Fondements de la révision dans un langage d'objets simple

Isabelle Crampé, Jérôme Euzenat

INRIA Rhône-Alpes
655 avenue de l'Europe, 38330 Montbonnot Saint-Martin (France)
{Isabelle.Crampe, Jerome.Euzenat}@inrialpes.fr

RÉSUMÉ : La révision d'une base de connaissance, rendue inconsistante suite à l'ajout d'une assertion, consiste à la rendre consistante en la modifiant. Résoudre ce problème est très utile dans l'assistance aux utilisateurs de bases de connaissance et s'appliquerait avec profit dans le contexte des objets. Afin de poser les bases d'un tel mécanisme, une représentation par objets minimale est formalisée. Elle est dotée de mécanismes d'inférence et d'une caractérisation syntaxique de l'inconsistance et de l'incohérence. La notion de base de connaissance révisée est définie sur ce langage. Un critère de minimalité, à la fois sémantique et syntaxique, permet de définir les bases révisées les plus proches de la base initiale.

Le problème de la révision d'une base de connaissance par objets consiste, lorsqu'une base de connaissance se révèle inconsistante (c'est-à-dire si elle contient des connaissances contradictoires), à trouver la (ou les) modification(s) à apporter à cette base afin qu'elle retrouve sa consistance. Ce problème est bien entendu délicat car il conduit à modifier la base de connaissance dans ses axiomes mêmes. Il est cependant très utile de trouver les solutions aux inconsistances qui apparaissent lors de la manipulation d'une base de connaissance. Il y a de nombreuses raisons à l'utilisation de la révision dans une base de connaissance [10]; notre ambition est d'aider un utilisateur à résoudre ce problème en lui proposant de choisir parmi les différentes modifications possibles [6]. Ceci évite de décider a priori quelle est la «meilleure» solution. Il est cependant utile de déterminer des critères de minimalité pour réduire la charge de l'utilisateur dans la sélection des révisions.

La révision a été étudiée longuement dans le domaine de la logique [1,14,3]. Les travaux en logique sont immédiatement applicables aux objets en traduisant les représentations par objets en logique. Cependant, l'opération de révision reste problématique [11]. En effet, deux types de problèmes se posent lors de la tentative de mise en œuvre de la révision : des problèmes de complexité (l'algorithme de révision nécessite en toute généralité un oracle décidant de la consistance) et des problèmes de complétude (les critères définis syntaxiquement ne correspondent pas aux critères sémantiques).

D'autre part, cette traduction n'est pas souhaitable car les systèmes perdent alors leur structure. En effet, les modèles à objets sont sans doute plus adaptés à la révision que les systèmes logiques, grâce à leurs multiples entités (classes, objets, attributs) et à des règles d'inférence adaptées. Cette multiplicité devrait permettre d'orienter la recherche des bases révisées et de les ordonner. Il semble donc que les gains à attendre de ce type de mécanisme justifient le développement de systèmes de révision spécifiques aux objets.

Afin d'explorer le bien-fondé et la faisabilité d'un mécanisme de révision dans une représentation de connaissance par objet, cette étude commence par définir un langage relativement restreint pour lequel des procédures rapides de déduction, inspirées des techniques couramment utilisées, sont possibles. Les propriétés du langage vis-à-vis de la déduction et de la révision sont établies formellement. En particulier, la caractérisation donnée de la révision fait correspondre les aspects syntaxique et sémantique. Ce langage sera ultérieurement étendu en cherchant à préserver les propriétés obtenues. Définir un langage spécifique et ne pas considérer d'emblée une logique d'objets établie (par exemple, F-logic [9]) permet de se restreindre aux cas simples. L'intérêt de cette approche est d'identifier les éventuels problèmes un par un plutôt que de se heurter de front aux problèmes d'indécidabilité ou de complexité de ces logiques [11]. L'inconvénient serait que l'approche n'autorise pas une extension continue vers d'autres systèmes plus élaborés.

La première partie de l'exposé définit le langage de représentation, sa sémantique et un système de déduction complet pour celle-ci. Une forme normalisée de base de connaissance (nommée forme première), utilisée par les algorithmes et par la révision, y est aussi introduite. La seconde partie définit la notion de base révisée et les critères de minimalité (syntaxique) et de proximité (sémantique) sur les bases révisées. Il est montré que les deux notions coïncident. La contribution de ce travail est alors replacée dans le cadre de son utilisation au sein d'un système réel, ce qui nécessite une vérification de la formulation et des extensions. Une définition plus complète de ce travail, incluant les preuves de toutes les propositions, est disponible sous forme de rapport de recherche [7].

1. Logique d'objets

Pour fixer le formalisme sur lequel la révision va s'appliquer, le langage d'expression des objets est défini (§1.1). Il est doté d'une sémantique correspondant à ce qui est attendu des représentations par objets considérées (§1.2). Un ensemble de règles d'inférence est proposé pour ce langage (§1.3); il permet d'assurer la complétude avec la sémantique (§1.4). Une forme normalisée des bases de connaissance, utilisée pour la révision, est ensuite présentée (§1.5).

1.1. Syntaxe

La syntaxe est découpée en deux parties : une première partie correspond à la grammaire donnant l'expression des entités de la base, alors que la seconde permet de considérer des types de données qui peuvent être ajoutés au système (entiers, booléens, mais aussi tout type défini par un type abstrait).

La figure 1 est la représentation graphique d'une base, contenant cinq classes (c_0, c_1, c_2, c_3, c_4), trois objets (ou instances) i, i' et i'' rattachées respectivement à c_2, c_3 et c_4 et dont un seul attribut a a été représenté.

Le langage permet de décrire des classes en relation de spécialisation, des objets — instances de ces classes — et des attributs typés dans les classes par un domaine et valués dans les objets par une valeur. La base de connaissance de la figure 1 peut s'exprimer syntaxiquement par :

$B=\{c_1 \leq c_0 ; c_2 \leq c_1 ; c_3 \leq c_2 ; c_4 \leq c_2 ; i \in c_2 ; i' \in c_3 ; i'' \in c_4 ; c_0.a \subseteq [-\infty\ +\infty]; c_1.a \subseteq [0\ 100]; c_2.a \subseteq [10\ 25]; c_4.a \subseteq [5\ 15]; i.a=24; i'.a=18\}$, selon la grammaire donnée ci-dessous.

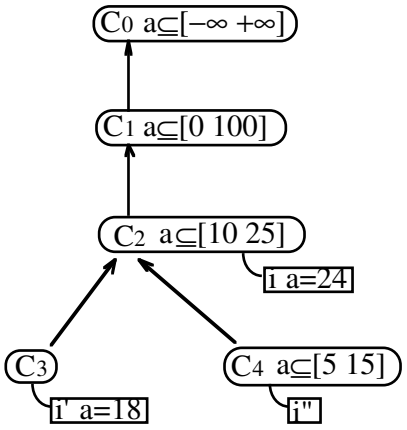

Figure 1 : exemple de base de connaissance représentée graphiquement.

Définition 1 : grammaire

<base_de_connaissance> ::= {<assertion>*}
<assertion> ::= <class_introduction> / <object_introduction> /
<class_slot_restriction> / <object_slot_valuation>
<class_introduction> ::= <class_name> ≤ <class_name>
<object_introduction> ::= <object_name> ∈ <class_name>
<class_slot_restriction> ::= <class_name>.<slot_name> ⊆ <domain>
<object_slot_valuation> ::= <object_name>.<slot_name> = <value>
<class_name> / <object_name> / <slot_name> ::= <identifier>

De plus, le graphe de la relation ≤ (appelée spécialisation) entre les <class_name> doit être sans circuit.

Opérations sur les types et les domaines

Un type définit un ensemble de valeurs et des opérations sur celles-ci. Les opérations suivantes sur les types sont utilisées [5]; elles sont indicées par le type τ des domaines ou des objets concernés. Soient d, d' des domaines et v, v' des valeurs.

$d \wedge_\tau d'$: intersection des domaines de valeurs;
$v :_\tau d$ ($\neg v :_\tau d$) : une valeur appartient (resp. n'appartient pas) à un domaine;
$d \wedge_\tau d' = \bot_\tau$: les deux domaines de valeurs sont disjoints pour le type τ;
$d \leq :_\tau d'$ ($d <:_\tau d'$) : le domaine d est inclus (resp. strictement) dans le domaine d';
$v \leq_\tau v'$: v est inférieur à v'; (τ, le type de v et v', étant ordonné).

Le symbole τ sera omis là où il ne sera pas nécessaire. Dans le cadre actuel, un sous-ensemble du modèle de connaissance est pris en compte en restreignant les valeurs aux entiers relatifs et les domaines aux intervalles d'entiers relatifs. Le seul type τ est donc entier :

<value> ::= <integer>
<domain> ::= [<bound-> <bound+>] / ⊥
<bound-> ::= <integer> / -∞
<bound+> ::= <integer> / +∞

1.2. Sémantique

La sémantique est définie par une fonction d'interprétation des objets vers un domaine en tenant compte des types utilisés dans la syntaxe. Les notions de modèle, conséquence, inconsistance et incohérence sont définies ci-dessous. Afin de caractériser le domaine de la fonction d'interprétation, on identifie d'abord les ensembles de symboles manipulés.

Définition 2 : ensembles d'entités

Les ensembles suivants sont définis :

T_C^B l'ensemble des classes (c) introduites dans la base B par une assertion du type $c \leq c'$ ou $c.a \subseteq d$.

T_O^B l'ensemble des objets (o) introduits dans la base B par une assertion du type $o.a = v$ ou $o \in c$.

T_A^B l'ensemble des attributs (a) introduits dans la base B par une assertion du type $o.a = v$ ou $c.a \subseteq d$.

T_V^B l'ensemble des valeurs des types introduits dans la base B (ici les entiers).

T_T^B l'ensemble des domaines des types introduits dans la base B (ici les intervalles d'entiers).

L'interprétation se définit pour les objets de la base, mais aussi pour les valeurs et domaines.

Définition 3 : interprétation de type

Soit un type τ, une interprétation des expressions (domaines et valeurs) du type τ est une fonction I: $T_V^B \to \tau$ et de $T_T^B \to 2^\tau$ satisfaisant les condition suivantes :

$I[[v\ v']] = \{v'' \in \tau,\ I[v] \leq_\tau v'' \leq_\tau I[v']\}$ (avec $\forall v \in \tau,\ I[-\infty] \leq_\tau v \leq_\tau I[+\infty]$)

$I[\bot_\tau] = \emptyset$

$I[d \wedge_\tau d'] = I[d] \cap I[d']$

$d <:_\tau d' \Leftrightarrow I[d] \subset I[d']$

$v :_\tau d \Leftrightarrow I[v] \in I[d]$

$\neg v :_\tau d \Leftrightarrow I[v] \notin I[d]$

$v \neq v' \Leftrightarrow I[v] \neq I[v']$.

Dans le cas présent, un domaine est interprété par un intervalle d'entiers et une valeur par un entier. La fonction d'interprétation va permettre de définir la relation entre la base et un domaine particulier.

Définition 4 : interprétation

Soit une base de connaissance B et un domaine D quelconque. Une interprétation est un couple $<D, I_B>$, I_B étant appelée fonction d'interprétation :

I_B: $T_C^B \to 2^D$

I_B: $T_O^B \to D$

I_B: $T_A^B \to (D \to \tau)$

I_B: $T_V^B \to \tau$

I_B: $T_T^B \rightarrow 2^\tau$

La restriction de I_B à T_V^B et T_T^B est une fonction d'interprétation du type τ (dans notre cas, τ=**Z**) et I_B est injective sur T_O^B (hypothèse de nom unique pour les objets).

Une interprétation du langage associe donc à un objet, un élément du domaine, à une classe, un ensemble de tels éléments, et à un attribut, une fonction du domaine vers le type. Un certain nombre d'observations peuvent être faites sur la sémantique du langage proposé :

(1) La multi-spécialisation (la possibilité pour une classes d'être sous-classe de plusieurs classes incomparables) est autorisée. Plus généralement, aucune structure n'est imposée au graphe de la relation ≤ (hors l'absence de circuit).

(2) La multi-instanciation (la possibilité pour un objet d'être rattaché à plusieurs classes incomparables) est autorisée.

(3) La multi-valuation (c'est-à-dire la faculté d'avoir plusieurs valeurs dans un attribut — comme c'est le cas d'une relation en général —, par opposition à avoir une collection de valeurs) est interdite par la sémantique des attributs qui sont des fonctions (à valeurs dans τ ici).

(4) La valeur d'attribut d'un objet ne peut être un autre objet (c'est la limitation principale de ce langage).

Notation :

Pour un attribut a, $I_B[a|c]$ exprime la restriction de $I_B[a]$ à $I_B[c]$ (c'est-à-dire aux éléments de l'interprétation de cette classe) et $I_B[a|c] \subseteq I_B[d]$ exprime que $I_B[a|c]$ est une fonction de $I_B[c]$ dans $I_B[d]$. L'indice B sera omis lorsqu'il n'y a pas d'ambiguïté.

La définition de la satisfiabilité d'une assertion (le fait qu'elle soit considérée comme vraie dans une interprétation) et de modèle (les interprétations pour lesquelles toutes les assertions de la base sont considérées comme vraies) sont données ci-dessous.

Définition 5 : satisfaction d'une assertion

$\models_M \delta$ signifie qu'une interprétation $M=\langle D,I\rangle$ satisfait une assertion δ :

$\models_M c \leq c'$	ssi	$I[c] \subseteq I[c']$
$\models_M c.a \subseteq d$	ssi	$I[a\|c] \subseteq I[d]$
$\models_M o \in c$	ssi	$I[o] \in I[c]$
$\models_M o.a = v$	ssi	$I[a](I[o]) = I[v]$

Définition 6 : modèle

$M=\langle D, I_B\rangle$ est un modèle de B, noté $\models_M B$, ssi M satisfait toutes les assertions de B.

Ces définitions introduisent des contraintes sur la fonction d'interprétation d'un modèle d'une base qui correspondent à ce qui est attendu d'un modèle à objets :

L'interprétation d'une classe doit être incluse dans l'interprétation de chacune de ses sur-classes et dans l'ensemble d'éléments du domaine qui satisfont les restrictions de ses attributs. Si aucune contrainte n'est posée sur la classe (par exemple pour la racine d'une hiérarchie), son interprétation est simplement inclue dans D.

L'interprétation d'un objet doit appartenir à l'interprétation de chacune des classes à laquelle il est rattaché et à l'ensemble des objets qui satisfont les contraintes posées sur la valeur de ses attributs.

L'interprétation d'un attribut d'une classe doit être inclue dans l'interprétation de ce même attribut pour les surclasses et dans l'interprétation du domaine, restriction de l'attribut pour cette classe particulière. Si aucune contrainte n'est posée sur un attribut, son domaine est simplement inclus dans τ.

L'interprétation d'un attribut d'un objet doit appartenir à l'interprétation de l'attribut de ses classes et aux interprétations des valeurs données pour cet attribut.

Les notions classiques de conséquence (ce qui est vrai dans tous les modèles d'une base particulière) et de base inconsistante (les bases n'ayant pas de modèle) sont maintenant introduites.

Définition 7 : conséquence

Soient B une base et δ une assertion, δ est une conséquence de B, noté $B \models \delta$, ssi :

$$\forall \text{M tel que } \models_M B,\ \models_M \delta .$$

La négation de la conséquence est notée $\not\models$.

Définition 8 : base inconsistante

B est inconsistante ssi il n'existe pas de modèle de B.

Par exemple, la base de la figure 1 à laquelle est simplement ajoutée l'assertion : $i''.a=7$ est inconsistante (figure 2). En effet, $(i''.a=7) \in B$, $(i'' \in c_4) \in B$, $(c_2.a \subseteq [10\ 25]) \in B$ et $(c_4 \leq c_2) \in B$. Tous les modèles doivent donc satisfaire : $I[a|c_4] \subseteq [10\ 15]$ d'où $I[a](I[i'']) \in \varnothing$, ce qui est impossible.

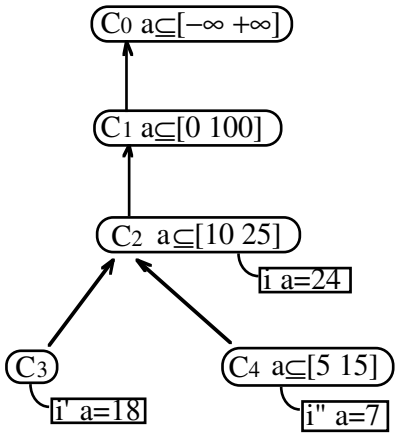

Figure 2 : exemple de base inconsistante.

Cette définition d'inconsistance n'est pas assez forte pour garantir que l'utilisateur d'une base de connaissance ne construit pas de classe inutile. La notion de base incohérente, dans laquelle une classe ne peut avoir d'instance dans aucun des modèles, permet de mieux contrôler son action.

Définition 9 : classe incohérente, base incohérente

Une classe c est incohérente dans une base B ssi elle est vide dans tout modèle de B ($I[c]=\varnothing$); une base B est incohérente ssi elle contient une classe incohérente.

1.3. Déduction syntaxique

Calculer l'ensemble des modèles d'une base de connaissance pour en connaître les conséquences est très coûteux. Afin d'accéder à ces conséquences, on définit un système déductif par un ensemble de règles d'inférence. La complétude sera étudiée au paragraphe suivant (§1.4).

Définition 10 : règles d'inférence

Pour tout $c, c', c'', c_i \in T_C^B$, $o \in T_O^B$, $a \in T_A^B$, $v \in T_V^B$ et $d, d', d_i \in T_T^B$:

(≤-reflexivité) $\dfrac{}{c \leq c}$ (d-intersection) $\dfrac{c.a \subseteq d \quad c.a \subseteq d'}{c.a \subseteq d \wedge_\tau d'}$

(≤-transitivité) $\dfrac{c \leq c' \quad c' \leq c''}{c \leq c''}$ (d-héritage) $\dfrac{c.a \subseteq d \quad c' \leq c}{c'.a \subseteq d}$

(∈-clôture) $\dfrac{c \leq c' \quad o \in c}{o \in c'}$ (d-expansion) $\dfrac{c.a \subseteq d}{c.a \subseteq d'}\ d <:_\tau d'$

(o-valuation) $\dfrac{\{c_i.a \subseteq d_i \quad o \in c_i\}}{o.a = v}\ \{v\} = \underset{i}{\wedge} d_i$

Les deux premières règles expriment la réflexivité et la transitivité de la relation de spécialisation, la troisième qu'un objet est instance de toutes les surclasses de la classe à laquelle il est rattaché. Les quatrième et cinquième règles signifient que la contrainte portant sur un attribut d'une classe est l'accumulation de toutes celles posées sur cet attribut dans la classe et dans ses surclasses. La sixième règle est une règle technique permettant d'obtenir la complétude (elle permet d'ajouter des contraintes plus faibles que la contrainte existante). Enfin, la dernière règle indique que si, pour un ensemble de classes, l'intersection des restrictions de domaine d'un attribut est réduit à un singleton, les objets qui appartiennent à l'ensemble de ces classes possèdent cette valeur pour l'attribut considéré.

La clôture déductive d'une base est obtenue en appliquant les règles tant que cela est possible.

Définition 11 : clôture d'une base

Soit B une base, la clôture de la base B est la base notée Cn(B) (pour «conséquence») telle que :

(1) $B \subseteq \mathrm{Cn}(B)$,

(2) Pour tout $f_1, \ldots f_n \in \mathrm{Cn}(B)$ et toute règle $\dfrac{f_1 \ldots f_n}{f}\ co$, si co est satisfaite, $f \in \mathrm{Cn}(B)$,

(3) Cn(B) ne contient pas d'autres assertions.

Définition 12 : déduction

B permet de déduire l'assertion δ, noté $B \vdash \delta$ ssi $\delta \in \mathrm{Cn}(B)$.

B permet de déduire la base B', noté $B \vdash B'$ ssi pour tout $\delta \in B'$, $B \vdash \delta$.

La négation de la déduction est notée $\nvdash$.

La déduction permet d'inférer les conséquences de la base. Dans les systèmes à objets, elle correspond à l'héritage, à l'inférence de type ou de valeur.

1.4. Complétude

Établir le pont entre la déduction syntaxique ($\vdash$) et la conséquence sémantique ($\models$) permet de s'assurer que ce qui est déduit syntaxiquement est bien valide sémantiquement et vice versa. Une caractérisation syntaxique d'une base inconsistante ou incohérente permet de ne pas utiliser la notion de modèle pour détecter l'inconsistance.

La correction est assez naturellement acquise, vu que les règles d'inférence décrivent un mécanisme de déduction conforme aux propriétés classiques d'une fonction d'interprétation.

Proposition 1 : correction

Si $B \vdash \delta$ alors $B \models \delta$.

L'expression syntaxique de l'inconsistance est plus délicate que dans les systèmes logiques :

Proposition 2 : inconsistance syntaxique

B est inconsistante ssi :

il existe o, c, a, d et v tels que $B \vdash o \in c$, $B \vdash c.a \subseteq d$, $B \vdash o.a=v$, et $\neg v:d$
ou il existe o, a, v et v' tels que $B \vdash o.a=v$ et $B \vdash o.a=v'$ et $v \neq v'$.

Cette proposition se vérifie sur l'exemple de base inconsistante (figure 2). En effet :
$(i''.a=7) \in B$, donc $B \vdash i''.a=7$,
$(i'' \in c_4) \in B$ et $(c_4 \leq c_2) \in B$ donc d'après la règle de ($\in$-transitivité) $B \vdash i'' \in c_2$,
$(c_2.a \subseteq [10\ 25]) \in B$ donc $B \vdash c_2.a \subseteq [10\ 25]$ et $\neg 7:[10\ 25]$.

La complétude faible est l'inverse de la correction (tout ce qui est vrai dans tous les modèles est dérivable). Elle est appelé ainsi pour la distinguer de la complétude forte qui rassemble correction et complétude. La complétude faible n'est acquise que lorsque la base est consistante. En effet, une base inconsistante n'ayant pas de modèle, elle permet de valider n'importe quelle assertion, ce qui n'est pas le cas des règles d'inférence. Ceci est une propriété de localité typique des systèmes à objets. La proposition précédente est alors indispensable car elle permet de caractériser syntaxiquement l'inconsistance.

Proposition 3 : complétude faible

Si B est consistante et $B \models \delta$ alors $B \vdash \delta$.

De même qu'il existe une notion d'inconsistance syntaxique, il est intéressant de caractériser syntaxiquement l'incohérence pour la détecter et l'éviter.

Proposition 4 : incohérence syntaxique

B est incohérente ssi il existe c et a tels que $B \vdash c.a \subseteq \perp_\tau$

1.5. Forme première

Les systèmes de représentation de connaissance ne manipulent pas les bases sous forme de modèles ou de clôture déductive (souvent infinie dans le cas simple présenté ici à cause des intervalles d'entiers), mais sous une forme normalisée permettant de déterminer rapidement inconsistance et conséquence. La forme première, qui sera utilisée lors de la

révision, permet de ramener une base de connaissance à une forme unique, qui contient le minimum d'assertions nécessaires à la déduction de la base. Elle supprime de la clôture déductive l'ensemble des assertions qui sont déductibles par les règles d'inférence.

Définition 13 : forme première

Soit $\mathrm{Cn}'(B)=\mathrm{Cn}(B)-\{c\leq c\in \mathrm{Cn}(B)\}$, la réduction réflexive de Cn(B).

La forme première d'une base B, notée FP(B), se définit par :

FP(B) = Cn′(B)

– $\{(c \leq c'');\ (c \leq c'),\ (c' \leq c'') \in \mathrm{Cn}'(B)\}$ (≤-réduction)

– $\{(o \in c);\ (o \in c'),\ (c' \leq c) \in \mathrm{Cn}'(B)\}$ (∈-réduction)

– $\{(c.a \subseteq d);\ (c \leq c'),\ (c'.a \subseteq d) \in \mathrm{Cn}'(B)\}$ (d-anti-héritage)

– $\{(c.a \subseteq d);\ (c.a \subseteq d') \in \mathrm{Cn}'(B)$ et $d' <:_\tau d\}$ (d-normalisation)

– $\{(o.a = v);\ \{c_i.a \subseteq d_i \quad o \in c_i\} \in \mathrm{Cn}'(B)$ et $\{v\} = \underset{i}{\wedge} d_i\}$ (o-anti-valuation)

Il est nécessaire de considérer l'ensemble Cn′(B), afin de ne faire que les retraits pertinents. La règle de (d-intersection) du système déductif ne trouve pas son pendant dans cette définition vu que ce qu'elle permet de déduire (l'assertion synthétisant toutes les contraintes sur un attribut pour une classe) est justement à conserver au détriment de toutes les autres assertions concernant les restrictions d'attribut (qui sont supprimées par la règle de (d-anti-héritage)).

Conséquences 5 :

(1) FP(B) $\vdash B$

(2) $B \vdash$ FP(B)

(3) FP(B) est unique.

Exemple :

La base prise comme exemple :

$B=\{c_1\leq c_0\ ;\ c_2\leq c_1\ ;\ c_3\leq c_2\ ;\ c_4\leq c_2\ ;\ i\in c_2\ ;\ i'\in c_3\ ;\ i''\in c_4\ ;\ c_0.a\subseteq[-\infty\ +\infty];\ c_1.a\subseteq[0\ 100];\ c_2.a\subseteq[10\ 25];\ c_4.a\subseteq[5\ 15];\ i.a=24;\ i'.a=18\}$

est déjà pratiquement sous forme première (ce qui montre l'aspect intuitif de la forme première) : $\mathrm{FP}(B)=\{c_1\leq c_0\ ;\ c_2\leq c_1\ ;\ c_3\leq c_2\ ;\ c_4\leq c_2\ ;\ i\in c_2\ ;\ i'\in c_3\ ;\ i''\in c_4\ ;\ c_0.a\subseteq[-\infty\ +\infty];\ c_1.a\subseteq[0\ 100];\ c_2.a\subseteq[10\ 25];\ \boldsymbol{c_4.a\subseteq[10\ 15]};\ i.a=24;\ i'.a=18\}$

L'intérêt théorique principal de la forme première réside dans sa minimalité qui fait que les retraits y sont effectifs (c'est-à-dire que, si une assertion est supprimée d'une base sous forme première, celle-ci ne peut plus être déduite).

Proposition 6 : minimalité de la forme première

Pour tout δ dans FP(B), FP(B)–$\{\delta\} \nvdash \delta$.

L'intérêt plus pratique de cette forme première (ou d'autres formes normalisées adaptées aux représentations par objets) est la possibilité de l'obtenir facilement (par son calcul à partir d'une base quelconque) et de la manipuler de façon efficace pour déterminer la consistance, la cohérence ou une conséquence particulière d'une base.

Les bases de connaissance gagnent à être stockées et maintenues sous forme normalisée (vu leur petite taille). De plus, pour déduire une assertion particulière (ou vérifier qu'une

assertion particulière est déductible), les procédures de déductions peuvent être guidées de manière à aboutir rapidement à une conclusion car il suffit de déclencher :

- *les règles* qui sont utiles pour le type de l'assertion à déduire (ainsi, pour déduire une assertion du type $c \leq c'$, nul besoin des règles de (d-héritage) ou de ($\in$-transitivité))
- *sur des assertions* dont les composants (objets ou classes) sont utiles en fonction de leur place respective par rapport à l'ordre de spécialisation entre les classes (par exemple, pour déduire une assertion du type $o \in c$, seules les assertions portant sur les classes supérieures à c (par rapport à $\leq$) peuvent être utiles dans les règles d'inférence).

La conception et l'évaluation de tels algorithmes n'ont pas été complètement réalisés pour l'instant mais il semble raisonnable de penser que pour un langage du type de celui présenté ici, il est possible de développer des algorithmes polynomiaux.

Au delà de cet aspect algorithmique, la forme première est utilisée dans la définition de la révision.

2. Révision

La définition de base révisée proposée correspond à celle donnée généralement. Les bases révisées peuvent s'exprimer sous une forme particulière utilisant la forme première (§2.1). Nous définissons la minimalité sémantique (la proximité) et une relation d'ordre entre bases révisées, basées sur les assertions supprimées et ajoutées, qui permet de donner une définition syntaxique de la minimalité des bases révisées (§2.2). L'équivalence de ces deux minimalités est montrée.

2.1. Base révisée

La notion de révision va agir sur des bases acceptables (c'est-à-dire qui correspondent à des bases que les utilisateurs acceptent).

Définition 14 : base acceptable, base inacceptable

Une base B est acceptable ssi elle est consistante et cohérente. Dans le cas contraire, elle est inacceptable.

Définition 15 : base révisée

Soient B une base acceptable et δ une assertion telle que $B \cup \{\delta\}$ soit inacceptable. B' est une base révisée de (B,δ) ssi $B' \cup \{\delta\}$ est acceptable et $\mathrm{Cn}(B') \subset \mathrm{Cn}(B)$ (ce qui se traduit sémantiquement par $B \models B'$).

La notion de révision présentée ici satisfait les six premiers postulats de [1], ainsi que ceux proposés dans [10, §7.1.2]. Elle correspond donc à la notion de révision couramment utilisée. En cas d'inconsistance ou d'incohérence, elle permet de se replier sur une base B' plus réduite que B mais compatible avec l'ajout de δ.

Les bases révisées vont être exprimées sous une forme plus adaptée à leur manipulation. Intuitivement, cette écriture d'une base révisée indique qu'à partir de la forme première, un certain nombre d'assertions (B'_-) sont supprimées — qui ne seront effectivement plus déductibles grâce à la minimalité de la forme première — et que certaines (B'_+), déjà

déductibles auparavant, sont cependant conservées. Pour avoir une forme plus simple, l'intersection de ces deux ensembles est vide (sinon des éléments retranchés seraient ré-ajoutés).

Proposition 7 :

Si B' est une base révisée de (B,δ), alors B' peut se caractériser de manière unique par l'écriture suivante : $B'=(\mathrm{FP}(B)-B'_-)\cup B'_+$ avec comme contraintes :

- $B'_- \cap B'_+ = \varnothing$,
- $B'_- \subseteq \mathrm{FP}(B)$ et
- $B'_+ \subseteq \mathrm{Cn}(B)$.

Jusqu'ici les bases révisées ne sont pas contraintes à être exprimées sous forme première : B'_+ peut contenir des informations déductibles du reste de la base. Différentes bases révisées peuvent donc avoir la même clôture déductive (et donc la même forme première). Dans la suite, on ne considérera plus que l'ensemble des bases révisées quotientées par la relation d'équivalence «avoir la même image par Cn (ou par FP)». L'expression «base révisée» désignera donc la représentante sous forme première d'une classe de bases révisées. Une base révisée est donc caractérisée par $B'=(\mathrm{FP}(B)-B'_-)\cup B'_+$ avec la contrainte supplémentaire qu'elle soit sous forme première.

Exemple :

Différentes bases révisées possibles de la base inconsistante de la figure 2 : $\mathrm{FP}(B)=\{c_1\leq c_0 ; c_2\leq c_1 ; c_3\leq c_2 ; c_4\leq c_2 ; i\in c_2 ; i'\in c_3 ; i''\in c_4 ; c_0.a\subseteq[-\infty\ +\infty]; c_1.a\subseteq[0\ 100]; c_2.a\subseteq[10\ 25]; c_4.a\subseteq[10\ 15]; i.a=24; i'.a=18\}$ avec δ=(i".a=7) peuvent se représenter graphiquement ainsi :

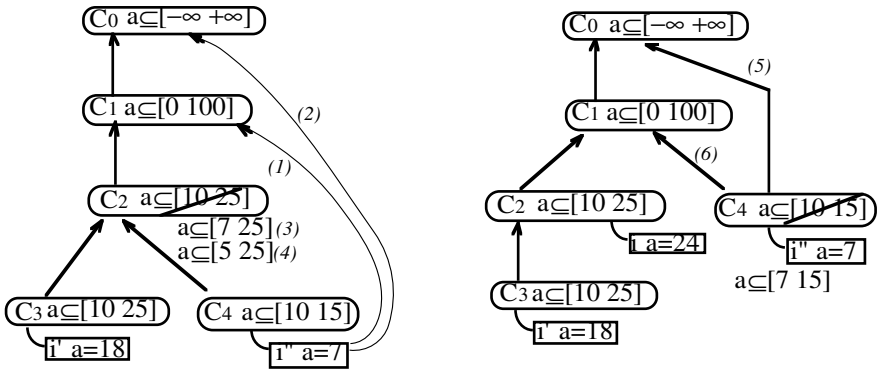

Figure 3 : exemples de bases révisées possibles : (1) et (2) : l'instance est déplacée sous une autre classe (suppression de son appartenance à sa classe d'origine et rattachement à l'une de ses surclasses (puisque les ajouts sont à faire dans $\mathrm{Cn}(B)$)). (3) et (4) : une assertion de restriction de domaine est supprimée (elle peut être remplacée par une autre — le domaine étant plus grand que le précédent). (5) et (6) : la classe est déplacée en supprimant une assertion de spécialisation et en la remplaçant par d'autres et sa restriction de domaine est modifiée.

Selon la notation donnée, ces solutions se détaillent comme suit :

$B_1 = \mathrm{FP}(B) - \{i''\in c_4\} \cup \{i''\in c_1\}$

$B_2 = \mathrm{FP}(B) - \{i''\in c_4\} \cup \{i''\in c_0\}$

$B_3 = \mathrm{FP}(B) - \{c_2.a\subseteq[10\ 25]; c_4.a\subseteq[10\ 15]\} \cup \{c_2.a\subseteq[7\ 25]; c_4.a\subseteq[7\ 15]\}$

$B_4 = \mathrm{FP}(B) - \{c_2.a\subseteq[10\ 25]; c_4.a\subseteq[10\ 15]\} \cup \{c_2.a\subseteq[5\ 25]; c_4.a\subseteq[5\ 15]\}$

B_5 = FP(B) – $\{c_4 \leq c_2;\ c_4.a \subseteq [10\ 15]\} \cup \{c_4 \leq c_0;\ c_4.a \subseteq [7\ 15]\}$

B_6 = FP(B) – $\{c_4 \leq c_2;\ c_4.a \subseteq [10\ 15]\} \cup \{c_4 \leq c_1;\ c_4.a \subseteq [7\ 15]\}$

Il s'agit bien de bases révisées. D'autres solutions existent (comme FP(B)–$\{c_4 \leq c_2;\ c_2.a \subseteq [10\ 25]\}$), mais elles ne correspondent pas à ce qui est attendu intuitivement; c'est pourquoi il faut définir une notion de minimalité permettant de trouver les bases révisées qui modifient le moins la base initiale. Pour ce faire, les notions de proximité (d'un point de vue sémantique) et de minimalité (d'un point de vue syntaxique) sont définies. Nous montrerons que ces deux notions coïncident. À noter que cette relation entre bases révisées possibles ne permet pas généralement de définir une unique base révisée minimale.

2.2. Proximité et minimalité

Lors de la révision, la base de connaissance doit être modifiée de manière à ce qu'elle devienne acceptable tout en s'éloignant le moins possible de la base initiale. Cette notion s'exprime sémantiquement par la notion de proximité.

Définition 16 : base révisée la plus proche possible

B' est une base révisée la plus proche possible de (B,δ) ssi

$$\forall \gamma \text{ tel que } B \models \gamma \text{ et } B' \cup \{\delta\} \not\models \gamma \text{ alors } \forall M \not\models_M B' \cup \{\delta\} \cup \{\gamma\}.$$

Intuitivement B' est (une base révisée) la plus proche possible de (B,δ), au sens où aucune base révisée de (B,δ) ne permet de satisfaire plus d'assertions (au sens de l'inclusion ensembliste) de B (que B'). Cette préservation des assertions de B se fait au détriment des modèles.

La notion de minimalité (des changements opérés sur la base) peut être définie par un ordre sur les modifications opérées. Cet ordre est défini sur l'écriture caractéristique de base révisée définie plus haut (proposition 7).

Définition 17 : ordre sur les bases révisées

Si B' et B'' sont deux bases révisées de (B,δ), B' est plus petite que B'', noté $B' \propto B''$, ssi $B'_- \subseteq B''_-$ et $B' \vdash B''_+$.

Intuitivement, cela montre que B' est plus petite que B'' si B' a moins d'assertions supprimées et qu'elle permet de déduire tout ce qui a été rajouté dans B''. Ceci permet de guider la recherche des bases révisées à l'aide de l'ordre. Bien entendu, ce travail reste complexe puisque la relation d'ordre fait appel à la clôture déductive.

Exemple :

Dans l'exemple de base inconsistante précédent, on a donc :

$B_1 \propto B_2$ (car $B_1 \vdash i'' \in c_0$),

$B_3 \propto B_4$ (car $B_3 \vdash c_2.a \subseteq [5\ 25]$ et $B_3 \vdash c_4.a \subseteq [5\ 15]$) et

$B_6 \propto B_5$, (car $B_6 \vdash c_4 \leq c_0$).

Les autres bases révisées ne sont pas comparables.

Proposition 8 :

Soient B' et B'' deux bases révisées de (B,δ), $B' \propto B''$ ssi Cn(B'')$\subseteq$Cn(B').

Définition 18 : base révisée minimale

B' est une base révisée minimale de (B,δ) ssi B' est une base révisée de (B,δ) et qu'il n'existe pas $B'' \neq B'$ telle que B'' soit une base révisée de (B,δ) et que $B'' \propto B'$.

Proposition 9 : équivalence de la proximité et de la minimalité

B' est une base révisée minimale de (B,δ) ssi B' est une base révisée la plus proche possible de (B,δ).

Ainsi, la notion de base révisée minimale s'exprime syntaxiquement et sémantiquement. Dès lors, les procédures de révision qui peuvent être mises en œuvre syntaxiquement auront le sens de «conserver le maximum d'assertions valides». Ceci est possible pour l'instant grâce à la restriction du langage proposé. L'objectif des travaux futurs va donc être (a) de chercher à conserver ce résultat pour des langages étendus et (b) d'évaluer la complexité théorique et l'applicabilité pratique des techniques développées.

Les mécanismes de révision ont été peu développés, jusqu'à présent, pour des raisons de complexité. En effet, tous les moyens de réviser une base de connaissance ont besoin d'un oracle leur disant si la base est consistante ou non (dans la définition de minimalité donnée ci-dessus, la déduction est utilisée encore plus généralement). Si la complexité de cette décision est exponentielle, la révision est véritablement impraticable [11].

Dans le cas présenté ci-dessus, cette déduction devrait être polynomiale (et semble-t-il quadratique) dans le pire des cas et l'expérience pratique montre que les systèmes à objets (du moins ceux relativement simples : sans problème de satisfaction de contraintes, ni concepts récursifs ou restriction de rôles ouverts) permettent de décider rapidement de la consistance du système. Alors, l'idée de trouver de bonnes heuristiques pour la révision de bases d'objets vaut la peine d'être explorée.

3. Retour aux objets

Les travaux présentés jusqu'à présent sont quelque peu éloignés des objets tels qu'ils sont habituellement considérés. Pour pouvoir envisager de les exploiter dans le cadre d'un système de représentation de connaissance concret, il faut disposer d'une articulation liant le système et le langage présenté ici (§3.1), ajouter un certain nombre d'expressions qui ne figurent pas dans le langage restreint et examiner si le résultat reste malgré tout praticable (§3.2).

3.1. Relation avec TROPES

Le langage qui a été présenté n'est pas celui utilisé par le système de représentation de connaissance TROPES [13] sur lequel la révision sera implémentée. Pour cela, la révision présentée ici devra être étendue à un langage plus riche et adaptée aux constructions de TROPES. La liaison entre les deux systèmes doit être détaillée et il faut donc considérer que le système TROPES a pour mission de s'assurer qu'il construit bien une base au sens énoncé plus haut.

En TROPES, les objets et les valeurs sont partitionnés en un ensemble de concepts et un ensemble de types. Aux valeurs correspond un domaine (qui sera T_V dans le langage présenté ici) et aux objets un domaine (qui sera T_O). Les objets des concepts sont munis d'attributs, chaque concept constituant un espace de nom différent pour les noms de ces

attributs. Ceux-ci sont transformés en un ensemble d'attributs T_A dans le langage ci-dessus. Les concepts peuvent être munis de différents points de vue qui sont importants pour nommer les classes mais n'ont aucune contrepartie dans le langage ci-dessus. Chaque point de vue contient une hiérarchie de classes ordonnées en un arbre par la relation de spécialisation. L'espace de nommage des classes étant le point de vue, ici encore chaque classe aura un nom unique et l'ensemble des classes constituera donc T_C. L'ordre correspondant à la spécialisation sera rendu par des assertions de type ($c \leq c'$) et les contraintes posées sur les attributs dans les classes seront traduites par des assertions de type ($c.a \subseteq d$). La racine de chaque hiérarchie recevra donc aussi les contraintes portant sur les attributs du concept. Chaque objet dans T_O fait partie d'un et d'un seul concept et est rattaché à une unique classe par point de vue ce qui est rendu par des assertions de type ($o \in c$). Il faut noter que l'attachement multiple n'a pas été prohibé jusqu'à maintenant. Enfin, les valeurs d'attributs des objets seront traduites par ($o.a=v$). Un exemple de base correspondant à l'exemple 1 est donné dans la version longue de l'article [7].

TROPES est capable de vérifier un certain nombre de propriétés dans la construction des bases : que les noms sont uniques, que les objets n'appartiennent qu'à un seul concept, que les valeurs et domaines des attributs sont bien typés, que les taxonomies sont des arbres, etc. Il est capable d'émettre des messages d'erreurs et de proposer des corrections immédiates suite à ces vérifications.

Une fois ce premier niveau de vérification passé, la base peut être considérée comme une base au sens énoncé ci-dessus et les mécanismes de révision pourront s'appliquer en retranchant ou en ajoutant des éléments à la base TROPES. Le langage d'expression accepte aussi des bases qui ne sont pas des bases TROPES (par exemple, la spécialisation multiple n'a pas été prohibée) mais il suffit, pour détecter des inconsistances et proposer des révisions dans les bases TROPES, de pouvoir exprimer celle-ci dans le langage proposé.

3.2. Extensions

Le langage présenté ici est très restreint. Il n'est qu'une base à partir de laquelle des extensions pourront être ajoutées en surveillant les propriétés de la déduction et de la révision. Dans le cadre du système TROPES, les extensions suivantes devront être considérées :

Il faut pouvoir considérer divers types de données pas forcément aussi simples à manipuler que les entiers (TROPES autorise des types tels que les ensembles de dates). Par ailleurs, le langage d'expression des types est plus riche que les intervalles d'entiers (par exemple, TROPES manipule des unions d'intervalles pour les entiers). Cette extension pose un problème intéressant pour la révision (quelle est la révision minimale dans les unions d'intervalles ?) et risque d'amener un surcroît de complexité au problème.

La plus importante des extensions est celle qui autorise la valeur d'un attribut à être un objet et non une simple valeur. Le problème est délicat (en particulier, la complexité risque d'en pâtir) s'il n'y a pas de référence circulaire; il le devient encore plus lorsque les références circulaires sont autorisées (c'est-à-dire lorsqu'un objet fait référence à un autre objet qui le réfère à son tour).

Une extension plus typique de TROPES concerne l'ajout de passerelles (expressions s'interprétant comme $I[c_1] \cap \ldots \cap I[c_n] \subseteq I[c]$). Ce genre d'extension ne semble pas poser de problèmes particuliers. Sa révision a été traitée en temps polynomial dans un cadre cependant plus restreint (pas de types ni d'attributs) [15].

Enfin, l'intégration à TROPES de problèmes de satisfaction de contraintes posera d'importants problèmes de complexité habituels dans ce contexte.

4. Comparaison avec d'autres travaux

Bernhard Nebel [10] a ouvert la voie des travaux concernant la révision dans les formalismes de représentation de connaissance dans un cadre différent de la logique. Mais la complexité du problème de la décision dans les systèmes considérés n'a pas permis de poursuivre. Ces travaux sont cependant précieux pour évaluer la complexité des problèmes considérés [11].

Récemment Norman Foo [8] a proposé quelques méthodes pour réviser les bases de graphes conceptuels et de nombreux travaux ont étudié la réorganisation de bases d'objets (à l'aide d'invariants à respecter et de règles conservant ces invariants) [4, 12, 2]. Si ceux-ci permettent de se faire une idée des méthodes pour retrouver une base acceptable, ils ne considèrent pas explicitement la sémantique du formalisme de représentation de connaissance.

Dans de récents travaux, Theodorados [15] s'est attaqué au même problème que le travail présenté ici. Il part cependant de bases quelque peu différentes : le langage considéré est plus puissant dans l'expression des rapports entre les classes mais les attributs ne sont pas pris en compte. À partir d'une modélisation logique du langage de définition de classes, il définit deux procédures de révisions classiques. Il montre (pour des notions particulières de révision) la co-NP-complétude du problème général et des résultats de polynomialité dans plusieurs restrictions intéressantes pour TROPES.

5. Conclusion

Le travail présenté ici aborde la révision (ou la correction d'erreurs) dans une base de connaissance par objets sous un aspect formel. À cette fin, un langage de représentation de connaissance par objets restreint a été défini. Il a été doté d'une sémantique et d'un système déductif correct et complet. Sur cette base, la notion de révision a été définie. Cette révision est pourvue de critères de minimalité (syntaxique) et de proximité (sémantique) permettant d'ordonner les bases révisées. Ces deux notions coïncident, ce qui permet de définir des procédures valides sémantiquement sans faire appel aux ensembles de modèles.

Au delà du simple résultat formel, la notion de forme première d'une base de connaissance permet d'envisager des techniques de réalisation efficace de ces outils. Ce genre d'outil ne se cantonne pas à la représentation de connaissance. Il devrait être applicable à un langage orienté objet typé pour aider un concepteur, voire une application, à corriger les problèmes qui peuvent apparaître.

Deux perspectives permettent de baliser le développement futur de ce travail :

- une extension du langage utilisé et un aménagement des notions introduites;
- la mise en évidence de l'intérêt et de la faisabilité de cette approche grâce à l'implémentation.

Le but à terme est d'intégrer au système TROPES un mécanisme permettant, lors de la détection d'une erreur, d'indiquer à l'utilisateur les façons minimales permettant de corriger celle-ci. Nous pensons qu'un tel résultat est possible et qu'il peut être exporté vers d'autres formalismes de représentation de connaissance et de programmation par objets.

Remerciements

Nous tenons à remercier Roland Ducournau pour avoir lu les premières versions de ce document et contribué à l'améliorer notablement.

Références

[1] C. Alchourrón, E. Gärdenfors, P. Makinson. *On the logic of theory change : partial meet contraction and revision functions*. Journal of symbolic logic. 50(2):510-530. 1985.

[2] J. Barnejee, W. Kim, H.J. Kim, H. Korth. *Semantics and implementation of schema evolution in object-oriented databases*. SIGMOD records 16(3):311-322. 1987

[3] C. Barral, S. Krauss, J. Minker, V.S. Subrahmanian. *Combining knowledge bases consisting of first-order theories.* Computational Intelligence. 8(1):45-71. 1992.

[4] A. Borgida. *Language features for flexible handling of exceptions in informations systems.* ACM transactions on database system. 10(4):565-603. 1985.

[5] C. Capponi. *Identification et exploitation des types dans un modèle de connaissances à objets.* Thèse d'informatique. Université Joseph Fourier. Grenoble (FR). 1995.

[6] I. Crampé, J. Euzenat. *Révision interactive dans une base de connaissance à objets.* Actes 10ième RFIA. Rennes. pp.615-623. 1996.

[7] I. Crampé, J. Euzenat. *Fondements de la révision dans un langage d'objets simple*. Rapport de recherche, INRIA Rhône-Alpes. Grenoble (FR). 1996 à paraître.

[8] N. Foo. *Ontology revision*. Actes 'ICCS-95'. Santa Cruz (CA US). LNCS 954. 1995.

[9] M. Kifer, G. Lausen, J. Wu. *Logical foundations of object-oriented and frame-based languages*. Journal of lhe ACM 42(4):741-843. 1995.

[10] B. Nebel. *Reasoning and revision in hybrid representation systems*. LNCS 422. Springer-Verlag (éds). 1990.

[11] B. Nebel. *Syntax-based approaches to belief revision*. in P. Gärdenfors (éd.). Belief revision. Cambridge tracts in theoretical computer science 29. Cambridge university press. Cambridge (GB). pp52-88. 1992.

[12] J. Penney, J. Stein. *Class modification in the GemStone object-oriented DBMS*. Actes 2nd OOPSLA. Orlando (FL US). pp.111-117. 1987.

[13] Projet Sherpa. *TROPES 1.0 reference manual*. Rapport interne INRIA Rhône-Alpes. Grenoble. 1995.

`ftp://ftp.inrialpes.fr/pub/sherpa/rapports/tropes-manual.ps.gz`

[14] Léa Sombé (éds). *Special issue on Revision and Updating in Knowledge Base.* International journal of intelligent systems 9(1). 1994.

[15] D. Theodoratos. *Updating object-oriented schema structures viewed as logical theories*. Rapport de recherche 12. ERCIM. Rocquencourt (FR). 1995.