dans: actes 10ième RFIA, Rennes (FR), pp615-623, 1996

Révision interactive dans une base de connaissance à objets

Isabelle Crampé, Jérôme Euzenat
INRIA Rhône-Alpes — IMAG-LEIBNIZ
46, av. Félix Viallet, 38031 Grenoble cedex 1
email: {Isabelle.Crampe,Jerome.Euzenat}@imag.fr

Résumé — Lors de la construction d'une base de connaissance, la présence d'une inconsistance peut laisser l'utilisateur démuni car il ne peut embrasser l'étendue de la base. Afin de résoudre ce problème, nous proposons un outil lui indiquant les solutions possibles. Les principes de la révision en logique s'appliquent à cette problématique, mais des résultats plus satisfaisants sont envisageables. En effet, afin d'obtenir des solutions minimisant la perte de connaissance, nous allons nous appuyer sur les structures impliquées dans les représentations par objet (ordre de spécialisation, inclusion des domaines). Par ailleurs, la prise en compte des préférences de l'utilisateur et de son statut permet d'organiser la recherche de solutions.

Mots-clés — révision, représentation de connaissance par objet, interaction système-utilisateur.

1 Introduction

Une base de connaissance permet de stocker de manière structurée des informations disponibles dans un domaine donné (par exemple en biologie moléculaire sur un gène particulier). En tant que reflet d'une réalité, elle doit être consistante, ce qui est une condition nécessaire pour utiliser en toute confiance son contenu.

Cependant la construction d'une base de connaissance, comme celle de beaucoup d'artefacts, n'est ni simple ni immédiate. Elle requiert un processus par essais-et-erreurs qui nécessite des remises en cause d'une partie du contenu de la base. La construction est donc incrémentale: de la connaissance s'ajoute à la base au fur et à mesure de son obtention (des résultats d'observations, d'expériences ou de nouvelles formalisations). Le problème de l'inconsistance de la connaissance apparaît dans divers contextes: lorsque la connaissance du domaine évolue ou lorsque différents acteurs interviennent (successivement ou simultanément) dans la construction de la base. C'est le cas, en particulier, lorsque des intervenants différents cherchent à mettre au point de manière coopérative une base de connaissance consensuelle [Euzenat95].

Ainsi, lorsque l'inconsistance est détectée, l'utilisateur se trouve confronté à deux interrogations: quelles sont les informations contradictoires? Que faudrait-il modifier afin de pouvoir réaliser l'assertion? Des questions qui restent sans réponse pour un utilisateur ne pouvant appréhender la totalité du contenu de la base. Cet article présente les principes d'un outil guidant l'utilisateur dans sa correction d'erreurs, outil lui permettant de trouver les diverses modifications possibles qui garantissent la consistance de la base dont la connaissance a augmenté (Figure 1). Pour aider l'utilisateur dans cette démarche de correction, il faut:

i) détecter l'inconsistance éventuelle de la base résultant de l'ajout de connaissance,
ii) trouver les causes possibles de l'erreur, et
iii) trouver des solutions permettant d'y remédier, c'est-à-dire les modifications qui conduisent à une base consistante avec la dernière assertion.

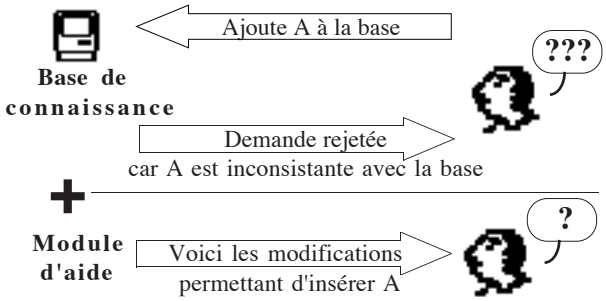

Figure 1. Un module rattaché à la base de connaissance aide l'utilisateur à corriger une inconsistance due à une assertion.

Ce problème a beaucoup de points communs avec la problématique de la révision logique. Cependant, le cadre considéré ici est plus contraint:

- les bases de connaissance sont exprimées dans un formalisme de représentation par objet, qui va diriger la recherche des solutions possibles, et
- le système se borne à aider l'utilisateur et ne cherche pas à résoudre automatiquement les problèmes.

Nous verrons comment ces contraintes interviennent dans le processus de révision. Dans la section 2, nous présentons la problématique de la révision et l'adaptation immédiate de l'approche logique de la révision à une base de connaissance à objets. Cette adaptation n'est cependant pas entièrement satisfaisante. C'est pourquoi, dans la section 3, nous détaillons une approche de la révision s'appuyant sur les caractéristiques propres à une représentation par objet; celle-ci conduit aux différentes possibilités de bases révisées. Dans la section 4, nous montrons alors comment l'interactivité avec l'utilisateur permet d'orienter le choix vers les solutions les plus satisfaisantes.

2 Révision logique et objets

Alchourrón, Gärdenfors et Makinson ont étudié le problème de la révision de bases de connaissance d'un point de vue logique [Alchourrón&85]. Parmi les travaux qui ont suivis, on peut distinguer les approches syntaxiques [Nebel90, Nebel94] et les approches sémantiques qui se basent sur les modèles [Dalal88]. Nous adoptons ici la première de ces approches et présentons rapidement quelques idées de base de ces travaux, avant de considérer leur adaptation aux représentations de connaissance par objet (notées RCO dans la suite). [Sombé94] propose un tour d'horizon des travaux sur la révision.

2.1 Approche logique de la révision

Une base de connaissance est un ensemble de formules du langage. Cet ensemble peut être, suivant les cas, déductivement clos (contient tout ce qui peut être déduit) ou non. Les modifications de la base — afin de lever une inconsistance introduite par une nouvelle information — peuvent être fondées sur la syntaxe de la base et donc sur ses mécanismes déductifs. La révision d'une base doit satisfaire des postulats rationnels et répondre à des exigences de minimalité. Ces critères ne conduisent cependant pas forcément à une seule possibilité de base révisée et il reste alors à déterminer (voire construire) la base révisée.

Postulats de rationalité

Afin de préciser l'opérateur de révision, des postulats ont été énoncés. Alchourrón, Gärdenfors et Makinson [Alchourrón&85] qui ont introduit l'idée de la révision de bases de connaissance, ont défini un ensemble de postulats de rationalité que devrait satisfaire tout opérateur de révision: soient A une théorie (ensemble de propositions déductivement clos, la clôture de la relation de déduction étant notée Cn), x une proposition et + l'opérateur de révision. Les six premiers axiomes (appelés "basic postulates" par Nebel) sont les suivants:

(1) $A+x$ est une théorie
(2) $x \in A+x$
(3) $A+x \subseteq \mathrm{Cn}(A \cup \{x\})$
(4) si $\neg x \notin A$, $A+x=\mathrm{Cn}(A \cup \{x\})$
(5) si $\neg x \notin \mathrm{Cn}(\emptyset)$, $A+x$ est consistant
(6) si $\mathrm{Cn}(x)=\mathrm{Cn}(y)$, alors $A+x=A+y$

La signification de ces axiomes est la suivante. La base révisée doit être fermée par déduction (1). La nouvelle information est prioritaire, elle doit appartenir à la base révisée (2). La base révisée est un sous-ensemble de la base obtenue par ajout et fermeture par déduction (3) et est exactement cet ensemble si l'ajout n'est pas inconsistant, réviser se réduit alors à ajouter et à fermer par déduction (4). La condition suffisante pour obtenir une base révisée consistante est que l'ajout ne soit pas faux en lui-même (5). La révision de la base doit être indépendante de la forme syntaxique de l'ajout (6). À ces axiomes s'en ajoutent deux autres qui concernent plus particulièrement les notions de conjonction et qui ne rentrent pas dans le cadre de cet article. Nous verrons dans quelle mesure l'approche que nous allons détailler par la suite satisfait ces postulats (section 5).

Nous nous plaçons donc dans le cas d'un opérateur de révision qui se décompose en une contraction de la base — c'est-à-dire un retrait d'éléments de la base pour obtenir un sous-ensemble consistant avec la nouvelle connaissance — suivie de l'ajout de la connaissance et de la clôture par déduction. Le problème majeur est de savoir quels sont les éléments à enlever. En effet, il peut exister plusieurs sous-ensembles de la base qui soient consistants avec la nouvelle information et à partir desquels il faut définir la nouvelle base.

Par ailleurs, il peut exister une partie de la connaissance contenue dans la base que l'on ne veut pas voir supprimer; elle reflète des principes du monde modélisé qui ne doivent pas être remis en question (par exemple, que tout être vivant est mortel, qu'un objet ne peut pas être à deux endroits à la fois, ou toute loi physique). Toutes ces informations portent souvent le nom de contraintes d'intégrité. Ken Satoh [Satoh88] différencie les croyances de la connaissance; seules les premières peuvent être remises en cause et doivent donc toujours être consistantes avec la connaissance qui, elle, est immuable.

Minimalité

Une contrainte supplémentaire concerne la notion de minimalité: on cherche à effectuer le moins de modifications possibles sur la base pour obtenir une base consistante avec la nouvelle information. Cette recherche de modifications minimales est importante car on souhaite perdre le moins de connaissance possible, conserver tout ce qu'il n'est pas nécessaire de supprimer.

La minimalité est évaluée différemment suivant les auteurs, suivant leur manière de comptabiliser les modifications effectuées sur la base. La base révisée de façon minimale est celle qui a conservé le plus grand nombre de faits (il n'existe pas une autre sous-base qui la contienne et qui soit consistante avec la nouvelle connaissance) [Alchourrón85, Satoh87] ou celle qui a un nombre minimum d'atomes dont la valeur de vérité est modifiée [Dalal88]. Le sous-problème complexe posé par la révision est de trouver ces sous-bases maximalement consistantes ou les sous-bases minimalement inconsistantes [Papini91].

Lors de la description de l'approche envisagée dans la section 3, nous mettrons en évidence la minimalité des bases révisées proposées, selon la définition donnée.

Choisir parmi les solutions possibles

Dans l'approche des sous-bases maximalement consistantes, il faut déterminer la base révisée à partir des différentes sous-bases. Il est possible d'adopter une approche prudente: la base révisée est l'intersection de toutes les sous-bases maximalement consistantes (seule la connaissance valide dans toutes les solutions est gardée). Au contraire, une approche audacieuse élit l'une des possibilités comme la base révisée ([Barral&92] examine ces deux approches). Entre ces deux extrêmes, prenant en compte ou toutes les solutions ou une seule, il est aussi envisagé de ne faire l'intersection que de certaines solutions, en utilisant une fonction de sélection (comme dans l'opérateur de "partial meet contraction" [Alchourrón&85]).

Dès qu'il s'agit de faire un choix entre solutions, des critères de préférence sur la connaissance sont sous-jacents. Ainsi, [Nebel94] définit un opérateur de révision fournissant une seule base en résultat. Celui-ci se base sur un ordre d'importance relative des connaissances et ne modifie des connaissances que si la modification de connaissances, considérées moins importantes, n'est pas concluante. Le problème de la minimalité sera abordé, pour l'approche envisagée, au paragraphe 3.2 et en section 4.

2.2 Modèle de représentation de connaissance par objet

Parmi les différentes techniques de représentation de connaissance, les RCO structurent la connaissance du monde autour de deux types d'objets — les classes et les instances — et de relations entre ces objets. Nous présentons ici le sous-ensemble du modèle de RCO TROPES [Sherpa95], nécessaire aux exemples traités dans la suite.

Syntaxe

Le but de la RCO est de représenter un ensemble d'individus d'un domaine particulier à l'aide d'objets (notés $o, o'\ldots$). Ces objets sont d'une part attachés à des classes (notées $c, c'\ldots$); ils sont appelés instances de leurs classes. D'autre part, ils sont décrits à l'aide d'attributs (notés $a_1,\ldots,a_n$) qui permettent également de les relier entre eux. On notera donc $o.a=v$ pour exprimer que l'attribut a de

l'objet o a pour valeur v (v pouvant être un autre objet) et $o \in c$ pour exprimer que l'objet o est rattaché à la classe c.

Les classes sont décrites à l'aide de domaines (d) que l'on restreindra ici à un ensemble de valeurs ($\{v_1,\ldots,v_n\}$), pouvant être des objets. Ainsi, $c.a=\{v_1,\ldots,v_n\}$ signifie que le domaine de l'attribut a pour la classe c est constitué des valeurs $v_1, \ldots, v_n$. De plus, ces classes sont organisées en hiérarchie par une relation de spécialisation notée $c \leq c'$ si c spécialise c' ($c<c'$ signifiant que $c \leq c'$ et $c \neq c'$). Une base de connaissance est un ensemble de telles assertions ($o \in c$, $c \leq c'$, $o.a=v$, $c.a=\{v_1,\ldots,v_n\}$).

Pour compléter la notation utilisée ci-dessus, on définit le domaine effectif de l'attribut (noté $[c.a]$) par l'intersection de tous les domaines définis dans la classe et dans ses surclasses. On a donc:

$[c.a]=c.a \cap$ ou encore $[c.a]=c.a \cap [c'.a]$ avec c' surclasse directe de c (on se restreint dans notre étude à la mono-spécialisation) et $[c.a]=c.a$ pour la classe racine de la hiérarchie. Le calcul de domaines effectifs est réalisé, en terme opérationnel, par le mécanisme d'héritage

Sémantique du modèle

Considérons un domaine à modéliser D; on peut établir l'ensemble des valeurs possibles d'attributs en y ajoutant les valeurs primitives (entiers, réels, chaînes…), ce qui correspond à l'ensemble D'. Les objets s'interprètent comme des éléments du domaine, les classes comme des parties de ce domaine et les attributs comme des fonctions partielles d'un élément du domaine vers une valeur.

La fonction d'interprétation $|.|$ est alors telle que:

pour tout objet o, $|o| \in D \qquad (|o| = |o'| \Rightarrow o = o')$

pour toute classe c, $|c| \subseteq D$

pour chaque attribut a, $|a| \in D \rightarrow D'$

pour tout domaine d d'attribut, $|d| \subseteq D'$

Les sous-ensembles correspondant aux classes subissent les contraintes suivantes:

pour toute sous-classe c' de c, $|c'| \subseteq |c|$

pour toute instance o de c, $|o| \in |c|$

pour tout domaine d d'un attribut a de c, $|a| \in |c| \rightarrow |d|$

Ainsi, la contrainte complète portant sur une classe est la suivante:

$$|c| \subseteq \bigcap_{c<c'} |c'| \cap \left[\bigcap_{\substack{a_i \in attributs(c) \\ d_{ij} \in domaines(a_i)}} \left\{ o \in D; |a_i|(o) \in d_{ij} \right\} \right]$$

Le premier terme correspond au domaine effectif de la surclasse et le second à la prise en compte de toutes les restrictions sur les attributs de la classe. Le caractère particulier de cette présentation par rapport à d'autres présentations plus classiques (telles que celles des logiques terminologiques) provient de la présence de relations d'inclusion au lieu d'égalité entre les termes. Ceci correspond à l'interprétation descriptive (et non définitionnelle) des classes qui ne sont décrites que par des conditions nécessaires à leur appartenance mais pas forcément suffisantes. Cette différence n'a cependant pas d'incidence ici.

Des règles et des contraintes

Le modèle de représentation admet des règles de déduction:

(1) la relation de spécialisation est transitive,
(2) une classe hérite des contraintes posées sur ses surclasses,

et des contraintes de consistance:

(3) une instance rattachée à une classe satisfait toutes les contraintes posées sur cette classe (domaine des attributs, contraintes numériques…),
(4) le domaine d'un attribut d'une classe n'est pas vide.

Ces règles et contraintes sont valides dans l'interprétation de la base.

Une base de connaissance est un ensemble d'assertions syntaxiquement correctes (par rapport à la syntaxe donnée ci-dessus). Elle est consistante si les contraintes sont satisfaites, c'est-à-dire si l'extension d'aucune de ses classes n'est vide (ce qui est le cas lorsque le domaine de l'un de ses attributs est réduit à Ø) et si toutes les instances vérifient les contraintes associées à leur classe. Nous supposons que la détection d'inconsistances est réalisée de manière efficace.

2.3 Adaptation de l'approche logique aux RCO

Traduction du formalisme de représentation

Une base de connaissance à objets peut se traduire dans un formalisme logique et il est donc possible d'adapter naïvement les opérateurs de révision pour réviser une base TROPES. Toute la connaissance de la base est exprimée sous forme de formules atomiques, les contraintes étant des propositions particulières qui ne pourront être invalidées. Les règles et contraintes énoncées ci-dessus s'écrivent en logique du premier ordre:

(1) $\forall c,c',c'', c \leq c' \wedge c' \leq c'' \Rightarrow c \leq c''$
(2) $\forall c,c', \forall a, c \leq c' \Rightarrow [c.a] \subseteq c.a \cap [c'.a]$
(3) $\forall o, \forall c, \forall a, o \in c \Rightarrow o.a \in [c.a]$
(4) $\forall c, \forall a, \neg([c.a] = \varnothing)$

Considérons une base très simple composée d'une hiérarchie de classes: les classes c, c' et c'' possèdent un seul attribut a de type entier dont le domaine est précisé (Figure 2). La nouvelle information rend manifestement la base inconsistante.

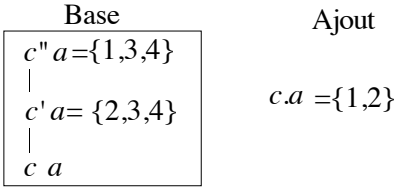

Figure 2. Exemple d'ajout inconsistant dans une base

Cette base peut se transcrire de la manière suivante: $\{c' \leq c''; c \leq c'; c''.a=\{1,3,4\}; c'.a=\{2,3,4\}; c.a=\{1,2\}\}$, sa fermeture déductive la complète par $\{c \leq c''; [c'.a]=\{3,4]; [c.a]=\varnothing\}$.

La dernière proposition est contradictoire avec l'une des contraintes, ce qui révèle l'inconsistance.

Méthode de révision

Pour réviser cette base, il faut lui appliquer tout d'abord l'opérateur de contraction, ce qui consiste à tester la consistance de tous les sous-ensembles possibles (résultant du retrait de chaque élément de la base) de la base augmentée de la nouvelle connaissance. Pour les sous-ensembles inconsistants, il est possible soit de leur enlever d'autres éléments, soit de les abandonner car étant plus éloignés de la base initiale, ils ne seront pas à considérer. Sur l'exemple précédent, l'opérateur revient donc à tester le retrait de chacun des 5 faits (Figure 3). Le retrait de $c \leq c'$ (B2), de $c''.a=\{1,3,4\}$ (B3) ou de $c'.a=\{2,3,4\}$ (B4) conduit à une base consistante, de même que le retrait de $c.a=\{1,2\}$ (B5) qui, lui, ramène à la base initiale. Pour le sous-ensemble B1, il

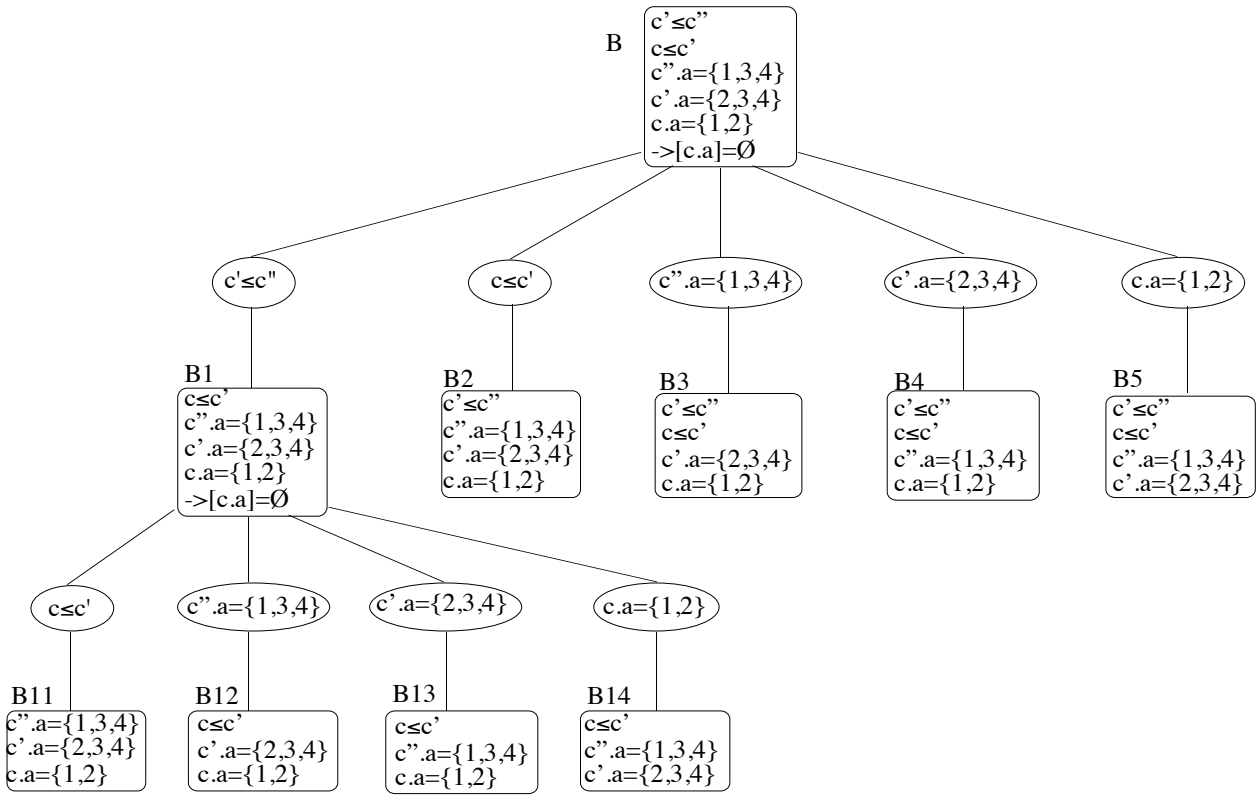

Figure 3. Les solutions sont recherchées en retirant successivement de la base initiale chaque élément et en testant la consistance; si celle-ci n'est pas atteinte, de nouveaux éléments sont retirés; les feuilles sont des bases consistantes, sous-bases de la base initiale.

est nécessaire de lui enlever d'autres éléments pour qu'il soit consistant, par exemple *c".a*={1,3,4} (B12). Sur cet exemple réduit, la révision définie ci-dessus peut s'appliquer: les bases révisées possibles sont trouvées au prix du parcours d'un arbre représentant les retraits successifs des éléments, les feuilles étant les bases consistantes. Parmi toutes les bases trouvées, certaines sont incluses dans d'autres (par exemple B12⊂B4); elles ne représentent donc pas une modification minimale et il n'est pas souhaitable de les garder. Après l'étape de recherche des sous-bases consistantes, il faut donc extraire de l'ensemble trouvé (8 bases dans l'exemple) celles qui sont minimales (il y en a quatre: B2, B3, B4 et B5).

Alternatives possibles

Ci-dessus, la recherche des sous-ensembles consistants a été faite sur la base non close. Or, si on ne dispose que de la base fermée par les règles de déduction, l'exploration est menée de manière identique, mais est beaucoup plus fastidieuse (huit faits à tester au lieu de cinq pour l'exemple précédent) sans apporter de solutions supplémentaires.

De plus, l'arbre obtenu ci-dessus conduit à l'exploration de solutions qui ne sont pas très intuitives, celles qui retirent des propositions apparemment sans rapport avec l'inconsistance soulevée (une base peut comporter des connaissances complètement indépendantes). Or, en procédant par abduction à partir de l'inconsistance détectée, il est possible de trouver les assertions ayant permis de déduire l'inconsistance à l'aide de règles d'inférence (le modus ponens ou la résolution par exemple dans les applications logiques); ainsi, la recherche de solutions se focalise sur les assertions pertinentes, vu l'inconsistance détectée.

Cependant, traduire le formalisme de représentation par objet dans le formalisme logique n'est sans doute pas la meilleure solution pour réviser une base de connaissance à objets. Les spécificités des représentations par objet vis-à-vis de la révision sont donc abordées dans la section suivante.

3 Révision caractéristique aux RCO

Les mécanismes d'inférence des RCO sont diversifiés (bien qu'ici seuls deux d'entre eux soient présentés) et les distinguer permet de contrôler la recherche de sous-bases maximalement consistantes (paragraphe 3.1). De plus, le retrait de connaissance n'est pas forcément le meilleur moyen d'obtenir une base révisée de manière minimale: l'ajout de connaissance sera envisagé (paragraphe 3.2). Enfin, nous montrerons comment l'arbre de solutions peut être généré (paragraphe 3.3).

3.1 Arbre des solutions minimales

On considère deux mécanismes d'inférence: la transitivité de la relation de spécialisation et l'héritage des contraintes posées sur les surclasses. Lorsqu'une inconsistance est détectée, on se base sur ces mécanismes pour localiser les causes potentielles de l'erreur, ce qui permet de n'enlever de la base que les assertions qui conduisent effectivement à l'inconsistance.

Reprenons le cas très simple d'une hiérarchie de classes dans laquelle une inconsistance est détectée: *c* spécialise *c'* mais les domaines de définition des attributs *c.a* et *c'.a* ont une intersection vide (ce qui entraîne que [*c.a*]=Ø).

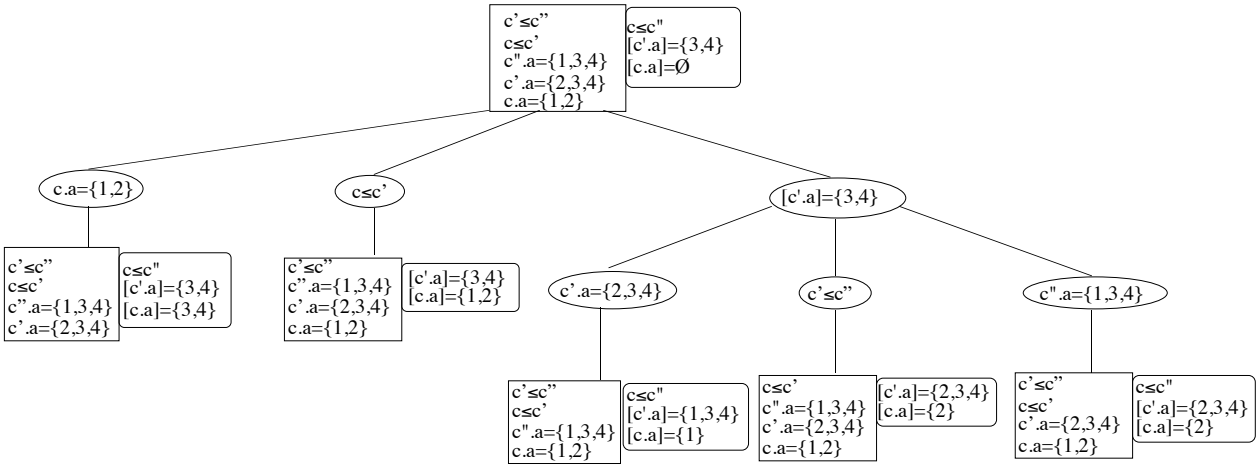

Figure 4. Un arbre donne les différentes solutions pour l'exemple de la hiérarchie de classes; les nœuds internes indiquent les assertions à supprimer et les feuilles représentent les bases solutions ainsi que leur fermeture transitive.

Rappelons que le mécanisme d'héritage de contraintes s'exprime par le formalisme suivant (cf. paragraphe 2.3): soient c et c' deux classes,

$c \leq c' \Rightarrow [c.a] \subseteq c.a \cap [c'.a]$.

L'erreur ($[c.a]=\emptyset$) peut donc provenir:

- de la relation de spécialisation entre c et c',
- de la définition de $c.a$,
- ou encore de la restriction de domaine hérité des surclasses.

Cette dernière source d'erreur peut elle-même être imputée à une relation de spécialisation, au domaine de l'attribut de la classe ou à un domaine hérité, qui à son tour peut provenir d'une mauvaise relation de spécialisation, etc. Ces différentes possibilités conduisent à autant de solutions envisageables qui sont représentables par un arbre.

Les nœuds de l'arbre sont de deux types, soit une base (et sa fermeture déductive), soit une assertion. Les fils d'un nœud base sont les différentes assertions dont la suppression lève l'inconsistance actuelle. Si l'assertion est élément de la fermeture déductive, elle est supprimée en supprimant l'une des assertions de la base qui permet de la déduire; les différentes suppressions possibles sont représentées par ses fils. Les modifications sont effectuées sur la base (nœud père) qu'il faut fermer par déduction pour obtenir la base globale et vérifier la consistance. Une feuille contient la base consistante résultant des différents retraits le long de la branche la reliant à la racine (Figure 4). À noter que le dernier retrait est celui d'un axiome dont la suppression n'a pas engendré d'autres inconsistances.

Remarquons que l'une des solutions consiste à enlever la dernière proposition, celle qui a soulevé l'inconsistance, ce qui ramène à la base initiale et n'est donc pas considéré classiquement comme de la révision.

Pour toute base inconsistante suite à l'ajout d'une assertion, un tel arbre peut être construit en se basant sur les règles d'inférence. Nous étudierons plus précisément la génération des arbres dans le paragraphe 3.3. Ils fournissent l'ensemble des solutions minimales, c'est-à-dire des bases qui ne diffèrent pas plus que nécessaire de la base initiale. Parmi toutes les informations contenues dans la base — et dont il faut a priori se défier quand une inconsistance est détectée —, seules celles qui sont responsables de l'inconsistance (au sens où sans chacune d'entre elles, il n'y aurait pas d'inconsistance) sont potentiellement supprimées. Toutes les informations indépendantes de l'inconsistance sont donc conservées.

Remarques:

- cet arbre est fini car la base est finie et que l'on ne fait que des suppressions d'assertions. Cela garantit la terminaison de l'exploration de toutes les solutions.
- Dans des exemples plus complexes de bases, des inconsistances peuvent survenir suite à un retrait permettant de lever une inconsistance précédente. Dans ce cas, les nouvelles inconsistances sont traitées comme la précédente en recherchant les différentes causes possibles.

Dans l'exemple précédent, toutes les suppressions envisageables pour retrouver la consistance ont été trouvées. Or, intuitivement d'autres modifications seraient possibles, comme d'augmenter le domaine de *c'.a* et *c".a* en {1,2,3,4}. Mais la simple adaptation des principes de la révision logique ne permettent pas de prendre en compte ce type de modification de la base.

3.2 Révision par ajout

L'approche logique consiste à supprimer des informations contradictoires avec la connaissance à insérer. Le retrait d'assertion dans la base permet effectivement de lever une inconsistance. Cependant, intuitivement, un retrait peut être nuancé par un ajout d'assertion afin de minimiser (sous un angle sémantique cette fois-ci) les modifications apportées à la base. Ainsi, dans l'exemple traité ci-dessus, l'une des solutions consiste à supprimer *c".a*={1,3,4}, le domaine de cet attribut devenant alors l'ensemble des entiers; or, modifier le domaine en {1,2,3,4} permet d'obtenir également une base consistante et qui est moins modifiée, dans un sens qui reste à définir.

Ce type de solution doit donc être envisagé et proposé car intuitivement il peut correspondre à une attente. Cependant, il faut identifier précisément les nuances à apporter aux suppressions car les ajouts ne peuvent être ni arbitraires, ni choisis systématiquement. Il s'agit donc de trouver les ajouts pertinents et le moyen d'étendre la notion de minimalité.

Les ajouts sont toujours associés à une suppression dans la base, afin de privilégier la modification à la suppression de connaissance. Les suppressions envisagées sont de trois types; à chacune d'elle correspond un type d'ajout (dont les paramètres sont instanciés en fonction de ceux de la suppression):

(1) supprimer la définition d'un domaine d'un attribut: définir le domaine de cet attribut afin qu'il soit consistant avec les contraintes;
(2) couper un lien de spécialisation: rattacher une classe sous une autre qu'elle spécialise effectivement;
(3) enlever une instance d'une classe: attacher l'instance à une classe qu'elle spécialise effectivement.

Il faut s'attacher à proposer des ajouts pertinents, ce qui oblige à tenir compte de la connaissance de la base et de l'inconsistance soulevée; ils doivent minimiser les erreurs qu'ils soulèvent à leur tour. L'idéal est qu'ils soient consistants avec la base mais ce n'est pas toujours envisageable: ils nécessitent parfois d'autres modifications (de même que les suppressions). Les ajouts doivent donc être les plus consistants possibles avec la base et s'attacher à minimiser la modification de la base initiale. Pour les ajouts de type (2) par exemple, il s'agit de minimiser les modifications de la clôture déductive des bases initiale et finale (c'est-à-dire le nombre de relations de spécialisation entre classes qui ont été modifiées).

Lors de révision par suppression uniquement, le critère de minimalité est basé sur l'inclusion des sous-bases. Les bases considérées ici sont fermées par déduction. Vu que les bases sont des sous-ensembles de la base initiale (complétés par la dernière assertion), il s'agit de minimiser la différence entre les bases initiale et finale. Soit *R(B,a)* l'ensemble des bases révisées possibles de la base *B* après l'ajout de *a* satisfaisant tous les postulats de rationalité de l'opérateur de révision. $Br \in R(B,a)$ est une base minimale si et seulement si quelle que soit $Br' \in R(B,a)$ telle que $Br \subseteq Br'$ alors $Br=Br'$ (ou encore: si $(Br'-B) \subseteq (Br-B)$ alors $Br=Br'$, «–» étant la différence ensembliste classique).

Dans le cas des ajouts, procédons de même en utilisant la différence symétrique notée / qui se définit par: $A/B = A \cup B - A \cap B$. Br/B contient l'ensemble des assertions qui ont été supprimées ou rajoutées dans *Br* par rapport à *B*. On pose alors:

Br est une base révisée minimalement si et seulement si quelle que soit $Br' \in R(B,a)$ telle que $(Br'/B) \subseteq (Br/B)$ alors $Br=Br'$.

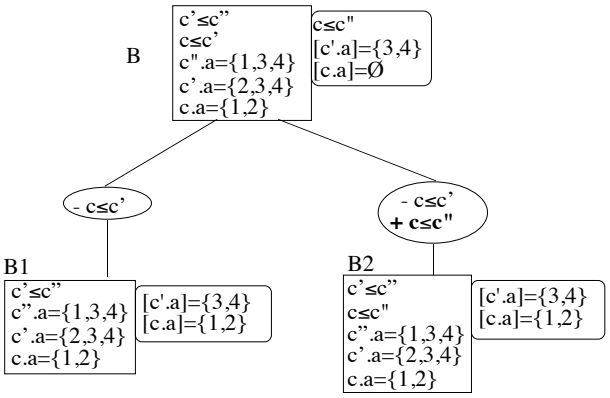

Figure 5. La base B2 obtenue grâce à un ajout est plus proche de la base initiale B que la base B1 qui a été obtenue par retrait uniquement.

Les bases non minimales sont alors celles qui font plus de suppressions ou d'ajouts que nécessaire (Figure 5), aucune priorité n'étant donnée aux unes par rapport aux autres. De plus, cette formulation correspond à la minimalité initiale lorsque l'on se restreint aux seules suppressions.

Cependant, ce critère de minimalité traite par exemple uniformément l'ajout de $c.a=\{1,2,3,4\}$ ou $c.a=N$, ce qui ne semble pas très intuitif. Pour les ajouts de type (1) et (3), il faut prendre en compte les objets mis en jeu dans l'assertion. Nous avons donc défini une relation d'ordre sur ces types d'assertion permettant de trouver celle qui réalise une modification minimale: celle qui minimise la variation de domaine d'un attribut (1) ou le déplacement d'une instance dans la hiérarchie de classes (3). Parmi l'ensemble d'ajouts associés à une suppression, ceux qui sont minimaux seront les seuls à être envisagés.

Les solutions, envisagées dans la suite afin de lever une inconsistance, se composent de suppressions et d'ajouts d'assertions qui fournissent une base révisée de façon minimale.

3.3 Génération des arbres

Les arbres de révision du type de celui présenté au paragraphe 3.1 ne peuvent être prédéfinis. Il faut donc un algorithme qui permette de construire l'arbre, complété par les ajouts correspondants aux retraits, fournissant toutes les solutions minimales d'une inconsistance donnée. À cette fin, l'algorithme est caractérisé par un ensemble de règles d'inférence dont l'application permet théoriquement d'obtenir un tel arbre. On présente ensuite la caractérisation des branches constituant les «bonnes» solutions.

Principe

Les inconsistances sont de deux types: soit le domaine d'un attribut d'une classe est vide, soit une instance ne satisfait pas les contraintes associées à sa classe. Pour chacun, un certain nombre de types de solutions existent et ce, quelle que soit la base concernée. Nous avions déjà remarqué que certaines solutions pouvaient entraîner d'autres inconsistances à lever à leur tour; nous proposons donc un processus de construction de l'arbre qui tienne compte de l'imbrication entre les solutions proposées et les inconsistances soulevées (Figure 6).

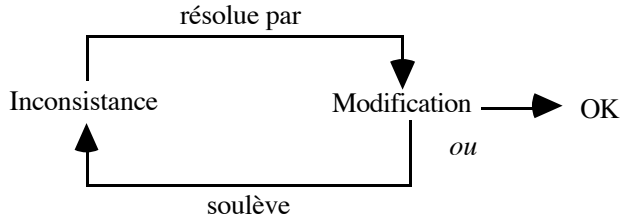

Figure 6. Pour lever une inconsistance, un certain nombre de modifications sont possibles; mais celles-ci peuvent soulever d'autres problèmes à résoudre récursivement.

Nous avons examiné toutes les modifications envisageables afin de lever les inconsistances dues à une classe vide ou une instance mal classifiée. Lorsqu'une classe est vide, on peut (comme cela a déjà été vu au paragraphe 3.1):

- modifier la définition de l'attribut dont le domaine est vide (en la supprimant ou en l'élargissant),
- couper le lien de spécialisation avec la surclasse (en rattachant la classe éventuellement sous l'une de ses surclasses)
- modifier le domaine hérité de l'attribut.

Si une instance ne vérifie pas les contraintes associées à sa classe, on peut:

- détacher l'instance de la classe (et éventuellement la reclassifier),
- supprimer l'instance,
- modifier les valeurs d'attributs inadéquats de l'instance,
- agrandir le domaine des attributs de la classe.

On peut remarquer que la plupart des solutions proposées combinent un retrait et un ajout afin de ne proposer que des solutions minimales.

Ces modifications proposées sont exprimées par des règles de la forme:

$$c \leq c', c'.a := c'.a \cup c.a \Rightarrow \neg([c.a] = \varnothing)$$

(*M*=>*I* signifiant que la modification *M* de la base est une solution permettant de lever l'inconsistance *I*.)

Ces règles sont au nombre de 11; elles couvrent des aspects de TROPES qui ne sont pas abordés dans cet article, mais pas encore toutes ses possibilités.

Parallèlement, nous avons précisé toutes les inconsistances qui pouvaient apparaître suite à une modification de la base, ce qui permet de détecter et localiser les inconsistances au cours du processus de révision. Ces inconsistances potentielles sont également exprimées par des règles de la forme suivante:

$$c.a := d \rightarrow [c.a] = \varnothing$$

($M \rightarrow I$ signifiant que la modification *M* peut soulever l'inconsistance *I*.)

Ces informations, exprimées sous forme de règles, sont instanciées pour une inconsistance particulière; elles fournissent alors l'ensemble des modifications envisageables. Si certaines des inconsistances potentielles liées aux modifications apparaissent, il faut les lever à l'aide de nouvelles modifications. Un moteur d'inférence sur ces règles permet donc de construire l'arbre conduisant aux solutions pour une inconsistance donnée.

Solutions

En se limitant au retrait de connaissance de la base, les branches de l'arbre d'exploration des solutions sont toutes finies. Or, en considérant également des ajouts dans la base, des cycles sont possibles: on élimine de la connaissance, ce qui entraîne une inconsistance qui est levée en ajoutant de la connaissance précédemment ôtée. Nous caractérisons une bonne solution comme étant une branche de l'arbre ainsi créé :

- ne contenant pas deux assertions qui s'annulent et
- finie (une branche infinie pouvant caractériser un comportement asymptotique dans le cas de domaines denses).

Cependant, de manière pratique, les cycles sont localisés: ils n'apparaissent que lors de résolutions particulières identifiées. Par exemple, lorsqu'une classe est vide, l'agrandissement du domaine d'un attribut de la surclasse peut créer une inconsistance avec sa propre surclasse, qui peut être résolue en rediminuant le domaine de l'attribut (ce qui ramène le problème à son état initial). Pour les deux types particuliers d'inconsistances dont la résolution peut boucler, des solutions plus globales permettent de lever l'inconsistance pour l'ensemble des objets potentiellement concernés. Elles fournissent les solutions minimales (et surtout elles permettent d'éviter les cycles).

Une fois toutes les révisions minimales connues, il reste à déterminer celle qui sera appliquée.

4 Préférence et interactivité

Toutes les bases consistantes, pouvant être le résultat de la révision de la base initiale, ne semblent pas équivalentes; selon quel critère peuvent-elles être classées? Comment trouver celle qui satisfait le mieux le critère de préférence choisi? Dans les paragraphes ci-dessous sont détaillées différentes notions de préférence et la manière de les mettre en œuvre.

4.1 Ordre sur les connaissances

Il est possible de demander à l'utilisateur de classer toutes les informations contenues dans la base selon l'importance qu'il leur accorde, pour ensuite lui proposer la base dont les modifications par rapport à la base initiale touchent les connaissances les moins importantes. Cette idée est liée à l'ordre selon la pertinence ("relevance ordering")[Nebel94]. Cela est cependant fastidieux et demanderait une connaissance de la base telle que le mécanisme de révision perdrait de son utilité.

4.2 Interaction avec l'utilisateur lors du parcours de l'arbre

Pour lever une inconsistance particulière, on dispose d'un arbre dont le parcours peut conduire, au moins, à toutes les solutions minimales. Laissons à l'utilisateur le soin de définir l'ordre d'exploration. Il parcourt l'arbre en choisissant à chaque nœud la branche traitant la cause d'erreur à laquelle il veut remédier; lorsque l'on atteint une feuille, l'ensemble des suppressions effectuées sur la base initiale au cours de ce parcours garantit qu'elle est consistante avec la dernière assertion. Ainsi, grâce à l'intervention de l'utilisateur à chacun des choix, la solution obtenue sera celle qu'il préfère, sachant que les retours-arrière sont possibles: l'une des voies choisie peut nécessiter plus de modifications que prévu, plus que ce qui peut être accepté, et donc être abandonnée.

Cependant, l'utilisateur n'est peut-être pas capable d'évaluer toutes les solutions possibles qui sont à sa disposition. Dans les cas pour lesquels on peut définir une préférence, les solutions peuvent lui être fournies de manière ordonnée .

4.3 Privilégier certains types de connaissances

Dans le cadre des RCO, il existe deux grandes classes de connaissance: la connaissance structurelle liée à la définition de classes et aux liens établis entre elles et la connaissance individuelle liée aux instances. Selon le contexte d'utilisation de la base, l'un ou l'autre de ces types de connaissance est le plus fiable (et donc à modifier le moins possible). Dans une démarche de construction de base à partir d'exemples (résultats d'expériences ou d'observations), les instances représentant ces exemples doivent être préservées au détriment de la connaissance structurelle. Celle-ci doit permettre de représenter ce qui provient du monde à modéliser et donc être modifiée si des incompatibilités apparaissent. En présence d'une inconsistance, la connaissance concernant les classes devra être remise en cause en priorité.

Au contraire, lors de l'utilisation d'une base construite et éprouvée, les instances seront plus facilement mises en doute que la structure de la base. De même, les utilisateurs considérés comme moins compétents que la personne ayant construit la base, ne sont pas autorisés à modifier la structure de la base. Les solutions proposées ne doivent alors concerner que des instances de la base (la valeur de leurs attributs ou leur classe d'appartenance).

Si un tel ordre entre la connaissance structurelle et individuelle a été établi, l'exploration de l'arbre doit s'orienter en tenant compte de ce critère. Il est facile de

classer les modifications à faire selon qu'elles touchent directement l'un ou l'autre de ces types de connaissance. En revanche, comme les modifications peuvent également en nécessiter d'autres (suite à de nouvelles inconsistances), il faut pouvoir tenir compte de ces répercussions plus lointaines afin d'évaluer les solutions préférées.

De plus, TROPES admet des points de vue dans sa représentation, ce qui permet de séparer la connaissance d'un domaine selon différents aspects, tout en les reliant par des passerelles. Il est envisageable pour un utilisateur de ne vouloir (ou n'être autorisé à) modifier que les informations contenues dans un point de vue donné. Lors de la révision de la base, les solutions proposées ne devront donc concerner que ces informations. Une telle révision selon les points de vue est proche des travaux sur la fusion d'informations provenant de différentes sources, la fiabilité de ces sources dépendant des thèmes sur lesquelles elles sont compétentes [Cholvy94].

5 Liens avec d'autres travaux

Nous confrontons ci-dessous la révision définie aux postulats de Alchourrón, Gärdenfors et Makinson, énoncés au paragraphe 2.1. Une fois choisies les modifications à apporter à la base, celle-ci est fermée par les deux relations de déduction, l'héritage des surclasses et la transitivité de la spécialisation (1). La nouvelle connaissance est insérée dans la base révisée (2) sauf si l'utilisateur indique explicitement qu'il rétracte son assertion (il peut très bien estimer que la dernière connaissance ne vaut pas les changements nécessaires). Nous avons envisagé la possibilité de rajouter de la connaissance; celle-ci vise à minimiser les modifications de la base mais l'inclusion ensembliste dans la base initiale complétée de la nouvelle connaissance n'est pas forcément vérifiée (3 n'est pas satisfait). Si aucune erreur n'est détectée lors de l'insertion de connaissance, elle est simplement rajoutée dans la base sans faire appel au processus de révision (4). Si la connaissance n'est pas inconsistante en elle-même (auquel cas, pratiquement, elle n'est pas insérée dans la base), le travail présenté dans le cadre de cet article garantit que la base révisée est consistante (5). La fermeture déductive d'un ensemble d'assertions qui sont autorisées dans une RCO est l'identité ((6) est trivial).

Il existe très peu de travaux liant la révision et la représentation par objet. Alexander Borgida [Borgida85] s'est cependant intéressé à la détection d'erreurs dans ce contexte, mais il se restreint à noter l'inconsistance, sans la corriger. Norman Foo [Foo95] s'est récemment intéressé à la révision de hiérarchies de classes descriptives. Il a ébauché des règles de révision de taxonomies (scission de classes et raffinement par sous-classes) qui se retrouvent également dans les règles que nous avons établies. Une partie de ce travail entretient également des rapports importants avec la réorganisation de bases d'objets [Penney&87, Skarra&87]: les protocoles établis permettent de modifier les bases d'objets de manière consistante. Les règles présentées ici, permettant les modifications, sont de même nature, à ceci près que les protocoles établis par ces auteurs sont totalement déterministes et servent à implémenter des politiques universelles sur lesquelles les utilisateurs (qui ne sont plus alors considérés comme des concepteurs) n'ont aucun contrôle.

6 Conclusion

Lorsque de la connaissance nouvelle est inconsistante avec une base de connaissance existante, il faut remettre en cause une partie de son contenu. Nous nous sommes attachés à préciser les caractéristiques d'un outil permettant la révision d'une base de connaissance à objets. Notre travail s'oriente selon deux aspects principaux:

- la représentation de connaissance par objet permet de ne pas se limiter aux suppressions pour lever une inconsistance: des ajouts nuancent les retraits, ce qui définit une nouvelle notion de minimalité plus intuitive pour les bases de connaissance à objets que l'inclusion ensembliste;
- l'interactivité lors de la recherche de solution permet à l'utilisateur d'obtenir la solution qu'il préfère; les solutions qui satisfont ses exigences de préférence sont privilégiées automatiquement .

Nous avons proposé un ensemble de règles permettant, grâce à un moteur d'inférence, de construire un arbre des solutions minimales. Il s'agit à présent d'établir la complétude de ce système de règles et de formaliser l'influence de la préférence sur le parcours de l'arbre proposé à l'utilisateur. L'approche de la révision que nous avons abordée est essentiellement basée sur la syntaxe. Cependant, nous la jugeons en regard de la sémantique et l'un de nos travaux futurs consistera à confronter les résultats à la sémantique de la base de connaissance. D'autre part, ce système est en cours d'implémentation sur le modèle de RCO TROPES afin de valider la faisabilité de l'approche et d'évaluer l'intérêt des préférences proposées.

Bibliographie

[Alchourrón&85] Carlos E. Alchourrón, Peter Gärdenfors & David Makinson. *On the logic of theory change: partial meet contraction and revision functions*. Journal of symbolic logic. Vol.50. No.2. pp.510-530. Juin 1985.

[Barral&92] Chitta Barral, Sarit Krauss, Jack Minker & Venkataram Subram Subrahmanian. *Combining knowledge bases consisting of first-order theories*. Computational Intelligence. Vol.8. No.1. pp.45-71. 1992.

[Borgida85] Alexander Borgida. *Language features for flexible handling of exceptions in informations systems*. ACM transactions on database system. Vol.10. No.4. Décembre 1985.

[Cholvy94] Laurence Cholvy. *Fusion de sources d'informations ordonnées en fonction des thèmes*. Actes 9ième RFIA. Paris. pp.487-494. Janvier 1994.

[Dalal88] Mukesh Dalal. *Investigations into a theory of knowledge base revision: preliminary report*. Actes 7th AAAI. Philadelphia (PA US). pp.475-479. 1988.

[Euzenat95] Jérôme Euzenat. *Building consensual knowledge bases: context and architecture*. Towards very large knowledge bases. Actes 'KB&KS-95'. Enschede (NL). pp.143-155. Avril 1995.

[Foo95] Norman Foo. *Ontology revision*. Actes 'ICCS-95'. Santa Cruz (CA US). LNCS 954. Août 1995.

[Nebel90] Bernhard Nebel. *Reasoning and revision in hybrid representation systems*. LNCS 422. 1990.

[Nebel94] Bernhard Nebel. *Base revision operations and schemes: semantics, representation and complexity*. Actes 11th ECAI. Amsterdam (NL). pp.341-345. Août 1994.

[Papini91] Odile Papini. *Révision des connaissances en calcul propositionnel*. Actes 8ième RFIA. Villeurbanne. pp.1293-1298. 1991.

[Penney&87] D. Jason Penney & Jacob Stein. *Class modification in the GemStone object-oriented DBMS*. Actes 2nd OOPSLA, Orlando (FL US). pp.111-117. 1987.

[Satoh87] Ken Satoh. *A minimal change of belief - a criterion of belief revision*. ICOT Technical report TR-297. Tokyo (JP). Septembre 1987.

[Satoh88] Ken Satoh. *Nonmonotonic reasoning by minimal belief revision*. Actes 'International Conference on fifth generation computer systems'. pp.455-462. Tokyo (JP). 1988.

[Sherpa95] Projet Sherpa. *TROPES 1.0 reference manual*. Rapport interne INRIA Rhône-Alpes. Grenoble. 1995.
`ftp://ftp.imag.fr/imag/SHERPA/rapports/tropes-manual.ps.gz`

[Skarra&87] Andrea Skarra & Stanley Zdonik. *Type evolution in an object-oriented database*. Bruce Shriver & Peter Wegner (éds.). Research directions in object-oriented programming. The MIT press, Cambridge (MA US). pp.393-415. 1987.

[Sombé94] Léa Sombé (éds). *Special issue on revision and updating in Knowledge Base*. International journal of intelligent systems.Vol.9. No.1. Janvier 1994.