HAL
archives-ouvertes.fr

Un algorithme d'adaptation avec des cas exprimés dans la logique de descriptions ALC

Julien Cojan, Jean Lieber

▶ To cite this version:

Julien Cojan, Jean Lieber. Un algorithme d'adaptation avec des cas exprimés dans la logique de descriptions ALC. 18ème Atelier " Raisonnement à Partir de Cas " RàPC 2010, Jun 2010, Strasbourg, France. pp.37-48. inria-00506094

HAL Id: inria-00506094
https://hal.inria.fr/inria-00506094

Submitted on 27 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Un algorithme d'adaptation avec des cas exprimés dans la logique de descriptions $\mathcal{ALC}$

Julien Cojan, Jean Lieber
UHP-Nancy 1 – LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy 2-UHP)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{Julien.Cojan, Jean.Lieber}@loria.fr

Résumé

Cet article décrit un algorithme d'adaptation pour un système de raisonnement à partir de cas où les cas et les connaissances du domaine sont exprimés dans la logique de descriptions expressive $\mathcal{ALC}$. Le principe consiste à supposer que le cas source à adapter résout le cas cible, ce qui entraîne des contradictions avec le contexte du cas cible et avec les connaissances du domaine. L'adaptation consiste alors à résoudre ces contradictions. L'algorithme est fondé sur une extension de la méthode classique des tableaux employée pour le calcul d'inférences déductives en $\mathcal{ALC}$.

Mots clés : adaptation, logique de descriptions, $\mathcal{ALC}$, algorithme des tableaux

1 Introduction

L'adaptation est une étape de certains systèmes de raisonnement à partir de cas (RÀPC) qui consiste à modifier un cas source pour qu'il réponde à une nouvelle situation, le cas cible. Une approche de l'adaptation consiste à utiliser un opérateur de révision des connaissances[1], i.e., un opérateur qui modifie de façon minimale un ensemble de connaissances afin d'être cohérent avec d'autres connaissances [1]. L'idée est de considérer la connaissance « Le cas source résout le problème cible » et de réviser cette connaissance par les contraintes données par le cas cible et les connaissances du domaine. Cela a été étudié pour des cas représentés en logique propositionnelle dans [12]. Puis, cela a été étudié dans un formalisme plus expressif, incluant des contraintes numériques [3] et, après cela, étendu à la problématique de combinaison des cas dans ce formalisme [4].

Dans cet article, cette approche de l'adaptation est étudiée pour les cas représentés dans la logique de descriptions (LD) $\mathcal{ALC}$ (qui est la plus simple des « LD expressives »). Le choix des LD comme formalismes pour le RÀPC peut être motivé de plusieurs façons. D'abord, elles étendent les formalismes attributs-valeurs, qui sont souvent utilisés pour le RÀPC (voir par exemple [11]) et ils sont similaires au formalisme des *memory organisation packets* (MOPs) utilisé dans certaines des premières applications du RÀPC [13]. Plus généralement, elles sont conçues comme des compromis entre l'expressivité et la « complexité pratique[2] ». Ensuite, elles ont une sémantique bien définie et elles sont étudiées systématiquement depuis plusieurs décennies maintenant. Enfin, plusieurs implantations efficaces sont disponibles gratuitement, offrant des services qui peuvent être utilisés par les systèmes de RÀPC, en particulier pour la remémoration et l'organisation de la base de cas.

La suite de l'article est organisée comme suit. La section 2 présente la LD $\mathcal{ALC}$ et, en particulier, l'algorithme des tableaux, qui est à la base des inférences déductives pour la plupart des implantations actuelles. Un exemple est présenté dans cette section pour illustrer des notions relativement complexes pour un lecteur qui ne serait pas familier des LD. Cet algorithme des tableaux est étendu pour effectuer un processus d'adaptation, ce qui fait l'objet de la section 3. La section 4 discute notre contribution et présente d'autres travaux utilisant

[1] Le terme anglais est *belief revision*, qui se traduit littéralement par « révision des croyances », que nous préférons traduire par « révision des connaissances », parce que, d'une part, le terme « croyance » a des connotations religieuses hors de propos et, d'autre part, que la distinction en débat *belief-knowledge* sort du cadre de nos travaux.

[2] La complexité dans le pire cas des LD expressives est très élevée, puisque la plupart des inférences sont PSPACE-difficiles. Cependant, en pratique, cette complexité reste raisonnable.

les LD pour le RÀPC, ainsi que d'autres travaux proches. La section 5 conclut l'article et présente quelques perspectives.

2 La logique de descriptions $\mathcal{ALC}$

Les logiques de descriptions [2] forment une famille de logiques classiques équivalentes à des fragments décidables de la logique du premier ordre (L1O). Elles prennent une importance grandissante en représentation des connaissances. $\mathcal{ALC}$ est la plus simple des *LD expressives*, c'est à dire des LD qui étendent la logique propositionnelle. L'exemple culinaire présenté dans cette section est inspiré par le *Computer Cooking Contest*.

2.1 Syntaxe

Les élément du langage de représentation de $\mathcal{ALC}$ sont les concepts, les rôles, les instances et les formules.

Intuitivement, un *concept* représente un sous-ensemble du domaine d'interprétation. Un concept est soit un *concept atomique* (c.-à-d. un nom de concept), ou une expression conceptuelle de l'une des forme suivantes : $\top$, $\bot$, $\neg\mathtt{C}$, $\mathtt{C} \sqcap \mathtt{D}$, $\mathtt{C} \sqcup \mathtt{D}$, $\exists\mathtt{r}.\mathtt{C}$, et $\forall\mathtt{r}.\mathtt{C}$, où C et D sont des concepts (atomiques ou non) et r est un rôle. À un concept peut être associé une formule du premier ordre avec une variable libre x. Par exemple, au concept

$$\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pomme} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \forall\mathtt{ing}.\neg\mathtt{Cannelle} \tag{1}$$

peut être associé la formule du premier ordre

$$\mathtt{Tarte}(x) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Pomme}(y)) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Pâte}(y)) \wedge (\forall y, \mathtt{ing}(x,y) \Rightarrow \neg\mathtt{Cannelle}(y))$$

Intuitivement, un *rôle* représente une relation binaire. Dans $\mathcal{ALC}$ les rôles sont atomiques, c.-à-d. des noms de rôles. Leur pendant en L1O sont les prédicats binaires. Le rôle apparaissant dans (1) est ing.

Intuitivement, une *instance* représente un élément du domaine d'interprétation. Dans $\mathcal{ALC}$ les instance sont atomiques, c.-à-d. des noms d'instances. Leur pendant en L1O sont les constantes.

Il y a quatre types de formules dans $\mathcal{ALC}$ (suivies par leurs significations) : (1) $\mathtt{C} \sqsubseteq \mathtt{D}$ (C est plus spécifique que D), (2) $\mathtt{C} \equiv \mathtt{D}$ (C et D sont des concepts équivalents), (3) $\mathtt{C}(\mathtt{a})$ (a est une instance de C), et (4) $\mathtt{r}(\mathtt{a},\mathtt{b})$ (r relie a à b), où C et D sont des concepts, a et b sont des instances, et r est un rôle. Les formules de types (1) et (2) sont appelées des *formules terminologiques*. Les formules de types (3) et (4) sont appelées des *formules assertionnelles*, ou simplement des *assertions*.

Une base de connaissances BC en $\mathcal{ALC}$ est un ensemble de formules $\mathcal{ALC}$. La partie terminologique (ou *TBox* pour *terminological box*) de BC est l'ensemble de ses formules terminologiques. La partie assertionnelle (ou *ABox* pour *assertional box*) de BC est l'ensemble de ses formules assertionnelles.

Par exemple, la TBox suivante représente les connaissances du domaine (CD) de notre exemple (avec en commentaire les significations) :

$$\begin{aligned} \mathtt{CD} = \{&\mathtt{Pomme} \sqsubseteq \mathtt{Piridion}, && \mathtt{Poire} \sqsubseteq \mathtt{Piridion}, && \textit{Les pommes et les poires sont des piridions.} \\ &\mathtt{Piridion} \sqsubseteq \mathtt{Pomme} \sqcup \mathtt{Poire}\} && && \textit{Un piridion est soit une pomme soit une poire.} \end{aligned} \tag{2}$$

Notez que la dernière formule est une simplification : en réalité, il y a d'autres piridions que des pommes et des poires. Cette simplification rendra l'exemple plus facile à appréhender mais, comme cela sera expliqué dans la note 7, on pourrait s'en passer.

Dans notre exemple, les seuls cas considérés sont le cas source et le cas cible. Ils sont représentés dans l'ABox suivante :

$$\{\mathtt{Source}(\sigma), \mathtt{Cible}(\gamma)\} \tag{3}$$

$$\text{avec} \quad \mathtt{Source} = \mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.\mathtt{Pomme} \qquad \text{et} \qquad \mathtt{Cible} = \mathtt{Tarte} \sqcap \forall\mathtt{ing}.\neg\mathtt{Pomme} \tag{4}$$

Le cas source est alors représenté par l'instance σ, qui est une tarte avec les types d'ingrédients pâte et pomme. Le cas cible est représenté par l'instance γ déclarant qu'une tarte sans pomme est demandée.

2.2 Sémantique

Une interprétation est un couple $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ où $\Delta_\mathcal{I}$ est un ensemble non vide (le *domaine d'interprétation*) et où $\cdot^\mathcal{I}$ associe à un concept $\mathtt{C}$ un sous-ensemble $\mathtt{C}^\mathcal{I}$ de $\Delta_\mathcal{I}$, à un rôle $\mathtt{r}$ une relation binaire $\mathtt{r}^\mathcal{I}$ sur $\Delta_\mathcal{I}$ (pour $x, y \in \Delta_\mathcal{I}$, x est lié à y par $\mathtt{r}^\mathcal{I}$ est noté $(x, y) \in \mathtt{r}^\mathcal{I}$) et, à une instance $\mathtt{a}$ un élément $\mathtt{a}^\mathcal{I}$ de $\Delta_\mathcal{I}$.

Étant donné une interprétation $\mathcal{I}$, les différents types d'expressions conceptuelles sont interprétées par :

$$\top^\mathcal{I} = \Delta_\mathcal{I} \qquad \bot^\mathcal{I} = \emptyset \qquad (\neg \mathtt{C})^\mathcal{I} = \Delta_\mathcal{I} \setminus \mathtt{C}^\mathcal{I} \qquad (\mathtt{C} \sqcap \mathtt{D})^\mathcal{I} = \mathtt{C}^\mathcal{I} \cap \mathtt{D}^\mathcal{I} \qquad (\mathtt{C} \sqcup \mathtt{D})^\mathcal{I} = \mathtt{C}^\mathcal{I} \cup \mathtt{D}^\mathcal{I}$$
$$(\exists \mathtt{r}.\mathtt{C})^\mathcal{I} = \{x \in \Delta_\mathcal{I} \mid \exists y, (x, y) \in \mathtt{r}^\mathcal{I} \text{ et } y \in \mathtt{C}^\mathcal{I}\} \qquad (\forall \mathtt{r}.\mathtt{C})^\mathcal{I} = \{x \in \Delta_\mathcal{I} \mid \forall y, \text{ si } (x, y) \in \mathtt{r}^\mathcal{I} \text{ alors } y \in \mathtt{C}^\mathcal{I}\}$$

Par exemple, si $\mathtt{Tarte}^\mathcal{I}$, $\mathtt{Pomme}^\mathcal{I}$, $\mathtt{Pâte}^\mathcal{I}$, et $\mathtt{Cannelle}^\mathcal{I}$ sont les ensembles de tartes, de pommes, de pâtes et de cannelles, et si $\mathtt{ing}^\mathcal{I}$ est la relation « a pour ingrédient », alors le concept de l'équation (1) représente l'ensemble des tartes avec des pommes et de la pâte, mais sans cannelle.

Étant donné une formule f et une interprétation $\mathcal{I}$, « $\mathcal{I}$ satisfait f » est dénoté par $\mathcal{I} \models f$. Un modèle de f est une interprétation $\mathcal{I}$ satisfaisant f. La sémantique des quatre types de formules est la suivante :

$$\mathcal{I} \models \mathtt{C} \sqsubseteq \mathtt{D} \quad \text{si } \mathtt{C}^\mathcal{I} \subseteq \mathtt{D}^\mathcal{I} \qquad \mathcal{I} \models \mathtt{C}(\mathtt{a}) \quad \text{si } \mathtt{a}^\mathcal{I} \in \mathtt{C}^\mathcal{I}$$
$$\mathcal{I} \models \mathtt{C} \equiv \mathtt{D} \quad \text{si } \mathtt{C}^\mathcal{I} = \mathtt{D}^\mathcal{I} \qquad \mathcal{I} \models \mathtt{r}(\mathtt{a}, \mathtt{b}) \quad \text{si } (\mathtt{a}^\mathcal{I}, \mathtt{b}^\mathcal{I}) \in \mathtt{r}^\mathcal{I}$$

Étant donné une base de connaissances $\mathtt{BC}$ et une interprétation $\mathcal{I}$, $\mathcal{I}$ satisfait $\mathtt{BC}$ – dénoté par $\mathcal{I} \models \mathtt{BC}$ – si $\mathcal{I} \models f$ pour chaque $f \in \mathtt{BC}$. Un *modèle* de $\mathtt{BC}$ est une interprétation satisfaisant $\mathtt{BC}$. Une base de connaissances $\mathtt{BC}$ entraîne une formule f – dénoté par $\mathtt{BC} \models f$ – si tout modèle de $\mathtt{BC}$ est un modèle de f. Une tautologie est une formule f satisfaite par toute interprétation. « f est une tautologie » s'écrit $\models f$. Deux bases de connaissances sont dites équivalentes si chaque modèle de l'une d'elles est un modèle de l'autre et vice-versa.

2.3 Inférences

Soit $\mathtt{BC}$ une base de connaissances. Parmi les inférences classiques associées à $\mathcal{ALC}$ il y a les tests de la forme $\mathtt{BC} \models f$, où f est une formule. Par exemple, tester si $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{D}$ est appelé le *test de subsomption* : il teste si, étant donné $\mathtt{BC}$, le concept $\mathtt{C}$ est plus spécifique que le concept $\mathtt{D}$, ce qui est utile pour organiser les concepts en hiérarchies (p. ex., les hiérarchies d'index de systèmes de RÀPC).

La *classification de concept* consiste, étant donné un concept $\mathtt{C}$, à trouver les concepts atomiques $\mathtt{A}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{A}$ (les subsumants de $\mathtt{C}$) et les concepts atomiques $\mathtt{B}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{B} \sqsubseteq \mathtt{C}$ (les subsumés de $\mathtt{C}$). La *classification d'instance* consiste, étant donné une instance $\mathtt{a}$, à trouver les concepts atomiques $\mathtt{A}$ apparaissant dans $\mathtt{BC}$ tels que $\mathtt{BC} \models \mathtt{A}(\mathtt{a})$. Ces deux inférences peuvent être utilisées pour l'étape de remémoration d'un système de RÀPC.

La *satisfiabilité d'une ABox* consiste à tester si, étant donné $\mathtt{BC}$, une ABox a un modèle. Des inférences importantes peuvent se ramener à cette inférence, p. ex. $\models \mathtt{C} \sqsubseteq \mathtt{D}$ ssi $\{(\mathtt{C} \sqcap \neg \mathtt{D})(\mathtt{a})\}$ est non satisfiable, où $\mathtt{a}$ est une nouvelle instance (n'apparaissant ni dans $\mathtt{C}$, ni dans $\mathtt{BC}$).

D'autres inférences sont définies dans la littérature, mais ne sont pas utiles pour cet article.

2.4 Une procédure de déduction classique pour $\mathcal{ALC}$: la méthode des tableaux

Soit $\mathtt{BC}$ une base de connaissances, $\mathcal{T}_0$, sa TBox, et $\mathcal{A}_0$, sa ABox. La procédure permet de tester si $\mathcal{A}_0$ est satisfiable, étant donné $\mathtt{BC}$.

Prétraitement. La première étape du prétraitement consiste à substituer $\mathcal{T}_0$ par une TBox équivalente $\mathcal{T}_0'$ de la forme $\{\top \sqsubseteq \mathtt{K}\}$, pour un certain concept $\mathtt{K}$. Cela peut être fait tout d'abord en substituant chaque formule $\mathtt{C} \equiv \mathtt{D}$ par deux formules $\mathtt{C} \sqsubseteq \mathtt{D}$ et $\mathtt{D} \sqsubseteq \mathtt{C}$. La TBox résultante est de la forme $\{\mathtt{C}_i \sqsubseteq \mathtt{D}_i\}_{1 \leqslant i \leqslant n}$ et on peut montrer qu'elle équivaut à $\{\top \sqsubseteq \mathtt{K}\}$, avec $\mathtt{K} = (\neg \mathtt{C}_1 \sqcup \mathtt{D}_1) \sqcap \ldots \sqcap (\neg \mathtt{C}_n \sqcup \mathtt{D}_n)$.

La deuxième étape du prétraitement consiste à mettre $\mathcal{T}_0$ et $\mathcal{A}_0$ sous forme normale négative (FNN), ce qui signifie que dans chaque concept apparaissant dans ces bases de connaissances, le signe de négation $\neg$

apparaît uniquement devant des concepts atomiques. Il est toujours possible d'effectuer une telle normalisation en appliquant, tant que c'est possible, les équivalences suivantes (de gauche à droite) :

$$\neg\top \equiv \bot \qquad \neg(\mathtt{C} \sqcap \mathtt{D}) \equiv \neg\mathtt{C} \sqcup \neg\mathtt{D} \qquad \neg\exists \mathtt{r}.\mathtt{C} \equiv \forall \mathtt{r}.\neg\mathtt{C} \qquad \neg\neg\mathtt{C} \equiv \mathtt{C}$$
$$\neg\bot \equiv \top \qquad \neg(\mathtt{C} \sqcup \mathtt{D}) \equiv \neg\mathtt{C} \sqcap \neg\mathtt{D} \qquad \neg\forall \mathtt{r}.\mathtt{C} \equiv \exists \mathtt{r}.\neg\mathtt{C} \qquad \mathtt{C} \sqcup \bot \equiv \mathtt{C}$$

Par exemple, le concept $\neg(\forall \mathtt{r}.(\neg\mathtt{A} \sqcup \exists \mathtt{s}.\mathtt{B}))$ est équivalent au concept $\exists \mathtt{r}.(\mathtt{A} \sqcap \forall \mathtt{s}.\neg\mathtt{B})$, qui est sous FNN.

La TBox de CD donnée dans l'équation (2) est équivalente à la TBox $\{\top \sqsubseteq \mathtt{K}\}$ sous FNN avec

$$\mathtt{K} = (\neg\mathtt{Pomme} \sqcup \mathtt{Piridion}) \sqcap (\neg\mathtt{Poire} \sqcup \mathtt{Piridion}) \sqcap (\neg\mathtt{Piridion} \sqcup \mathtt{Pomme} \sqcup \mathtt{Poire})$$

Processus principal. Étant donné une TBox $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ et une ABox $\mathcal{A}_0$, toutes deux sous FNN, la méthode des tableaux manipule des ensembles d'ABox, en partant du singleton $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$, avec

$$\mathcal{A}_0^{\mathtt{K}} = \mathcal{A}_0 \cup \{\mathtt{K}(\mathtt{a}) \mid \mathtt{a} \text{ est une instance apparaissant dans } \mathcal{A}_0\}$$

Un tel ensemble d'ABox $\mathcal{D}$ doit être interprété comme une disjonction : $\mathcal{D}$ est satisfiable ssi au moins une $\mathcal{A} \in \mathcal{D}$ est satisfiable.

Chacune des étapes suivantes de l'algorithme consiste à transformer l'ensemble d'ABox courant $\mathcal{D}$ en un autre ensemble d'ABox $\mathcal{D}'$, en appliquant des règles de transformation sur les ABox : quand une règle de transformation ϱ, applicable sur une ABox $\mathcal{A} \in \mathcal{D}$, est sélectionnée par le processus, alors $\mathcal{D}' = (\mathcal{D} \setminus \{\mathcal{A}\}) \cup \{\mathcal{A}^1, \ldots, \mathcal{A}^p\}$ où les $\mathcal{A}^i$ sont obtenues par application de ϱ sur $\mathcal{A}$ (voir plus loin une description de ces règles).

Le processus termine quand aucune règle de transformation n'est applicable.

Une ABox est *fermée* quand elle contient un *conflit* (*clash* en anglais), c.-à-d., une contradiction « évidente ». Pour $\mathcal{ALC}$, les conflits sont les paires d'assertions de la forme $\{\mathtt{A}(\mathtt{a}), (\neg\mathtt{A})(\mathtt{a})\}$.

Par conséquent, une ABox fermée est insatisfiable. Une ABox *ouverte* est une ABox non fermée.

Une ABox est *complète* si aucune règle de transformation ne peut être appliquée sur elle.

Soit $\mathcal{D}_{\text{final}}$, l'ensemble des ABox à la fin du processus, c.-à-d., quand chacune des $\mathcal{A} \in \mathcal{D}$ est complète. Il a été montré (voir, p. ex., [2]) que, avec les règles de transformation présentées ci-dessous, le processus termine toujours, et $\mathcal{A}_0$ est satisfiable étant donné $\mathcal{T}_0$ ssi $\mathcal{D}_{\text{final}}$ contient au moins une ABox ouverte.

Les règles de transformation. Il y a quatre règles de transformation pour la méthode des tableaux appliquée à $\mathcal{ALC}$: $\longrightarrow_{\sqcap}$, $\longrightarrow_{\sqcup}$, $\longrightarrow_{\forall}$, et $\longrightarrow_{\exists}^{\mathtt{K}}$. Aucune de ces règles n'est applicable sur une ABox fermée. L'ordre de ces règles affecte seulement la performance du système, à l'exception de la règle $\longrightarrow_{\exists}^{\mathtt{K}}$ qui doit être appliquée seulement quand aucune autre règle n'est applicable sur l'ensemble d'ABox actuel (ceci afin d'assurer la terminaison). Ces règles correspondent à des étapes de déduction au sens où elles ajoutent des assertions déduites d'assertions déjà existantes[3].

La règle $\longrightarrow_{\sqcap}$ est applicable sur une ABox $\mathcal{A}$ si cette dernière contient une assertion de la forme $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$, et est telle qu'au moins une assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$ n'appartient pas à $\mathcal{A}$. Cette règle retourne l'ABox $\mathcal{A}'$ définie par

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leqslant k \leqslant p\}$$

La règle $\longrightarrow_{\sqcup}$ est applicable sur une ABox $\mathcal{A}$ si cette dernière contient une assertion de la forme $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$ mais aucune assertion de la forme $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$. Cette règle retourne les ABox $\mathcal{A}^1, \ldots, \mathcal{A}^p$ définies, pour $1 \leqslant k \leqslant p$, par

$$\mathcal{A}^k = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a})\}$$

La règle $\longrightarrow_{\forall}$ est applicable sur l'ABox $\mathcal{A}$ si cette dernière contient deux assertions des formes respectives $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ et $\mathtt{r}(\mathtt{a}, \mathtt{b})$ (avec le même $\mathtt{r}$ et le même $\mathtt{a}$), et si $\mathcal{A}$ ne contient pas l'assertion $\mathtt{C}(\mathtt{b})$. Cette règle retourne l'ABox $\mathcal{A}'$ définie par

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}(\mathtt{b})\}$$

La règle $\longrightarrow_{\exists}^{\mathtt{K}}$ est applicable sur une ABox $\mathcal{A}$ si

[3] Plus précisément, chacune de ces règles transforme une disjonction d'ABox $\mathcal{D}$ en une autre disjonction d'ABox $\mathcal{D}'$, telle que, étant donné $\mathcal{T}_0$, $\mathcal{D}'$ est satisfiable ssi $\mathcal{D}$ l'est. Cette forme d'équivalence (l'équivalence des satisfiabilités) est celle que l'on retrouve dans la skolémisation des formules de logique du premier ordre (qui remplace les quantificateurs existentiels par des symboles de fonction).

(i) $\mathcal{A}$ contient une assertion de la forme $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;

(ii) $\mathcal{A}$ ne contient pas à la fois une assertion de la forme $\mathtt{r}(\mathtt{a},\mathtt{b})$ et une assertion de la forme $\mathtt{C}(\mathtt{b})$ (avec le même $\mathtt{b}$, et avec les mêmes $\mathtt{C}$ et $\mathtt{a}$ que dans la condition précédente) ;

(iii) Il n'y a aucune instance $\mathtt{c}$ telle que $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \mathcal{A}\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \mathcal{A}\}$ (cette troisième condition est introduite pour assurer la terminaison de l'algorithme).

Si ces conditions sont remplies, soit $\mathtt{b}$ une nouvelle instance, cette règle retourne

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{r}(\mathtt{a},\mathtt{b}), \mathtt{C}(b)\} \cup \{\mathtt{K}(\mathtt{b})\}$$

On peut noter que la TBox $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ est utilisée ici : étant donné qu'une nouvelle instance $\mathtt{b}$ est introduite, cette instance doit satisfaire la TBox, ce qui correspond à l'assertion $\mathtt{K}(\mathtt{b})$.

Remarque 1 *Après l'application d'une de ces règles sur une ABox de $\mathcal{D}$, la disjonction d'ABox résultante est équivalente à $\mathcal{D}$ (voir note 3).*

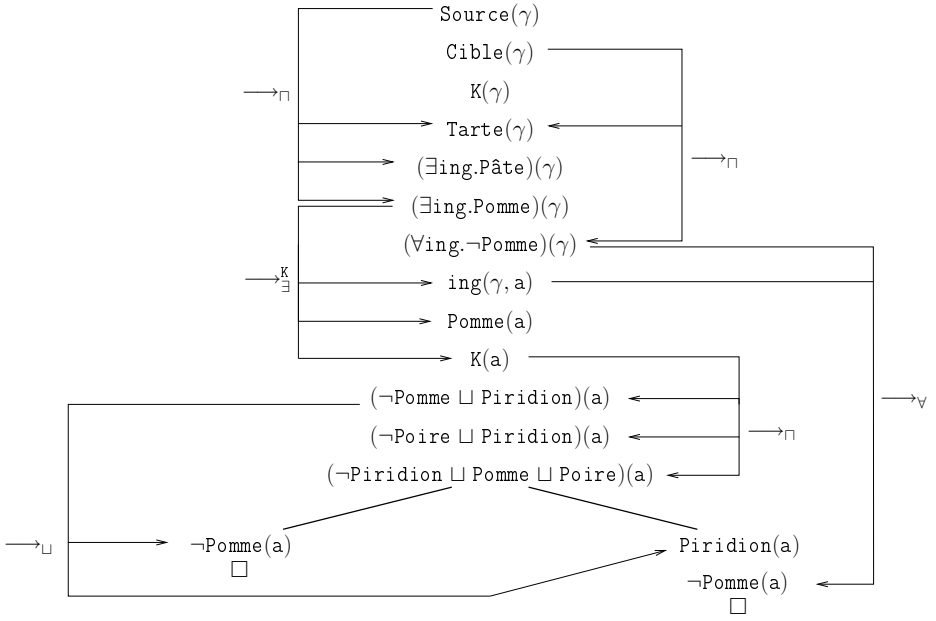

FIG. 1 – Application de la méthode des tableaux montrant que l'ABox $\{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ est non satisfiable, étant donné la TBox $\{\top \sqsubseteq \mathtt{K}\}$ (l'ordre d'application des règles ne respecte pas dans cet exemple la contrainte « $\longrightarrow_{\exists}^{\mathtt{K}}$ doit être appliquée en dernier », cela afin de rendre l'exemple plus simple.).

Exemple. Considérons l'exemple donné à la fin de la section 2.1. Prétendre qu'un cas source représenté par l'instance σ peut être appliqué au cas cible représenté par l'instance γ revient à identifier ces deux instances, p. ex., en substituant σ par γ. Cela conduit à l'ABox $\mathcal{A}_0 = \{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ (avec $\mathtt{Source}$ et $\mathtt{Cible}$ définis dans (4)). La figure 1 représente ce processus. L'arbre en entier représente l'ensemble d'ABox $\mathcal{D}_{\text{final}}$: chacune de ces branches représente une ABox complète $\mathcal{A} \in \mathcal{D}_{\text{final}}$. Au début du processus, les seuls nœuds de cet arbre sont $\mathtt{Source}(\gamma)$, $\mathtt{Cible}(\gamma)$ et $\mathtt{K}(\gamma)$: cela correspond à $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$. Puis, les règles de transformation sont appliquées. On peut noter que seule la règle $\longrightarrow_{\sqcup}$ conduit à la création de plusieurs nœuds-fils d'un nœud. Quand un conflit a été détecté sur une branche (p. ex. $\{\mathtt{Pomme}(\mathtt{a}), (\neg\mathtt{Pomme})(\mathtt{a})\}$) la branche représente une ABox fermée (le conflit est symbolisé par $\square$). Notons que les deux ABox finales sont fermées, ce qui signifie que $\{\mathtt{Source}(\gamma), \mathtt{Cible}(\gamma)\}$ est non satisfiable : le cas source doit être adapté pour pouvoir être réutilisé dans le contexte du cas cible.

3 Un algorithme d'adaptation en $\mathcal{ALC}$

Cette section présente un algorithme d'adaptation en $\mathcal{ALC}$: ses paramètres, son résultat, ses étapes et certaines de ses propriétés.

3.1 Paramètres et résultat de l'algorithme

Les paramètres de l'algorithme sont $\mathtt{CD}$, $\mathcal{A}^{\sigma}_{\mathtt{srce}}$, $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ et $\mathtt{coût}$. Son résultat est $\mathcal{D}$.

$\mathtt{CD}$ est une base de connaissances en $\mathcal{ALC}$ représentant les connaissances du domaine. Dans l'exemple suivi le long de cet article, son ABox est vide, mais, de façon générale, $\mathtt{CD}$ peut contenir des assertions.

Les cas source et cible sont représentés par deux ABox supposées satisfiable étant donné $\mathtt{CD}$: $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$, respectivement. Plus précisément, le cas source est réifié par une instance σ et $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ contient les assertions à son sujet. Dans l'exemple ci-dessus, $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ contient une seule assertion, $\mathtt{Source}(\sigma)$. De façon similaire, le cas cible est réifié par une instance γ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ contient les assertions concernant γ (une seule assertion dans l'exemple : $\mathtt{Cible}(\gamma)$).

Le paramètre $\mathtt{coût}$ est une fonction associant à un littéral ℓ une valeur numérique $\mathtt{coût}(\ell) > 0$, où un littéral est soit un concept atomique (littéral positif) soit un concept de la forme $\neg\mathtt{A}$ où $\mathtt{A}$ est atomique (littéral négatif). Intuitivement, plus $\mathtt{coût}(\ell)$ est grand, plus il est difficile de renoncer à la vérité d'une assertion $\ell(\mathtt{a})$.

L'algorithme retourne $\mathcal{D}$, un ensemble d'ABox $\mathcal{A}$ résolvant le cas cible en adaptant le cas source : $\mathcal{A} \models \mathcal{A}^{\gamma}_{\mathtt{cible}}$ et $\mathcal{A}$ réutilise « autant que possible » $\mathcal{A}^{\sigma}_{\mathtt{srce}}$. Il peut arriver que $\mathcal{D}$ contienne plusieurs ABox. Dans ce cas, les connaissances du système (en particulier, ceux associés à la fonction $\mathtt{coût}$) sont insuffisantes pour faire un choix, et c'est à l'utilisateur de faire ce choix.

3.2 Étapes de l'algorithme

Prétraitement. Soient $\mathcal{T}_{\mathtt{CD}}$ et $\mathcal{A}_{\mathtt{CD}}$ la TBox et l'ABox de $\mathtt{CD}$. Soit $\mathtt{K}$ un concept sous forme normale négative (FNN) tel que $\mathcal{T}_{\mathtt{CD}}$ soit équivalent à $\{\top \sqsubseteq \mathtt{K}\}$. $\mathcal{A}_{\mathtt{CD}}$ est simplement ajoutée aux ABox :

$$\mathcal{A}^{\sigma}_{\mathtt{srce}} \leftarrow \mathcal{A}^{\sigma}_{\mathtt{srce}} \cup \mathcal{A}_{\mathtt{CD}} \qquad\qquad \mathcal{A}^{\gamma}_{\mathtt{cible}} \leftarrow \mathcal{A}^{\gamma}_{\mathtt{cible}} \cup \mathcal{A}_{\mathtt{CD}}$$

$\mathcal{A}^{\sigma}_{\mathtt{srce}}$ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ sont ensuite mises sous FNN.

Prétendre que le cas source résout le problème. La réutilisation de $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ pour l'instance γ réifiant le cas cible est obtenue en assimilant les instances σ et γ. On obtient alors l'ABox $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ par substitution de σ par γ dans $\mathcal{A}^{\sigma}_{\mathtt{srce}}$. Soit $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}} = \mathcal{A}^{\gamma}_{\mathtt{srce}} \cup \mathcal{A}^{\gamma}_{\mathtt{cible}}$. Si $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$ est satisfiable, étant donné $\mathtt{CD}$, alors la réutilisation directe du cas source n'entraîne aucune contradiction avec les spécifications du cas cible, il n'apporte que de nouvelles informations à son propos. Par exemple, considérons $\mathcal{A}^{\sigma}_{\mathtt{srce}} = \{\mathtt{Source}(\sigma)\}$ donnée par l'équation (3), $\mathcal{A}^{\gamma}_{\mathtt{cible}} = \{\mathtt{Tarte}(\gamma), \mathtt{ing}(\gamma, \mathtt{p}), \mathtt{PâteFeuilletée}(\mathtt{p})\}$ (c.-à-d., « Je veux une tarte avec de la pâte feuilletée »), et les connaissances du domaine $\mathtt{CD}' = \mathtt{CD} \cup \{\mathtt{PâteFeuilletée} \sqsubseteq \mathtt{Pâte}\}$, avec $\mathtt{CD}$ définies dans (2). On peut alors montrer qu'étant donné $\mathtt{CD}'$, $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$ est satisfiable et correspond à une tarte aux pommes avec de la pâte feuilletée.

Il arrive cependant souvent que $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$ ne soit pas satisfiable, étant donné $\mathtt{CD}$. Ceci est vrai pour l'exemple courant. Le principe de l'algorithme d'adaptation consiste alors à réparer $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$. Par « réparation » de $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$ nous voulons dire sa modification pour la rendre complète et sans conflit, et donc consistante. La suppression des conflits n'est pas suffisante, les formules dont ces conflits dérivent doivent aussi être supprimées. Cela motive l'introduction dans la section 3.2 des AGraphes qui étendent les ABox en conservant la trace de l'application des règles. De plus, pour obtenir une adaptation plus fine, $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ sont complétées par la méthode des tableaux avant d'être combinées.

Appliquer la méthode des tableaux sur $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ et sur $\mathcal{A}^{\gamma}_{\mathtt{cible}}$, en mémorisant l'application des règles de transformation. L'implantation de cette étape et des suivantes fait intervenir la notion de *graphe assertionnel* (ou *AGraphe*) introduite ici. Un AGraphe $\mathcal{G}$ est un graphe simple dont l'ensemble des nœuds, $\mathit{Nœuds}(\mathcal{G})$, est une ABox, et dont les arcs sont étiquetés par des règles de transformation : si $(\alpha, \beta) \in \mathit{Arcs}(\mathcal{G})$, l'ensemble des arcs orientés, alors $\lambda_{\mathcal{G}}(\alpha, \beta) = \varrho$ indique que β a été obtenu en appliquant ϱ sur α et, peut-être, sur d'autres assertions ($\lambda_{\mathcal{G}}$ est la fonction d'étiquetage du graphe $\mathcal{G}$).

La méthode des tableaux sur les AGraphes repose sur les règles de transformation $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow^{\mathtt{K}}_{\exists}$. Elles sont similaires aux règles de transformation sur les ABox $\mathcal{ALC}$, à quelques différences près.

La règle $\Longrightarrow_{\sqcap}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α de la forme $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$;

(ii) $\mathcal{G} \neq \mathcal{G}'$ (c.-à-d., $\mathit{Nœuds}(\mathcal{G}) \neq \mathit{Nœuds}(\mathcal{G}')$ ou $\mathit{Arcs}(\mathcal{G}) \neq \mathit{Arcs}(\mathcal{G}')$) avec $\mathcal{G}'$ défini par

$$\begin{aligned}
\mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leqslant k \leqslant p\} \\
\mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a})) \mid 1 \leqslant k \leqslant p\} \\
\lambda_{\mathcal{G}'}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcap} \text{ pour } 1 \leqslant k \leqslant p \\
\lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G})
\end{aligned}$$

Sous ces conditions, l'application de cette règle renvoie $\mathcal{G}'$.

La principale différence entre la règle $\longrightarrow_{\sqcap}$ sur ABox et la règle $\Longrightarrow_{\sqcap}$ sur AGraphes est que cette dernière peut être appliquée à $\alpha = (\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$ même quand $\mathtt{C}_k(\mathtt{a}) \in \mathit{Nœuds}(\mathcal{G})$ pour tout k, $1 \leqslant k \leqslant p$. Dans ce cas de figure, $\mathit{Nœuds}(\mathcal{G}') = \mathit{Nœuds}(\mathcal{G})$ mais $\mathit{Arcs}(\mathcal{G}') \neq \mathit{Arcs}(\mathcal{G})$: un nouvel arc $(\alpha, \mathtt{C}_k)$ indique ici que $\alpha \models \mathtt{C}_k(\mathtt{a})$ et donc, que si $\mathtt{C}_k(\mathtt{a})$ doit être supprimé, alors α doit aussi être supprimé (voir plus loin, l'étape de réparation de l'algorithme).

De même, les règles $\longrightarrow_{\sqcup}$, $\longrightarrow_{\forall}$, et $\longrightarrow_{\exists}^{\mathtt{K}}$ sont modifiées en $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow_{\exists}^{\mathtt{K}}$, détaillées dans la figure 2.

On peut appliquer la méthode des tableaux présentée en section 2.4 avec une TBox $\{\top \sqsubseteq \mathtt{K}\}$ et une ABox $\mathcal{A}_0$. Avec comme seule différence que ce sont des AGraphes qui sont manipulés à la place de d'ABox, ce qui entraîne que (1) un AGraphe initial $\mathcal{G}_0^{\mathtt{K}}$ doit être construit à partir de $\mathcal{A}_0^{\mathtt{K}}$ (il est défini par $\mathit{Nœuds}(\mathcal{G}_0^{\mathtt{K}}) = \mathcal{A}_0^{\mathtt{K}}$ et $\mathit{Arcs}(\mathcal{G}_0^{\mathtt{K}}) = \emptyset$), (2) les règles $\Longrightarrow_{\cdot}$ sont utilisées à la place des règles $\longrightarrow_{\cdot}$, et (3) le résultat est un ensemble d'AGraphes complets et ouverts (qui est vide ssi, étant donné $\{\top \sqsubseteq \mathtt{K}\}$, $\mathcal{G}_0^{\mathtt{K}}$ n'est pas satisfiable).

Soient $\{\mathcal{G}_i\}_{1 \leqslant i \leqslant m}$ et $\{\mathcal{H}_j\}_{1 \leqslant j \leqslant n}$ des ensembles d'AGraphes ouverts et complets obtenus par l'application de la méthode des tableaux respectivement sur $\mathcal{A}_0 = \mathcal{A}^{\gamma}_{\mathtt{src}}$ et $\mathcal{A}_0 = \mathcal{A}^{\gamma}_{\mathtt{cible}}$. Si, étant donné $\{\top \sqsubseteq \mathtt{K}\}$, $\mathcal{A}^{\gamma}_{\mathtt{src}}$ et $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ sont satisfiables, alors $m \neq 0$ et $n \neq 0$. Si $m = 0$ ou $n = 0$, l'algorithme s'arrête avec la valeur $\mathcal{D} = \{\mathcal{A}^{\gamma}_{\mathtt{cible}}\}$.

Générer des conflits explicites à partir de $\mathcal{G}_i$ et $\mathcal{H}_j$. On considère un nouveau type d'assertion, réifiant la notion de conflit : l'*assertion de conflit* $\Box\pm\mathtt{A}(\mathtt{a})$ réifie le conflit $\{\mathtt{A}(\mathtt{a}), (\neg\mathtt{A})(\mathtt{a})\}$. Elles sont générées par la règle $\Longrightarrow_{\Box}$. Cette règle est applicable sur l'AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient deux nœuds $\mathtt{A}(\mathtt{a})$ et $(\neg\mathtt{A})(\mathtt{a})$ (avec le même $\mathtt{A}$ et le même $\mathtt{a}$) ;

(ii) $\mathcal{G} \neq \mathcal{G}'$ avec $\mathcal{G}'$ défini par

$$\begin{aligned}
\mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\Box\pm\mathtt{A}(\mathtt{a})\} \\
\mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})), ((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a}))\} \\
\lambda_{\mathcal{G}'}(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) &= \lambda_{\mathcal{G}'}((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) = \Longrightarrow_{\Box} \\
\lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G})
\end{aligned}$$

Sous ces conditions, la règle renvoie $\mathcal{G}'$.

L'étape suivante de l'algorithme consiste à appliquer la méthode des tableau sur chaque $\mathcal{G}_i \cup \mathcal{H}_j$, pour tout i et j, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, en utilisant les règles de transformation $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, $\Longrightarrow_{\exists}^{\mathtt{K}}$, et $\Longrightarrow_{\Box}$. À la différence de la méthode des tableaux présentée plus haut où il était inutile d'appliquer les règles sur les ABox fermées (ou les AGraphes fermés), ici, lorsqu'une règle est applicable sur un AGraphe contenant une assertion de conflit, elle est appliquée. Plusieurs conflits peuvent donc être générés dans le même AGraphe.

Remarque 2 *Si une assertion de conflit $\Box\pm\mathtt{A}(\mathtt{a})$ est générée, alors ce conflit est la conséquence d'assertions venant à la fois de $\mathcal{G}_i$ et de $\mathcal{H}_j$, autrement, ce conflit aurait été généré à l'étape précédente de l'algorithme (puisque ces deux AGraphes sont ouverts et complets).*

Une condition nécessaire pour appliquer $\Longrightarrow_{\sqcup}$ sur un AGraphe $\mathcal{G}$ est que $\mathcal{G}$ contienne un nœud α de la forme $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$. Si c'est le cas, alors deux conditions peuvent être considérées :

(a) $\mathcal{G}$ ne contient aucune assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leqslant k \leqslant p)$. Sous cette condition, la règle renvoie les AGraphes $\mathcal{G}^1, \ldots, \mathcal{G}^p$ définis, pour $1 \leqslant k \leqslant p$, par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}^k) &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a})\} \\ \mathit{Arcs}(\mathcal{G}^k) &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a}))\} \\ \lambda_{\mathcal{G}^k}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcup} \\ \lambda_{\mathcal{G}^k}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

(b) $\mathcal{G}$ contient une ou plusieurs assertion $\beta_k = \mathtt{C}_k(\mathtt{a})$ telle que $(\alpha, \beta_k) \notin \mathit{Arcs}(\mathcal{G})$. Sous cette condition, $\Longrightarrow_{\sqcup}$ renvoie l'AGraphe $\mathcal{G}'$ obtenu en ajoutant cet arc (α, β_k) à $\mathcal{G}$, avec $\lambda_{\mathcal{G}'}(\alpha, \beta_k) = \Longrightarrow_{\sqcup}$.

La règle $\Longrightarrow_{\forall}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α_1 de la forme $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ et un nœud α_2 de la forme $\mathtt{r}(\mathtt{a}, \mathtt{b})$;

(ii) $\mathcal{G} \neq \mathcal{G}'$ avec $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{C}(\mathtt{b})\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha_1, \mathtt{C}(\mathtt{b})), (\alpha_2, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha_1, \mathtt{C}(\mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha_2, \mathtt{C}(\mathtt{b})) = \Longrightarrow_{\forall} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous ces conditions, la règle renvoie $\mathcal{G}'$.

La règle $\Longrightarrow^{\mathtt{K}}_{\exists}$ est applicable sur un AGraphe $\mathcal{G}$ si

(i) $\mathcal{G}$ contient un nœud α de la forme $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;

(ii) (a) Soit $\mathcal{G}$ ne contient à la fois $\mathtt{r}(\mathtt{a}, \mathtt{b})$ et $\mathtt{C}(\mathtt{b})$ pour aucune instance $\mathtt{b}$;

(b) Soit $\mathcal{G}$ contient deux assertions $\beta_1 = \mathtt{r}(\mathtt{a}, \mathtt{b})$ et $\beta_2 = \mathtt{C}(\mathtt{b})$, telles que $(\alpha, \beta_1) \notin \mathit{Arcs}(\mathcal{G})$ ou $(\alpha, \beta_2) \notin \mathit{Arcs}(\mathcal{G})$;

(iii) Il n'y a pas d'instance $\mathtt{c}$ telle que $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \mathit{Nœuds}(\mathcal{G})\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \mathit{Nœuds}(\mathcal{G})\}$ (condition du *set-blocking*, servant à assurer la terminaison de l'algorithme).

Si la condition (ii-a) est satisfaite, soit $\mathtt{b}$ une nouvelle instance. La règle renvoie $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \cup \{\mathtt{r}(\mathtt{a}, \mathtt{b}), \mathtt{C}(\mathtt{b}), \mathtt{K}(\mathtt{b})\} \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})), (\alpha, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha, \mathtt{C}(\mathtt{b})) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

Sous la condition (ii-b), la règle renvoie $\mathcal{G}'$ défini par

$$\begin{aligned} \mathit{Nœuds}(\mathcal{G}') &= \mathit{Nœuds}(\mathcal{G}) \\ \mathit{Arcs}(\mathcal{G}') &= \mathit{Arcs}(\mathcal{G}) \cup \{(\alpha, \beta_1), (\alpha, \beta_2)\} \\ \lambda_{\mathcal{G}'}(\alpha, \beta_1) &= \lambda_{\mathcal{G}'}(\alpha, \beta_2) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ pour } e \in \mathit{Arcs}(\mathcal{G}) \end{aligned}$$

FIG. 2 – Les règles de transformation $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, et $\Longrightarrow^{\mathtt{K}}_{\exists}$.

Réparer les assertions de conflit. L'étape précédente a produit, pour chaque $\mathcal{G}_i \cup \mathcal{H}_j$, un ensemble non vide $\mathcal{S}_{ij}$ d'AGraphes. L'étape de réparation consiste à réparer chacun de ces AGraphes $\Gamma \in \mathcal{S}_{ij}$ et à ne conserver que ceux qui minimisent le coût de réparation[4]. Soit $\Gamma \in \mathcal{S}_{ij}$. Si Γ ne contient aucune assertion de conflit, ceci implique que $\mathcal{G}_i \cup \mathcal{H}_j$ est satisfiable et donc $\mathcal{A}^{\gamma}_{\texttt{srce,cible}}$ aussi : il n'y a pas d'adaptation nécessaire. Si Γ contient $\delta \geq 1$ assertions de conflit, alors l'une d'entre elles est choisie et sa réparation produit un ensemble d'AGraphes réparés Γ' contenant $\delta - 1$ conflits. La réparation est ensuite reprise sur Γ', jusqu'à ce qu'il n'y ait plus de conflit[5]. Le coût de la réparation globale est la somme des coûts de chaque réparation. Dans la suite, nous verrons comment est réparé un conflit de Γ.

Le principe de la réparation d'un conflit consiste à supprimer des assertions de Γ de sorte à empêcher que les conflit soient regénérées par l'application des règles. Ainsi, la réparation de tous les conflits doit produire des AGraphes satisfiables (ce qui est une conséquence de la complétude de l'algorithme des tableaux sur $\mathcal{ALC}$). Pour cela, on suit le principe suivant, exprimé comme une règle d'inférence :

$$\frac{\varphi \models \beta \qquad \beta \text{ doit être supprimé}}{\varphi \text{ doit être supprimé}} \tag{5}$$

où β est une assertion et φ est un ensemble minimal d'assertions tel que $\varphi \models \beta$ (φ doit être interprété comme la conjonction de ses formules). Supprimer φ revient à oublier une des assertions $\alpha \in \varphi$: lorsque $\text{card}(\varphi) \geq 2$, il y a plusieurs manières de supprimer φ, et donc, plusieurs AGraphes Γ' peuvent être obtenus à partir de Γ. La relation $\models$ reliant φ et β est représentée par les arcs de Γ. Donc, suivant (5), la suppression d'assertions est propagée suivant les arcs (α, β), de β vers α.

Soit $\beta = \Box\pm\texttt{A}(\texttt{a})$, le conflit de Γ qu'il faut supprimer. Soient $\alpha^+ = \texttt{A}(\texttt{a})$ et $\alpha^- = \neg\texttt{A}(\texttt{a})$. α^+ ou α^-, l'un des deux au moins, doit être supprimé. $\mathcal{H}_j$ étant un AGraphe ouvert et complet, soit $\alpha^+ \notin \mathcal{H}_j$, soit $\alpha^- \notin \mathcal{H}_j$ (cf. remarque 2). Trois situations sont possibles :

- Si $\alpha^+ \in \mathcal{H}_j$, alors α^+ ne peut être supprimé : c'est une assertion générée à partir de $\mathcal{A}^{\gamma}_{\texttt{cible}}$. Donc α^- doit être supprimée.
- Si $\alpha^- \in \mathcal{H}_j$, alors α^+ doit être supprimé.
- Si $\alpha^+ \notin \mathcal{H}_j$ et $\alpha^- \notin \mathcal{H}_j$, le choix de l'assertion à supprimer repose sur la minimisation du coût. Si $\texttt{coût}(\texttt{A}) < \texttt{coût}(\neg\texttt{A})$, α^+ doit être supprimée. Si $\texttt{coût}(\texttt{A}) > \texttt{coût}(\neg\texttt{A})$, α^- doit être supprimée. Si $\texttt{coût}(\texttt{A}) = \texttt{coût}(\neg\texttt{A})$, deux AGraphes sont générés : l'un en supprimant α^+, l'autre en supprimant α^-.

Si une assertion β doit être supprimée, la propagation de la suppression suivant un arc (α, β) tel que $\lambda_{\mathcal{G}}(\alpha, \beta) \in \{\Longrightarrow_{\sqcap}, \Longrightarrow_{\sqcup}, \Longrightarrow^{\texttt{K}}_{\exists}\}$ consiste à supprimer α (et à propager la suppression depuis α).

Soit β une assertion à supprimer qui a été déduite par la règle $\Longrightarrow_{\forall}$. Il existe alors deux assertions telles que $\lambda_{\mathcal{G}}(\alpha_1, \beta) = \lambda_{\mathcal{G}'}(\alpha_2, \beta) = \Longrightarrow_{\forall}$. Dans cette situation, deux AGraphes sont générés, l'un fondé sur la suppression de α_1, l'autre sur la suppression de α_2 (lorsque α_1 ou α_2 est dans $\mathcal{H}_j$, un seul AGraphe est généré).

À la fin du processus de réparation, on obtient un ensemble non vide $\{\Gamma_k\}_{1 \leqslant k \leqslant p}$ d'AGraphes sans conflit. Seuls ceux dont le coût de réparation est minimal sont gardés. Soit $\mathcal{A}_k = \textit{Nœuds}(\Gamma_k)$. Le résultat de la réparation est $\mathcal{D} = \{\mathcal{A}_k\}_{1 \leqslant k \leqslant p}$.

Transformer la disjonction d'ABox $\mathcal{D}$. Si $\mathcal{A}, \mathcal{B} \in \mathcal{D}$ vérifient $\mathcal{A} \models \mathcal{B}$, alors les disjonctions d'ABox $\mathcal{D}$ et $\mathcal{D} \setminus \{\mathcal{A}\}$ sont équivalentes. Cela sert à simplifier $\mathcal{D}$ en supprimant de tels $\mathcal{A}$([6]). Après cette simplification, chaque $\mathcal{A} \in \mathcal{D}$ est réécrit pour enlever les instances i introduites par l'algorithme des tableaux. D'abord, les i qui ne sont reliées à aucune instance nommée, ni directement, ni indirectement, par des assertions $\texttt{r}(\texttt{a}, \texttt{b})$ sont supprimées, ce qui signifie que des assertions avec de tels i sont supprimées (cela peut arriver à cause de l'étape de réparation qui « déconnecte » i des instances nommées). Ensuite, une étape de « dé-skolemisation » est effectuée en remplaçant les instances introduites i par des assertions du type $(\exists\texttt{r}.\texttt{C})(\texttt{a})$. Par exemple, l'ensemble

[4] Dans notre prototype ceci a été amélioré en éliminant les réparations en cours dont le coût dépasse le minimum en cours.

[5] Il se peut que des nœuds supplémentaires doivent être supprimés pour assurer la consistance de l'AGraphe réparé. Ils sont déterminés par analyse des *set-bockings* (cf. figure 2, $\Longrightarrow^{\texttt{K}}_{\exists}$, condition (iii)).

[6] Dans nos tests, nous nous sommes servis de conditions nécessaires à $\mathcal{A} \models \mathcal{B}$ portant sur l'inclusion ensembliste, avec ou sans renommage d'instance introduite. Cela a entraîné une réduction conséquente de la taille de $\mathcal{D}$, ce qui incite à penser que l'algorithme présenté ici peut être grandement amélioré, en évitant la génération d'ABox inutiles.

$\{\mathtt{r}(\mathtt{a},\mathtt{i}_1), \mathtt{A}(\mathtt{i}_1), \mathtt{s}(\mathtt{i}_1,\mathtt{i}_2), \neg\mathtt{B}(\mathtt{i}_2)\}$ est remplacé par $\{(\exists \mathtt{r}.(\mathtt{A} \sqcap \exists \mathtt{s}.\neg\mathtt{B}))(\mathtt{a})\}$. L'algorithme renvoie la valeur finale de $\mathcal{D}$.

Exemple. En reprenant l'exemple de la fin de la section 2.1. L'ensemble des étapes de l'algorithme est trop important pour être développé ici, nous ne verrons que les réparations.

Plusieurs AGraphes sont générés et doivent être réparés mais ils partagent tous le même conflit $\square\pm\mathtt{Pomme}(\mathtt{a})$. Il y a deux réparation possibles et l'ensemble $\mathcal{D}$ obtenu ne dépend que des coûts $\mathtt{coût}(\mathtt{Pomme})$ et $\mathtt{coût}(\neg\mathtt{Pomme})$.

Si $\mathtt{coût}(\mathtt{Pomme}) < \mathtt{coût}(\neg\mathtt{Pomme})$ alors $\mathcal{D}$ est équivalente à $\{\{(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.\mathtt{Poire})(\gamma)\}\}$. L'adaptation proposée est une tarte aux poires[7].

Si $\mathtt{coût}(\mathtt{Pomme}) \geq \mathtt{coût}(\neg\mathtt{Pomme})$ alors $\mathcal{D}$ est équivalente à $\{\{(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte})(\gamma)\}\}$. Seule la pâte a été réutilisée du cas source.

3.3 Propriétés de l'algorithme

L'algorithme d'adaptation termine. Cela peut être prouvé en s'appuyant sur la terminaison de l'algorithme de tableau sur les ABox [2] : comme la réparation enlève au moins un nœud des AGraphes à chaque étape, et que ceux-ci sont finis, elle termine également.

Chaque ABox $\mathcal{A} \in \mathcal{D}$ satisfait les contraintes de $\mathtt{Cible} : \mathcal{A} \models \mathcal{A}^{\gamma}_{\mathtt{cible}}$.

Si $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ est satisfiable alors toute $\mathcal{A} \in \mathcal{D}$ est satisfiable. En d'autres termes, à moins que le cas cible soit en contradiction avec les connaissances du domaine, l'adaptation produit un résultat cohérent. Si $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ est non satisfiable alors $\mathcal{D}$ est équivalente à $\{\mathcal{A}^{\gamma}_{\mathtt{cible}}\}$. Cela signifie que si une ABox $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ de la base de cas est dénuée de sens[8], alors $\mathcal{A}^{\gamma}_{\mathtt{cible}}$ n'est pas modifié (on n'infère rien d'un cas source insatisfiable).

Si le cas source est applicable sous des contraintes du cas cible ($\mathcal{A}^{\gamma}_{\mathtt{srce,cible}} = \mathcal{A}^{\gamma}_{\mathtt{srce}} \cup \mathcal{A}^{\gamma}_{\mathtt{cible}}$ est satisfiable) alors $\mathcal{D}$ contient une seule ABox qui est équivalente à $\mathcal{A}^{\gamma}_{\mathtt{srce,cible}}$: le cas source est réutilisé sans modification pour résoudre le cas cible.

L'adaptation présentée ici peut être considérée comme une approche par généralisation et spécialisation de l'adaptation. Les ABox $\mathcal{A} \in \mathcal{D}$ sont obtenues en « généralisant » $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ en $\mathcal{A}'$ (en supprimant des formules : $\mathcal{A}^{\gamma}_{\mathtt{srce}} \models \mathcal{A}'$) puis $\mathcal{A}'$ est « spécialisée » en $\mathcal{A} = \mathcal{A}' \cup \mathcal{A}^{\gamma}_{\mathtt{cible}}$.

4 Discussion et travaux proches

Au-delà d'un processus d'adaptation fondé sur l'appariement ? Il y a deux types d'algorithmes pour les inférences déductives classiques en LD : l'algorithme des tableaux présenté ci-dessus et les algorithmes structurels. Le premier est utilisé pour les LD expressives (c.-à-d., pour $\mathcal{ALC}$ et pour les LD étendant $\mathcal{ALC}$). Les autres sont utilisées pour les autres LD (pour lesquelles au moins certaines des inférences déductives sont polynomiales). L'algorithme structurel pour le test de subsomption $\mathtt{BC} \models \mathtt{C} \sqsubseteq \mathtt{D}$ consiste, après une étape de prétraitement, à *apparier* les descripteurs de D avec les descripteurs de C. Cette procédure d'appariement est assez proche de celles utilisées dans la plupart des procédures d'adaptation, de façon explicite ou non (si les cas ont une structure attribut-valeur fixe, en général, les cas source et cible sont appariés attribut par attribut, et le processus d'appariement n'a pas besoin d'être explicite). Les algorithmes structurels apparaissent comme inappropriés pour les LD expressives pour lesquelles la méthode des tableaux est utilisée à la place. L'algorithme d'adaptation présenté dans ce papier, qui s'appuie sur les principes de la méthode des tableaux, n'a pas d'étape d'appariement (même si on peut toujours apparier *a posteriori* les descripteurs du cas source et du cas cible adapté). En partant de ces observations, nous faisons l'hypothèse que, au-delà d'un certain niveau d'expressivité du langage de représentation des connaissances, il devient difficile d'utiliser des techniques d'appariement pour l'adaptation en tenant pleinement compte des connaissances du domaine.

[7] En l'absence de l'axiome $(\mathtt{Pomme} \sqcup \mathtt{Poire} \sqsubseteq \mathtt{Piridion}) \in \mathtt{CD}$, le résultat serait $\mathcal{D} = \{\mathcal{A}\}$ avec $\mathcal{A}$ équivalent à $(\mathtt{Tarte} \sqcap \exists\mathtt{ing}.\mathtt{Pâte} \sqcap \exists\mathtt{ing}.(\mathtt{Piridion} \sqcap \neg\mathtt{Pomme}))(\gamma)$. Autrement dit, l'adaptation consiste à remplacer les pommes par d'autres piridions (des poires, des coings, des nèfles).

[8] Dans un cadre logique, une base de connaissances non satisfiable est équivalente à n'importe quelle base de connaissances insatisfiable et donc, est dénuée de sens.

Autres travaux sur le RÀPC et les LD. Malgré les avantages de l'usage des LD en RÀPC (motivés en introduction), il y a relativement peu de travaux sur le RÀPC et les LD. Dans [10], les concepts d'une LD sont utilisés comme index pour la remémoration d'un planificateur à partir de cas, alors que l'adaptation est effectuée dans un autre formalisme. Dans [14], une LD non expressive est utilisée pour la remémoration et pour l'organisation de la base de cas. Ce travail s'appuie en particulier sur la notion de plus petit subsumant commun (PPSC) pour réifier la similarité entre concepts représentant les cas source et cible : le PPSC des concepts $\mathtt{C}$ et $\mathtt{D}$, quand il existe, est le plus spécifique des subsumants communs à $\mathtt{C}$ et à $\mathtt{D}$ et il met en évidence les caractéristiques communes aux deux concepts. Ainsi, le calcul du PPSC peut être vu comme un algorithme d'appariement, utilisable lors d'un processus d'adaptation. Dans une LD expressive, le PPSC de $\mathtt{C}$ et $\mathtt{D}$ est $\mathtt{C} \sqcup \mathtt{D}$ (ou un concept équivalent) qui n'exprime rien à propos des caractéristiques communes de $\mathtt{C}$ et $\mathtt{D}$.

À notre connaissance, les seules tentatives pour définir un algorithme d'adaptation pour les LD sont [8] et [6]. [8] présente une modélisation du cycle de vie du RÀPC s'appuyant sur les LD. En particulier, il présente une approche d'adaptation par substitution qui consiste à apparier les descripteurs des cas source et cible par une chaîne de rôles (similaire à une chaîne d'assertion $\mathtt{r}(\mathtt{a}_1, \mathtt{a}_2)$, $\mathtt{r}(\mathtt{a}_2, \mathtt{a}_3)$, etc.) afin de mettre en évidence quelles substitutions peuvent être faites. [6] utilise des règles d'adaptation (des reformulations) et une représentation multi-points de vue pour le RÀPC, incluant une étape d'adaptation complexe, alors que l'algorithme présenté ici utilise principalement les connaissances du domaine pour effectuer l'adaptation. Une perspective de recherche sera de voir comment ces approches de l'adaptation peuvent être combinées.

Travaux sur la réparation d'ontologies. Certains travaux sur le web sémantique et la gestion d'ontologies se penchent sur le problème de la réparation d'ontologies et la fusion ou révision d'ontologies contradictoires. On peut trouver une synthèse dans [7]. Il serait intéressant de comparer de façon précise ces travaux avec le nôtre. Notons néanmoins que la plupart de ces travaux, notamment pour des raisons de temps d'exécution, manipulent les ontologies à un niveau syntaxique (suppression ou affaiblissement de formules) alors que notre travail tente de manipuler les connaissances à un niveau sémantique (conséquences de formules, modèles, etc.).

5 Conclusion et perspectives

Cet article présente un algorithme pour l'adaptation dédié aux systèmes de RÀPC dont les cas et les connaissances du domaine sont représentés dans la LD expressive $\mathcal{ALC}$. La première question soulevée par un problème d'adaptation est « Qu'est-ce qui doit être adapté ? » L'algorithme traite cette question d'abord en prétendant que le cas source résout le problème cible, puis en mettant en évidence des inconsistances logiques : ces dernières correspondent aux parties du cas source qui doivent être modifiées afin de répondre à la situation posée par le cas cible. Ces principes sont appliqués à $\mathcal{ALC}$, LD pour laquelle les contradictions sont réifiées par les conflits générés par la méthode des tableaux. La deuxième question soulevée par un problème d'adaptation est « Comment le cas source doit-il être adapté ? » Pour y répondre, l'algorithme répare les contradictions en enlevant (temporairement) des connaissances associées au cas source, jusqu'à ce que la cohérence soit restaurée.

Actuellement, seul un prototype très simple de cet algorithme d'adaptation a été implanté, et il n'est pas très efficace. Une perspective sera de l'implanter efficacement et de façon extensible, en prenant en compte les extensions possibles mentionnées ci-dessous. On peut noter que les travaux sur l'amélioration de la méthode des tableaux pour les LD a conduit à des gains de temps de calcul spectaculaires (voir, en particulier, [9]).

La deuxième perspective sera d'étendre l'algorithme vers d'autres LD expressives. Les inférences déductives pour ces LD sont la plupart du temps implantées grâce à la méthode des tableaux, avec de nouvelles règles de transformation d'ABox et de nouveaux types de conflits. Pour étendre notre approche de l'adaptation, il faut définir de nouvelles règles de transformation sur les AGraphes et les méthodes de réparation des nouveaux types de conflits. En particulier, nous envisageons de l'étendre à $\mathcal{ALC}(\mathtt{D})$, où $\mathtt{D}$ est le domaine concret des n-uplets de nombres (entiers et réels) avec des prédicats de contraintes linéaires. Cela signifie que les cas pourront avoir des attributs numériques et que les connaissances du domaine pourront contenir des contraintes linéaires sur ces attributs. Cette perspective sera aussi une perspective de [4].

Dans notre algorithme, la valeur précise de $\mathtt{coût}(\ell)$, pour un littéral ℓ n'a guère d'importance. Ce qui importe est le signe et la nullité de $\delta(\mathtt{A}) = \mathtt{coût}(\neg\mathtt{A}) - \mathtt{coût}(\mathtt{A})$. $\delta(\mathtt{A}) = 0$ signifie qu'aucune préférence entre

renoncer à $\mathtt{A}(\mathtt{a})$ ou à $\neg\mathtt{A}(\mathtt{a})$ n'est connue du système : celui-ci retournera les deux possibilités et l'utilisateur fera son choix. Ce choix pourra être retenu pour une utilisation future. L'étude détaillée de cet apprentissage interactif est une perspective de ce travail et il pourra s'appuyer sur [5].

L'algorithme d'adaptation présenté ci-dessus peut être considéré comme une approche de l'adaptation par généralisation et spécialisation (cf. section 3.3). En revanche, l'algorithme [6] est une approche de l'adaptation utilisant des règles, une règle (ou reformulation) spécifiant une substitution pertinente pour une certaine classe de cas sources. Une piste pour intégrer ces deux approches est d'utiliser les règles d'adaptation durant le processus de réparation : à la place d'enlever les assertions conduisant à un conflit, une telle règle, si elle est disponible, peut être utilisée pour trouver des substitutions.

Comme cela a été écrit dans l'introduction, ce travail succède à des travaux sur l'adaptation fondée sur la révision des connaissances, bien qu'on ne puisse pas dire que cet algorithme, sous sa forme actuelle, implante un algorithme de révision pour $\mathcal{ALC}$ (p. ex., il ne permet pas la révision d'une TBox par une ABox). Dans [4], l'adaptation fondée sur la révision est généralisée en une combinaison (de plusieurs cas) fondé sur la fusion. Une telle généralisation devrait être applicable à l'algorithme défini dans ce papier : l'ABox $\mathcal{A}^{\gamma}_{\mathtt{srce}}$ est remplacée par plusieurs ABox et les réparations sont appliquées sur ces ABox. Définir précisément cet algorithme et étudier ses propriétés est une autre perspective.

Remerciements. Les auteurs remercient les relecteurs dont les remarques ont permis d'améliorer cet article.

Références

[1] C. E. Alchourrón, P. Gärdenfors et D. Makinson : On the Logic of Theory Change : partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi et P. Patel-Schneider, éditeurs. *The Description Logic Handbook*. Cambridge University Press, cambridge, UK, 2003.

[3] J. Cojan et J. Lieber : Conservative adaptation in metric spaces. *In Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany. Proceedings*, pages 135–149, 2008.

[4] J. Cojan et J. Lieber : Belief Merging-based Case Combination. *In* D. C. Wilson et L. McGinty, éditeurs : *8th International Conference on Case-Based Reasoning (ICCBR 2009)*, volume 5650 de *LNAI*, pages 105–119, Seattle, 2009. Springer.

[5] A. Cordier : *Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning*. Thèse de doctorat, Université Lyon 1, France, november 2008.

[6] M. d'Aquin, J. Lieber et A. Napoli : Decentralized Case-Based Reasoning for the Semantic Web. *In* Y. Gil et E. Motta, éditeurs : *Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, LNCS 3729, pages 142–155. Springer, November 2005.

[7] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis et G. Antoniou : Ontology change : classification and survey. *Knowledge Eng. Review*, 23(2):117–152, 2008.

[8] M. Gómez-Albarrán, P. A. González-Calero, B. Díaz-Agudo et C. Fernández-Conde : Modelling the CBR Life Cycle Using Description Logics. *In* K.-D. Althoff, R. Bergmann et L. K. Branting, éditeurs : *Proc. of the 3rd International Conference on Case-Based Reasoning Research and Development (ICCBR-99)*, LNAI 1650, pages 147–161. Springer, 1999.

[9] I. Horrocks : *Optimising Tableaux Decision Procedures for Description Logics*. Thèse de doctorat, University of Manchester, 1997.

[10] J. Koehler : Planning from Second Principles. *Artificial Intelligence*, 87:145–186, 1996.

[11] J. Kolodner : *Case-Based Reasoning*. Morgan Kaufmann, Inc., 1993.

[12] J. Lieber : Application of the Revision Theory to Adaptation in Case-Based Reasoning : the Conservative Adaptation. *In Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR-07)*, LNAI 4626, pages 239–253. Springer, Belfast, 2007.

[13] C. K. Riesbeck et R. C. Schank : *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1989.

[14] S. Salotti et V. Ventos : Study and Formalization of a Case-Based Reasoning System Using a Description Logic. *In* B. Smyth et P. Cunningham, éditeurs : *Fourth European Workshop on Case-Based Reasoning, EWCBR-98*, Lecture Notes in Artificial Intelligence 1488, pages 286–297. Springer, 1998.