Logiques pour le changement de croyances : une histoire de tout, ou presque

Andreas Herzig

Logiques pour le changement de croyances : une histoire de tout, ou presque

Andreas Herzig

Université de Toulouse, CNRS, IRIT
herzig@irit.fr

Résumé

Nous examinons plusieurs opérations de changement de croyances à la lumière d'une logique dynamique avec affectations propositionnelles. Cette dernière permet d'encoder les mises à jour comme des programmes particuliers, ce qui fournit une méthode de mise à jour syntaxique.

Abstract

We examine several belief change operations in the light of a dynamic logic of propositional assignments. We show that update operations can be encoded as particular programs of that logic. This provides a syntactical update method.

1 Introduction

Intégrer une nouvelle information A à une base de croyances logique B consiste à construire une nouvelle base de croyances qui contient A. Écrivons $B \circ A$ pour désigner la base résultante du changement. Deux problèmes sont alors intéressants :

- Le *problème de changement de croyances* est un problème fonctionnel : il s'agit de trouver une formule logique représentant la base modifiée $B \circ A$.
- Le *problème de conséquence d'un changement de croyances* est un problème de décision : étant donné une formule C, il s'agit de décider si C est une conséquence de $B \circ A$, c.-à-d de décider si l'implication

$$(B \circ A) \rightarrow C$$

est valide.

On peut trouver beaucoup d'opérateurs de changement de croyances dans la littérature en intelligence artificielle, à commencer par l'approche des modèles possibles (*possible models approach*, PMA) de Winslett [16,17], l'opérateur de mise à jour de Forbus [3], la sémantique standard de Winslett (*Winslett's standard semantics*, WSS) [18], ainsi que l'opérateur de révision de Dalal [2]. Une vue d'ensemble de ces opérateurs peut être trouvée dans [10, 15]. Depuis Katsuno et Mendelzon [14] on distingue entre opérateurs de mise à jour et opérateurs de révision : les trois premiers sont des opérateur de mise à jours, notés usuellement $\diamond$ au lieu de $\circ$, tandis que le dernier est un opérateur de révision, noté usuellement $*$ au lieu de $\circ$.

Et la révision et la mise à jour ont été principalement étudiées d'un point de vue sémantique. Pour un langage fini, l'identification de l'ensemble de modèles de $B \circ A$ permet de construire une formule représentant $B \circ A$, à savoir la disjonction des formules décrivant chacun de ces modèles (ces dernières étant des conjonctions de littéraux).

Existe-t-il de meilleures procédures syntaxiques nous permettant de construire $B \circ A$? Force est de constater que de telles procédures sont plutôt rares : une procédure de mise à jour pour le WSS (l'opération de mise à jour la plus simple) qui est basée sur l'oubli de variables a été étudiée dans [10] et plus récemment dans [7] ; une axiomatisation du PMA a été proposée dans [6] qui est cependant basée sur des formules en forme normale disjonctive.

Dans le présent article nous proposons un cadre logique à la fois puissant et simple permettant de raisonner sur les opérateurs de changement de croyance d'une manière très générale : la logique dynamique des affectations propositionnelles (*Dynamic Logic de Propositional Assignments*), DL-PA. DL-PA est une instantiation simple de la logique dynamique propositionnelle (*Propositional Dynamic Logic*, PDL) [4,5] dont les programmes atomiques sont des affectations de formules à des variables propositionnelles de la forme $p \leftarrow \varphi$. Des programmes complexes peuvent alors être construits moyennant les opérateurs de programme standards de PDL. Par exemple, tossHeads $\cup$ tossTails est un programme nondeterministe qui décrit l'action de tirer à pile ou face, qui est ainsi identifié au choix nondeterministe entre 'tirer pile' et 'tirer face'. Nous ajoutons

ici l'opérateur de converse qu'on trouve moins souvent en PDL. Tout comme PDL, notre logique a des formules de la forme $\langle\pi\rangle\varphi$ et $[\pi]\varphi$, où π est un programme et φ est un formule. La formule $\langle\pi\rangle\varphi$ exprime que φ est vrai après *au moins une* exécution possible de π, et $[\pi]\varphi$ exprime que φ est vrai après *toute* exécution possible de π. Par exemple, $\langle(\mathsf{tossHeads} \cup \mathsf{tossTails})^*\rangle\mathsf{Heads}$ nous dit qu'il y a une manière d'exécuter la séquence finie d'actions 'lancer une pièce' tel que la pièce atterrit sur pile. DL-PA se distingue de PDL par le fait que chaque formule peut y être réduite à une formule propositionnelle équivalente. Cette dernière peut être obtenue moyennant les axiomes dites de réduction. Par exemple, pour des variables propositionnelles différentes r et p, la formule $\langle p{\leftarrow}q\rangle(p \vee r)$ est successivement équivalente à $\langle p{\leftarrow}q\rangle p \vee \langle p{\leftarrow}q\rangle r$ et à $q \vee r$.

Nous allons montrer que les opérateurs de changement de croyance peuvent être toutes exprimés dans le fragment de DL-PA sans l'opérateur d'itération $(.)^*$ (qu'on appelle aussi 'l'étoile de Kleene'). À cette fin nous traduisons d'abord la nouvelle information A dans un programme approprié π_A ; le changement de B par A peut alors être exprimé par

$$B \circ A \;=\; \langle(\pi_A)^{-1}\rangle B$$

Cette identité peut être lue : être dans un état où B a été mis à jour par A est la même chose qu'être dans un état où B était vrai avant l'exécution du programme π_A.

L'article est organisé comme suit. En section 2 nous donnons quelques notions et notations de base. En Section 3 nous introduisons la logique DL-PA. En Section 4 nous introduisons quelques abréviations de programmes DL-PA particuliers qui nous permettront de rédiger les différents programmes de changement de croyance π_A d'une manière concise. Dans les sections restantes nous traduisons plusieurs opérateurs de changement de croyances sémantique en DL-PA : la sémantique standard WSS de Winslett en Section 5, le possible modèles approach PMA de Winslett en Section 6 et l'opérateur de Forbus en Section 7. Section 8 conclut.

2 Quelques notations

Les notations et conventions suivantes vont être utilisées tout au long de l'article.

Pour commencer, $\mathbb{P} = \{p, q, \ldots\}$ dénote un ensemble dénombrable de *variables propositionnelles*.

Une *valuation* associe une valeur de vérité à chaque variable propositionnelle. Nous identifions une valuation avec un sous-ensemble de $\mathbb{P}$ et nous utilisons les symboles v, v', v_1, v_2, etc. pour les dénoter. L'ensemble de toutes les valuations est $\mathbb{V} = 2^{\mathbb{P}}$.

Les *formules booléennes* (aussi appelées *formules propositionnelles*) sont construites à partir des variables propositionnelles moyennant les connecteurs booléens standards. Nous allons en particulier employer la disjonction exclusive $\oplus$. Les formules propositionnelles sont désignées par A, B, C, etc. (ce qui contraste avec les *formules modales* qui seront introduites dans la section suivante et qui seront désignées par φ, ψ, etc.). Étant donné une formule booléenne A, l'ensemble des variables apparaissant dans A est noté $\mathbb{P}_A$. Par exemple, $\mathbb{P}_{p\wedge\neg q} = \{p, q\}$.

Une valuation permet d'associer une valeur de vérité aux formules booléennes. Une *A-valuation* est une valuation où la formule booléenne A est vraie.

D'habitude les logiques modales ont besoin de structures plus riches que les valuations pour être interprétées, à savoir les modèles de Kripke ; dans cet article les valuation va également suffire pour donner une valeur de vérité aux formules modales de notre logique DL-PA (et un puriste pourrait donc objecter qu'il est un peu abusif d'appeler ces formules modales).

3 La logique dynamique des affectations propositionnelles DL-PA

Dans la présente section nous définissons syntaxe et sémantique de notre logique dynamique des affectations propositionnelles DL-PA et nous rappellons les résultats de complexité pertinents. Le fragment sans itération de DL-PA a été introduit dans [8], où il a été montré qu'il contient la *Coalition Logic of Propositional Control* de van der Hoek et Wooldridge [11–13]. Il a été montré dans [9] que cette logique permet également une analyse des systèmes normatifs en termes de règles constitutives et règles régulatrices. Le langage entier de DL-PA avec itération a été davantage étudié dans l'article [1]. Dans le présent article nous étendons ce langage davantage : *primo*, nous ajoutons l'opérateur d'exécution à l'envers d'un programme ; *secundo*, en lieu d'affectations de variables propositionnelles à seulement vrai ou faux nous permettons des affectations à des formules arbitraires. Nous continuons à appeler cette logique DL-PA. Cela va être justifié par le fait qu'elle a la même expressivité et la même complexité que la logique DL-PA de [1].

3.1 Langage

Le langage de DL-PA est défini par la grammaire suivante :

$$\begin{array}{lcl} \pi & ::= & p{\leftarrow}\varphi \mid \varphi? \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \pi^{-1} \\ \varphi & ::= & p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \end{array}$$

où p est une variable propositionnelle de $\mathbb{P}$. Ainsi, un *programme atomique* du langage de DL-PA est un programme de la forme $p{\leftarrow}\varphi$. Les opérateurs de composition séquentielle (";"), composition non-déterministe ("$\cup$"), itération ("$(.)^*$", dit 'l'étoile de Kleene'), et test ("$(.)?$") sont familiers de PDL. L'opérateur "$(.)^{-1}$" est l'opérateur d'exécution à l'envers d'un programme.

Nous abrégeons les connecteurs logiques $\wedge$, $\rightarrow$ et $\leftrightarrow$ de la manière habituelle. De plus, $[\pi]\varphi$ abrège $\neg\langle\pi\rangle\neg\varphi$. Le programme skip abrège $\top?$: "rien ne se passe" (il pourrait aussi bien être défini par $p{\leftarrow}p$, pour un p arbitraire). Pour $n \geq 0$, la n-ième itération de π est défini inductivement comme :

$$\pi^0 = \mathsf{skip}$$
$$\pi^{n+1} = \pi^n; \pi$$

Le langage des programmes de DL-PA permet d'exprimer les primitives des langages de programmation standards. Par exemple, la boucle "while A do π" peut être exprimée en DL-PA par le programme $(A?; \pi)^*; \neg A?$.

La *longueur* d'une formule φ, notée $|\varphi|$, est le nombre de symboles nécessaires pour l'écrire φ, sans toutefois "$\langle$", "$\rangle$", paranthèses et commas. Par exemple, $|q \wedge r| = |\neg(\neg q \vee \neg r)| = 6$ et $|\langle q{\leftarrow}\top\rangle(q \wedge r)| = 2{+}6 = 8$. La longueur du programme π, notée $|\pi|$, est défini de la même manière. Par exemple, $|p{\leftarrow}\bot; p?| = 5$.

Le *fragment sans étoile* ('*star-free fragment*') de DL-PA est le sous-ensemble du langage consistant en les formules sans l'étoile de Kleene "$(.)^*$".

3.2 Semantics

L'interprétation d'une formule φ est un ensemble de valuations : l'ensemble des valuations où φ est vraie. Les programmes de DL-PA sont interprétés au moyen d'une relation (unique) entre valuations : un programme atomique $p{\leftarrow}\varphi$ met les valuations à jour de la manière attendue, et les programmes complexes sont interprétés comme en PDL par récursion mutuelle. La table 1 donne l'interprétation des connecteurs de DL-PA.

Deux formules φ_1 et φ_2 sont équivalentes si $\|\varphi_1\| = \|\varphi_2\|$. Deux programmes π_1 et π_2 sont équivalentes si $\|\pi_1\| = \|\pi_2\|$. Dans le dernier cas nous écrivons alors $\pi_1 \equiv \pi_2$. Par exemple, nous avons les équivalences de programme suivantes :

$$\begin{aligned}
\pi; \mathsf{skip} &\equiv \pi \\
p{\leftarrow}\varphi &\equiv (\varphi?; p{\leftarrow}\top) \cup (\neg\varphi?; p{\leftarrow}\bot) \\
p{\leftarrow}\top^{-1} &\equiv p?; (\mathsf{skip} \cup p{\leftarrow}\bot) \\
&\equiv p? \cup (p?; p{\leftarrow}\bot) \\
p{\leftarrow}\bot^{-1} &\equiv \neg p?; (\mathsf{skip} \cup p{\leftarrow}\top) \\
&\equiv \neg p? \cup (\neg p?; p{\leftarrow}\top) \\
p{\leftarrow}\neg p^{-1} &\equiv (p?; p{\leftarrow}\bot) \ \cup \ (\neg p?; p{\leftarrow}\top) \\
p{\leftarrow}p^{-1} &\equiv p{\leftarrow}p \\
p{\leftarrow}q^{-1} &\equiv p{\wedge}q? \cup (p{\wedge}q?; p{\leftarrow}\bot) \cup \\
&\qquad \neg p{\wedge}\neg q? \cup (\neg p{\wedge}\neg q?; p{\leftarrow}\top)
\end{aligned}$$

Dans la dernière équivalence, p et q doivent être différents.

$$\begin{aligned}
\|p{\leftarrow}\varphi\| &= \{(v, v') \ : \ v' = v \cup \{p\} \text{ if } v \in \|\varphi\| \text{ et} \\
&\qquad\qquad v' = v \setminus \{p\} \text{ if } v \notin \|\varphi\|\} \\
\|\varphi?\| &= \{(v, v) \ : \ v \in \|\varphi\|\} \\
\|\pi; \pi'\| &= \|\pi\| \circ \|\pi'\| \\
\|\pi \cup \pi'\| &= \|\pi\| \cup \|\pi'\| \\
\|\pi^*\| &= \bigcup_{k \in \mathbb{N}_0} (\|\pi\|)^k \\
\|\pi^{-1}\| &= \|\pi\|^{-1} \\
\|p\| &= \{v \ : \ p \in v\} \\
\|\top\| &= \mathbb{V} \ = \ 2^{\mathbb{P}} \\
\|\bot\| &= \emptyset \\
\|\neg\varphi\| &= 2^{\mathbb{P}} \setminus \|\varphi\| \\
\|\varphi \vee \psi\| &= \|\varphi\| \cup \|\psi\| \\
\|\langle\pi\rangle\varphi\| &= \{v \ : \ \text{il existe est } v' \text{ t.q. } (v, v') \in \|\pi\| \text{ et } v' \in \|\varphi\|\}
\end{aligned}$$

T 1 – Interprétation des connecteurs de DL-PA

Une *expression* est une formule ou un programme. L'équivalence est préservée par le remplacement d'une sous-expression par une expression équivalente.

Dans l'article [1] il est montré que contrairement à PDL, l'étoile de Kleene peut être éliminé en DL-PA : pour chaque programme π il existe un programme équivalent π' sans étoile de Kleene π'. Cependant, l'élimination est non polynomiale. Montrons que l'opérateur converse peut également être éliminé. La table 2 contient les équivalences pertinents (aussi appelés axiomes de réduction). Observons que tout comme pour l'étoile de Kleene, la réduction est non polynomiale. Fort heureusement, pour nos besoins dans le présent article des affectations de variables p à $\top$, $\bot$, et à une autre variable propositionnelle q suffisent. De telles affectations peuvent être réduites au prix d'une croissance linéaire de la taille du programme.

Une formule φ est DL-PA *valide* si elle est équivalente à $\top$, c.-à-d si $\|\varphi\| = 2^{\mathbb{P}}$. Elle est DL-PA *satisfaisable* si elle n'est pas équivalente à $\bot$, c.-à-d si $\|\varphi\| \neq \emptyset$. Par exemple, les formules $\langle p{\leftarrow}\top\rangle\top$ et $\langle p{\leftarrow}\top\rangle\varphi \leftrightarrow \neg\langle p{\leftarrow}\top\rangle\neg\varphi$ sont DL-PA valides. D'autres exemples de validités sont $\langle p{\leftarrow}\top\rangle p$ et $\langle p{\leftarrow}\bot\rangle\neg p$. Les validités $\varphi \rightarrow [\pi]\langle\pi^{-1}\rangle\varphi$ et $\varphi \rightarrow [\pi^{-1}]\langle\pi\rangle\varphi$ sont hérités de la logique PDL avec converse (elles sont parfois appelés les axiomes de conversion). De plus, $\varphi \rightarrow [\pi]\varphi$ est valide si et seulement si $\langle\pi^{-1}\rangle\varphi \rightarrow \varphi$ est valide. Les deux sens du "si et seulement si" correspondent à deux règles d'inférence qu'on appelle les *règles de conversion*.

Observons que si p n'apparait pas dans φ alors les formules $\varphi \rightarrow \langle p{\leftarrow}\top\rangle\varphi$ et $\varphi \rightarrow \langle p{\leftarrow}\bot\rangle\varphi$ sont toutes les deux valides. Ceci est dû à la propriété sémantique suivante qui

$$
\begin{aligned}
(\varphi?)^{-1} &\equiv \varphi? \\
(\pi_1;\pi_2)^{-1} &\equiv \pi_2^{-1};\pi_1^{-1} \\
(\pi_1 \cup \pi_2)^{-1} &\equiv \pi_1^{-1} \cup \pi_2^{-1} \\
(\pi^*)^{-1} &\equiv (\pi^{-1})^* \\
(p{\leftarrow}\varphi)^{-1} &\equiv ((\varphi?; p{\leftarrow}\top) \cup (\neg\varphi?; p{\leftarrow}\bot))^{-1} \\
&\equiv (p{\leftarrow}\top^{-1}; \varphi?^{-1}) \cup (p{\leftarrow}\bot^{-1}; (\neg\varphi?)^{-1} \\
&\equiv ((p? \cup (p?; p{\leftarrow}\bot)); \varphi?) \cup \\
&\quad ((\neg p? \cup (\neg p?; p{\leftarrow}\top)); \neg\varphi?) \\
&\equiv (p?;\varphi?) \cup (p?; p{\leftarrow}\bot; \varphi?) \cup \\
&\quad (\neg p?; \neg\varphi?) \cup (\neg p?; p{\leftarrow}\top; \neg\varphi?)
\end{aligned}
$$

T 2 – Axiomes de réduction pour l'opérateur converse

nous sera utile plus tard.

Proposition 1. *Soit $\mathbb{P}_\varphi \cap P = \emptyset$, c.-à-d aucune des variables de P apparait dans φ. Alors $v \cup P \in \|\varphi\|$ ssi $v \setminus P \in \|\varphi\|$.*

3.3 Complexité du langage entier

Il est démontré dans [1] que la vérification de modèle et le test de satisfaisabilité sont tous les deux EXPTIME complets pour le fragment de DL-PA sans l'opérateur de conversion et où les formules φ dans les programmes atomiques $p{\leftarrow}\varphi$ sont restreintes à ou bien $\top$ ou $\bot$. Pour ce qui est de notre langage, les bornes inférieures pour les deux problèmes sont bien-sûr hérités du DL-PA de [1].

La borne supérieur pour le problème de satisfaisabilité est établie dans [1] moyennant une transformation polynomiale dans le problème de satisfaisabilité de PDL. Une inspection de la démonstration montre que :

- elle peut être généralisée à des affectations quelconques ;
- la transformation peut être étendue afin d'inclure l'opérateur converse, traduisant ainsi notre langage en PDL avec converse.

Le problème de satisfaisabilité de notre DL-PA a donc la même complexité que celui de PDL avec converse : il est EXPTIME complet.

La borne supérieur pour le problème de vérification de modèle peut être établi de la même manière qu'en [1], à savoir par une transformation polynomiale vers le problème de satisfaisabilité : nous utilisons que $v \in \|\varphi\|$ si et seulement si la formule $\varphi \wedge (\bigwedge_{p\in v} p) \wedge (\bigwedge_{p\notin v} \neg p)$ est satisfaisable. En conséquence, le problème de vérification de modèle de notre DL-PA est également EXPTIME complet.

$$
\begin{aligned}
\mathsf{chSome}(P) &= (p_1{\leftarrow}\top \cup p_1{\leftarrow}\bot); \cdots; (p_n{\leftarrow}\top \cup p_n{\leftarrow}\bot) \\
\mathsf{chOne}(P) &= p_1{\leftarrow}\neg p_1 \cup \cdots \cup p_n{\leftarrow}\neg p_n \\
\mathsf{store}(P) &= p'_1{\leftarrow}p_1; \cdots; p'_n{\leftarrow}p_n \\
\mathsf{restoreOne}(P) &= (p_1 \oplus p'_1?; p_1{\leftarrow}p'_1) \cup \cdots \cup \\
&\quad (p_n \oplus p'_n?; p_n{\leftarrow}p'_n) \\
\mathsf{restoreSome}(P) &= (\mathsf{skip} \cup (p_1 \oplus p'_1?; p_1{\leftarrow}p'_1)); \cdots; \\
&\quad (\mathsf{skip} \cup (p_n \oplus p'_n?; p_n{\leftarrow}p'_n))
\end{aligned}
$$

T 3 – Quelques programmes DL-PA utiles

3.4 Complexité du fragment sans étoile de Kleene

La complexité des problèmes de décision pour le fragment sans étoile de Kleene du langage de [1] est établi dans [8], où il est montré qu'il est PSPACE complet et pour la vérification de modèle et pour le test de satisfaisabilité.

En ce qui concerne la vérification de modèle, la borne inférieure est héritée clairement par notre fragment sans étoile (qui possède en plus l'opérateur converse et des programmes atomiques plus généraux). De plus, l'algorithme de vérification de modèle de [8] (qui travaille dans un espace polynomial) peut être étendu à notre fragment plus général. [1]

En ce qui concerne le test de satisfaisabilité, la borne inférieur de [8] est héritée. La borne supérieure peut être établie de la même manière qu'en [8] : étant donné une formule φ, on devine (non-deterministiquement) une valuation v et vérifie ensuite si $v \in \|\varphi\|$. La vérification de modèle étant en PSPACE, le test de satisfaisabilité doit donc être en NPSPACE, et la classe NPSPACE est identique à la classe PSPACE grâce au théorème de Savitch.

4 Quelques programmes utiles

La table 3 contient quelques programmes DL-PA qui vont nous être utiles dans la suite. Dans cette table, P est l'ensemble de variables $\{p_1, \ldots, p_n\}$ et P' est l'ensemble de variables p' tel que p est dans P et p' est 'nouveau' : nous supposons que p' n'apparait dans la formule qui nous intéresse.

Le programme $\mathsf{chSome}(P)$ modifie d'une manière non-déterministe la valeur de vérité de quelques-unes des va-

1. Comme nous l'avons annoncé déjà, les programmes atomiques $p{\leftarrow}\varphi$ dont nous avons besoin dans la section suivante ne nécessitent pas l'affectation de formules complexes φ : ces dernières peuvent être restreints à $\top$, $\bot$ ou des variables propositionnelles. L'opérateur converse peut être éliminé de telles affectations au prix d'une croissance seulement linéaire de la taille du programme : il suffit d'appliquer d'abord les équivalences de la table 2 afin de pousser l'opérateur converse vers l'intérieur des programmes pour ensuite l'éliminer lorsqu'il fait face à une variable propositionnelle moyennant les équivalences de programme pour $p{\leftarrow}\top^{-1}$, $p{\leftarrow}\bot^{-1}$, $p{\leftarrow}q^{-1}$ et $p{\leftarrow}\neg p^{-1}$ de la section 3.1.

riables de P. Considérons le programme $\mathsf{chSome}(\mathbb{P}_A); A?$: il permet d'accéder a toutes les A-valuations. Le fait qu'une formule booléenne A est satisfaisable peut être exprimée en DL-PA par la formule $\langle \mathsf{chSome}(\mathbb{P}_A)\rangle A$.

Le programme $\mathsf{chOne}(P)$ modifie la valeur de vérité d'une des variables de P.

Le programme $\mathsf{store}(P)$ stocke la valeur de vérité de chaque variable p dans la variable nouvelle p'. Si p et p' ont des valeurs de vérité différentes alors nous disons que p est *marqué* ; sinon nous disons que p est *non-marqué*.

Le programme $\mathsf{restoreOne}(P)$ restaure une des variables marquées p_k de P, c.-à-d qui diffère de sa copie p'_k.

Finalement, $\mathsf{restoreSome}(P)$ restaure zéro ou plus des variables marquées de P.

Notons que la longueur de chacun des programmes de la Table 3 est linéaire dans la cardinalité de P.

5 WSS

Winslett's standard sémantics (WSS) [18] fournit le plus faible des opérateurs de mise à jour, dans le sens qu'il préserve relativement peu d'informations de la base. C'est aussi un opérateur sensible à la syntaxe.

5.1 Semantique du WSS

Soit v une valuation et A une formule propositionnelle. La *mise à jour WSS de v par A* est

$$v \diamond^{\mathsf{wss}} A = \{v' \ : \ v' \in \|A\| \text{ et pour tout } p \notin \mathbb{P}_A, \ p \in v \text{ ssi } p \in v'\}$$

L'ensemble $v \diamond^{\mathsf{wss}} A$ est donc l'ensemble de A-valuations en accord avec v sur toutes les variables n'apparaissant pas dans A. Par exemple,

$$\emptyset \diamond^{\mathsf{wss}} p \vee q \ = \ \{\{p\}, \{q\}, \{p, q\}\}$$

L'opérateur WSS de mise à jour dépend de la syntaxe : en fonction de leur signature, des formules logiquement équivalentes A et A' peuvent modifier la valuation v de manière différente.

Soient A et B des formules propositionnelles. La *mise à jour WSS de B par A* est obtenue en collectionnant les mises à jours de chaque B-valuation par A :

$$B \diamond^{\mathsf{wss}} A = \bigcup_{v \in \|B\|} v \diamond^{\mathsf{wss}} A$$

Ainsi, l'opération $\diamond^{\mathsf{wss}}$ prend en entrée deux formules propositionnelles et retourne un ensemble de valuations. Par exemple,

$$\begin{aligned}
\neg p \wedge \neg q \diamond^{\mathsf{wss}} p &= \|p \wedge \neg q\| \\
\neg p \wedge \neg q \diamond^{\mathsf{wss}} p \vee q &= \|p \vee q\| \\
\neg q \diamond^{\mathsf{wss}} p &= \|p \wedge \neg q\| \\
\neg p \vee \neg q \diamond^{\mathsf{wss}} p &= \|p\|
\end{aligned}$$

5.2 Expression de l'opérateur WSS en DL-PA

Montrons maintenant comment exprimer les mises à jour WSS en DL-PA.

Proposition 2. *Soient A, B des formules propositionnelles. Soit π_A^{wss} le programme $\mathsf{chSome}(\mathbb{P}_A); A?$. Alors*

$$B \diamond^{\mathsf{wss}} A = \|\langle (\pi_A^{\mathsf{wss}})^{-1}\rangle B\|$$

Par exemple, considérons $A = p$. Alors

$$\begin{aligned}
\pi_p^{\mathsf{wss}} &= \mathsf{chSome}(\{p\}); p? \\
&= (p{\leftarrow}\top \cup p{\leftarrow}\bot); p? \\
&\equiv (p{\leftarrow}\top; p?) \cup (p{\leftarrow}\bot; p?) \\
&\equiv p{\leftarrow}\top
\end{aligned}$$

Son converse $(\pi_p^{\mathsf{wss}})^{-1}$ peut être simplifié moyennant les équivalences de programme de la section 3.1 comme suit :

$$\begin{aligned}
(\pi_p^{\mathsf{wss}})^{-1} &\equiv (p{\leftarrow}\top)^{-1} \\
&\equiv p? \ \cup \ (p?; p{\leftarrow}\bot)
\end{aligned}$$

Considérons maintenant la base $B = \neg p \wedge q$ et la nouvelle information $A = p$. Nous avons :

$$\begin{aligned}
\langle (\pi_A^{\mathsf{wss}})^{-1}\rangle B &\leftrightarrow \langle p? \cup (p?; p{\leftarrow}\bot)\rangle(\neg p \wedge q) \\
&\leftrightarrow \langle p?\rangle(\neg p \wedge q) \vee \langle p?\rangle\langle p{\leftarrow}\bot\rangle(\neg p \wedge q) \\
&\leftrightarrow (p \wedge \neg p \wedge q) \vee \langle p?\rangle\langle p{\leftarrow}\bot\rangle(\neg p \wedge q) \\
&\leftrightarrow \bot \vee \langle p?\rangle\langle p{\leftarrow}\bot\rangle(\neg p \wedge q) \\
&\leftrightarrow \langle p?\rangle(\langle p{\leftarrow}\bot\rangle \neg p \wedge \langle p{\leftarrow}\bot\rangle q) \\
&\leftrightarrow \langle p?\rangle(\top \wedge q) \\
&\leftrightarrow p \wedge q
\end{aligned}$$

Ainsi et sans surprises, la mise à jour de $\neg p \wedge q$ par p est équivalent à $p \wedge q$.

6 PMA

L'opérateur de mise à jour du *Possible Models Approach* (PMA) de Winslett [16, 17] est plus conservateur que le WSS, dans le sens qu'il préserve plus d'informations de la base B. Ceci est obtenu au travers un principe de changement minimal.

6.1 Sémantique de l'opérateur PMA

Nous devons d'abord définir une notion de distance entre deux valuations. La *différence symétrique* entre deux valuations $v, v' \subseteq \mathbb{P}$ est définie par

$$v \dot{-} v' = (v \setminus v') \cup (v' \setminus v)$$

Par exemple, $\{p, q\} \dot{-} \{q, r, s\} = \{p, r, s\}$.

La *mise à jour PMA de v par la formule A* est

$$v \diamond^{\mathsf{pma}} A = \{v' \ : \ v' \in \|A\| \text{ et il n'y a pas de } v'' \text{ tel que } v \dot{-} v'' \subset v \dot{-} v'\}$$

Ainsi, l'ensemble $v \diamond^{\mathsf{pma}} A$ est l'ensemble d'A-valuations qui sont les plus proches de v par rapport à la différence symétrique. Par exemple,

$$\emptyset \diamond^{\mathsf{pma}} p \vee q = \{\{p\}, \{q\}\}$$
$$\emptyset \diamond^{\mathsf{pma}} (p \wedge q) \vee r = \{\{p, q\}, \{r\}\}$$

Soient A et B deux formules propositionnelles. La *mise à jour PMA de B par A* collectionne les mises à jour de chaque B-valuation par A :

$$B \diamond^{\mathsf{pma}} A = \bigcup_{v \in \|B\|} v \diamond^{\mathsf{pma}} A$$

Par exemple,

$$\begin{aligned} \neg p \wedge \neg q \diamond^{\mathsf{pma}} p &= \|p \wedge \neg q\| \\ \neg q \diamond^{\mathsf{pma}} p &= \|p \wedge \neg q\| \\ \neg p \wedge \neg q \diamond^{\mathsf{pma}} p \vee q &= \|p \oplus q\| \\ \neg p \wedge \neg q \wedge \neg r \diamond^{\mathsf{pma}} (p \wedge q) \vee r &= \|(p \wedge q) \oplus r\| \end{aligned}$$

6.2 Expression de l'opérateur PMA en DL-PA

Nous définissons maintenant des programmes de mise à jour π_A^{pma} dont la longueur est linéaire dans la longueur de A. Ceci permet de transformer polynomialement la mise à jour $B \diamond^{\mathsf{pma}} A$ en DL-PA.

Il est tentant de définir π_A^{pma} comme étant le programme $\mathsf{while}\ \neg A\ \mathsf{do}\ \mathsf{chOne}(\mathbb{P}_A)$ (où while est défini en section 3.1). Tout en lui ressemblant, ce programme n'a pas exactement le même comportement que la mise à jour PMA. En effet, l'interprétation de

$$\mathsf{while}\ \neg((p \wedge q) \vee r)\ \mathsf{do}\ \mathsf{chOne}(\{p, q, r\})$$

relie la valuation vide aux quatre valuations $\{p, q\}$, $\{r\}$, $\{p, r\}$, et $\{q, r\}$. Cependant, nous avons vu plus haut que la mise à jour PMA de la valuation vide résulte seulement en les premiers deux de ces valuations.

La contre-partie DL-PA exacte de la mise à jour PMA est un peu plus compliqué que ça. La manière la plus simple est de définir π_A^{pma} comme étant le programme

$$A? \ \cup \ \left(\neg A?; \ \bigcup_{\emptyset \subset P \subseteq \mathbb{P}_A} \mathsf{dist}(A, P)? \ ; \ \mathsf{chSome}(P) \ ; \ A? \right)$$

où $\mathsf{dist}(A, P)$ est défini comme :

$$\mathsf{dist}(A, P) = \langle \mathsf{chSome}(P) \rangle A \wedge \neg \langle \bigcup_{p \in P} \mathsf{chSome}(P \setminus \{p\}) \rangle A$$

pour P non vide. La formule $\mathsf{dist}(A, P)$ est vraie dans une valuation v exactement quand il y a une plus proche A-valuation dont la distance symétrique à v est P. Malheureusement la longueur de ce programme est exponentielle en la longueur de A.

Notre transformation polynomiale du PMA utilise le stockage des valeurs des variables.

Proposition 3. *Soient A, B des formules propositionnelles. Soit* π_A^{pma} *le programme suivant :*

$\mathsf{chSome}(\mathbb{P}_A)$;
$A?$;
$\mathsf{store}(\mathbb{P}_A)$;
$([\mathsf{restoreOne}(\mathbb{P}_A); \mathsf{restoreSome}(\mathbb{P}_A)]\neg A)?$

Alors $B \diamond^{\mathsf{pma}} A = \|\langle (\pi_A^{\mathsf{pma}})^{-1} \rangle B\|$.

Intuitivement, le programme π_A^{pma} va vers une A-valuation et vérifie si tous les chemins y menant contiennent exclusivement des $\neg A$-valuations. En d'autres termes, une A-valuation est écartée si une autre A-valuation peut être atteinte en changeant moins de variables. Par exemple,

$$\begin{aligned} \pi_p^{\mathsf{pma}} = \ & (p{\leftarrow}\top \cup p{\leftarrow}\bot); \\ & p?; \\ & p'{\leftarrow}p; \\ & ([p \oplus p'?; p{\leftarrow}p'; (\mathsf{skip} \cup (p \oplus p'?; p{\leftarrow}p'))]\neg p)? \\ \equiv \ & p{\leftarrow}\top; p'{\leftarrow}p \end{aligned}$$

Le dernier pas est correct : le dernier test dans la séquence

$$([p \oplus p'?; p{\leftarrow}p'; (\mathsf{skip} \cup (p \oplus p'?; p{\leftarrow}p'))]\neg p)?$$

est équivalent à skip car le premier test $p \oplus p'?$ à l'intérieur de l'opérateur modal [.] échoue. Comme p' est nouveau, l'effet net de ce programme de mise à jour π_A^{pma} est juste $p{\leftarrow}\top$, comme espéré.

Observons que la longueur de π_A^{pma} est linéaire en la longueur de la formule d'entrée A.

7 Forbus

L'opérateur de mise à jour de Forbus [3] est encore plus conservateur que le PMA de Winslett. Il est basé sur la minimisation de la distance de Hamming entre valuations.

7.1 Sémantique de l'opérateur de Forbus

La *distance de Hamming* entre deux valuations v et v' est définie comme étant $\mathsf{card}(v \dot{-} v')$, c.-à-d la cardinalité de la différence symétrique entre v et v'. Par exemple, la distance de Hamming entre $\{p, q\}$ et $\{q, r, s\}$ est $\mathsf{card}(\{p, r, s\}) = 3$.

Forbus définit la mise à jour de v par A comme suit :

$$v \diamond^{\text{forbus}} A = \{v' : v' \in \|A\| \text{ et il n'y a pas de } v'' \text{ tel que } \mathsf{card}(v \dot{-} v'') < \mathsf{card}(v \dot{-} v')\}$$

L'ensemble $v \diamond^{\text{forbus}} A$ est donc l'ensemble d'A-valuations qui sont les plus proches de v par rapport à la distance de Hamming. Par exemple,

$$\emptyset \diamond^{\text{forbus}} p \vee q = \{\{p\}, \{q\}\}$$
$$\emptyset \diamond^{\text{forbus}} (p \wedge q) \vee r = \{\{r\}\}$$

Soient A et B des formules propositionnelles. Tout comme la mise à jour PMA, *la mise à jour de Forbus de B par A* est la mise à jour point-par-point des B-valuations par A :

$$B \diamond^{\text{forbus}} A = \bigcup_{v \in \|B\|} v \diamond^{\text{forbus}} A$$

Par exemple,

$$\neg p \wedge \neg q \diamond^{\text{forbus}} p \vee q = \|p \oplus q\|$$
$$\neg p \wedge \neg q \wedge \neg r \diamond^{\text{forbus}} (p \wedge q) \vee r = \|\neg p \wedge \neg q \wedge r\|$$

Le dernier exemple illustre la différence avec l'opérateur de mise à jour PMA.

7.2 Expression de l'opérateur de Forbus en DL-PA

Définissons d'abord la formule $\mathsf{hdist}(A, m)$, pour $m \geq 1$:

$$\mathsf{hdist}(A, m) = \langle(\mathsf{chOne}(P))^m\rangle A \wedge \neg\langle(\mathsf{chOne}(P))^{m-1}\rangle A$$

La formule $\mathsf{dist}(A, m)$ est vraie à la valuation v exactement quand les plus proches A-valuations dans le sens de la distance de Hamming diffèrent en m variables de v.

Nous définissons maintenant des programmes de mise à jour π_A^{forbus} dont la longueur est cubique en la longueur de A. Ceci permet une transformation polynomiale du problème de mise à jour $B \diamond^{\text{forbus}} A$ en DL-PA.

Proposition 4. *Soient A, B des formules propositionnelles. Soit π_A^{forbus} le programme suivant :*

$$A? \cup \left(\bigcup_{1 \leq m \leq \mathsf{card}(\mathbb{P}_A)} \mathsf{hdist}(A, m)?; (\mathsf{chOne}(\mathbb{P}_A))^m; A? \right)$$

Alors $B \diamond^{\text{forbus}} A = \|\langle(\pi_A^{\text{forbus}})^{-1}\rangle B\|$.

La longueur de π_A^{forbus} est cubique en la longueur de A.

8 Conclusion

Nous avons montré comment exprimer plusieurs opérateurs de changement de croyance bien connus dans un cadre unique : la logique dynamique des affectations propositionnelles DL-PA. Ceci nous a permis de munir ces opérateurs jusqu'à maintenant seulement définis d'une manière sémantique d'une véritable contre-partie syntaxique.

Revenons au problème de conséquence d'un changement de croyances : décider si l'implication

$$(B \circ A) \rightarrow C$$

est valide. Il s'agit d'un problème de décision, et des bornes exactes sont connus pour chacun des opérateurs que nous avons considéré : il est NP complet pour l'opérateur WSS de mise à jour, tandis qu'il est Σ_p^2 complet pour le PMA de Winslett et l'opérateur de Forbus. Nos traductions sont dans le fragment sans étoile de Kleene du langage de DL-PA. La borne supérieure PSPACE de ce fragment nous donne ainsi une borne supérieure PSPACE pour la complexité de décider si $(B \circ A) \rightarrow C$ est valide. Cette borne est donc clairement suboptimale. Cependant, l'enchâssement des opérateurs modaux dans nos programmes de mise à jour est borné. Nous prévoyons dans des travaux futurs d'étudier de tels fragments de DL-PA sans l'étoile de Kleene où l'alternance des quantificateurs est bornée.

Remerciements

Un grand merci est dû aux deux relecteurs d'IAF 2013 : leur commentaires compétents m'ont permis de corriger plusieurs petites erreurs dans la version soumise, ainsi qu'une erreur plus importante. Ils m'ont également défié d'étendre les traductions (qui dans la version soumise couvraient seulement les mises à jour WSS et PMA) et m'ont motivé de raconter une histoire plus complète qui tient davantage les promesses du titre.

Références

[1] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments : a well-behaved variant of PDL. In Orna Kupferman, editor, *Logic in Computer Science (LICS), New Orleans, June 25-28, 2013*, http ://www.ieee.org/, juin 2013. IEEE.

[2] Mukesh Dalal. Investigations into a theory of knowledge base revision : preliminary report. In *Proc. 7th Conf. on Artificial Intelligence (AAAI'88)*, pages 475–479, 1988.

[3] Kenneth D. Forbus. Introducing actions into qualitative simulation. In N. S. Sridharan, editor, *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, pages 1273–1278. Morgan Kaufmann Publishers, 1989.

[4] David Harel. Dynamic logic. In Dov M. Gabbay and Franz Günthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. D. Reidel, Dordrecht, 1984.

[5] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[6] Andreas Herzig. The PMA revisited. In Luigia Carlucci Aiello and Stuart Shapiro, editors, *Proc. Int. Conf. on Knowledge Representation and Reasoning (KR'96)*, pages 40–50. Morgan Kaufmann Publishers, November 1996.

[7] Andreas Herzig, Jérôme Lang, and Pierre Marquis. Propositional update operators based on formula/literal dependence. *ACM Transactions on Computational Logic (TOCL)*, to appear.

[8] Andreas Herzig, Emiliano Lorini, Frédéric Moisan, and Nicolas Troquard. A dynamic logic of normative systems. In Toby Walsh, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 228–233, Barcelona, 2011. IJCAI/AAAI. Erratum at `http://www.irit.fr/~Andreas.Herzig/P/Ijcai11.html`.

[9] Andreas Herzig, Emiliano Lorini, and Nicolas Troquard. A dynamic logic of institutional actions. In João Leite and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems (CLIMA)*, volume 6814 of *LNCS/LNAI*, pages 295–311, Barcelona, juillet 2011. Springer Verlag.

[10] Andreas Herzig and Omar Rifi. Propositional belief base update and minimal change. *Artificial Intelligence Journal*, 115(1) :107–138, November 1999.

[11] Wiebe van der Hoek, Dirk Walther, and Michael Wooldridge. On the logic of cooperation and the transfer of control. *J. of AI Research (JAIR)*, 37 :437–477, 2010.

[12] Wiebe van der Hoek and Michael Wooldridge. On the dynamics of delegation, cooperation and control : a logical account. In *Proc. AAMAS'05*, 2005.

[13] Wiebe van der Hoek and Michael Wooldridge. On the logic of cooperation and propositional control. *Artif. Intell.*, 164(1-2) :81–119, 2005.

[14] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In Peter Gärdenfors, editor, *Belief revision*, pages 183–203. Cambridge University Press, 1992. (preliminary version in Allen, J.A., Fikes, R., and Sandewall, E., eds., Principles of Knowledge Representation and Reasoning : Proc. 2nd Int. Conf., pages 387–394. Morgan Kaufmann Publishers, 1991).

[15] Jérôme Lang. Belief update revisited. In *Proc. of the 10th International Joint Conference on Artificiel Intelligence (IJCAI'07)*, pages 2517–2522, 2007.

[16] Mary-Anne Winslett. Reasoning about action using a possible models approach. In *Proc. 7th Conf. on Artificial Intelligence (AAAI'88)*, pages 89–93, St. Paul, 1988.

[17] Mary-Anne Winslett. *Updating Logical Databases*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[18] Mary-Anne Winslett. Updating logical databases. In Dov M. Gabbay, Chris J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 133–174. Oxford University Press, 1995.