$ d'ACTL décrivent le fait que les préfixes finis de toutes les séquences de transitions issues d'un état doivent avoir une certaine forme, ce qui n'est pas exprimable uniquement avec les opérateurs de PDL. En revanche, ces propriétés d'inévitabilité — ou plutôt leurs négations — peuvent être exprimées en PDL-Δ grâce à l'opérateur de répétition infinie ($\Delta\beta$). Considérons la propriété d'inévitabilité $\mathsf{A}(\mathsf{tt}_{\mathsf{tt}}\mathsf{U}\ \varphi)$, exprimant le fait que toutes les séquences issues d'un état mènent, après un nombre fini de pas, à des états satisfaisant φ. Ceci peut être décrit de manière équivalente comme la conjonction de deux propriétés :

1. Il n'existe pas de séquence de transitions issue de l'état qui conduit à un état de blocage avant de rencontrer un état satisfaisant φ. Cette propriété peut être exprimée en PDL par la formule $\neg\,\langle(\neg\varphi?;\mathsf{tt})^*\rangle\,(\neg\varphi\wedge[\mathsf{tt}]\,\mathsf{ff}) = [(\neg\varphi?;\mathsf{tt})^*]\,(\varphi\vee\langle\mathsf{tt}\rangle\,\mathsf{tt})$.

2. Il n'existe pas de séquence de transitions infinie issue de l'état qui ne passe pas par des états satisfaisant φ. Cette propriété peut être exprimée en PDL-Δ par la formule $\neg\Delta(\neg\varphi?;\mathsf{tt})$.

Cette observation conduit à l'égalité $\mathsf{A}(\mathsf{tt}_{\mathsf{tt}}\mathsf{U}\ \varphi) = [(\neg\varphi?;\mathsf{tt})^*]\,(\varphi \vee \langle\mathsf{tt}\rangle\,\mathsf{tt}) \wedge \neg\Delta(\neg\varphi?;\mathsf{tt})$, qui sera justifiée formellement par la suite. La même idée sert de base à l'égalité utilisée en Ctl [CES86] : $\mathsf{A}(\varphi_1\ \mathsf{U}\ \varphi_2) = \neg\mathsf{E}(\neg\varphi_2\ \mathsf{U}\ \neg\varphi_1 \wedge \neg\varphi_2) \wedge \neg\mathsf{EG}\neg\varphi_2$, qui permet de réduire le problème de la vérification de formules Ctl à la recherche de séquences de transitions finies (resp. infinies) satisfaisant des formules de chemins de la forme $\varphi_1\ \mathsf{U}\ \varphi_2$ (resp. $\mathsf{G}\varphi$).

La traduction d'Actl vers Pdl-Δ que nous proposons est définie dans le tableau 6. Les opérateurs "next" $\mathsf{EX}_\alpha\varphi$ et $\mathsf{AX}_\alpha\varphi$ sont traduits au moyen d'opérateurs booléens et modaux. Les opérateurs de potentialité $\mathsf{E}(\varphi_{1\alpha}\mathsf{U}\ \varphi_2)$ et $\mathsf{E}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$ sont traduits comme des modalités de possibilité contenant des formules régulières. Les opérateurs d'inévitabilité $\mathsf{A}(\varphi_{1\alpha_1}\mathsf{U}\ \varphi_2)$ et $\mathsf{A}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$ sont traduits comme des conjonctions entre des modalités de nécessité contenant des formules régulières et des négations d'opérateurs de répétition, en généralisant le schéma illustré ci-dessus pour la traduction de $\mathsf{A}(\mathsf{tt}_{\mathsf{tt}}\mathsf{U}\ \varphi)$.

Opérateur	Traduction
$\mathsf{EX}_\alpha\varphi$	$\langle\alpha\rangle\,\varphi$
$\mathsf{EX}_\tau\varphi$	$\langle\tau\rangle\,\varphi$
$\mathsf{AX}_\alpha\varphi$	$\langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg\alpha]\,\mathsf{ff} \wedge [\alpha]\,\varphi$
$\mathsf{AX}_\tau\varphi$	$\langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg\tau]\,\mathsf{ff} \wedge [\tau]\,\varphi$
$\mathsf{E}(\varphi_{1\alpha}\mathsf{U}\ \varphi_2)$	$\langle(\varphi_1?;\alpha \vee \tau)^*\rangle\,\varphi_2$
$\mathsf{E}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$	$\langle(\varphi_1?;\alpha_1 \vee \tau)^*;\varphi_1?;\alpha_2\rangle\,\varphi_2$
$\mathsf{A}(\varphi_{1\alpha}\mathsf{U}\ \varphi_2)$	$[(\neg\varphi_2?;\alpha \vee \tau)^*]\,(\varphi_2 \vee (\langle\varphi_1?;\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\,\mathsf{ff})) \wedge$ $\neg\Delta(\neg\varphi_2?;\alpha \vee \tau)$
$\mathsf{A}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$	$[(\neg\alpha_2 \cup ((\alpha_1 \wedge \alpha_2);\neg\varphi_2?))^*]\,((\langle\varphi_1?;\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\neg(\alpha_1 \vee \alpha_2 \vee \tau)]\,\mathsf{ff} \wedge$ $[\alpha_2 \wedge \neg\alpha_1]\,\varphi_2) \wedge \neg\Delta(\neg\alpha_2 \cup ((\alpha_1 \wedge \alpha_2);\neg\varphi_2?))$

Table 6: Traduction d'Actl vers Pdl-Δ

La traduction des opérateurs $\mathsf{E}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$, $\mathsf{A}(\varphi_{1\alpha}\mathsf{U}\ \varphi_2)$ et $\mathsf{A}(\varphi_{1\alpha_1}\mathsf{U}_{\alpha_2}\varphi_2)$ duplique les sous-formules φ_1 et/ou φ_2, ce qui peut conduire, pour des formules Actl contenant des imbrications de ces opérateurs, à des formules Pdl-Δ de taille exponentielle. Cependant, la traduction peut être rendue succincte en utilisant une représentation des formules Pdl-Δ qui factorise les sous-formules communes — la même que celle utilisée pour les formules de μ-calcul produites par traduction d'Actl et de Pdl-Δ (voir le tableau 5) — ce qui entraîne au plus une augmentation linéaire de leur taille par rapport aux formules Actl.

Le théorème suivant établit la correction de la traduction.

Théorème 1 (traduction d'ACTL vers PDL-Δ) *Soit φ une formule* Actl *et φ' sa traduction en* Pdl-Δ *donnée par le tableau 6. Alors:*

$$[\![\varphi]\!] = [\![\varphi']\!]$$

où les interprétations des formules sont définies sur un Ste *quelconque $M = (S, A, T, s_0)$.*

Preuve Soit φ une formule Actl. Nous montrons que la traduction d'Actl vers Pdl-Δ définie dans le tableau 6 est compatible avec les traductions d'Actl et de Pdl-Δ vers le μ-calcul modal définies dans le tableau 5.

Cas $\varphi = \mathsf{EX}_\alpha \varphi_1$ (preuve similaire pour $\varphi = \mathsf{EX}_\tau \varphi_1$).

$\mathsf{EX}_\alpha \varphi_1 =$	traduction ACTL $\to$ PDL-Δ
$\langle \alpha \rangle \varphi_1 =$	traduction PDL-Δ $\to$ μ-calcul
$\langle \alpha \rangle \varphi_1$	traduction ACTL $\to$ μ-calcul

Cas $\varphi = \mathsf{AX}_\alpha \varphi_1$ (preuve similaire pour $\varphi = \mathsf{AX}_\tau \varphi_1$).

$\mathsf{AX}_\alpha \varphi_1 =$	traduction ACTL $\to$ PDL-Δ
$\langle \mathsf{tt} \rangle \mathsf{tt} \wedge [\neg\alpha]\, \mathsf{ff} \wedge [\alpha]\, \varphi_1 =$	traduction PDL-Δ $\to$ μ-calcul
$\langle \mathsf{tt} \rangle \mathsf{tt} \wedge [\neg\alpha]\, \mathsf{ff} \wedge [\alpha]\, \varphi_1$	traduction ACTL $\to$ μ-calcul

Cas $\varphi = \mathsf{E}(\varphi_{1\alpha} \mathsf{U}\ \varphi_2)$.

$\mathsf{E}(\varphi_{1\alpha} \mathsf{U}\ \varphi_2) =$	traduction ACTL $\to$ PDL-Δ
$\langle (\varphi_1?; \alpha \vee \tau)^* \rangle \varphi_2 =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X.\varphi_2 \vee \langle \varphi_1?; \alpha \vee \tau \rangle X =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X.\varphi_2 \vee \langle \varphi_1? \rangle \langle \alpha \vee \tau \rangle X =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X.\varphi_2 \vee (\varphi_1 \wedge \langle \alpha \vee \tau \rangle X)$	traduction ACTL $\to$ μ-calcul

Cas $\varphi = \mathsf{E}(\varphi_{1\alpha_1} \mathsf{U}_{\alpha_2} \varphi_2)$.

$\mathsf{E}(\varphi_{1\alpha_1} \mathsf{U}_{\alpha_2} \varphi_2) =$	traduction ACTL $\to$ PDL-Δ
$\langle (\varphi_1?; \alpha_1 \vee \tau)^*; \varphi_1?; \alpha_2 \rangle \varphi_2 =$	traduction PDL-Δ $\to$ μ-calcul
$\langle (\varphi_1?; \alpha_1 \vee \tau)^* \rangle \langle \varphi_1?; \alpha_2 \rangle \varphi_2 =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X. \langle \varphi_1?; \alpha_2 \rangle \varphi_2 \vee \langle \varphi_1?; \alpha_1 \vee \tau \rangle X =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X. \langle \varphi_1? \rangle \langle \alpha_2 \rangle \varphi_2 \vee \langle \varphi_1? \rangle \langle \alpha_1 \vee \tau \rangle X =$	traduction PDL-Δ $\to$ μ-calcul
$\mu X.(\varphi_1 \wedge \langle \alpha_2 \rangle \varphi_2) \vee (\varphi_1 \wedge \langle \alpha_1 \vee \tau \rangle X) =$	calcul propositionnel
$\mu X.\varphi_1 \wedge (\langle \alpha_2 \rangle \varphi_2 \vee \langle \alpha_1 \vee \tau \rangle X)$	traduction ACTL $\to$ μ-calcul

Cas $\varphi = \mathsf{A}(\varphi_{1\alpha} \mathsf{U}\ \varphi_2)$ (preuve similaire pour $\varphi = \mathsf{A}(\varphi_{1\alpha_1} \mathsf{U}_{\alpha_2} \varphi_2)$).

$\mathsf{A}(\varphi_{1\alpha} \mathsf{U}\ \varphi_2) =$	traduction ACTL $\to$ PDL-Δ
$[(\neg\varphi_2?; \alpha \vee \tau)^*]\, (\varphi_2 \vee (\langle \varphi_1?; \mathsf{tt} \rangle \mathsf{tt} \wedge$	
$[\neg(\alpha \vee \tau)]\, \mathsf{ff})) \wedge \neg\Delta(\neg\varphi_2?; \alpha \vee \tau) =$	traduction PDL-Δ $\to$ μ-calcul
$\nu X.((\varphi_2 \vee (\langle \varphi_1?; \mathsf{tt} \rangle \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff})) \wedge$	
$[\neg\varphi_2?; \alpha \vee \tau]\, X) \wedge \neg\nu X.(\langle \neg\varphi_2?; \alpha \vee \tau \rangle X) =$	traduction PDL-Δ $\to$ μ-calcul
$\nu X.((\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff})) \wedge$	
$(\varphi_2 \vee [\alpha \vee \tau]\, X)) \wedge \neg\nu X.(\neg\varphi_2 \wedge \langle \alpha \vee \tau \rangle X) =$	traduction PDL-Δ $\to$ μ-calcul
$\nu X.(\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)) \wedge$	
$\mu X.(\varphi_2 \vee [\alpha \vee \tau]\, X)$	

Pour obtenir l'équivalence de cette dernière formule avec la traduction d'ACTL vers le μ-calcul modal définie dans le tableau 5, il reste à montrer l'égalité suivante :

$\mu X.(\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)) =$
$\nu X.(\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)) \wedge \mu X.(\varphi_2 \vee [\alpha \vee \tau]\, X)$

L'implication "$\Rightarrow$" découle immédiatement par monotonie. Pour l'implication inverse, nous considérons un Ste $M = (S, A, T, s_0)$ et nous montrons l'inégalité suivante entre les interprétations sur M des formules en partie gauche et droite :

$[\![\nu X.(\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X))]\!] \cap$
$[\![\mu X.(\varphi_2 \vee [\alpha \vee \tau]\, X)]\!] \subseteq$
$[\![\mu X.(\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X))]\!]$

Soient $\Phi : 2^S \to 2^S$ et $\Psi : 2^S \to 2^S$ les fonctionnelles définies ci-dessous (où $U \subseteq S$ et $[U/X]$ est le contexte associant U à X) :

$\Phi(U) = [\![\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)]\!]\, [U/X]$
$\Psi(U) = [\![\varphi_2 \vee [\alpha \vee \tau]\, X]\!]\, [U/X]$

L'inégalité à montrer peut être réécrite en termes de Φ et Ψ (en utilisant l'interprétation des formules de μ-calcul définie dans le tableau 4) comme suit :

$\bigcap_{n \geq 0} \Phi^n(S) \cap \bigcup_{n \geq 0} \Psi^n(\emptyset) \subseteq \bigcup_{n \geq 0} \Phi^n(\emptyset)$

ou bien, de manière équivalente :

$\bigcup_{k \geq 0} \left(\left(\bigcap_{n \geq 0} \Phi^n(S) \right) \cap \Psi^k(\emptyset) \right) \subseteq \bigcup_{k \geq 0} \Phi^k(\emptyset)$

Pour montrer la dernière inégalité, nous prouvons la relation suivante par induction sur k :

$\forall k \geq 0. \left(\bigcap_{n \geq 0} \Phi^n(S) \right) \cap \Psi^k(\emptyset) \subseteq \Phi^k(\emptyset)$

Base. $(\bigcap_{n \geq 0} \Phi^n(S)) \cap \Psi^0(\emptyset) = (\bigcap_{n \geq 0} \Phi^n(S)) \cap \emptyset = \emptyset \subseteq \Phi^0(\emptyset)$.

Induction. Avec l'hypothèse d'induction et les définitions de Φ et Ψ, nous obtenons :

$\Phi^{k+1}(\emptyset) = \Phi(\Phi^k(\emptyset)) \supseteq$
$\Phi((\bigcap_{n \geq 0} \Phi^n(S)) \cap \Psi^k(\emptyset)) =$
$[\![\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)]\!]\, [(\bigcap_{n \geq 0} \Phi^n(S)) \cap \Psi^k(\emptyset)/X] =$
$[\![\varphi_2 \vee (\varphi_1 \wedge \langle \mathsf{tt} \rangle\, \mathsf{tt} \wedge [\neg(\alpha \vee \tau)]\, \mathsf{ff} \wedge [\alpha \vee \tau]\, X)]\!]\, [(\bigcap_{n \geq 0} \Phi^n(S))/X] \cap$
$[\![\varphi_2 \vee [\alpha \vee \tau]\, X]\!]\, [\Psi^k(\emptyset)/X] =$
$\Phi(\bigcap_{n \geq 0} \Phi^n(S)) \cap \Psi(\Psi^k(\emptyset)) =$
$(\bigcap_{n \geq 0} \Phi^n(S)) \cap \Psi^{k+1}(\emptyset)$.

$\square$

Utilisée conjointement avec la traduction entre CTL et ACTL donnée en [NV90], la traduction définie dans le tableau 6 permet également d'exprimer les opérateurs de CTL en PDL-Δ. Cependant, nous pouvons concevoir une traduction directe de CTL vers PDL-Δ en utilisant un schéma similaire à celui du tableau 6 (nous illustrons ci-dessous les traductions des opérateurs temporels principaux de CTL) :

$$\begin{array}{rcl} \mathsf{E}\,[\varphi_1 \;\mathsf{U}\; \varphi_2] & = & \langle(\varphi_1?;\mathsf{tt})^*\rangle\,\varphi_2 \\ \mathsf{A}\,[\varphi_1 \;\mathsf{U}\; \varphi_2] & = & [(\neg\varphi_2?;\mathsf{tt})^*]\,(\varphi_2 \vee \langle\varphi_1?;\mathsf{tt}\rangle\,\mathsf{tt}) \wedge \neg\Delta(\neg\varphi_2?;\mathsf{tt}) \end{array}$$

Le lecteur intéressé peut vérifier la correction de ces deux égalités par rapport à la traduction des opérateurs CTL en μ-calcul modal [EL86] : $\mathsf{E}\,[\varphi_1 \;\mathsf{U}\; \varphi_2] = \mu X.\varphi_2 \vee (\varphi_1 \wedge \langle\mathsf{tt}\rangle\, X)$ et $\mathsf{A}\,[\varphi_1 \;\mathsf{U}\; \varphi_2] = \mu X.\varphi_2 \vee (\varphi_1 \wedge \langle\mathsf{tt}\rangle\,\mathsf{tt} \wedge [\mathsf{tt}]\, X)$.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399