Laboratoire d'Informatique, de
Robotique et de
Microélectronique de
Montpellier

Unité Mixte CNRS - Université Montpellier II
C 05526

Rapport de Recherche

Des langages d'objets aux logiques de descriptions : les systèmes classificatoires

Roland Ducournau

3 décembre 1996 R.R.LIRMM 96-???

161, Rue Ada - 34392 Montpellier Cedex 5 - France
Tél: (33) 67 41 85 85 - Fax: (33) 67 41 85 00

Résumé

Les différentes approches de représentation des connaissances fondées plus ou moins sur la notion d'objet et de relation — qu'il s'agisse des logiques de descripion (ou logiques terminologiques, LT), des représentations des connaissances à objets (RCO), voire des langages de programmation par objets (LPO) ou des graphes conceptuels (GC) — permettent de faire émerger la notion de *système classificatoire* (SC).

Un système classificatoire est constitué par une *hiérarchie* de *classes*, définies en *intension* par un ensemble de *propriétés*, et munies d'une interprétation *extensionnelle.* Un SC peut ainsi être considéré comme un système de déduction, donc une logique, avec une sémantique directe en théorie des modèles, sur le modèle de celle(s) établie(s) depuis quelques années dans l'approche des LT.

Ce travail est à voir comme une tentative de définition d'un cadre formel très général, valable aussi bien pour le noyau de tous les systèmes à objets, que pour les logiques terminologiques, les ψ-termes, les hiérarchies de concepts, les graphes conceptuels, etc. D'une certaine manière, c'est la suite de la discussion entamée dans le groupe "classification et objets" du PRC-IA [Carré *et al.*, 1995], avec deux objectifs : donner un contenu à la sémantique du tronc commun à toutes les approches objet et, sur la base de cette sémantique, approfondir la comparaison entre systèmes à objets et logiques terminologiques [1].

Mots-clefs : classification, logique terminologique, représentation des connaissances par objets, intension, extension

1. Ce rapport est une version sensiblement corrigée et allongée des RR LIRMM 95-038 *Les systèmes classificatoires* (printemps 95) et 96-030 *Des langages à objets aux logiques terminologiques : les systèmes classificatoires* (printemps 96).

Table des matières

Chapitre 1

Introduction

Les gens sont terribles, il faut tout leur expliquer.
Samuel Beckett, *Fin de partie*

Les différentes approches de représentation des connaissances fondées plus ou moins sur la notion d'objet et de relation — qu'il s'agisse des logiques terminologiques (LT), des représentations des connaissances à objets (RCO), voire des langages de programmation par objets (LPO), des bases de données d'objets (BDO) ou même des graphes conceptuels (GC) — permettent de faire émerger un modèle abstrait de représentation.

Le succès des objets est en effet justifié, habituellement, par les qualités que le « génie logiciel » a mises en avant : modularité, extensibilité ou réutilisabilité sont des références obligées. Pourtant, une raison moins reconnue est sans doute aussi importante : les objets (informatiques) ont une capacité « naturelle » de représentation (des objets) du « monde ». Cette capacité est à l'origine, en « intelligence artificielle » de la représentation des connaissances par objets, autant que des méthodes d'analyse et de conception, en génie logiciel. En fait, on peut conjecturer que ces bonnes qualités de représentation, par leur « déclarativité » [Winograd, 1975; Carré *et al.*, 1995], sont aussi à l'origine des qualités revendiquées par le génie logiciel. L'étude de la sémantique de représentation des diverses approches objet et leur comparaison dans un cadre commun est ainsi une étape nécessaire, en particulier dans le but de leur intégration [Carré *et al.*, 1995]. Or, parallèlement aux développements de l'approche objet dans le courant principal du Génie Logiciel, un courant important de la représentation des connaissances et plus particulièrement des *réseaux sémantiques* a conduit à la définition d'une famille de langages, les *logiques de descriptions*, dites aussi *logiques terminologiques*. Cette approche s'est trouvée confrontée, dès sa naissance, à ces questions de sémantique et les a résolues d'une manière tout à fait satisfaisante. Il était donc tentant d'appliquer cette recette éprouvée aux systèmes à objets plus usuels. Le cadre des *systèmes classificatoires* (SC) est une abstraction de ces diverses approches objet qui constitue une première proposition en ce sens.

Un système classificatoire est constitué, d'une part d'une *hiérarchie* de *classes*, décrites ou définies en *intension* par un ensemble de *propriétés* et munies d'une interprétation *extensionnelle*, d'autre part d'un ensemble d'assertions formant une description, incomplète, du monde. L'ensemble peut être considéré comme une logique, avec une sémantique directe en théorie des modèles, sur le modèle justement de celles que les logiques terminologiques ont établies depuis quelques années [Nebel, 1990], et un système de déduction constitué d'un ensemble de règles d'inférences et/ou d'algorithmes spécialisés. Il s'agit donc d'une abstraction du cadre sémantique général des LT, dans le but d'y faire rentrer les systèmes d'objets. L'instanciation de cette sémantique abstraite par tel ou tel système particulier peut bien entendu s'avérer triviale : notre objectif en reste à l'élaboration et à l'analyse de ce cadre abstrait pour, d'une part, l'appliquer à des langages existants, d'autre part, étudier la possibilité de spécifier de « meilleurs » langages.

Techniquement, ce cadre unificateur très général permet de poser divers problèmes :

- De quel ordre sont les déductions d'un système classificatoire ? Quels sont ses problèmes ? Par exemple : classification « automatique », subsomption, satisfaisabilité — comme le pratiquent les LT — mais aussi classification incertaine à l'image des RCO ;

- Pour chacun de ces problèmes, quelles sont la décidabilité et la complexité du problème, la complétude des solutions, etc. ?

- Quelles sont les propriétés formelles d'un SC ? Un SC se caractérise par 3 relations différentes : l'ordre de la hiérarchie et les préordres des subsomptions intensionnelles et extensionnelles : quelles sont les relations entre ces trois relations ? Quelles propriétés structurelles ou graphiques sont possibles ou souhaitables ?

- Comment les différents SC connus s'insèrent-ils dans ce cadre ? La question concerne d'abord la famiile des LT, dans son ensemble, et les systèmes de RCO comme FROME ou TROPES, mais aussi les ψ-termes ou les graphes conceptuels, voire la classification de molécules suivant leurs structures et sans doute de nombreux autres systèmes *a priori* classificatoires.

Cependant, des systèmes tout aussi classificatoires *a priori* (classification phylogénétique des espèces, par exemple) rentrent plus difficilement dans ce cadre. Il ne s'agit donc pas d'une théorie entièrement générale et abstraite des systèmes classificatoires, comme Euzenat (1993a) a pu en proposé une : la théorie présentée ici a pour spécificité son caractère « objet » qui se manifeste par le fait que les classes et les objets sont décrits par des *attributs*.

La justification de « classificatoire » est double : ces systèmes offrent une « classification des espèces », au sens de la hiérarchie de classes, et ils se prêtent à une « classification des individus », au sens du mécanisme d'inférence qui reconnaît qu'un individu appartient à une classe.

Cette approche classificatoire a été reconnue et mise en œuvre à la charnière des années 70 et 80 dans les langages dits de *frames* [Minsky, 1975; Fikes et Kehler, 1985] qui ont vu l'émergence d'un raisonnement basé sur la classification [Napoli, 1991; Napoli, 1992] et ont donné naissance à deux courants principaux (figure 1.1). L'un des critères discriminant ces deux courants est le caractère *définitionnel* ou *descriptif* des classes. Les logiques terminologiques (LT) ont introduit la distinction entre concept (classe) *défini* ou *primitif* [Nebel, 1990] : pour une classe définie, la description de la classe constitue une *condition nécessaire et suffisante* (CNS) d'appartenance d'un objet à son extension. En revanche, pour une classe primitive, la description n'est qu'un ensemble de conditions nécessaires (CN).

Des classes définies autorisent la classification automatique des instances que pratiquent les LT sous le nom de *réalisation* ou, plus récemment, d'*instanciation*. En contre-partie, les classes définies nécessitent une aide à leur définition, aide qui peut être apportée par la classification automatique des classes. Du côté des systèmes de représentation des connaissances par objets (RCO), la sémantique des classes est purement descriptive : aussi, depuis SHIRKA [Rechenmann, 1988] et comme ses successeur ou émule TROPES [Mariño, 1993; Euzenat et Rechenmann, 1995] et FROME [Dekker, 1994], certains de ces systèmes pratiquent, pour remédier à l'impossibilité d'une classification automatique, une classification que l'on peut qualifier d'*incertaine* puisqu'elle attribue à la relation classe-instance une modalité parmi *sûre*, *possible* et *impossible*.

Chacune de ces approches présente une limitation importante : pour les logiques terminologiques, les classes primitives constituent une barrière quasiment infranchissable, dans la mesure où l'appartenance à une classe primitive ne peut en général pas se déduire, alors que, de son côté, la représentation des connaissances par objets ne peut effectuer presqu'aucune inférence automatique valide.

L'objectif à moyen terme de cette étude est de contribuer à l'analyse des points communs et des différences entre ces diverses approches basées sur la classification et les objets, avec trois sous-objectifs principaux :

- clarification de la sémantique des systèmes de RCO existants : en particulier, analyse de la classification incertaine que certains mettent en œuvre pour remédier à l'impossibilité d'une classification automatique, et vérification de sa validité ;

- convergence entre RCO et LT, dans l'hypothèse d'une fusion possible voire nécessaire : cette convergence peut s'effectuer dans les deux sens, soit en modifiant la sémantique des logiques de description pour qu'elle prenne en compte une caractéristique utile de l'approche objet — explication des attributs, classification incertaine —, soit en modifiant la spécification des systèmes de RCO existants pour qu'ils puissent réaliser pleinement leur sémantique implicite ;

- intégration entre LPO d'un côté et RCO ou LT de l'autre, pour fournir un cadre maîtrisé de programmation et de représentation, et une coexistence harmonieuse entre déclaratif et procédural.

Cet ouvrage constitue une première avancée dans ce programme. D'une certaine manière, c'est la poursuite personnelle de la discussion entamée dans le groupe « classification et objets » du PRC-IA [Carré *et al.*, 1995], avec deux objectifs à court terme : donner un contenu à la sémantique du tronc commun à toutes les approches objet et, sur la base de cette sémantique, approfondir la comparaison entre systèmes d'objets et logiques terminologiques et vérifier la correction sémantique des premiers, en particulier en ce qui concerne la classification incertaine qui constitue la principale originalité des langages de la famille SHIRKA et en fait une alternative aux logiques de description pour la mise en œuvre d'un raisonnement basé sur la classification [Napoli, 1991; Napoli, 1992].

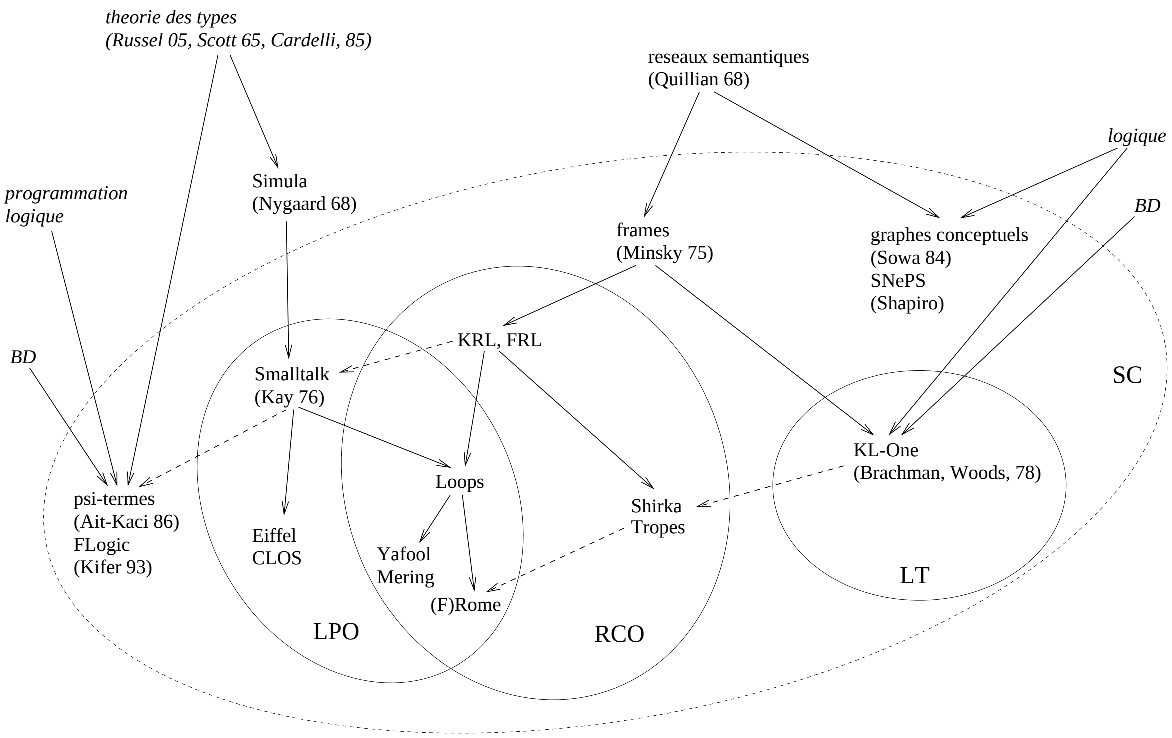

FIG. 1.1 – *Histoire géographie des systèmes classificatoires.*

Plan

Après cette introduction, le chapitre 2 présente, de manière pseudo formelle et intuitive, le cadre général des SC, cadre dans lequel peut se poser le problème de la classification d'instances, d'une façon un peu orthogonale à la présentation de [Euzenat, 1993a; Euzenat, 1994]. Le problème est posé avec une intention de généralité — réduite seulement par le fait que l'intension des classes se définit par un ensemble de propriétés — et avec une réussite partielle puisque certains constructeurs des *logiques terminologiques* (LT) ne rentrent pas bien dans le cadre proposé ici. En ce qui concerne les LT, la référence principale est [Nebel, 1990] dont la démarche est suivie d'assez près : la connaissance de ses chapitres 3 et 4 peut aider le lecteur mais n'est pas indispensable. Outre les références précédentes, le lecteur pourra aussi consulter [MacGregor, 1991; Woods, 1991; Woods et Schmolze, 1992; Borgida, 1992b; Buchheit *et al.*, 1993; Borgida et Patel-Schneider, 1994; Donini *et al.*, 1994] : ces références constituent un point de départ suffisant pour couvrir la quasi totalité de la bibliographie sur les LT.

Le chapitre 3 dresse ensuite un catalogue de propriétés que l'on pourrait vouloir imposer à un SC spécifique : il reprend en particulier les critères proposés par Euzenat (1993a) pour caractériser les différents systèmes classificatoires, tout en insistant sur les rapports fondamentaux entre la hiérarchie de classes, et les subsomptions extensionnelle et intensionnelle.

Jusque là le discours est assez informel, même si une mathématisation minimale permet de garder une certaine précision. Le chapitre 4 s'applique alors à formaliser toutes ces notions, en fixant à la fois le cadre de l'expression de l'intension des classes, et celui de leur interprétation extensionnelle : c'est la sémantique des SC, exprimée classiquement en théorie des modèles, comme le fait Nebel. Le chapitre suivant étudie enfin la déduction, en en restant à un assez haut niveau d'abstraction.

Le chapitre 6 s'intéresse ensuite au problème plus particulier de la classification d'instances, en distinguant en particulier la classification automatique des LT de la classification incertaine et interactive de la RCO. Cette dernière est analysée dans le détail et définie de façon rigoureuse.

Enfin, le dernier chapitre décrit les principaux systèmes classificatoires identifiés jusqu'ici, en essayant de les apparier à ce cadre théorique.

L'ouvrage se termine par plusieurs annexes, dont la première inclut tout ce qu'il faut savoir sur les ensembles ordonnés et introduit les notations usuelles, et la seconde fait de même pour les logiques de description. Les index sont à la fois copieux et exhaustifs : c'est l'hyper-texte du pauvre, qui peut servir à suivre à la trace les différentes notions, en plus des renvois explicites dans le texte.

NOTE *1.1* *Naïvement formel ou formellement naïf?*

Ami lecteur, patience ! Compte-tenu de la généralité et de l'abstraction de son approche, ce rapport procède souvent par un premier exposé pseudo formel mais intuitif, suivi d'un schéma de formalisation rigoureux, quitte à faire coexister les deux par la suite, la formalisation rigoureuse n'étant qu'un cadre général de formalisation possible. Par ailleurs, de nombreuses affirmations péremptoires reposent plus sur l'intuition que sur une argumentation sans faille : l'auteur attend avec impatience la contradiction.

NOTE *1.2* *Cadre unificateur :* RCO, LPO *et* BDO

La prétention de l'auteur de présenter un cadre unificateur pour les trois sommets de ce triangle repose sur les considérations suivantes. En ce qui concerne les bases de données, il faut bien distinguer un niveau logique qui est de l'ordre du modèle de données et un niveau physique qui concerne le stockage et les accès. À ces deux niveaux, les bases de données d'objets (BDO) ont ajouté une composante de programmation empruntée aux langages de programmation par objets (LPO). Le modèle de données des BDO n'est pas différent, dans son essence, du modèle de représentation des systèmes de représentation des connaissances par objets (RCO) [1], même s'il l'est souvent, en pratique. De la même manière, on peut distinguer dans les langages de programmation par objets un modèle de représentation et un modèle de programmation. On déplace ainsi ce premier triangle RCO-LPO-BDO vers un triangle représentation-programmation-stockage.

La prétention se ramène ainsi à l'hypothèse qu'il existerait un modèle de représentation universel sous-jacent aux modèles de représentation élaborés dans ces trois disciplines. Cette prétention ne se justifie que par l'argument d'un commun besoin de représentation « du monde » mais elle n'exclut pas la possibilité d'une incompatibilité entre deux besoins. Ainsi, le modèle de représentation impose une covariance du domaine des attributs, alors que la théorie des types exige leur invariance (cf. note 4.3, page 58) : cela ne signifie pas que l'on ne puisse pas intégrer la représenttaion et la programmation dans des programmes « sûrs », mais simplement que ces programmes ne seraient pas « sûrs du point de vue des types ».

1. Ainsi la communauté des logiques terminologiques se revendique clairement des bases de données, par le titre de ses publications plus que par les journaux ou conférences où elle publie. Cela ne signifie pas que la communauté des bases de données la reconnaisse comme telle.

NOTE *1.3* *Cadre unificateur : objets, logiques de description, ψ-termes, etc.*
L'auteur prétend aussi unifier des approches aussi différentes que les « objets », les logiques de description, les ψ-termes, et tant d'autres qui ne seront que brièvement examinées au chapitre 7 et dont la figure 1.1 essaie de montrer la généalogie. « *Qui trop embrasse mal étreint* » : le risque est grand en effet de n'en plus rien retenir, comme Wittgenstein avec les jeux (cf. note 3.7, page 53). Aussi cette unification concernera prioritairement les objets, dans leurs diverses approches, et les logiques de description.

NOTE *1.4* *Classification(s)*
Le terme de *classification* est utilisé classiquement — par exemple en analyse de données ou en apprentissage — pour désigner la construction d'un ensemble (partition ou hiérarchie) de classes à partir d'un ensemble d'individus. Malgré l'antériorité de cet usage, non breveté heureusement, les LT et la RCO ont repris ce même terme pour désigner l'association d'un individu à une classe, le terme de *catégorisation* prenant alors quelquefois ce premier sens. La langue française (ou anglaise) permet manifestement cette polysémie, même si des synonymes — par exemple *classement* — permettraient théoriquement, par l'édiction d'une norme appropriée, de réduire l'ambiguïté. Que les tenants du premier usage nous pardonnent : nous suivrons le second, malgré leurs droits d'antériorité manifestes.

NOTE *1.5* *Langages à machins ou logiques de trucs ?*
Pour éviter l'anglicisme « orienté objet », la francophonie est tombée dans le « à objets » [Masini *et al.*, 1989], voire le « à classes » ou « à prototypes » [Mulet et Cointe, 1993; Mulet, 1995], dont la dissonance aurait dû nous protéger. Pourtant d'autres, voire les mêmes, proposent des « de classes », « de *frames* » ou « de descriptions » que la consonance fait préférer.

En attendant que l'Académie Française se prononce, et pour l'inciter à le faire dans le bon sens, le titre de ce rapport est à considérer comme une pétition pour l'usage de « langage d'objets », à l'instar de [Crampé et Euzenat, 1996].

NOTE *1.6* *Remerciements*
L'idée de ce travail m'est venue — à moins que je ne sois moi-même venu à cette idée — en 1994, à la suite de la rédaction d'un article collectif de débat pour les *Journées nationales du GDR PRC-IA* [Carré *et al.*, 1995]. Au fil des années, de nombreuses personnes ont participé indirectement à ce travail ou l'ont soutenu, que je voudrais remercier ici : d'abord, les membres du groupe *classification et objets* du PRC-IA pour les nombreuses discussions, et pour leur lecture critique des premières versions de ce rapport, en particulier J. Euzenat et A. Napoli, M. Habib qui m'a évité quelques barbarismes sur les ensembles ordonnés ou M. Huchard sur les treillis de Galois, G. Simonet qui m'a soufflé le terme de « classe (in)satisfaisable » et Ph. Vismara qui m'a transmis une petite partie de son expertise en LaTeX, dont j'ai essayé de faire le meilleur usage. Enfin, plusieurs promotions d'étudiants de DEA ont essuyé les nombreux plâtres de cours approximatifs en servant de cobayes à l'exposé de ces idées : qu'ils soient eux aussi remerciés.

Chapitre 2

Cadre général

Eu égard à la difficulté de transcender notre patron de pensée orienté vers l'objet, il convient plutôt de l'examiner de l'intérieur.
W.V. Quine, *Parler d'objets*, in [Quine, 1977]

LE PROBLÈME DE LA CLASSIFICATION se pose dans un cadre qui consiste en une *hiérarchie* de *classes* décrites en *intension* et interprétées en *extension*. Cette hiérarchie constitue une première composante, *intensionnelle*, ce que les LT nomment la *TBox* ; il lui correspond une seconde composante, *extensionnelle*, la *ABox* des LT dans la définition restreinte de Nebel (1990) ou de Buchheit *et al.* (1993). L'ensemble sera appelé un *système classificatoire* (SC). Le fait de se placer *a priori* dans un cadre de représentation par objets (RPO), au sens large, se traduit par la restriction fondamentale que l'intension des classes s'exprime en fonction d'*attributs* des classes et des objets.

Les différents termes employés ne sont pas supposés munis d'une quelconque signification préalable, même si celle que nous visons est proche de leur sens usuel ou intuitif. La composante intensionnelle offre une *classification des espèces* qui sert de support à la *classification des individus* de la composante extensionnelle, le terme de « classification » étant pris dans son double sens de structure et de processus.

2.1 La composante intensionnelle : les classes

La composante intensionnelle des SC est constituée d'une hiérarchie de classes décrites en intension par des *propriétés* qui s'expriment au travers d'*attributs* ; s'y ajoutent diverses relations entre ces classes.

2.1.1 La hiérarchie

$\mathcal{X}$ est un ensemble fini d'*entités génériques* ou *classes*, ordonnées par une relation de *spécialisation*, notée $\prec$, qui est un ordre partiel strict (et $\preceq$ est l'ordre partiel associé). Comme c'est l'habitude dans les approches objet, on parlera de *sous-classe* ou de *super-classe* en référence à cette relation. On placera aussi la super-classe « en haut » et la sous-classe « en bas », autorisant ainsi toutes les métaphores spatiales et en particulier les mouvements *ascendant* et *descendant*.

Pour désigner les successeurs directs d'une classe par la relation $\prec$, ses *super-classes directes*, on introduit la fonction $Super : \mathcal{X} \longrightarrow 2^{\mathcal{X}}$ telle que :

$$Super(\alpha) = \{\beta \mid \alpha \prec \beta \ \& \ \nexists\gamma : \alpha \prec \gamma \prec \beta\} \tag{2.1}$$

On soulignera aussi le fait que la hiérarchie est donnée « en extension », par la relation $\preceq$, plus précisément par la fonction $Super$.

2.1.2 Cadre extensionnel

Cette hiérarchie s'interprète dans un cadre extensionnel dont nous présentons d'abord un aperçu intuitif et relativement informel : Ω est un ensemble d'*entités individuelles* ou *objets*, dont des parties sont associées aux classes par l'application $Ext : \mathcal{X} \longrightarrow 2^{\Omega}$. $Ext(\alpha)$ est appelé l'*extension* de la classe α : c'est l'ensemble des objets qui « tombent sous le concept » α, le vérifient ou le représentent[1]. Lorsque $o \in Ext(\alpha)$, o est une *instance de* α, et α est *une* (et non pas *la*) *classe de o*. Inversement, l'application $Isa : \Omega \longrightarrow 2^{\mathcal{X}}$ associe à tout objet o les classes dont il est instance, *ses* classes :

$$Isa(o) \stackrel{\text{def}}{=} \{\alpha \in \mathcal{X} \mid o \in Ext(\alpha)\} \tag{2.2}$$

Ext est une fonction monotone (mais pas strictement) relativement à $\preceq$.

$$\alpha \preceq \beta \Longrightarrow Ext(\alpha) \subseteq Ext(\beta) \tag{2.3}$$

Les instances d'une classe sont aussi les instances de ses super-classes. Il s'ensuit que $Isa(o)$ est un *idéal*, caractérisé par son *antichaîne* minimale $\underline{Isa}(o)$ (cf. annexe A.2.3).

La relation de *subsomption extensionnelle* est un *préordre* sur $\mathcal{X}$, noté $\unlhd$, induit par l'inclusion des extensions :

$$\alpha \unlhd \beta \stackrel{\text{def}}{\Longleftrightarrow} Ext(\alpha) \subseteq Ext(\beta) \tag{2.4}$$

Les circuits de $\unlhd$ s'interprètent comme des classes extensionnellement équivalentes (ou co-extensives).

On admettra aussi qu'il existe dans $\mathcal{X}$ un maximum $\top$, d'extension Ω, ainsi qu'un minimum $\bot$, d'extension vide :

$$Ext(\bot) = \emptyset \qquad \min_{\prec}(\mathcal{X}) = \{\bot\} \tag{2.5}$$

$$Ext(\top) = \Omega \qquad \max_{\prec}(\mathcal{X}) = \{\top\} \tag{2.6}$$

1. Depuis Aristote jusqu'à nos jours, la variation terminologique est inépuisable : dans tous les cas, notre notion d'extension est tout à fait orthodoxe, et s'identifie parfaitement à la notion usuelle d'extension d'un concept ou d'une classe.

Enfin, $\min_{\prec}(\mathcal{X} - \{\bot\})$ désigne les *classes terminales*, ou *puits*, de la hiérarchie à l'exclusion de $\bot$. Il arrivera souvent que l'on oublie $\bot$.

L'ensemble de ces notions représente l'interprétation extensionnelle des systèmes d'objets. Il faut noter que l'ensemble Ω n'est en général pas donné, pas plus que les extensions, et qu'il faut donc les voir comme des ensembles d'instances *potentielles*. Le cadre extensionnel ne sert qu'à établir la sémantique des systèmes considérés et il n'intervient explicitement dans l'utilisation de ces systèmes que par la manipulation d'instances *effectives* qui constituent le *niveau extensionnel* qui sera discuté plus longuement plus loin (cf. paragraphe 2.2.1, page 27). Ce cadre extensionnel naïf trouvera dans la suite deux réalisations différentes : la première dans la composante extensionnelle qui décrit les instances effectives, la seconde dans la sémantique formelle du SC qui considère les instances *potentielles*.

NOTE *2.1* *Classe et ensemble*

L'emploi des termes de *classe*, *sous-classe* et *super-classe* [2] se justifie ainsi par l'interprétation extensionnelle et par la synonymie usuelle des mots « classe » et « ensemble ». Russel par exemple utilise le mot classe dans le sens d'ensemble [Russel, 1921]. Notons qu'en mathématiques, le terme de *classe* est utilisé pour ensemble dans deux sens précis opposés :

- d'une part, en théorie axiomatique des ensembles, comme *proto-ensemble* : dans l'axiomatique de Gödel, un ensemble est une classe particulière, axiomatisée — seuls les ensembles peuvent être éléments d'une classe —, ce qui permet de parler de la classe de tous les ensembles sans tomber sur l'*imprédicativité* à la base des paradoxes de Cantor-Russel [Gochet et Gribomont, 1994].
- d'autre part, comme ensemble particulier, classe d'équivalence par exemple.

Par ailleurs, en théorie des ensembles, une relation R est qualifiée d'*extensionnelle* lorsque, à l'image de l'appartenance ensembliste $\in$, $R^{-1}(x) = R^{-1}(y)$ ssi $x = y$: deux ensembles de mêmes éléments sont identiques. La relation d'appartenance entre les objets et leurs classes n'est extensionnelle que si $\trianglelefteq$ est un ordre. En toute généralité, il n'est donc pas question d'identifier la classe et son extension, même si l'on pourra parler abusivement d'appartenance à une classe.

Notons que l'usage de classe, qu'il soit philosophique [Lalande, 1926] ou mathématique, correspond à une définition par une condition nécessaire et suffisante (cf. citation page 85).

NOTE *2.2* *Subsomption*

La subsomption extensionnelle correspond exactement à la subsomption des LT, ainsi qu'à la *règle de subsomption* de la théorie des types [Bruce et Longo, 1990] :

$$\frac{C, A \vdash e : \sigma \qquad C \vdash \sigma \leq \tau}{C, A \vdash e : \tau}$$

qui se lit : si e a le type σ et que σ est un sous-type de τ, alors e a le type τ.

2. Il ne paraît pas souhaitable de dire *sur-classe* : « super » est un préfixe couramment utilisé en français (superstructure, etc.), dans le sens de « qui est au-dessus » alors que « sur » est plus souvent employé dans le sens de « qui dépasse » (surcoût, surcapacité, etc.)

2.1.3 Cadre intensionnel

Un cadre *intensionnel* se superpose à cette interprétation extensionnelle : chaque classe est accompagnée d'une « description » *en intension*, par une collection de « propriétés » qui s'expriment par l'appartenance d'un *attribut* à un *domaine*[3].

Attributs et domaines

$\mathcal{P}$ est un ensemble fini d'attributs et, $\forall \alpha \in \mathcal{X}$, $\mathcal{P}_\alpha$ représente l'ensemble des attributs de α. L'*héritage de propriétés* consiste d'abord en une inclusion des ensembles d'attributs associés à des classes qui se spécialisent :

$$\alpha \prec \beta \Longrightarrow \mathcal{P}_\alpha \supseteq \mathcal{P}_\beta \tag{2.7}$$

La fonction $\alpha \mapsto \mathcal{P}_\alpha$ est donc une fonction (pas strictement) monotone de $(\mathcal{X}, \preceq)$ dans $(\mathcal{P}, \supseteq)$.

Inversement, la fonction $Def : \mathcal{P} \longrightarrow 2^{\mathcal{X}}$ associe à tout attribut P l'ensemble

$$Def(P) \stackrel{\text{def}}{=} \{\alpha \mid P \in \mathcal{P}_\alpha\} \tag{2.8}$$

des classes qui le possèdent : $Def(P)$ est aussi un idéal et $\overline{Def}(P)$ $(= \max_\prec(Def(P))$, cf. annexe A.2.3) sera l'*ensemble de définition* de P, ou encore l'ensemble des classes qui *introduisent* P.

Dans chaque classe, un attribut a un *domaine*[4] défini par une application

$$Dom : \mathcal{X} \times \mathcal{P} \longrightarrow 2^{\mathcal{D}}$$

où $\mathcal{D}$ est un domaine quelconque contenant au moins Ω et 2^Ω. On notera aussi Dom_P (resp. Dom_α) la fonction obtenue en fixant P (resp. α) :

$$Dom_P : Def(P) \longrightarrow 2^{\mathcal{D}}$$
$$Dom_\alpha : \mathcal{P}_\alpha \longrightarrow 2^{\mathcal{D}}$$
$$Dom_P(\alpha) \stackrel{\text{def}}{=} Dom_\alpha(P) \stackrel{\text{def}}{=} Dom(\alpha, P)$$

Dom_α et Dom_P sont des fonctions totales, alors que Dom n'est qu'une fonction partielle sur $\mathcal{X} \times \mathcal{P}$[5].

---

3. Le terme de *propriété* n'est pas utilisé ici pour désigner les attributs, mais réservé à son usage prédicatif usuel, par exemple pour dire qu'un attribut d'un objet appartient à un certain domaine.

4. Il serait plus juste de parler de *co-domaine*, puisque l'attribut est interprété comme une fonction ou une relation, mais « domaine » est, en pratique, plus fréquent. Il est en fait inutile de préciser le *domaine* (au sens strict) : c'est l'extension de la classe.

5. Mais on pourrait aussi en faire une fonction totale, en posant que $Dom(\alpha, P) = \perp$, où $\perp$ est une valeur distinguée, lorsque $P \notin \mathcal{P}_\alpha$.

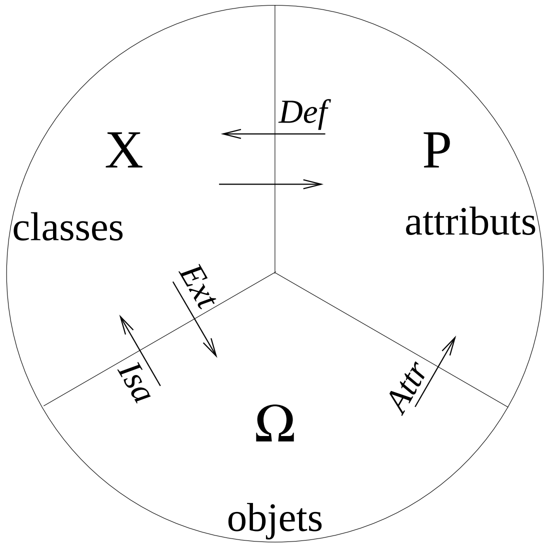

FIG. 2.1 – *Composante intensionnelle : classes, attributs, objets et leurs relations.*

NOTE *2.3* *Attributs, relations, rôles*

On ne cherchera pas à préciser ici ce que c'est qu'un attribut : est-ce un nom ? ou une entité abstraite ? et dans ce dernier cas a-t-elle un nom ? et quelles autres propriétés ? Voir par exemple [Ducournau *et al.*, 1995], qui discute ce point.

On pourrait aussi faire une typologie des attributs, en distinguant des autres attributs un attribut dont le domaine est inclus dans Ω ou 2^{Ω}, et l'appeler une *relation*. On ne le fera pas car on s'intéresse ici essentiellement à eux. Les autres attributs ont leur domaine inclus dans les *types de base* et posent d'autres genres de problèmes (cf. paragraphe 4.2.4, page 75).

Les LT utilisent le terme de *rôle* à propos des premiers : on les suivra, à leur propos (des LT) exclusivement. De même, comme les LT parlent de *concept* au lieu de classe, nous ferons souvent comme elles, pour elles.

L'extension des attributs

Le cadre intensionnel est relié au cadre extensionnel comme suit.

D'abord, chaque objet o possède tous les attributs décrits dans ses classes, que l'on peut lui associer au moyen d'une application $Attr : \Omega \longrightarrow 2^{\mathcal{P}}$:

$$Attr(o) \stackrel{\text{def}}{=} \bigcup_{\alpha \in Isa(o)} \mathcal{P}_\alpha = \bigcup_{\alpha \in \underline{Isa}(o)} \mathcal{P}_\alpha \tag{2.9}$$

On verra que cette relation entre les attributs des classes et ceux des objets pose deux questions : un objet peut-il avoir plus d'attributs que ses classes (cf. paragraphe 4.3.3, page 81) ? Une classe peut-elle avoir moins d'attributs que ses instances (cf. paragraphe 4.3.2, page 79) ? Sous l'hypothèse que les attributs d'une classe sont exactement les attributs communs à toutes ses instances, l'inclusion des ensembles d'attributs (2.7) est une conséquence de l'inclusion des extensions (2.3).

Ensuite, pour tout objet o, à chacun de ses attributs P ($P \in Attr(o)$) est associée une *valeur*, notée $P(o)$. $Dom(\alpha, P)$ représente les valeurs admissibles de l'attribut P pour les instances de α :

$$o \in Ext(\alpha) \ \& \ P \in \mathcal{P}_\alpha \Longrightarrow P(o) \in Dom(\alpha, P) \tag{2.10}$$

Chaque attribut P peut être considéré comme une fonction partielle $P : \Omega \longrightarrow \mathcal{D}$, totale sur l'extension des classes qui introduisent P[6].

L'*héritage de propriétés* n'est pas seulement l'héritage des attributs : c'est aussi l'héritage de leur domaine et une contrainte d'inclusion en cas de *redéfinition* de ces domaines :

$$\alpha \prec \beta \ \& \ P \in \mathcal{P}_\beta \Longrightarrow Dom_P(\alpha) \subseteq Dom_P(\beta) \tag{2.11}$$

Dom_P est donc une fonction monotone de $(\mathcal{X}, \preceq)$ dans $(2^{\mathcal{D}}, \subseteq)$.

Note *2.4* *Héritage de propriétés*

L'héritage défini par les formules 2.7 et 2.11 rentre complètement dans le cadre plus général d'héritage de propriété présenté dans [Ducournau *et al.*, 1995], mais nous ne nous intéressons ici qu'à l'*héritage de nom*, sans *conflit*, et à un *héritage de valeur* de l'aspect « domaine », avec masquage compatible et résolution de conflit correcte. L'interprétation formelle des attributs qui sera proposée plus loin justifie l'héritage de nom et complète la présentation qui en a été faite dans [Ducournau *et al.*, 1995].

Note *2.5* *Monotonie de* Dom_P

L'inclusion des domaines exprimée par (2.11) n'est pourtant pas une conséquence logique des formules précédentes. On peut considérer qu'il s'agit de la suite de la définition de l'héritage et que c'est la notion intuitive d'héritage qui l'impose. Mais en fait, la sémantique des domaines est exprimée uniquement par (2.10) et elle n'entraîne l'inclusion que pour la « partie utile » des domaines, ce qui explique l'hypothèse de *minimalité des domaines* (cf. paragraphe 4.3.1, page 76).

Il y a donc monotonie si l'on considère le *domaine effectif*, c'est-à-dire l'ensemble des valeurs effectivement « réalisées », dans un sens à définir, mais la monotonie n'est plus nécessaire si l'on considère le *domaine exprimé* par l'utilisateur, bien qu'elle reste alors une propriété clarificatrice souhaitable. Le domaine effectif est bien sûr toujours inclus dans le domaine exprimé.

Pour [Crampé et Euzenat, 1996], la monotonie n'est une propriété que pour ce qu'ils appellent la *forme première* d'une base de connaissances : sans cette propriété, il peut exister des bases équivalentes (dans un sens à définir) mais non isomorphes et les bases « minimales » ont cette propriété de monotonie.

6. Ou bien une fonction totale, avec la convention qu'il existe dans $\mathcal{D}$ une valeur distinguée $\perp$ pour désigner l'absence de valeur due au fait que l'attribut n'est pas défini dans les classes de l'objet : $P(o) = \perp \iff P \notin Attr(o)$. A ne pas confondre avec le fait que la valeur de l'attribut puisse être (provisoirement) inconnue. En fait, une valeur distinguée pour dénoter l'inconnu n'est pas nécessaire : les attributs étant multi-valués, $\emptyset$ suffit.

NOTE *2.6* *Domaine d'un attribut dans un objet*

La notion de domaine d'un attribut P peut s'étendre des classes aux objets eux-mêmes, par la fonction $Dom_o : Attr(o) \longrightarrow 2^{\mathcal{D}}$, qui vérifie :

$$Dom_o(P) = \bigcap_{\alpha \in Def(P) \cap Isa(o)} Dom(\alpha, P) \tag{2.12}$$

L'introduction du domaine d'un attribut dans un objet devient nécessaire lorsque l'on abandonne l'hypothèse d'indépendance des attributs et que l'on introduit des contraintes entre plusieurs attributs (cf. paragraphe 4.1.2, page 59) ou simplement pour tenir compte de la valeur partiellement connue de l'attribut (cf. paragraphe 2.2.1, page 27). Dans le premier cas, le domaine de l'attribut dans l'objet a exactement la même interprétation que le domaine d'une variable dans un CSP.

Le filtrage, ou réduction de domaine, des CSP s'applique alors à ce domaine des attributs dans les objets, en réduisant le domaine effectif à partir du domaine exprimé.

L'intension des classes

On appellera *intension* de α, noté $Int(\alpha)$, le couple $(\mathcal{P}_\alpha, Dom_\alpha)$. On posera aussi, au prix d'un léger abus[7], que $Int(\top) = (\emptyset, \emptyset)$ et que $Int(\bot) = (\mathcal{P}, \emptyset)$. Les intensions sont munies d'une relation d'ordre $\sqsupseteq$ similaire à l'inclusion comme suit :

$$Int(\alpha) \sqsupseteq Int(\beta) \overset{\text{def}}{\Longleftrightarrow} \begin{cases} \mathcal{P}_\alpha \supseteq \mathcal{P}_\beta \\ \forall P \in \mathcal{P}_\beta : Dom_P(\alpha) \subseteq Dom_P(\beta) \end{cases} \tag{2.13}$$

Int est donc (non strictement) monotone :

$$\alpha \prec \beta \Longrightarrow Int(\alpha) \sqsupseteq Int(\beta) \tag{2.14}$$

La relation de *subsomption intensionnelle* est un *préordre* sur $\mathcal{X}$, noté $\trianglerighteq$, induit par l'inclusion des intensions :

$$\alpha \trianglerighteq \beta \overset{\text{def}}{\Longleftrightarrow} Int(\alpha) \sqsupseteq Int(\beta) \tag{2.15}$$

Les opérations ensemblistes usuelles trouvent leurs analogues intensionnels par la définition de deux opérateurs, $\sqcap$ et $\sqcup$, d'intersection et d'union :

$$Int(\alpha) \sqcap Int(\beta) \overset{\text{def}}{=} (\mathcal{P}_\alpha \cap \mathcal{P}_\beta, Dom_{\alpha\cup\beta}) \tag{2.16}$$

$$\text{où } Dom_{\alpha\cup\beta}(P) \overset{\text{def}}{=} Dom_\alpha(P) \cup Dom_\beta(P)$$

$$Int(\alpha) \sqcup Int(\beta) \overset{\text{def}}{=} (\mathcal{P}_\alpha \cup \mathcal{P}_\beta, Dom_{\alpha\cap\beta}) \tag{2.17}$$

$$\text{où } Dom_{\alpha\cap\beta}(P) \overset{\text{def}}{=} \begin{cases} Dom_\alpha(P) \cap Dom_\beta(P) & \text{si } P \in \mathcal{P}_\alpha \cap \mathcal{P}_\beta \\ Dom_\alpha(P) & \text{si } P \in \mathcal{P}_\alpha - \mathcal{P}_\beta \\ Dom_\beta(P) & \text{si } P \in \mathcal{P}_\beta - \mathcal{P}_\alpha \end{cases}$$

7. En fait, considérées comme des parties de produit cartésien, les deux intensions sont vides car la fonction Dom_α est vide, dans la première parce que son domaine l'est, dans la seconde parce que son co-domaine l'est.

$$\begin{tikzcd}
(\mathcal{P} \to 2^{\mathcal{D}}) \arrow[d, "\sqsupseteq"'] & \mathcal{X} \arrow[l, "Int"'] \arrow[r, "Ext"] \arrow[d, "\prec"'] & 2^{\Omega} \arrow[d, "\subseteq"] \\
(\mathcal{P} \to 2^{\mathcal{D}}) & \mathcal{X} \arrow[l, "Int"] \arrow[r, "Ext"'] & 2^{\Omega}
\end{tikzcd}
\qquad
\begin{tikzcd}
2^{\mathcal{D}} \arrow[d, "\subseteq"'] & \mathcal{X} \arrow[l, "Dom_P"'] \arrow[r, "\alpha \mapsto \mathcal{P}_\alpha"] \arrow[d, "\prec"'] & 2^{\mathcal{P}} \arrow[d, "\supseteq"] \\
2^{\mathcal{D}} & \mathcal{X} \arrow[l, "Dom_P"] \arrow[r, "\alpha \mapsto \mathcal{P}_\alpha"'] & 2^{\mathcal{P}}
\end{tikzcd}$$

FIG. 2.2 – *Contravariance des extensions et des intensions, et des attributs et des domaines : un diagramme pseudo-commutatif.*

Dans cette dernière définition, si jamais l'un des $Dom_\alpha(P) \cap Dom_\beta(P)$ est vide (si P n'a aucune valeur admissible), on pose alors que $Int(\alpha) \sqcup Int(\beta) = (\mathcal{P}, \emptyset) = Int(\bot)$ [8]. $\sqsupseteq$, $\sqcap$ et $\sqcup$ sont analogues aux relations et opérations ensemblistes correspondantes et en vérifient bien toutes les propriétés. En particulier, $Int(\alpha) \sqcap Int(\beta)$ (resp. $Int(\alpha) \sqcup Int(\beta)$) est bien la borne supérieure (resp. inférieure) (au sens de $\sqsupseteq$) de $Int(\alpha)$ et de $Int(\beta)$, ce qui donne aux intensions une structure de *treillis* (cf. A.3).

NOTE *2.7* *Contravariance des intensions et des extensions*

Les notations $\sqsupseteq$ et $\trianglerighteq$ — dans leur opposition aux notations $\subseteq$ et $\trianglelefteq$ [9] — mettent l'accent sur la *contravariance* des intensions par rapport aux extensions (cf. figure 2.2) : plus les classes sont spécialisées (basses dans la hiérarchie), plus l'extension est petite et plus l'intension (ensemble d'attributs) est grande, mais plus les domaines sont petits !

NOTE *2.8* *Treillis des intensions*

Formellement, l'espace des intensions est l'ensemble $\mathcal{P} \to 2^{\mathcal{D}}$ des fonctions partielles de $\mathcal{P}$ dans l'ensemble des parties de $\mathcal{D}$. Tout élément f de cet ensemble peut être considéré comme l'intension d'une classe α, avec $\mathcal{P}_\alpha = \mathrm{dom}(f)$ et $Dom_\alpha = f$. Les définitions de $\sqsupseteq$, $\sqcap$ et $\sqcup$ s'étendent à cet ensemble, en lui donnant une structure de treillis, sans se confondre pour autant avec les opérateurs ensemblistes usuels $\supseteq$, $\cap$ et $\cup$. C'est le *treillis des intensions*.

Dans les définitions précédentes, il ne s'agit donc pas de faire des opérations algébriques sur le langage des intensions, mais sur les ensembles qu'elles expriment : les opérations ensemblistes ci-dessus correspondent à des intersections et unions sur les domaines et co-domaines des fonctions, par exemple, pour $\sqcap$:

$$h = f \sqcap g \Longrightarrow \forall x \in \mathrm{dom}(h) = \mathrm{dom}(f) \cap \mathrm{dom}(g) : h(x) = f(x) \cup g(x)$$

Nous avons conservé l'expression relative aux classes, mais ces intensions abstraites ne correspondent pas forcément à une classe, *a fortiori* pour le résultat d'une opération sur des intensions, bien que toute intension (bien formée) puisse définir une classe.

Toute cette algèbre des intensions est très proche de celle d'Hautamäki [Hautamäki, 1986], aussi bien dans l'esprit que dans la lettre (techniquement).

8. Mais on pourrait aussi bien vouloir conserver des intensions incohérentes (d'extension vide) distinctes : il n'y a aucun rapport entre une `Licorne`, un `[Milliardaire heureux]` et un `[30 février]`.

9. Bien sûr, le symétrique du symbole d'une relation n'a (presque) rien à voir avec le symbole de la relation symétrique ! En revanche, les symboles de relations de préordre strict et non strict tels que $<$ et $\leq$, $\prec$ et $\preceq$, $\triangleleft$ et $\trianglelefteq$ sont bien deux à deux corrélés comme le veut l'intuition : $\trianglelefteq = (\triangleleft \cup =)$.

NOTE *2.9* *Compréhension de l'intension*

La notion d'intension proposée ici s'écarte partiellement de la notion classique, de Port-Royal à Frege, de *définition en intension* (ou *en compréhension*). Avec une véritable définition en intension, qu'il faut comprendre comme une condition nécessaire et suffisante (CNS), l'introduction d'une classe définit *la classe des objets ayant ces propriétés* (c'est-à-dire dont les attributs sont dans les domaines considérés). Pour l'instant, nous nous contenterons d'une interprétation plus floue, basée sur de simples conditions nécessaires (CN), avec *une classe d'objets ayant ces propriétés*. Cf. paragraphes 3.4.2 et 4.5.3.

À propos d'« intension », on évitera de l'écrire « intention » sous prétexte que ce ne serait pas un mot français : si « intension » est bien absent des dictionnaires français courants, on le trouve dans le Lalande (1926) qui le considère comme ancien, citations à l'appui de Leibniz [10]. On utilisera donc sans complexe aussi bien « intension » que le néologisme « intensionnel ».

NOTE *2.10* *Intension et type des objets*

L'intension des classes correspond en fait à ce que Abadi et Cardelli (1996) appellent le type des objets (*object type*), ou à ce que Capponi (1995) appelle type. C'est en effet la subsomption intensionnelle qui est le bon critère de substitution dans les langages de programmation : la spécialisation (c'est-à-dire le sous-typage) n'en est qu'une approximation.

NOTE *2.11* *Complémentaire relatif*

Pour poursuivre avec les intensions l'analogie ensembliste, on peut aussi envisager un complémentaire intensionnel relatif : lorsque $\alpha \trianglerighteq \beta$, $Int(\beta) \backslash Int(\alpha)$ peut être défini par l'intension minimale (au sens de $\sqsupseteq$) $Int(\gamma)$ telle que $Int(\alpha) = Int(\beta) \sqcup Int(\gamma)$. L'unicité n'est pas toujours vérifiée [Borgida, 1995a].

Soulignons que l'on cherche ici à déterminer le complément d'intension permettant de définir une classe à partir d'une autre, en aucun cas, de déterminer l'intension du complément extensionnel, c'est-à-dire de $Ext(\beta) \backslash Ext(\alpha)$, lorsque $\alpha \trianglelefteq \beta$.

2.1.4 Relation d'incompatibilité

À ce cadre général, il faut ajouter une notion d'incompatibilité entre classes, qui s'exprime par une relation $\nabla : \mathcal{X} \times \mathcal{X}$, et qui qui peut se définir extensionnellement :

$$\alpha \nabla \beta \overset{\text{def}}{\Longleftrightarrow} Ext(\alpha) \cap Ext(\beta) = Ext(\bot) \tag{2.18}$$

aussi bien qu'intensionnellement :

$$\alpha \nabla \beta \overset{\text{def}}{\Longleftrightarrow} Int(\alpha) \sqcup Int(\beta) = Int(\bot) \tag{2.19}$$

10. Le lecteur sceptique, et l'auteur lui-même l'a d'abord été, se demandera en quoi un philosophe allemand peut être une référence pour la langue française : la réponse est que Leibniz écrivait exclusivement en latin et en français !

cette dernière définition étant équivalente à :

$$(2.20) \qquad \alpha \nabla \beta \iff \exists P \in \mathcal{P}_\alpha \cap \mathcal{P}_\beta, Dom_P(\alpha) \cap Dom_P(\beta) = \emptyset$$

La définition intensionnelle implique la définition extensionnelle, mais ne lui est pas équivalente : il peut donc être nécessaire de déclarer explicitement certaines relations d'incompatibilité, en plus des déclarations intensionnelles implicites.

L'une ou l'autre définition entraîne que la relation d'incompatibilité est « héritable » (par 2.3 ou 2.7 et 2.11) :

$$(2.21) \qquad \alpha' \preceq \alpha \ \& \ \beta' \preceq \beta \ \& \ \alpha \nabla \beta \Longrightarrow \alpha' \nabla \beta'.$$

Si deux classes sont incompatibles, les sous-classes de la première sont incompatibles avec les sous-classes de la seconde.

Soit $\mathcal{X}^*$ l'ensemble des antichaînes de $(\mathcal{X}, \preceq)$ (cf. annexe A.2.3). En toute généralité, il faut étendre les définitions qui précèdent à une *hyper-relation* (au sens des hyper-graphes), c'est-à-dire une relation d'arité variable, supérieure à 2. Sans perte de généralité compte tenu de sa symétrie, la relation d'incompatibilité s'exprime alors par un sous-ensemble maximal d'antichaînes de $(\mathcal{X}, \preceq)$, minimales au sens de $(\mathcal{X}^*, \preceq_b)$ du fait de cette héritabilité, en fait comme une antichaîne de $(\mathcal{X}^*, \preceq_b)$. La relation d'incompatibilité est binaire si toutes ces antichaînes minimales sont de cardinal 2.

NOTE *2.12* *Incompatibilité et négation*

La relation d'incompatibilité diffère de la négation des LT — restreinte dans un langage comme $\mathcal{AL}$, ou généralisée dans $\mathcal{ALC}$ ou $\mathcal{ALCNR}$ [Schmidt-Schauß et Smolka, 1991; Buchheit *et al.*, 1993; Donini *et al.*, 1994][11] — par son caractère non définitionnel (au sens du paragraphe 3.4.2) au contraire de la négation. Par ailleurs, l'expression de l'incompatibilité est symétrique, ce qui n'est pas le cas de la négation, dont la dissymétrie peut s'étendre au caractère primitif ou défini des classes : sous peine de sexisme, il faut considérer que `Mâle` et `Femelle` sont deux classes primitives incompatibles[12], et non que l'une est primitive et l'autre définie par la négation de la première, bien que cela semble être le cas du point de vue de l'*ontogénèse*. La négation représente un complémentaire extensionnel absolu.

2.1.5 L'expression du système classificatoire

Dans ce qui précède, nous avons souvent mélangé les plans de l'expression et du contenu. Comment s'exprime la hiérarchie, c'est-à-dire comment un utilisateur peut-il en introduire les divers composants ?

Introduction des classes

En pratique, une classe α est introduite au moyen de ses super-classes directes $Super(\alpha)$, explicitement désignées (par défaut, $\{\top\}$) et par son *intension propre*, no-

11. Voir à l'annexe B.3, figure B.3, comment se déclinent ces différents langages.
12. Que les `escargots` nous pardonnent !

tée $Int^*(\alpha)$, par laquelle sont ajoutés des attributs ou restreints des domaines. $\mathcal{P}^*_\alpha$ désignera le sous-ensemble d'attributs de α figurant dans $Int^*(\alpha)$. Alors :

$$Int(\alpha) = \Big(\bigsqcup_{\beta \in Super(\alpha)} Int(\beta)\Big) \sqcup Int^*(\alpha). \tag{2.22}$$

On suppose bien sûr que le domaine de tout attribut *hérité* sans redéfinition s'obtient par intersection des domaines hérités :

$$\forall \mathcal{P} \in \mathcal{P}_\alpha - \mathcal{P}^*_\alpha : Dom_P(\alpha) = \bigcap_{\beta \in Super(\alpha) \cap Def(P)} Dom_P(\beta) \tag{2.23}$$

Ces deux formules peuvent être considérées comme la définition de l'héritage, les deux contraintes (2.7) et (2.11) en étant des conséquences. Notons que nous n'avons pas encore abordé l'expression du domaine des attributs (Dom) : ce sera fait au chapitre 4.

La relation $\prec$ est alors définie comme la fermeture transitive de la relation explicite entre les classes et leurs super-classes directes.

Cette prise en compte de l'introduction des classes amène à distinguer deux notions subtilement différentes relatives à l'intension d'une classe α : d'un côté, $Int(\alpha)$ qui représente uniquement les *propriété*s de la classe, c'est-à-dire ses attributs et leurs domaines, de l'autre, le couple formé par l'intension propre $Int^*(\alpha)$ et les super-classes directes $Super(\alpha)$. On appellera ce couple la *description* d'une classe. La description permet bien de calculer l'intension (d'une certaine manière, c'est l'*héritage*), mais l'inverse n'est pas vrai, du moins en toute généralité.

En ce qui concerne les classes, on distinguera donc très précisément quatre notions voisines :

- l'*introduction* d'une classe consiste à ajouter une nouvelle classe à la composante intensionnelle et, en général, à la nommer ;
- la *description* d'une classe est associée à la classe lors de son introduction : elle est constituée des super-classes directes (par leurs noms) et de l'intension propre ;
- l'*intension* de la classe est calculée après son introduction, par héritage des intensions de toutes ses super-classes ;
- la *définition* d'une classe constitue une condition nécessaire et suffisante (CNS) d'appartenance d'un objet à l'extension de la classe (cf. paragraphes 3.4.2, page 49, et 4.5, page 85) ; cette CNS peut *a priori* s'exprimer aussi bien en fonction de la description que de l'intension.

Ces distinctions pourraient s'appliquer aussi, dans une moindre mesure, aux attributs. Pour ces derniers, on distinguera :

- l'*introduction* d'un attribut consiste à introduire une classe avec l'attribut considéré dans son intension propre et sans qu'aucune de ses super-classes n'ait l'attribut ;

- la *redéfinition* d'un attribut consiste à introduire une classe avec l'attribut considéré dans son intension propre, en spécialisant ainsi une de ses super-classes qui a déjà l'attribut ;
- la *définition* serait l'une ou l'autre des opérations précédentes, mais c'est un usage délicat : la suite montrera qu'il est possible d'avoir des attributs *définis*, au même titre que les classes (cf. paragraphe 4.1.2, page 59).

NOTE *2.13* *Transitivité*
Une classe peut être introduite au moyen d'un ensemble de super-classes pas forcément directes, contenant donc $Super(\alpha)$ mais aussi des super-classes indirectes. Ces éventuels arcs de transitivité ne changent rien à la formule précédente. En revanche, les circuits sont exclus, une classe ne pouvant être introduite qu'une fois, et les super-classes devant être introduites au préalable : c'est la raison pour laquelle $\preceq$ n'est pas un préordre.

Pour la notation, si nous avons choisi * pour marquer la multiplicité, que ce soit pour la fermeture transitive ou les antichaînes, il aurait sans doute été plus cohérent d'inverser les rôles de $\mathcal{P}$ et $\mathcal{P}^*$ (resp. de Int et Int^*).

NOTE *2.14* *Héritage multiple et conflit de domaine*
L'héritage des domaines de la formule (2.23) correspond donc soit à un simple héritage du domaine, soit à ce que Ducournau *et al.* (1995) appellent une résolution correcte de conflit de valeur, sur l'aspect domaine de l'attribut P. Dans ce cas précis, il est strictement équivalent de résoudre le conflit sur les super-classes directes ou sur le véritable ensemble de conflit des super-classes minimales qui introduisent P. Ce n'est pas le cas pour tout procédé de résolution de conflit : une condition nécessaire (et suffisante ?) est que la résolution soit *monotone* et on peut considérer que cette propriété est un bon critère, sinon de « correction » (vis-à-vis de quel critères ?) de la résolution, du moins de son caractère intuitif [Ducournau *et al.*, 1995].

NOTE *2.15* *Aristote et l'héritage multiple*
L'introduction d'une classe suit donc l'approche aristotélicienne[13], par *genre* (*Super*) et *différence* (Int^*), à la seule différence que le genre n'est pas *a priori* unique (bien qu'il le soit pour les GC, d'après Leclère (1995)). C'est triste à dire, mais il ne semble pas qu'Aristote se soit intéressé à l'héritage multiple !

NOTE *2.16* *Haut, bas, fragile*
Au moins une construction échappe à cette introduction *descendante* des classes : le constructeur de disjonction (`or`) des LT permet d'introduire une classe de façon *ascendante* en la définissant extensionnellement par l'union des extensions de ses sous-classes. Une classe introduite par OR n'a pas d'intension propre : son intension se réduit à l'intersection ($\sqcap$) des intensions de ses sous-classes, mais l'intension résultante ne définit pas la classe (cf. paragraphe 3.4.2, page 52). L'opérateur `cover` a un effet assez similaire de définition par union des extensions tout en conservant une introduction descendante.

13. D'après Sowa (1984) confirmé par le Lalande (1926), recoupé par Johnson (1964) suivant Hautamäki (1986).

Subsomption et incompatibilité extensionnelles explicites

Outre l'introduction intensionnelle des classes, par l'explicitation de leurs attributs et de la relation de spécialisation, il peut être nécessaire d'expliciter les relations extensionnelles qui ne se déduisent pas de l'intension. Il faut donc compléter l'expression du SC :

- par l'expression de la part non implicite de la relation d'incompatibilité ∇, l'implicite pouvant être dû à une incompatibilité intensionnelle ou aux propriétés structurelles spécifiques du système ; on veut ici pouvoir dire qu'il n'existe pas, dans le(s) domaine(s) modélisé(s), d'objets de `[forme ronde]` et de `[couleur rouge]`.

- par l'expression de la part non implicite de la relation de subsomption extensionnelle $\trianglelefteq$: cette part non implicite ne doit pas se déduire de $\preceq$, et ne doit pas pouvoir y être incluse, soit parce qu'elle n'est pas incluse dans $\trianglerighteq$, soit parce que cela contredirait les propriétés structurelles spécifiques du système : c'est alors un équivalent des *passerelles* de TROPES (cf. paragraphe 7.2.3, page 168), dont le rôle est purement déductif. On veut ici pouvoir dire que, dans le(s) domaine(s) modélisé(s), tous les objets de `[forme carrée]` sont de `[couleur bleue]`.

Comme il ne faut pas confondre la part non implicite des relations ∇ et $\trianglelefteq$ avec leur part implicite déduite des définitions intensionnelles et des propriétés formelles du SC, on notera les premières, respectivement, ∇_a et $\lhd_a$ (a pour affirmé).

On posera que $\trianglelefteq_a = (\prec \cup \lhd_a \cup =)^*$, où * dénote la fermeture transitive : c'est un préordre, voire un ordre partiel. Par analogie avec TROPES, on appellera *passerelle* les éléments de $\lhd_a$.

NOTE *2.17* *À revoir*
L'introduction des passerelles conduit à modifier quelques-unes des définitions et contraintes précédentes pour y remplacer $\preceq$ par $\trianglelefteq_a$: définition de $Isa_{\mathcal{W}}$, inclusion des domaines (2.11), héritage de l'incompatibilité (2.21), et sans doute quelques autres (par exemple, celles qui utilisent *Super*). On verra néanmoins qu'il ne s'agit pas forcément de modifier les définitions et les formules mais qu'il peut être nécesssaire de dédoubler les notions pour tenir compte de la différence sémantique entre $\prec$ et $\lhd_a$ (cf. paragraphe 4.5.5, page 91).

Expression de l'incompatibilité

Nebel (1990) limite l'expression de la relation d'incompatibilité aux concepts primitifs, avec la construction `disjoint`, sous prétexte que les concepts définis (cf. paragraphe 3.4.2, page 49) doivent l'impliquer par leurs définitions mêmes, mais il paraît nécessaire de pouvoir exprimer la modélisation d'un monde dans lequel il n'existe pas d'objets, à la fois, de `[forme ronde]` et de `[couleur verte]` alors que toutes les autres combinaisons de valeurs forment des concepts tout à fait instanciables. Pour une raison qui n'est pas claire, il limite aussi l'incompatibilité au cas binaire, mais ce n'est une perte de généralité qu'en cas de restriction aux concepts primitifs[14] : `(disjoint`

14. L'argument de Nebel (p. 49) est aussi tout à fait spécieux. Il oublie que la définition est en fait relative à des super-classes primitives (cf. paragraphe 4.5.2, page 86), et que l'incompatibilité de deux

a b c) équivaut à (disjoint a (and b c)). CLASSIC offre une limitation analogue avec le constructeur `disjoint-primitive`.

NOTE *2.18* *Incompatibilité implicite*

Il est sans doute fastidieux de devoir énumérer, pour chaque classe, toutes les classes ou combinaisons de classes avec lesquelles cette classe est incompatible, ce qui n'est pas sans rappeler le fameux *frame problem*. Un `professeur d'université` est ainsi incompatible avec les `camions`, `maisons`, `nuages` (encore que ...), etc. La propriété d'héritage de l'incompatibilité (2.21) permet heureusement d'en limiter l'énumération au strict nécessaire, à savoir l'incompatibilité entre `êtres animés` et `êtres inanimés`[15].

Si l'incompatibilité est majoritaire, il peut être plus simple de n'expliciter que la *compatibilité* comme le fait [Dennebouy, 1993] dans son modèle de BDO. Mais la compatibilité présente le gros inconvénient de ne pas être héritable : si deux classes sont compatibles, il n'y a aucune raison que les sous-classes de l'une soient compatibles avec les sous-classes de l'autre. Cette compatibilité peut aussi être contrainte, comme en ROME où la multi-instanciation n'est possible qu'entre des classes dites *de représentation* sous-classes d'une même classe *d'instanciation* : limitée par cet espace et les éventuelles incompatibilité intensionnelles, la compatibilité peut alors rester implicite.

Enfin, l'incompatibilité peut être entièrement déterminée par une propriété spécifique du SC, l'exclusivité ou l'univocité par exemple (cf. paragraphe 3.1.3, page 39). C'est ainsi le cas des systèmes d'objets habituels, où la relation d'incompatibilité est implicite : tout ensemble de classes sans sous-classe commune est incompatible (cf. paragraphe 7.2.1, page 165).

La composante intensionnelle

On peut ainsi résumer la composante intensionnelle d'un SC à un sextuplet $\mathcal{H} = (\mathcal{X}, \mathcal{P}^*, Dom^*, \prec, \lhd_a, \nabla_a)$, où $\mathcal{P}^* : \mathcal{X} \to 2^{\mathcal{P}}$ associe à chaque classe α l'ensemble de ses attributs propres $\mathcal{P}^*_\alpha$ et $Dom^*_\alpha : \mathcal{P}^*_\alpha \to 2^D$ associe son domaine à chaque attribut propre.

2.2 La composante extensionnelle : les instances

La hiérarchie des classes munies de leurs intensions constitue la composante intensionnelle des SC, à laquelle on associe une composante extensionnelle. Ces deux composantes permettent de représenter, respectivement, des entités génériques ou individuelles. La composante extensionnelle est constituée par la description de toutes les entités individuelles, instances *effectives* des classes introduites dans la composante intensionnelle. Dans la suite, on supposera une composante intensionnelle particulière $\mathcal{H}$.

classes définies se ramène à des contraintes d'incompatibilité plus complexes (n-aires par ex.) entre leurs super-classes primitives.

15. À condition bien entendu de ne pas considérer dans le domaine modélisé `golems`, `zombies` et autres `monstres de Frankenstein`. Quant aux `maisons` et `camions`, l'incompatibilité est plus subtile : et les `camping-cars` ?

2.2.1 Description du monde

La composante extensionnelle — aussi dite assertionnelle ou factuelle — des SC est une *description du monde* notée $\mathcal{W}$. C'est une réalisation particulière de l'interprétation extensionnelle du SC, dont la sémantique précise sera discutée au chapitre 4. Intuitivement, cette description du monde doit respecter toutes les contraintes formulées dans l'exposé de l'interprétation extensionnelle des SC. Soulignons que cette description est *incomplète*, le premier but du SC étant de la compléter.

Style relationnel

La description du monde peut être vue comme un ensemble fini d'assertions, analogue à une base de données relationnelle, ces assertions pouvant être de deux sortes :

- $(\alpha\ o) \in \mathcal{X} \times \Omega$ affirme[16] l'existence de l'*objet* $o \in Ext(\alpha)$;
- $(P\ o\ o') \in \mathcal{P} \times \Omega \times \Omega$ associe à l'attribut P de l'objet o la valeur atomique o'.

Dans les deux cas, on dira que o (mais pas o') est l'« objet » de la description.

La signification de la première catégorie ne pose pas de problème particulier. La seconde pose par contre le problème des valeurs manquantes : la valeur de l'attribut P de l'objet o, notée $P(o)$, est constituée par éventuellement plusieurs assertions élémentaires : $(P\ o\ v_i), 1 \leq i \leq n$, et $v = \{v_i \mid 1 \leq i \leq n\}$. L'ensemble de ces assertions peut s'interpréter comme une égalité, $P(o) = v$, ou comme une simple inclusion, $P(o) \supseteq v$. Le second cas est plus général à condition de pouvoir affirmer l'éventuelle complétude de l'attribut : les LT offrent ainsi la possibilité de spécifier, pour les instances aussi (cf. paragraphe 4.1.1, page 55), des cardinalités maximum[17] ou minimum[18]. C'est aussi compatible avec la possibilité que $P(o)$ soit complètement inconnu : $v = \emptyset$. C'est une hypothèse du *monde ouvert* sur la valeur des attributs.

Style objet

Les instances s'expriment tout aussi bien dans un style objet — c'est-à-dire en faisant apparaître des entités avec des attributs —, par un ensemble $\mathcal{O}$ d'instances o, appartenant à des classes $Isa_{\mathcal{W}}(o)$ et munies d'un ensemble d'attributs $Attr_{\mathcal{W}}(o)$ dont la valeur est notée $P_{\mathcal{W}}(o)$:

$$\mathcal{O} \stackrel{\text{def}}{=} \{o \mid (\alpha\ o) \in \mathcal{W}\} \qquad (2.24)$$

16. La composante extensionnelle pourrait être appelée assertionnelle : chaque tuple est une assertion. Or « assertion » est un nom sans équivalent verbal. Le Lalande (1926) distingue bien « assertion » de « affirmation », mais il ne donne malheureusement pas « asserter » : « affirmer » sera donc utilisé sans préjuger de la valeur de vérité de ce qui est affirmé. Seule une opportunité favorable nous évitera d'ajouter à la perle classique « avérer faux » un « affirmer faux » de la plus belle eau : il s'avère que nous n'avons pas, à proprement parler, de négation.

17. Certaines LT comme CLASSIC permettent de dire qu'un rôle est *fermé*, ce qui est notablement plus subtil puisque cela signifie qu'il n'y a pas d'autres valeurs que celles qui se *déduisent* logiquement de $\mathcal{W}$ (cf. paragraphe 6.3.2, page 132).

18. Ce qui permet d'affirmer l'existence d'objets sans avoir à les définir totalement : on peut affirmer qu'un `Humain` particulier a des `parents`, sans qu'il soit nécessaire de remonter jusqu'à `Adam` et `Eve`.

$$(2.25) \qquad Isa_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{\beta \in \mathcal{X} \mid (\alpha\ o) \in \mathcal{W}\ \&\ \alpha \preceq \beta\}$$

$$(2.26) \qquad Attr_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{P \mid (P\ o\ v) \in \mathcal{W}\}$$

$$(2.27) \qquad P_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{v \mid (P\ o\ v) \in \mathcal{W}\}$$

Ces trois fonctions caractérisent donc ce qui a été explicitement affirmé de l'objet, sans aucune inférence, à part la fermeture par $\preceq$ pour $Isa_{\mathcal{W}}$. $P_{\mathcal{W}}(o)$ n'est bien sûr défini que pour $P \in Attr_{\mathcal{W}}(o)$. En ce qui concerne $Attr_{\mathcal{W}}(o)$ on verra plus loin une définition alternative, fonction des attributs des classes de $Isa_{\mathcal{W}}(o)$ (cf. paragraphe 4.3.3, page 81).

La fonction $Ext_{\mathcal{W}}$, qui représente les instances connues d'une classe, se définit de façon symétrique à $Isa_{\mathcal{W}}$.

Il peut sembler impossible d'affirmer la présence d'un attribut dont aucune valeur n'est connue, sauf à spécifier qu'il a au moins 0 valeur. Cependant, affirmer l'appartenance d'un objet à une classe implique la présence de tous les attributs de la classe. Inversement, vouloir affirmer la présence d'un attribut qui n'appartient pas à l'intension des classes de l'objet est contestable (cf. paragraphe 4.3.3, page 81).

NOTE *2.19* *Style objet ?*

Nebel (1990) ne fait pas apparaître ce style objet, sans doute parce que son expérience personnelle d'implémentation, le système BACK, est réalisé en PROLOG. Il refuse explicitement certaines opérations pour la classification d'instances sous prétexte d'inefficacité, alors que ce serait en fait beaucoup plus efficace dans un *style objet*. C'est cette vision des objets comme des entités *de première classe*, et en opposition avec une vision relationnelle, qui fonde l'approche objet dans toute sa généralité. Nebel s'en réclame pourtant explicitement, mais objets et relations sont, d'une certaine manière, sémantiquement équivalents et c'est finalement l'implémentation (la structure de données) qui fait la différence et ce *style objet*. Cette critique s'applique tout aussi bien aux langages d'objets basés sur PROLOG comme OBJLOG [Dugerdil, 1988] ou OBJLOG+ [Faucher, 1992; Faucher, 1991]. Formellement, le style objet se rend bien par l'usage de fonctions, et non de relations. Pratiquement, il s'énonce en termes d'implémentation [Perrot, 1996]. Pourtant, le style relationnel donne un point de vue ensembliste formellement pratique sur les descriptions du monde : nous le garderons pour ce seul usage.

NOTE *2.20*

Soulignons que les objets n'ont pas d'autre *propriété* que celles qui résultent de la description de leurs attributs et de l'intension de leurs classes.

2.2.2 Composantes intensionnelle et extensionnelle

La séparation entre les deux composantes n'est pas totale. Il y a deux façons de voir leur possible interaction : l'introduction de classes dont la description tienne compte de certaines instances ou bien, en sens inverse, l'introduction de classes lors de la description d'instances.

Certaines LT comme celles de la famille $\mathcal{AL}$ (cf. annexe B.3) autorisent implicitement l'*introduction* de nouvelles classes dans la composante extensionnelle, puisque, dans les assertions (α o), α peut être n'importe quelle description de classe[19]. C'est par ce biais que les restrictions de cardinalité sont imposées aux instances. Dans son formalisme assertionnel $\mathcal{AF}$, Nebel n'autorise que des concepts atomiques ou des *restrictions de cardinalité* (`atleast` et `atmost`) : comme la conjonction (`and`) est implicite, seule la *restriction de valeur* (`all`) est en fait interdite.

En sens inverse, il est envisageable de décrire des classes en fonction d'instances. CLASSIC propose ainsi les constructeurs `one-of` et `fills`. le premier permet d'énumérer l'extension d'une classe, ce qui revient à la définir en extension tandis que le second permet d'imposer une ou plusieurs valeurs à un attribut d'objet. Ces constructeurs présentent beaucoup de difficulté puisque la hiérarchie de classes peut dépendre des classes de certaines instances : compléter une instance pourrait modifier la hiérarchie. Aussi, la description de la classe ne prend en compte l'objet que dans l'état où il est lors de l'introduction de la classe [Brachman *et al.*, 1991; Resnick *et al.*, 1993; Borgida et Patel-Schneider, 1994]. Dans son formalisme $\mathcal{TF}$, Nebel exclut toute construction à partir d'individus.

Pour ce qui est des rapports entre les composantes intensionnelle et extensionnelle, on adoptera un point de vue qualifié d'*aristotélicien* — Aristote est censé ne pas avoir accepté l'idée platonicienne de l'existence d'idées, les fameux idéaux platoniciens (cf. paragraphe 7.5, page 177) — en ce qu'il donne la priorité aux objets (la composante extensionnelle) sur les classes (la composante intensionnelle) qui ne sont qu'un moyen commode d'abstraction et de factorisation.

2.3 La classification

> [S]'il y avait [..] un principe d'induction purement logique, il n'y aurait pas de problème lié à l'induction car [..] toutes les inférences inductives devraient être considérées comme des transformations purement logiques ou tautologiques, tout comme les inférences pratiquées dans la logique déductive.
>
> Karl R. Popper, La logique de la découverte scientifique, 1934

Comme leur nom l'indique, les systèmes classificatoires sont destinés essentiellement à faire de la classification, mais il ne s'agit que d'un objectif emblématique parmi d'autres (à moins de considérer que c'est l'objectif de classification qui ne peut pas se réaliser seul).

2.3.1 Les objectifs d'un système classificatoire

La construction de la hiérarchie elle-même mise à part, un système classificatoire peut être utilisé, au niveau des instances, de plusieurs façons :

l'héritage permet de connaître *déductivement* les attributs d'un objet — pas leurs valeurs, au moins dans un premier temps (cf. paragraphe 6.4.4, page 151), mais

19. Notons qu'il s'agit toujours d'une classe définie, si ce n'est pas une classe atomique.

leur présence et leur « intension » —, étant données les classes auxquelles il appartient : c'est un mécanisme *ascendant* ;

la classification permet de reconnaître *inductivement* un objet comme instance d'une classe, compte-tenu de ses attributs connus[20] : c'est un mécanisme *descendant* ;

des assertions $\{(\alpha\ o)\ (P\ o\ v)\}$, il est possible de déduire que $v \in Dom(\alpha, P)$, mais il est aussi possible d'induire une classe β, éventuellement sous-classe de α, telle que

$$v \in Dom(\beta, P)\ \&\ o \in Ext(\beta).$$

le filtrage détermine toutes les instances d'une classe, ou, plus généralement, tous les objets qui vérifient certaines propriétés qui peuvent s'exprimer comme l'intension d'une classe : dans les LT, toute requête est considérée comme une classe dont il ne reste plus qu'à collecter les instances, une fois qu'elle a été correctement insérée dans la hiérarchie. C'est aussi l'approche suivie dans FROME [Dekker, 1993]. Cela suppose que les langages de requête et d'expression des classes soient identiques, ou bien seule la partie commune est réifiée.

la construction d'instance [Girard, 1995] consiste à valuer les attributs d'un objet, éventuellement en en créant d'autres, qu'il faut récursivement construire et classer.

Mais la distinction de ces différents problèmes est relativement artificielle : il s'agit d'abord d'un problème uniforme de déduction. Ce n'est qu'ensuite que l'on peut introduire des spécificités comme une classification ou un filtrage incertains. Mis à part le filtrage, dans tous les problèmes considérés ici, la hiérarchie $(\mathcal{X}, \prec)$ est supposée constante.

Rappelons que la description du monde est essentiellement incomplète : le premier objectif de cette déduction est de la compléter. Ce n'est que par la suite que l'on peut envisager de la réviser, pour suivre l'évolution du monde.

2.3.2 Classification de classe ou d'instance

Les SC peuvent aussi présenter divers problèmes au niveau intensionnel, comme la subsomption — une classe subsume-t-elle une autre classe — ou la classification de classes qui consiste à insérer une classe « à sa place » dans la hiérarchie. Mais, suivant le point de vue aristotélicien que nous venons d'énoncer, ces problèmes du niveau intensionnel ne sont que des problèmes techniques que nous ne traiterons pas pour eux-mêmes, les vrais objectifs étant au niveau extensionnel. Historiquement, la classification d'instances a souvent été traitée, dans les LT, comme un cas particulier de la classification de classes. L'éventuelle équivalence des classifications de classe et d'instance, donc du filtrage et de la classification, n'est pas discutable *a priori* : elle dépend de l'expressivité du SC considéré. Le filtrage ne sera pas plus discuté ici : il

20. Les LT réservent le terme de classification à l'opération d'insertion d'une nouvelle classe dans la hiérarchie préexistante, par une succession de calculs de subsomption. Pour la classification d'instances, ils utilisent plutôt le terme de *réalisation* [Nebel, 1990] ou, plus récemment, d'*instanciation*.

relève de la classification automatique de classes. L'héritage lui-même n'est abordé que dans les termes très généraux qui précèdent : c'est la classification des instances qui constituera, au chapitre 6, l'essentiel de notre problématique.

Comme il l'a été noté plus haut, l'héritage peut être qualifié de déductif puisqu'il représente une forme de syllogisme correcte et très naturelle, pour déduire les propriétés de l'objet de ses classes. Comme la classification revient à inverser ce syllogisme, pour déduire les classes de l'objet de ses propriétés, il est normal de la qualifier d'inductive. Mais comme il n'est pas question de chercher à démentir Karl Popper, il s'agira bien sûr de trouver le (ou un) sous-ensemble déductif de cette problématique généralement inductive.

2.4 Premières discussions

Toute idée fausse finit dans le sang mais c'est toujours le sang des autres
Albert Camus

À ce stade, il se présente déjà quelques grandes catégories de questions métaphysiques, qui obligent à contester, avec raison, la prétention de notre approche à une totale généralité.

2.4.1 Dualité classe-attribut

La première discussion est de l'ordre de la poule et de l'œuf : qui est premier de la classe ou de l'attribut ? [Wand, 1989] et les LT semblent pencher pour les seconds, et les systèmes d'objets pour les premières. Mais c'est sans doute une question sans objet, comme pour les points et les droites en géométrie. Cependant, suivant la réponse qui lui est donnée, les attributs peuvent être interprétés comme des relations — dont le domaine est la totalité du domaine lui-même, tout objet a tous les attributs — ou comme des fonctions *partielles*, ce que nous préférons. Cette question départage en fait deux camps : d'un côté les systèmes à objets, de l'autre les LT, TROPES et, curieusement, l'analyse de données. Les LT sont ainsi très justement nommées *attributive language* dans la famille des langages $\mathcal{AL}$ (cf. annexe B, page 191). À l'opposé, les langages « d'objets » sont souvent « de classes ». Nous nous plaçons dans le premier, mais il faudrait voir s'ils sont vraiment aussi antagonistes ou, au contraire, équivalents que ça.

En première approximation, il paraît strictement équivalent d'interpréter les attributs par des relations ou par des fonctions, à condition que ce soit des relations de $\Omega \times \mathcal{D}$ et des fonctions de $\Omega \rightarrow 2^{\mathcal{D}}$. Outre le *style objet* déjà mentionné, des fonctions *partielles*[21] ont l'avantage de distinguer l'absence de valeur (l'attribut n'est pas défini) de la valeur $\emptyset$ (mais le même résultat s'obtient avec des fonctions totales et une

21. Mais la fonction associée à l'attribut P de $\mathcal{P}_\alpha$ est néanmoins totale sur l'extension de α :

$$\mathrm{dom}(P) = \bigcup_{\alpha \in \overline{Def}(P)} Ext(\alpha).$$

valeur distinguée $\perp$). Les fonctions partielles permettent ainsi d'imposer l'antériorité des classes sur les attributs. En revanche, des fonctions de $\Omega \rightarrow \mathcal{D}$ restreindraient le problème aux attributs mono-valués.

À part ce léger avantage des fonctions, les deux interprétations ont leurs avantages et inconvénients respectifs au niveau des notations : les relations favorisent par exemple la notation de la composition (au sens mathématique du terme : $P \circ Q$). Le paragraphe 7.2.1 présentera quelques implications de ces deux interprétations.

2.4.2 Classes et ensembles, réflexivité et imprédicativité

> Si je dis par exemple : « Napoléon avait toutes les qualités d'un grand général », je dois définir ce que sont ces « qualités » d'une manière telle que ce que je suis en train de dire n'en fasse pas partie, *i.e.* « avoir toutes les qualités d'un général » ne doit pas elle-même être une qualité au sens qu'on vient d'admettre.
>
> Bertrand Russel, *Introduction à la philosophie mathématique*, 1921.

Le problème de la définition des ensembles en intension est à la source des nombreux paradoxes (ou antinomies) de la théorie des ensembles, dus en particulier à Cantor et Russel, — le barbier qui (ne) se rase (pas) lui-même, l'ensemble de tous les ensembles[22], l'ensemble des ensembles qui ne se contiennent pas eux-mêmes, ou « avoir toutes les propriétés d'un général » n'est pas une propriété de général —, paradoxes causés pas l'*imprédicativité* de certaines propriétés (voir [Gochet et Gribomont, 1994] par exemple).

D'une certaine manière, les SC constituent une variation sur la théorie des ensembles qui doit bien entendu, si ce n'est éviter ces difficultés, du moins les résoudre. Dans cet ordre d'idées, Dionne *et al.* (1992) proposent une interprétation des circularités dans une théorie d'ensembles non fondés (c'est-à-dire sans l'*axiome de fondation*).

La *réification* des classes, par l'introduction de *méta-classes*, ou classes de classes, conduit au même genre de difficultés. Une première idée conduit à introduire une *stratification* analogue à la *théorie des types* de Russel, avec une partition des objets en objets primitifs, classes d'objets, classes de classes, etc. Avec une telle stratification, une classe a le niveau immédiatement supérieur de de celui de ses instances, qui sont toutes de même niveau ; en corollaire, la spécialisation ne peut relier que des classes de même niveau :

$$o \in Ext(\alpha) \Longrightarrow level(\alpha) = level(o) + 1 \tag{2.28}$$

$$\alpha \prec \beta \Longrightarrow level(\alpha) = level(\beta) \tag{2.29}$$

La difficulté est double. D'une part, l'intuition tend à considérer les méta-classes comme des classes particulières ce qui conduit à :

$$o \in Ext(\alpha) \Longrightarrow level(\alpha) > level(o) \tag{2.30}$$

$$\alpha \prec \beta \Longrightarrow level(\alpha) \geq level(\beta) \tag{2.31}$$

en posant deux conditions :

$$Isa(o) = Isa(o') \Longrightarrow level(o) = level(o') \tag{2.32}$$

22. Dont l'antinomie découle de l'impossibilité que $E \supseteq 2^E$, par le théorème de Cantor.

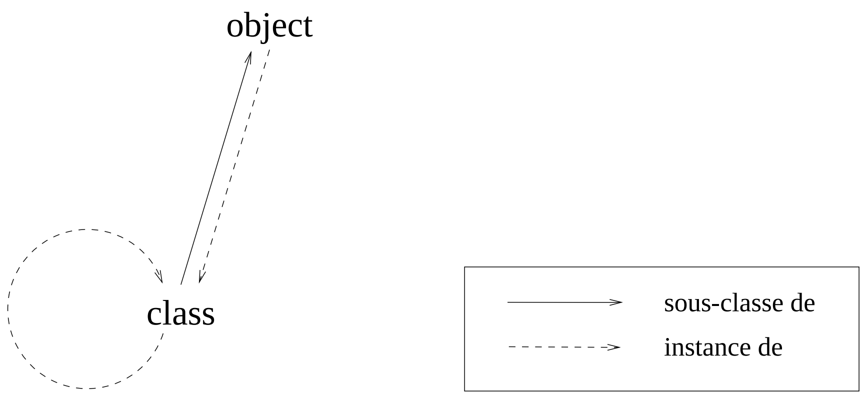

FIG. 2.3 – *Le noyau d'*OBJVLISP

$$\forall \alpha, \exists o \in Ext(\alpha) : level(\alpha) = level(o) + 1 \tag{2.33}$$

D'autre part, singulièrement dans les LPO, mais aussi de façon moins visible dans les autres SC, c'est la classe qui crée l'instance, ce qui conduit à une induction en sens inverse des niveaux, c'est-à-dire selon un ordre non inductif : il faudrait en fait réécrire (2.28) sous la forme :

$$o \in Ext(\alpha) \Longrightarrow level(o) = level(\alpha) - 1 \tag{2.34}$$

ce qui suppose de fixer à l'avance le niveau maximum d'un SC.

Dans la pratique, les langages ont replié sur elle-même la racine des classes, avec comme dans le modèle OBJVLISP [Cointe, 1987], une classe `Class` instance d'elle-même (figure 2.3).

La stratification reste possible, mais soit elle est finie et instable — il faut faire des cas particuliers pour `Class` et l'ajout de nouvelles méta-classes de niveau supérieur peut obliger à augmenter le niveau de classes préexistantes — soit elle est infinie, ce qui oblige à introduire un niveau infini ω[23] et des *ω-classes* comme le font Motschnig-Pitrik et Mylopoulos (1992) pour TELOS[24]. On peut alors espérer avoir :

$$o \in Ext(\alpha) \Longrightarrow \begin{cases} level(o) = level(\alpha) - 1 \\ \quad \text{ou bien} \\ level(\alpha) = \omega \end{cases} \tag{2.35}$$

$$\alpha \prec \beta \Longrightarrow \begin{cases} level(\alpha) = level(\beta) \\ \quad \text{ou bien} \\ level(\beta) = \omega \end{cases} \tag{2.36}$$

On constate que le niveau d'une instance est fixée par sa classe sauf si celle-ci est une ω-classe : il faut alors le fixer à l'instanciation.

23. ω est le premier ordinal infini (voir [Russel, 1921] ou [Gochet et Gribomont, 1994] pour une introduction sympathique, à tendance respectivement philosophique ou mathématique). Le problème est que l'addition n'est plus commutative pour les ordinaux infinis : $\omega + 1$ (l'ordinal suivant) n'est pas égal à ω (contrairement à $1 + \omega$) ce qui rend difficile, malgré le passage à l'infini, de conserver (2.28).

24. À ne pas confondre avec le noyau objet de ILOGTALK [ILO, 1995].

Dans tous les cas, la circularité initiale de la racine des classes rend impossible une interprétation ensembliste intuitive dans une théorie des ensembles classique, c'est-à-dire avec l'axiome de fondation : il existe des chaînes infinies de la relation d'appartenance censée interpréter la relation d'instanciation [25].

2.4.3 Intension sans attributs, ou sans extension

Malgré notre parti pris de généralité, certains systèmes qui relèvent manifestement de la problématique de la classification ne rentrent pas, ou mal, dans notre cadre.

Tous les systèmes classificatoires ont un cadre intensionnel, mais il ne s'exprime pas forcément en termes d'attributs : ce n'est pas le cas, par exemple, du système de classification de molécules chimiques suivant les sous-structures qu'elles contiennent, décrit dans [Napoli *et al.*, 1994; Vismara, 1996] et dont nous reparlerons (cf. paragraphe 7.5.1, page 173).

Inversement, il existe des systèmes classificatoires sans interprétation extensionnelle évidente : les systèmes à prototypes qui ne distinguent pas les classes des instances en sont un premier exemple, informatique. La biologie, qui fournit déjà le prototype des systèmes classificatoires avec la classification des espèces, offre aussi un prototype de système classificatoire sans interprétation extensionnelle : l'*arbre phylogénétique* de l'évolution des espèces (voir par exemple [Caraux *et al.*, 1995]), qui pose *a priori* les mêmes problèmes de classification — insérer une nouvelle espèce (en l'occurrence une séquence d'ADN ou ARN) dans un arbre phylogénétique — que ceux que nous posons ici [26]. Il est particulièrement notable que l'*héritage* des langages d'objets est une métaphore empruntée à la version *ontogénétique* — au niveau des instances — de l'évolution des espèces.

Ces deux paradigmes [27] classificatoires en biologie sont à rapprocher du couple *métaphore-métonymie* que le philosophe Patrick Tort met à la base de la classification [Tort, 1989], et ce dernier du couple *spécialisation-composition* que les méthodologies objet mettent souvent à la base de la conception (voir aussi [Napoli, 1992; Magnan, 1994]). La composition a son équivalent réthorique dans la *synecdoque* [28] — la partie pour le tout, ou vice-versa — qui est un cas particulier de *métonymie*, laquelle ratisse plus large puisqu'elle vise aussi bien la contiguïté temporelle (l'évolution ou la

25. F-LOGIC utilise lui aussi une relation d'appartenance non *bien fondée* pour exprimer par $u \in u$ le fait que u est un représentant typique de lui-même [Kifer *et al.*, 1993] (note 7, page 15), cf. paragraphe 7.3.2, page 171

26. Ce contre-exemple est quand même à pondérer : chaque séquence génétique peut s'interpréter par l'ensemble des animaux (ou des espèces) qui ont possédé — modulo les variations individuelles — cette séquence. Mais cette interprétation ne vérifie pas l'inclusion des extensions de la formule (2.3). Il faut donc interpréter chaque séquence comme l'ensemble des animaux (ou espèces) qui en sont issu(e)s. Mais bien sûr, rien n'empêche (théoriquement, et hors question de calculabilité) de considérer l'ADN complet d'un animal vivant, dont l'extension ne peut être que lui-même. Ce qui nous ramène très exactement aux prototypes.

27. Le mot *paradigme* exprime remarquablement bien ces deux paradigmes : le français lui donne plutôt le sens de classe (d'équivalence), alors que son sens anglais serait plutôt celui de prototype ! Faut-il y voir une différence culturelle : cartésianisme contre pragmatisme ?

28. *Métaphore, métonymie* et *synecdoque* sont des figures de réthorique, c'est-à-dire des *tropes* ! (cf. paragraphe 7.2.3, page 168)

généalogie), que la contiguïté spatiale ou fonctionnelle (en particulier la composition). Enfin, les notions de versions et d'évolution qui sont des sujets actifs de recherche dans la communauté objet ([Carré, 1989] par exemple), que ce soit au niveau ontogénétique ou au niveau phylogénétique, sont, elles aussi, proches de cette problématique classificatoire et généalogique de la biologie.

NOTE *2.21* *Ontogénèse et phylogénèse*

Rappelons que l'*ontogénèse* désigne la naissance et l'évolution des « êtres » — c'est-à-dire des individus, platoniciens s'abstenir — alors que la *phylogénèse* désigne la naissance et l'évolution des espèces. Le couple phylogénèse/ontogénèse reproduit donc fidèlement l'opposition classe/instance. L'une des grandes métaphores de l'histoire naturelle est que « l'ontogénèse reproduit la phylogénèse » : cette métaphore doit se comprendre par le fait que l'évolution de l'embryon reproduit les phases embryonnaires des espèces antérieures. La méthode `initialize-instance` des langages d'objets comme CLOS [Keene, 1989; Steele, 1990] ne fait pas autre chose, quand la combinaison des méthodes la fait passer par toutes les étapes d'initialisation correspondant aux diverse super-classes de la classe considérée.

Inversement, si la phylogénèse des espèces naturelles se conçoit de façon continue comme la succession d'innombrables ontogénèses particulières, la phylogénèse des SC repose soit sur un processus de *catégorisation* qui abstrait les classes à partir d'un ensemble d'instances (ce qui n'est pas si différent), soit sur une ontogénèse particulière dans les système réflexifs où la classe n'est qu'une instance particulière.

Les systèmes classificatoires ont ainsi doublement à voir avec les sciences naturelles : ce sont leurs premiers exemples historiques et elles en offrent l'ultime métaphore.

2.4.4 Pourquoi « systèmes classificatoires » ?

La théorie des systèmes classificatoires (SC) d'Euzenat (1993a) repose sur un *critère de catégorisation* général, non précisé, qui doit s'interpréter, de façon abstraite comme un prédicat, et qui représente donc l'inclusion des intensions des classes ($\trianglerighteq$). Le problème est que ce critère abstrait doit s'instancier dans chaque système classificatoire sous une forme plus spécifique pour éviter les problèmes d'imprédicativité. C'est ce que nous avons fait en posant que l'intension s'exprime au travers des domaines des attributs des classes.

Pourquoi avoir repris ce terme de « système classificatoire » ? Qu'il s'applique à notre propos est incontestable, mais il est tout autant incontestable que la notion d'Euzenat est plus générale. D'autres dénominations auraient pu être plus adéquates, mais déjà utilisées : « langage d'attribut » (*attributive language*) renvoie par exemple à $\mathcal{AL}$ (cf. annexe B.3) et aux logiques terminologiques. En fait, la plupart des approches — logiques terminologiques, structures attributives typées (*typed feature structure*) [Carpenter, 1992], ψ-termes — ont tendance à partir des attributs et s'appliquent difficilement aux « objets » tels que les pratiquent LPO et RCO. Nous verrons en particulier que ce qualificatif de « *attributive language* » est tout à fait adéquat pour les LT, mais qu'elles devraient pouvoir prétendre à être de véritables « langages de classes et d'attributs » (cf. paragraphe 4.3.3, page 81).

Chapitre 3

Propriétés spécifiques des systèmes classificatoires

Avoir toutes les propriétés d'un système classificatoire
n'est pas une propriété d'un système classificatoire
d'après Bertrand Russel.

Après avoir présenté un cadre général , nous nous intéressons ici aux propriétés formelles que l'on peut, soit vouloir imposer, soit chercher à démontrer, pour un SC donné. La justification de ces propriétés incombe bien entendu aux concepteurs du SC concerné, qui jugent utile de les imposer.

Nous ne considérons pas ici les propriétés *a posteriori* de calculabilité — décidabilité et complexité des problèmes, complétude et complexité des algorithmes, etc. —, ce que nous ferons au chapitre 5, mais uniquement les propriétés *a priori* ayant une influence potentielle sur la sémantique du SC.

3.1 Propriétés structurelles

3.1.1 Rapports entre spécialisation et subsomptions

La problématique trouve d'abord ses degrés de libertés dans les rapports existant entre la relation de spécialisation (ordre $\preceq$) et les subsomptions extensionnelle (préordre $\trianglelefteq$) et intensionnelle ($\trianglerighteq$) (ou éventuellement les ordres quotients qu'ils induisent, en particulier si les circuits de ces préordres sont inclus dans les chaînes de $\preceq$), rapports qu'il faudrait examiner en détail et dont nous ne faisons qu'énumérer les combinaisons possibles. Ces trois relations peuvent avoir les rapports suivants :

(3.1) $\preceq \;\subseteq\; \trianglelefteq \;\cap\; \trianglerighteq$: cas général

(3.2) $\preceq \;\subseteq\; \trianglelefteq \;\subseteq\; \trianglerighteq$

(3.3) $\preceq \;\subseteq\; \trianglerighteq \;\subseteq\; \trianglelefteq$: ce sont les *classes définies*

(3.4) $\preceq \;=\; \trianglelefteq \;\cap\; \trianglerighteq$

(3.5) $\preceq \;=\; \trianglelefteq \;\subseteq\; \trianglerighteq$: *hiérarchie extensionnelle*

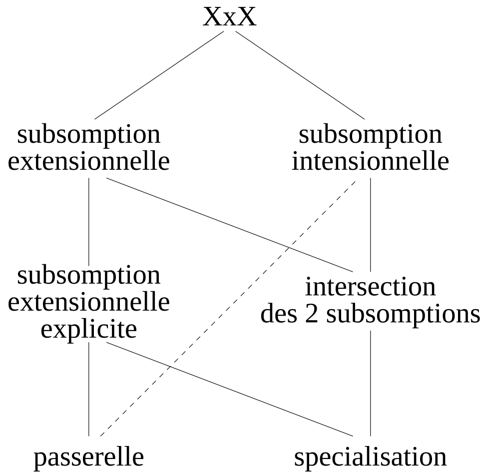

FIG. 3.1 – *Spécialisation, subsomptions et passerelles : inclusions et disjonctions dans le cas général.*

(3.6) $\preceq \;=\; \trianglerighteq \;\subseteq\; \trianglelefteq$: *hiérarchie intensionnelle*

(3.7) $\preceq \;\subseteq\; \trianglelefteq \;=\; \trianglerighteq$ $\qquad (= 3.2 + 3.3)$

(3.8) $\preceq \;=\; \trianglerighteq \;=\; \trianglelefteq$ $\qquad (= 3.5 + 3.6 + 3.7)$

(3.5) (resp. (3.6)) est un cas particulier de (3.2) (resp. (3.3)). (3.6) impose la *classification automatique des classes*. Ces comparaisons directes paraissent difficiles à exploiter.

Le cas 3.4 présente une discussion intéressante : si $\alpha \trianglelefteq \beta$ et $\alpha \trianglerighteq \beta$, on ne voit pas bien pourquoi β ne serait pas une sous-classe de α, si bien sûr cela n'introduit pas de circuit dans $\prec$. Lors de l'introduction de α, la subsomption intensionnelle par β peut être supposée consciente pour l'utilisateur (sinon, le système pourrait la lui signaler). Il est alors raisonnable d'exiger de l'utilisateur, soit qu'il affirme la non subsomption extensionnelle de α par β, soit qu'il fasse de la première une spécialisation de la seconde. Il paraît donc raisonnable de supposer que $\preceq = \trianglelefteq \cap \trianglerighteq$, modulo les circuits de $\trianglelefteq \cap \trianglerighteq$ qui seraient inclus dans des chaînes de $\prec$. Inversement, il faut bien supposer que chaque arc de $\trianglerighteq - \preceq$ n'appartient pas à $\trianglelefteq$. En suivant cet argument, on supposera aussi que $\triangleleft_a \cap \trianglerighteq \; = \emptyset$: si l'on veut que $\alpha \trianglelefteq \beta$ alors que $\alpha \trianglerighteq \beta$, autant dire tout de suite que α est une sous-classe de β.

Cet argument s'applique aussi au cas 3.2 (resp. 3.3), en lui préférant le 3.5 (resp. 3.6). C'est cet argument qui est implicitement à la base de la classification automatique des classes dans les LT : si la subsomption extensionnelle est calculable, autant la calculer et en faire la relation de spécialisation. On y perd juste la spécificité des passerelles, qui sont alors traitées à part, comme par les *règles* de CLASSIC.

3.1.2 La structure d'ordre de la hiérarchie

Les propriétés graphiques des relations $\preceq$, $\trianglelefteq$ et $\trianglerighteq$ peuvent elles-aussi être questionnées : sont-ce des préordres ou des ordres, voire des demi-treillis ou treillis, ou simplement des arbres ? Sur toutes ces notions d'ordre, voir l'annexe A.

A la limite, même si $\preceq$ était aussi un préordre, comme $\trianglelefteq$ et $\trianglerighteq$, les propriétés graphiques pertinentes sont celles de l'ordre quotient induit par ces préordres.

De façon purement graphique, $(\mathcal{X}, \preceq)$ peut être :

une arborescence : il s'agit de ce que l'on appelle habituellement *héritage simple*. (NB. On ne considère alors pas $\perp$).

un treillis (ou demi-treillis inférieur) : la structure d'ordre est alors caractérisée par l'existence (et l'unicité) d'une borne inférieure (et supérieure) pour tout couple de sommets (cf. annexe A.3). $\wedge$ (resp. $\vee$) désigne la borne inférieure (resp. supérieure) de ce treillis.

un ordre partiel quelconque : il s'agit dans ce cas d'*héritage multiple* ordinaire.

3.1.3 Propriétés extensionnelles

Mais souvent tel objet, qui par une ou plusieurs de ses propriétés
a été placé dans une classe, tient à une autre classe par d'autres propriétés [...].
d'Alembert, *Discours préliminaire* à l'Encyclopédie, cité par Eco (1994).

Dans sa théorie abstraite des SC, Euzenat (1993a) propose de les caractériser par quelques propriétés structurelles. On peut qualifier ces propriétés d'*extensionnelles* dans la mesure où elles concernent essentiellement les extensions des classes, même si certaines ont des conséquences sur la structure d'ordre de la hiérarchie. Il s'agit de propriétés possibles de ces systèmes, qui peuvent servir à caractériser telle ou telle famille, en aucun cas de propriétés nécessaires.

exhaustivité : tout objet est instance d'un (de) *puits* de la relation $\preceq$ ($\perp$ mis à part) :

$$\underline{Isa}(o) \subseteq \min_{\prec}(\mathcal{X} - \perp). \tag{3.9}$$

C'est le cas des *espèces naturelles* : il n'existe pas de pur `mammifère`, qui ne soit pas membre d'une sous-catégorie, `chien` ou `chat`.

univocité : tout objet a une unique classe minimale,

$$|\underline{Isa}(o)| = 1 \tag{3.10}$$

C'est la *mono-instanciation* des systèmes d'objets habituels.

exclusivité : deux classes non comparables ont des extensions disjointes :

$$\alpha \not\preceq \beta \ \&\ \beta \not\preceq \alpha \Longrightarrow Ext(\alpha) \cap Ext(\beta) = \emptyset \tag{3.11}$$

L'exclusivité impose bien sûr l'identité entre $\preceq$ et $\trianglelefteq$: c'est le cas de la formule (3.5)[1]. Elle implique aussi l'univocité.

1. Modulo les circuits de $\trianglelefteq$ qui peuvent être inclus dans les chaînes de $\preceq$.

L'exclusivité fournit ainsi une relation d'incompatibilité implicite binaire et totale au sens où elle forme avec $\preceq$ une relation totale :

$$\nabla \cup \preceq \ = \mathcal{X} \times \mathcal{X}.$$

C'est la plus grande relation d'incompatibilité possible. L'univocité fournit aussi une relation d'incompatibilité implicite : tout ensemble de classes sans sous-classe commune est incompatible. L'incompatibilité est complète au sens où elle est entièrement déterminée par la propriété : tout nouveau couple d'incompatibles est soit redondant, soit contradictoire (au sens où l'extension d'une classe deviendrait vide). L'incompatibilité affirmée (∇_a) devient inutile.

Une propriété graphique — par exemple, le fait que $(\mathcal{X}, \preceq)$ soit un treillis — ne peut suffire à entraîner une propriété extensionnelle, par exemple l'univocité. Seul l'inverse est possible. Ainsi, l'exclusivité implique une hiérarchie arborescente, à condition du moins de ne pas considérer les classes d'extension vide ($\perp$). En fait, l'*exclusivité* est équivalente à l'*univocité* plus l'*arborescence*.

A priori, tout SC se complète de façon naturelle en un SC exhaustif et univoque, sans que sa « signification » en soit modifiée pour autant : il suffit de rajouter les classes d'équivalence de la relation « avoir exactement les mêmes classes » lorsque ces classes d'équivalence ne sont pas extensionnellement équivalentes à des classes déjà existantes. Le problème est que l'intension des classes rajoutées peut ne pas être significative. Exhaustivité et univocité n'ont donc pas, isolément, d'influence sur la sémantique du SC : ce ne sont que des propriétés formelles. La suite montrera néanmoins que leur association avec des propriétés plus sémantiques leur donne une sémantique propre.

NOTE *3.1* *Classification et classicisme*

Les termes d'exhaustivité et d'exclusivité doivent se comprendre relativement aux sous-classes directes d'une classe : elles sont mutuellement exclusives et constituent collectivement une sous-classification exhaustive de (l'extension de) la classe. Hautamäki (1986) reprend ces termes de [Johnson, 1964] en considérant les termes comme classiques et les propriétés comme classiquement associées à la notion de classification. Les classifications d'espèces naturelles sont effectivement univoques, exhaustives et exclusives.

NOTE *3.2* *Propriétés de la sémantique ou de la déduction ?*

Euzenat attribue ces propriétés au processus de classification plutôt qu'au système classificatoire lui-même : cela revient à caractériser le système de déduction (chapitre 5), ce qui est assez légitime, mais en faisant d'abord l'hypothèse implicite de ces propriétés pour le système classificatoire lui-même, c'est-à-dire pour sa sémantique (chapitre 4).

Exclusivité et univocité

Il est tentant de chercher à affaiblir l'exclusivité pour autoriser l'héritage multiple en ne rendant incompatibles que deux classes sans sous-classe commune (d'extension non vide) :

$$Ext(\alpha) \cap Ext(\beta) \neq \emptyset \iff \exists \gamma \neq \perp : \gamma \preceq \alpha \ \& \ \gamma \preceq \beta \tag{3.12}$$

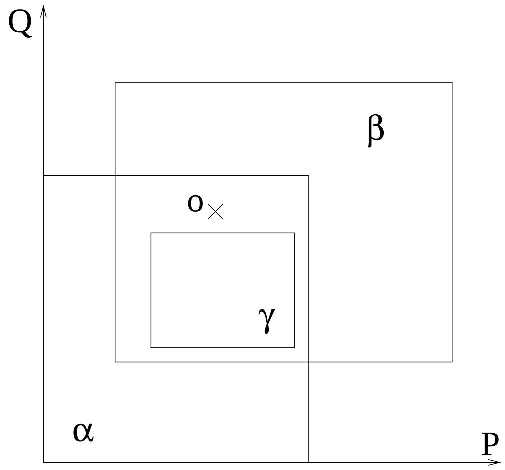

FIG. 3.2 – *L'exclusivité faible sans univocité : l'existence de o implique celle de γ, sans qu'il y ait de rapport direct entre les deux.*

L'univocité implique l'exclusivité faible : en fait, c'en est une version un peu plus forte où l'intersection des extensions est recouverte par les extensions d'un ensemble de sous-classes communes :

$$\forall\alpha,\beta : \exists\Gamma \subset \mathcal{X} : \begin{cases} \forall\gamma \in \Gamma : \gamma \preceq \alpha \ \& \ \gamma \preceq \beta \\ Ext(\alpha) \cap Ext(\beta) = \bigcup_{\gamma\in\Gamma} Ext(\gamma) \end{cases} \tag{3.13}$$

C'est d'ailleurs l'exclusivité faible qui donne à l'univocité sa relation d'incompatibilité implicite et complète. Mais l'exclusivité faible sans univocité présente clairement peu d'intérêt : essayons d'imaginer que $o \in Ext(\alpha) \cap Ext(\beta)$ sans que $o \in Ext(\gamma)$ (figure 3.2) !

3.2 Histoires de treillis

L'ubiquité de la structure de treillis dans les SC tient à la proximité du treillis ensembliste des extensions ($\cap$ et $\cup$) d'une part, du treillis syntaxique des intensions ($\sqcap$ et $\sqcup$) d'autre part, même si ces deux treillis restent virtuels.

3.2.1 La structure d'ordre des extensions et intensions

La structure d'ordre de la hiérarchie n'est rien si elle n'a pas de correspondant au niveau des ensembles associés, extensions ou intensions. Ainsi, une propriété fondamentale est celle de la fermeture de $\mathcal{X}$ par intersection des extensions.

$\mathcal{X}$ est *fermé par intersection des extensions* ssi :

$$\forall\alpha,\beta \in \mathcal{X}, \exists\gamma \in \mathcal{X} : Ext(\gamma) = Ext(\alpha) \cap Ext(\beta). \tag{3.14}$$

Cette propriété implique que l'ordre induit par $\trianglelefteq$ est un demi-treillis inférieur, mais la réciproque ne tient pas. Elle implique aussi l'*univocité*. La propriété de fermeture de $\mathcal{X}$ par union des intensions se définit de façon analogue.

En toute généralité :

- le graphe $(\mathcal{X}, \preceq)$ est un ordre partiel quelconque, et les extensions n'ont aucune propriété particulière ;
- l'ensemble des extensions, ordonné par $\subseteq$, forme un sous-graphe du treillis de sa fermeture par union et intersection ;
- l'ensemble des intensions, muni de la relation $\sqsupseteq$, constitue un sous-graphe du treillis complet obtenu par la combinatoire de tous les attributs munis des parties de $\mathcal{D}$ comme domaine (cf. 3.2.2).

Lorsque la hiérarchie est un (demi)treillis [2], on peut considérer plusieurs couplages successifs entre ces trois niveaux :

treillis extensionnel : $\mathcal{X}$ est fermé par intersection des extensions et $(\mathcal{X}, \preceq)$ est un treillis, ce qui assure que

(3.15) $$Ext(\alpha \wedge \beta) = Ext(\alpha) \cap Ext(\beta).$$

Si o est une instance de α et β, c'est une instance de $\alpha \wedge \beta$. Un treillis extensionnel implique l'*univocité*. En fait un treillis qui ne serait pas extensionnel n'aurait pas plus de sens que l'exclusivité faible sans univocité. Plus précisément, un treillis extensionnel est exactement la conjonction d'un treillis et de l'univocité. De même, l'exclusivité extensionnelle est exactement la conjonction d'un treillis extensionnel et d'une arborescence.

Cette propriété peut nécessiter de quotienter $\mathcal{X}$ par les circuits de $\trianglelefteq$ pour réduire les classes extensionnellement équivalentes. Une fois ce quotient effectué, $\preceq$ et $\trianglelefteq$ coïncident.

treillis intensionnel : $\mathcal{X}$ est fermé par union des intensions et $(\mathcal{X}, \preceq)$ est un treillis, ce qui assure que

(3.16) $$Int(\alpha \wedge \beta) = Int(\alpha) \sqcup Int(\beta).$$

Un treillis intensionnel correspond en fait à la présence d'un opérateur de *conjonction* (`and`) et à la fermeture de la hiérarchie par cet opérateur. Il implique l'égalité de $\preceq$ et $\trianglerighteq$, modulo les circuits de ce dernier.

treillis classificatoire : c'est la conjonction des treillis intensionnel et extensionnel. Dans un treillis classificatoire, le fait que o soit à la fois un α et un β ne lui donne pas plus de propriété que celles dues au fait qu'il est un α et un β. $\alpha \wedge \beta$ est en fait une classe *définie* par une CNS à partir de α et β. L'opérateur de conjonction est alors un *constructeur*.

Dans un simple treillis extensionnel, $\alpha \wedge \beta$ pourrait avoir un attribut que ni α ni β ne possèdent, ou ne pas avoir dans un domaine une valeur présente à la fois dans les domaines de α et β.

2. Seules les propriétés du demi-treillis inférieur sont explicitées par la suite.

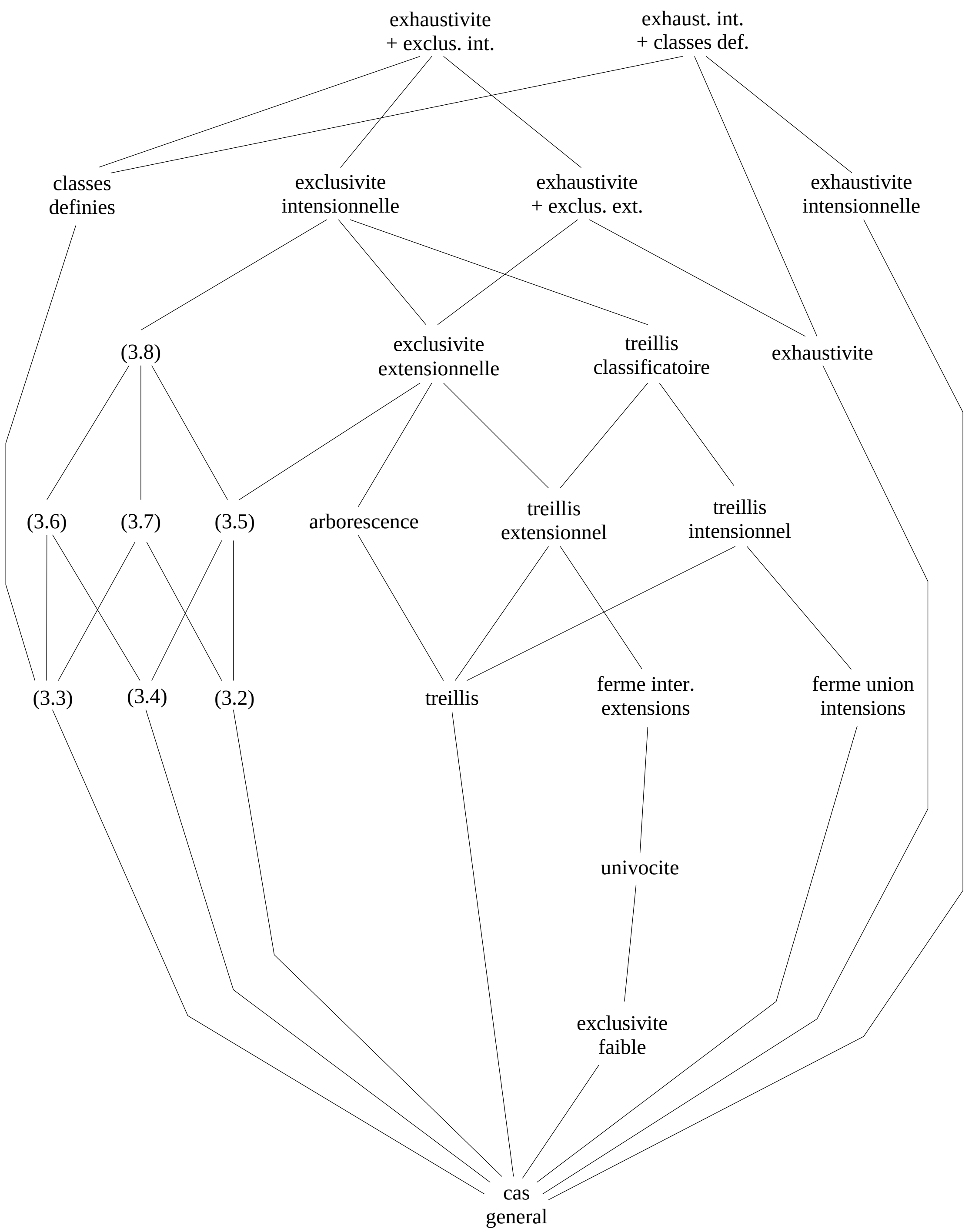

FIG. 3.3 – *Le graphe d'implication de toutes les propriétés globales des* SC.

Un treillis classificatoire devrait entraîner l'égalité des trois relations, modulo les circuits des préordres, mais comme ces circuits n'ont aucune raison de coïncider, ce n'est sans doute pas si simple. Un treillis classificatoire assure la commutativité du double diagramme suivant :

$$\begin{array}{ccccc} (\mathcal{P} \to 2^{\mathcal{D}}) & \xleftarrow{Int} & \mathcal{X} & \xrightarrow{Ext} & 2^{\Omega} \\ \sqcup\big\downarrow & & \wedge\big\downarrow & & \big\downarrow\cap \\ (\mathcal{P} \to 2^{\mathcal{D}}) & \xleftarrow[Int]{} & \mathcal{X} & \xrightarrow[Ext]{} & 2^{\Omega} \end{array}$$

NOTE *3.3* *Treillis et demi-treillis*

Compte-tenu de l'existence de $\top$ et de $\perp$, il n'y a pas lieu de distinguer, pour la hiérarchie, entre treillis et demi-treillis et le cas de l'arborescence n'est qu'un cas particulier de treillis. Cependant, dans les propriétés précédentes, on n'a considéré que des demi-treillis inférieurs, mais les propriétés duales de fermeture par intersection ou union des extensions (resp. intensions) sont indépendantes, ce qui maintient la distinction entre demi-treillis extensionnels (resp. intensionnels) inférieurs et supérieurs et donc entre treillis et demi-treillis. Le demi-treillis extensionnel (resp. intensionnel) supérieur qui s'interprète par une union des extensions (resp. disjonction des intensions) présente plusieurs difficultés et est moins souvent utile.

NOTE *3.4* *Treillis de Galois*

Les treillis extensionnel et intensionnel ont beaucoup à voir avec les treillis de Galois (cf. annexe A.3.3). Leurs relations devraient être précisées. Il s'agit en fait de voir dans quelles conditions la hiérarchie d'un SC est le treillis de Galois d'une relation, à déterminer, entre l'intension et l'extension des classes. Cette relation pourrait être $P \in Attr(o)$, $P \in \mathcal{P}_\alpha$, ou mieux $(P, E) \subseteq Int(\alpha)$, où E est une partie de $\mathcal{D}$. C'est un problème à creuser.

On peut associer à tout SC le treillis de Galois de diverses relations, par exemple celui de la relation R de $\Omega \times \mathcal{P}$, oRP ssi $P \in Attr(o)$, sans tenir compte du domaine des attributs. Avec le domaine, il faut considérer un ensemble plus complexe que $\mathcal{P}$, par exemple l'ensemble des idéaux de $\mathcal{ATTR}$ (cf. paragraphe 3.2.2, page 44 : à vérifier). La question est de savoir quels rapports ce treillis de Galois entretient avec les autres relations ($\prec$ en particulier), ou de voir dans quelle mesure la hiérarchie elle-même peut être assimilée à ce treillis de Galois.

Les treillis de Galois — souvent appelés *treillis de concepts* [Wille, 1982; Wille, 1992], cf. paragraphe 7.4.1, page 172 — s'interprètent naturellement par un couple intension / extension (respectivement E et F). Voir [Godin *et al.*, 1995] pour leurs divers usages conceptuels, en particulier dans la problématique de réorganisation de hiérarchies [Dicky *et al.*, 1994; Dicky *et al.*, 1996], ainsi que pour les diverses façons de les présenter ou de les élaguer.

3.2.2 Les treillis de types induits

Il est possible d'extraire d'un système classificatoire différentes notions de type qui toutes expriment une certaine notion d'équivalence et constituent un treillis. Bien

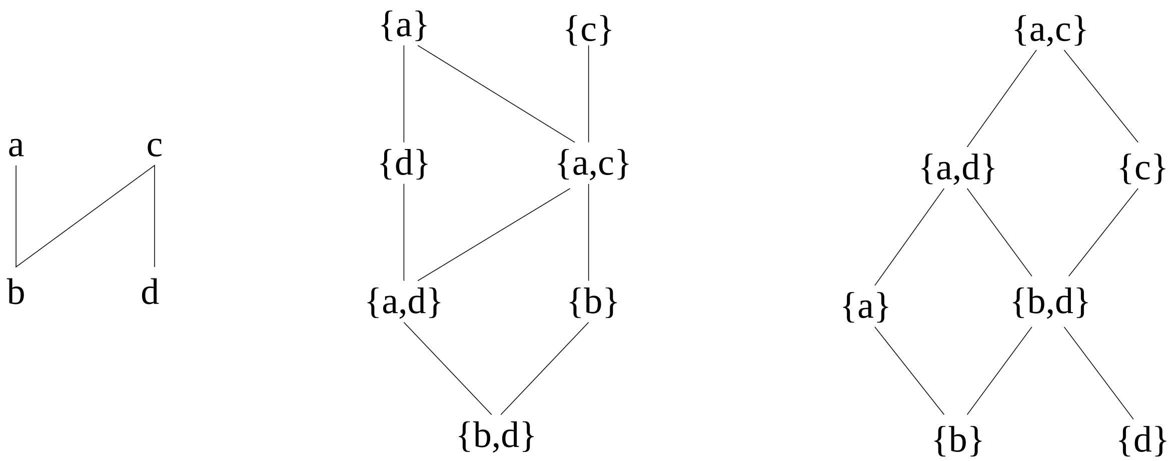

FIG. 3.4 – *La hiérarchie de classes (gauche) s'étend en un treillis de types conjonctif (milieu) ou disjonctif (droite) suivant que l'interprétation est équivoque ou univoque.*

entendu, la hiérarchie se plonge par un morphisme d'ordre dans chacun de ces treillis de types.

Treillis de types intensionnel

On a remarqué que l'ensemble des classes d'un objet est un idéal (cf. annexe A.2.3). Lorsque le SC est univoque, il n'y a pas grand chose à tirer de cette propriété parce que chaque objet a une unique classe minimale — l'idéal est alors *principal* — et que l'ordre d'inclusion des idéaux principaux est isomorphe à l'ordre d'origine.

En revanche, si le SC n'est pas univoque, tout idéal peut être un ensemble de classes d'un objet (à condition que l'idéal ne contienne pas un élément de la relation d'incompatibilité ∇) : les « vraies » classes du SC, au sens de classes d'équivalence de la relation « avoir exactement les mêmes classes » — ce que l'on pourrait appeler un *type* [3] —, sont les idéaux de la hiérarchie. Or l'ensemble des idéaux d'un ordre a une structure de treillis distributif (cf. annexe A.3.1).

On transforme ainsi facilement un SC non univoque en un SC univoque en prenant le treillis des idéaux et en enlevant tous les idéaux contenant un élément de ∇ que l'on fusionne en $\perp$. L'avantage est que ce SC univoque équivalent est un treillis distributif[4] et classificatoire, au sens que nous lui avons donné plus haut, alors qu'un SC univoque n'a même pas forcément une structure de treillis.

Carpenter (1992) construit ainsi un treillis de types à partir d'une hiérarchie d'héritage — sous la forme d'un réseau d'héritage à la [Touretzky, 1986], qui constitue une version élémentaire de SC (cf. paragraphe 7.7.4, page 176). Mais Carpenter propose aussi un treillis analogue dans le cas d'une hiérarchie d'héritage univoque (figure 3.4).

Dans les deux cas, chaque « type » est caractérisé par une antichaîne de $(\mathcal{X}, \prec)$ mais le premier treillis est *conjonctif* dans la mesure où chaque type représente une

3. Est-ce que ces types et ces idéaux sont les mêmes que ceux de [Scott, 1976], c'est une autre histoire.

4. Il est distributif si l'on ne tient pas compte de l'incompatibilité : lorsque l'on fusionne dans $\perp$ toutes les « classes » incompatibles, la distributivité est perdue.

conjonction des classes de l'antichaîne, alors que le deuxième treillis est *disjonctif* car chaque type représente une disjonction des classes de l'antichaîne.

Treillis de types intensionnel

Étant donné un système classificatoire, c'est-à-dire l'ensemble des classes munies de leurs intensions, considérons l'ensemble des descriptions élémentaires d'attributs

$$\mathcal{ATTR} \stackrel{\text{def}}{=} \{(P, Dom(\alpha, P)) \mid \alpha \in \mathcal{X}, P \in \mathcal{P}_\alpha\} \tag{3.17}$$

ordonné par la relation $\sqsupseteq$ des intensions :

$$(P, Dom(\alpha, P)) \sqsupseteq (Q, Dom(\beta, Q)) \stackrel{\text{def}}{\Longleftrightarrow} \begin{cases} P = Q \\ Dom(\alpha, P) \subseteq Dom(\beta, Q) \end{cases} \tag{3.18}$$

en particulier si $\alpha \preceq \beta$.

Ce graphe n'est pas connexe (pas de relation entre deux attributs distincts) et chaque classe de $\mathcal{X}$ peut être vue comme une antichaîne de $(\mathcal{ATTR}, \sqsupseteq)$. La réciproque est fausse, car chaque composante connexe n'est pas forcément une chaîne. Cet ensemble se complète naturellement en un demi-treillis inférieur $(\mathcal{INT}, \sqsupseteq)$ de toutes les intensions virtuelles du système classificatoire, par fermeture par l'opérateur d'union des intensions $\sqcup$, qui correspond au constructeur `and` des LT. Cette opération n'est pas sans analogie avec les produits de SC de [Euzenat, 1994].

NOTE *3.5*
On pourrait (et il faudrait) étendre ces définitions aux sous-rôles (cas où $P \preceq Q$) et aux *role-value-map* (cf. paragraphe 4.1.2, page 59).

NOTE *3.6* *Treillis des types et des intensions*
$\mathcal{ATTR}$ équivaut à peu près aux termes élémentaires calculés par la fonction **CUNFOLD** de [Nebel, 1990], et $\mathcal{INT}$ au « treillis des types » de [Capponi, 1993; Capponi, 1995; Euzenat, 1993a; Euzenat, 1994]. Tout SC se plonge dans ce treillis des types associé et ce dernier se plonge lui-même dans le treillis des intensions défini de façon abstraite sur le domaine $\mathcal{D}$. $\mathcal{INT}$ est l'ensemble des fonctions partielles sur $\mathcal{P}$ qui associent à un attribut P un élément de la fermeture par intersection de $\{Dom_P(\alpha) \mid \alpha \in Def(P)\}$, alors que le treillis des intensions est l'ensemble des fonctions partielles de $\mathcal{P}$ dans $2^{\mathcal{D}}$. Le premier dépend des classes de la hiérarchie, alors que le second ne dépend que du domaine et ne représente que l'espace abstrait de toutes les intensions.

3.3 Propriétés d'ordre supérieur

« Libre de toute contrainte » sonne à l'oreille comme un pléonasme, mais, en fait, la contrainte est bien une liberté de second ordre : elle libère du poids des décisions.
W.V. Quine, « Liberté » in [Quine, 1993][5]

Il faut remarquer que ces propriétés des systèmes classificatoires, quelles qu'elles soient, peuvent elles-mêmes être dotées de propriétés d'ordre supérieur.

5. Mais du coup, la liberté n'est-elle pas une contrainte de second ordre ?

3.3.1 Objectivité, relativité

Une propriété est **objective** si elle est formellement établie par construction, et maintenue, par le système. Une propriété est **subjective** si elle repose sur la discipline de l'utilisateur, qui doit la vérifier lui-même, ou construire, manuellement, sa représentation pour qu'elle la vérifie. La correction syntaxique d'un programme édité sous un éditeur purement textuel est une propriété subjective. Elle est par contre objective avec un éditeur structurel. Une propriété peut aussi être *faiblement objective*, dans la mesure où le système peut la vérifier mais pas la maintenir : c'est le cas de la correction syntaxique lorsque l'éditeur textuel est couplé avec un compilateur.

Il est certain qu'une propriété objective est préférable, mais une propriété faiblement objective, voire purement subjective, peut parfois suffire.

D'autre part, comme l'incomplétude des objets peut induire une indétermination partielle de leurs classes, il faut aussi distinguer les propriétés du système classificatoire dans son interprétation **définitive**, de ses propriétés dans sa représentation **provisoire**, donc distinguer les propriétés de Isa, des propriétés de $Isa_{\mathcal{W}}$. Ainsi, si la classification doit être, *in fine*, exhaustive, il peut être nécessaire de savoir considérer des instances de classes non terminales, au moins provisoirement. Si le SC est univoque, il faut un treillis extensionnel pour ne pas avoir à considérer une multi-instanciation provisoire.

Enfin, si la dernière distinction revient à une certaine relativité par rapport au *temps*, il faut aussi envisager une relativité analogue par rapport à l'*espace* : ces propriétés peuvent être **globales** ou seulement **locales** à certaines parties de $\mathcal{X}$. Il faut donc s'intéresser aussi aux propriétés spécifiques des classes.

3.3.2 Propriétés intensionnelles

En règle générale, le meilleur moyen de rendre *objective* une propriété extensionnelle est d'imposer un analogue intensionnel forcément plus fort. Si cela marche bien pour l'exclusivité, c'est moins clair pour l'exhaustivité, ou pour l'univocité.

Exclusivité intensionnelle

L'exclusivité a la particularité d'avoir aussi une version intensionnelle car sa définition repose sur celle de l'incompatibilité,

$$\alpha \not\preceq \beta \ \& \ \beta \not\preceq \alpha \Longrightarrow \alpha \nabla \beta. \tag{3.19}$$

Elle offre donc le choix entre les définitions extensionnelle ou intensionnelle de ∇. Dans l'*exclusivité intensionnelle*, les sous-classes doivent être deux à deux *intensionnellement incompatibles* :

$$\alpha \not\preceq \beta \ \& \ \beta \not\preceq \alpha \Longrightarrow Int(\alpha) \sqcup Int(\beta) = Int(\bot) \tag{3.20}$$

deux sous-classes non comparables doivent donc avoir un même attribut avec des domaines disjoints.

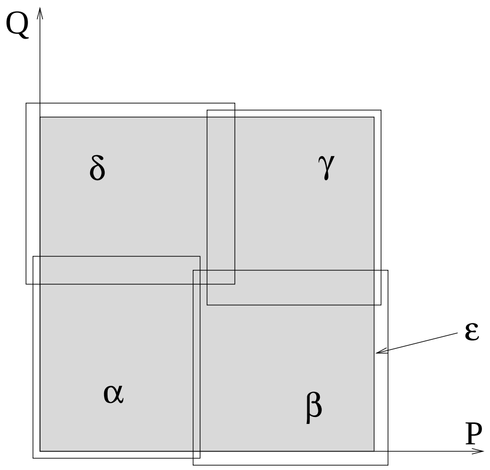

FIG. 3.5 – *Exhaustivité intensionnelle : l'intension de la classe ϵ avec deux attributs* `P` *et* `Q` *est recouverte par celles de ses quatre sous-classes α, β, γ et δ.*

L'exclusivité intensionnelle entraîne l'égalité $\preceq = \trianglelefteq = \trianglerighteq$, modulo les circuits des préordres qui peuvent être inclus dans les chaînes de $\preceq$. Alliée à l'exhaustivité, elle implique aussi des classes définies.

Exhaustivité ou univocité intensionnelles

Une version intensionnelle de l'exhaustivité est aussi envisageable : elle consisterait à imposer que l'intension de chaque classe soit « recouverte » par la réunion des intensions de ses sous-classes (directes), au sens du produit cartésien des domaines (cf. paragraphe 4.3.1, page 76) :

$$\prod_{P \in \mathcal{P}_\alpha} Dom(\alpha, P) = \bigcup_{\beta \prec \alpha} \prod_{P \in \mathcal{P}_\alpha} Dom(\beta, P) \tag{3.21}$$

de sorte que chaque instance de α pourrait aussi bien être une instance d'une sous-classe de α (figure 3.5). Exhaustivité et exclusivité intensionnelles font que ce recouvrement est une partition. L'exhaustivité intensionnelle est une condition suffisante pour que tout objet soit compatible avec l'intension d'au moins une classe terminale, donc pour permettre une exhaustivité extensionnelle sans risque d'*objet paradoxal*. Mais l'exhaustivité intensionnelle ne suffit pas à entraîner l'exhaustivité extensionnelle.

L'analogue intensionnel de l'univocité serait que l'intersection des intensions de deux classes soit recouverte par la réunion des intensions de leurs sous-classes communes. Comme pour l'exhaustivité, cela n'implique pas la propriété extensionnelle, mais assure simplement l'absence d'objets paradoxaux (figure 3.6).

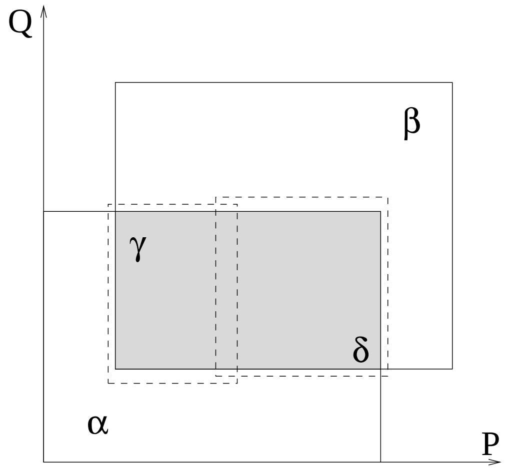

FIG. 3.6 – *Univocité intensionnelle : l'intersection des intensions des classe α et β (avec deux attributs* `P` *et* `Q`*) est recouverte par celles de deux sous-classes γ et δ.*

3.4 Propriétés spécifiques des classes

3.4.1 Propriétés extensionnelles

À chaque propriété extensionnelle globale du SC peut correspondre une propriété locale, soit à une classe, soit à la classe et à toutes ses sous-classes. Ainsi une classe α est *exhaustive* ssi :

$$Ext(\alpha) = \bigcup_{\gamma \prec \alpha} Ext(\gamma). \tag{3.22}$$

ce qui signifie qu'elle n'a pas d'*instance propre*[6]. Une version plus forte imposerait que toutes les sous-classes non terminales soient elles-aussi exhaustives.

Dans la version locale de l'exclusivité, les sous-classes directes de la classe considérée sont alors deux à deux incompatibles.

Pour ce qui est de l'univocité, sa version locale pourrait être que toute instance de la classe n'a qu'une seule classe minimale : les sous-classes d'une classe univoque sont univoques. Dans une version plus faible, les instances d'une classe univoque n'auraient comme classes minimales (<u>*Isa*</u>) que des sous-classes de la classe.

Bien entendu, tout cela nécessite d'augmenter d'autant l'expressivité du SC.

3.4.2 Propriétés intensionnelles

Outre les traductions locales des propriétés intensionnelles globales, la principale propriété intensionnelle d'une classe est son caractère défini ou primitif.

Classes définies ou primitives

Les LT ont introduit et rendu classique la distinction entre classe *définie* et classe *primitive* (ou *descriptive*). La classification d'instance repose sur la comparaison d'un

6. Dans un langage de programmation par objets, ce serait une *classe abstraite*.

objet avec l'intension des classes candidates. On pose d'abord que *o vérifie l'intension de* α, noté $o \sqsubseteq Int(\alpha)$, ssi :

$$\forall P \in \mathcal{P}_\alpha : P(o) \in Dom_P(\alpha) \tag{3.23}$$

alors la formule (2.10) se réécrit :

$$o \in Ext(\alpha) \Longrightarrow o \sqsubseteq Int(\alpha) \tag{3.24}$$

Une classe est dite *définie* lorsque la réciproque tient :

$$o \in Ext(\alpha) \iff o \sqsubseteq Int(\alpha). \tag{3.25}$$

Sinon, elle sera simplement *descriptive* (ou encore *primitive*).

Une classe définie est donc toujours *la classe des objets* qui vérifient son intension — c'est l'usage habituel de *définition en intension* —, alors qu'une classe descriptive n'est qu'*une classe d'objets* qui vérifient son intension.

Dans une classe définie, les attributs expriment une CNS, alors que ce ne sont que de simples CN dans une classe descriptive. Plus exactement, chaque attribut est une condition nécessaire, et, pour les classes définies seulement, tous les attributs constituent, ensemble, une condition suffisante.

Définition et propriétés spécifiques

Le caractère défini des classes a des effets importants sur leurs intensions, ainsi que sur les rapports entre leurs intensions d'une part, entre les subsomptions intensionnelle et extensionnelle d'autres part.

Si β est définie et si $\alpha \trianglerighteq \beta$, alors $\alpha \trianglelefteq \beta$: noter que α n'a pas besoin d'être définie. La restriction (en fait la restriction du codomaine) des subsomptions aux classes définies vérifie le cas (3.3) : $\trianglerighteq \ \subseteq \ \trianglelefteq$. L'inclusion de l'intension d'une classe définie implique l'inclusion dans son extension. De plus, il paraîtrait bizarre que la relation $\trianglelefteq$ soit partiellement calculable à partir de $\trianglerighteq$ et que le système n'en tienne pas compte pour aider l'utilisateur à exprimer ses classes. C'est l'argument du cas (3.4) qui mène ici au cas (3.6) : les classes définies rendent en fait inutile la distinction entre $\trianglelefteq$ et $\preceq$, et il serait souhaitable d'avoir $\trianglerighteq \ \subseteq \ \preceq$. Cette dernière propriété mérite bien sûr d'être *objective*, ce qui impose de classer automatiquement les classes, à l'instar des LT.

Si α et β sont définies et si α et β sont incompatibles, elles devraient être intensionnellement incompatibles si elles ont les mêmes attributs (*exclusivité intensionnelle*) : elles devraient donc avoir des domaines disjoints sur un attribut au moins. Il faudrait sinon considérer qu'une instance satisfaisant les intensions des deux classes est un objet *paradoxal* : il ne peut exister, et c'est une erreur comme une division par zéro, ou bien elle rend l'ensemble du SC inconsistant, comme une contradiction en logique classique.

Lorsque les classes incompatibles sont primitives, cette condition n'est plus nécessaire : on pourrait dire que l'intersection des intensions des deux classes correspond à deux sous-classes paradoxales, de même intension mais incompatibles : rien (aucun attribut) ne permet de discerner leurs instances et l'attribution de l'une ou l'autre classe

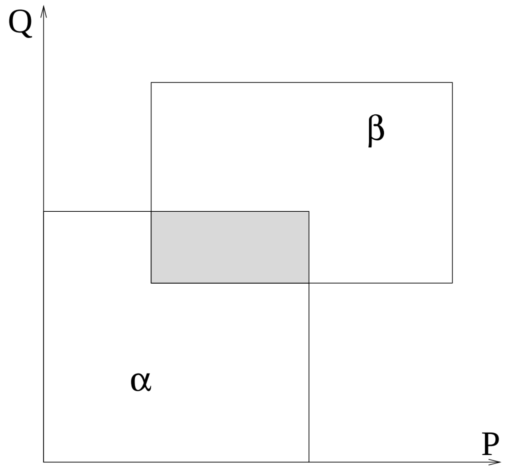

FIG. 3.7 – *L'intersection des intensions des classes α et β avec deux attributs* `P` *et* `Q` *recouvre une extension vide et des objets paradoxaux si les classes sont définies et incompatibles, ou bien deux sous-classes paradoxales de même intension et d'extensions disjointes si l'on accepte l'hypothèse de discrimination des intensions.*

à l'instance relève d'un *deus ex machina* extérieur au SC, le monde réel[7] (figure 3.7). Comme la suite le montrera, la notion de classe définie a d'autres effets sémantiques importants, et il est nécessaire de s'y intéresser dès que la classification d'instances est abordée (chapitre 6).

On remarquera qu'une propriété structurelle très forte comme l'exclusivité intensionnelle ne suffit pas à rendre les classes définies. Cependant, ajouter l'exhaustivité fait bien obtenir cette propriété. Ce n'est pas une coïncidence : l'exhaustivité et la définition (en intension) des classes constituent l'une et l'autre un tropisme descendant, qui force les instances à « descendre » quand elles le peuvent. La définition en intension est donc une forme atténuée d'exhaustivité. Inversement, l'exhaustivité intensionnelle, alliée au caractère défini des classes, induit bien une exhaustivité extensionnelle.

Discrimination des intensions

> [...] dire d'une chose quelconque qu'elle est identique à elle-même est évidemment trivial. Dès lors, comme le faisait remarquer Wittgenstein, à quoi bon parler d'identité ?
>
> W.V. Quine, « Identité » in [Quine, 1993]

Poussons le paradoxe un peu plus loin, en remarquant que deux objets identiques en ce qui concerne les attributs de ces deux sous-classes, peuvent très bien ne pas appartenir à la même. Cette incomplétude de la représentation est suffisamment paradoxale pour chercher à l'éviter en faisant l'hypothèse de cohérence suivante. Soient o_1, o_2 et α :

$$\big(\forall P \in \mathcal{P}_\alpha : P(o_1) = P(o_2)\big) \Longrightarrow \big(o_1 \in Ext(\alpha) \iff o_2 \in Ext(\alpha)\big) \tag{3.26}$$

Ce sera l'*hypothèse de discrimination des intensions* : elle signifie que l'appartenance d'un objet à une classe est entièrement déterminée par leurs intensions respectives,

7. J'y verrais bien un *démon de Maxwell*.

même si cette détermination n'est pas exprimable dans le langage de description du SC (cf. paragraphe 4.3.1, page 76).

Une hypothèse complémentaire serait un analogue du *principe d'exclusion de Pauli* de la mécanique quantique — version moderne du principe d'*identité des indiscernables* de Leibniz —, qui veut que deux particules ne puissent pas avoir les mêmes valeurs propres sans être identiques. Wand (1989), suivant en cela l'ontologie de [Bunge, 1977; Bunge, 1979], en déduit que deux objets ayant exactement les mêmes valeurs d'attributs sont identiques. Capponi (1995) fait aussi cette hypothèse. C'est ce qui permet(trait) d'identifier les instances de types structurés à des valeurs. Cette hypothèse signifie en fait que la représentation doit être suffisamment fine pour pouvoir distinguer deux objets par leurs descriptions : à chaque individu peut être associée la classe *définie* par la description de l'individu, dont il est la seule instance.

Cette hypothèse devient problématique lorsque l'on considère une sous-classe α qui ajoute à ses super-classes β quelques nouveaux attributs : deux objets instances propres respectivement de α et β peuvent être indiscernables du point de vue de β tout en étant incomparables du point de vue de α, l'un d'eux n'en étant pas instance. Notons que cela ne peut pas se produire en cas d'*exhaustivité*.

Une version un peu plus forte de cette hypothèse repose sur la notion de *clé*, commune aux bases de données et à TROPES, qui représente un sous-ensemble d'attributs suffisant pour identifier l'objet, c'est-à-dire le distinguer des autres.

Définition en extension

A) animaux sur lesquels on fait des paris,
B) animaux dont la chasse est interdite du 1^{er} avril au 15 septembre,
C) baleines échouées,
D) animaux dont l'entrée sur le territoire national est soumis à quarantaine,
E) animaux en copropriété, F) animaux empaillés, G) *et cætera*[8],
H) animaux susceptibles de communiquer la lèpre, I) chiens d'aveugle,
J) animaux bénéficiaires d'héritages importants,
K) animaux pouvant être transportés en cabine, L) chiens perdus sans collier,
M) ânes, N) juments présumées pleines.

Georges Perec, *Penser/Classer*, 1982

En fait, une classe localement exhaustive est une classe *définie en extension*, par la réunion des extensions de ses sous-classes. L'intension de cette classe peut bien sûr se définir par l'intersection ($\sqcap$) des intensions de ses sous-classes : mais cela n'en fait pas une classe *définie en intension* pour autant. Une véritable définition en intension nécessiterait l'introduction d'un opérateur de *disjonction* qui ne fait pas partie de notre langage.

Il est facile de rendre une classe α localement exhaustive : il suffit de lui ajouter comme sous-classe la *classe de ses instances propres*. C'est à nouveau une classe définie en extension, cette fois par la différence entre l'extension de α et celles des sous-classes de α. Mais cette nouvelle sous-classe n'a aucune intension spécifique (elle a celle d'α), ce qui exclut qu'elle puisse être *définie*. Ainsi, si l'exhaustivité seule n'a aucun effet sur

8. *Cet « etc. » n'a rien de surprenant en soi ; c'est seulement sa place dans la liste qui le rend curieux* (N.d.A.).

la sémantique des classes, ce n'est plus le cas pour les classes définies : une classe *définie et localement exhaustive* dont les sous-classes sont définies a une propriété irréductible à ses caractères exhaustif et défini.

L'exhaustivité a des effets sur l'existence d'un *domaine universel* et peut contredire l'*hypothèse de minimalité des domaines* (cf. paragraphe 4.3.1, page 76). Certains objets apparemment bien formés (leurs attributs respectent les contraintes de leur classe) sont *paradoxaux* : ils ne peuvent être rattachés à aucune sous-classe (figure 3.7). C'est le cas si l'exhaustivité extensionnelle n'est pas couplée à l'exhaustivité intensionnelle.

NOTE *3.7* *Usages et abus*

L'usage de *définition en extension* est un peu abusif : l'usage classique concerne une définition par explicitation des instances, comme par le constructeur `one-of` des LT. Il s'agit en fait ici d'une définition en intension au sens général du terme, pour laquelle il n'existe pas d'intension exacte, au sens précis que nous lui donnons ici, mais pour laquelle l'extension de la classe se calcule à partir des extensions d'autres classes.

Au niveau du langage, la définition en extension se réaliserait par un constructeur de *disjonction* (`or`) et permettrait de représenter les familles de concepts que Wittgenstein illustre par les *jeux*. L'intension d'une telle classe s'obtient par intersection des intensions des sous-classes, mais elle peut être vide : il n'y a rien de commun aux différents jeux, si ce n'est que nous les reconnaissons comme tels [Wittgenstein, 1945] [9]. La définition en extension s'illustre aussi parfaitement par la définition des `animaux`, vus par l'administration française, en exergue de ce paragraphe.

L'opérateur `cover` des LT est cependant plus proche de cette notion de classe exhaustive puisqu'il n'empêche pas d'introduire normalement la classe, avec sa description. Il est aussi plus précis puisqu'il spécifie les sous-classes qui recouvrent l'extension de la classe : `(cover Humain Homme Femme)`.

3.4.3 Expression des propriétés spécifiques

Il y a *a priori* deux façons d'exprimer qu'un SC ou une classe a une propriété spécifique :

- inclure dans la sémantique du SC la ou les « équations » nécessaires à la propriété (cf. chapitre 4) ;
- pour une propriété spécifique à une classe, forcer cette propriété par les expressions appropriées qui l'impliquent, en jouant, soit de l'expression de l'incompatibilité, soit de l'introduction d'attributs particuliers.

N'importe quelle classe primitive α peut être rendue définie, en rajoutant à son intension un attribut *être-un-*α à valeur quelconque : c'est un peu ce que fait Nebel

9. Cette invocation des mânes de Wittgenstein est un peu déloyale : *Mais si quelqu'un disait : « Donc quelque chose est commun à toutes ces formations, — à savoir la disjonction de toutes ces caractéristiques communes » je répondrais : ici vous ne faites que jouer sur les mots.* Un jeu réflexif, donc.

(1990) en introduisant des *composants primitifs* [10]. Il n'empêche que cet attribut doit être affirmé pour permettre de déduire cette appartenance. De plus, il faut bien, d'une manière ou d'une autre, introduire ce composant primitif dans l'intension de ses sous-classes.

L'exclusivité se fera aussi bien par une déclaration d'incompatibilité, que par l'ajout d'un attribut prenant des valeurs différentes dans toutes les sous-classes.

D'autres fonctionnalités pourraient s'exprimer au moyen d'attributs *artificiels* de ce genre : ainsi l'incompatibilité s'exprime simplement par un même attribut avec des domaines disjoints, ces domaines disjoints pouvant être eux-mêmes exprimés soit par un constructeur comme `one-of`, soit par des restrictions de cardinalité incompatibles, soit, récursivement, par des types eux-mêmes incompatibles. On soulignera quand même le caractère artificiel de ces techniques qui, outre leur côté fastidieux et leur nuisibilité vis-à-vis de la clarté de la représentation des connaissances, présentent des désagréments importants en ce qui concernent l'exploitation du SC, par exemple pour la classification d'instances. Les composants primitifs poseront en particulier des problèmes vis-à-vis des classes définies en extension (cf. paragraphe 3.4.2, page 52).

En fait, il semble délicat d'obtenir exactement la sémantique souhaitée. Pour les classes primitives, par exemple, il faut que le domaine de l'attribut supplémentaire soit réduit à un singleton, de sorte que l'on ne se préoccupe pas de sa valeur, seulement de son existence : un attribut booléen exprimerait déjà plus, donc trop, par exemple la négation de l'appartenance à une classe[11].

10. Nebel ne traite en fait ni les types de base (booléen par exemple), ni le constructeur `one-of`. CLASSIC, qui inclut l'un et l'autre, le fait avec des restrictions fortes (type de base d'extension infinie, donc pas les booléens, et sémantique spéciale pour `one-of`) qui ne paraissent pas empêcher d'utiliser `one-of(true)` dans ce cas [Borgida et Patel-Schneider, 1994] (voir aussi note 11).

11. On retrouverait alors le *méta-héritage* proposé dans [Ducournau et Habib, 1991; Ducournau *et al.*, 1994].

Chapitre 4

Sémantique des SC

La sémantique n'est pas davantage un moyen astucieux de prouver qu'hormis l'auteur et ses amis, tous les autres disent des inepties.
Alfred Tarski, *The semantic conception of truth...* [Tarski, 1952].

L'INTERPRÉTATION EXTENSIONNELLE présentée au paragraphe 2.1 était totalement informelle, la définition du domaine ayant été soigneusement évitée. Par ailleurs, la définition purement extensionnelle du domaine des attributs est impossible, et il faut essayer d'en trouver une définition intensionnelle, la plus générale possible.

4.1 Du côté de l'intension

D'habitude, le problème est pris en sens inverse : une syntaxe est d'abord donnée, arbitraire même si judicieusement choisie en fonction de ses objectifs et des « conditions historiques », à laquelle est associée une sémantique qui a bien sûr guidé, dans un stade intuitif, le choix de la syntaxe. Ici nous partons d'une idée très générale d'une sémantique dont nous voulons faire apparaître, non pas une syntaxe, mais les entités à la base de l'expression. À moins de créer *ex nihilo* de nouvelles entités, nous sommes condamnés à réutiliser celles que nous avons déjà, les classes par exemple, ou des entités universelles comme les entiers.

4.1.1 Le domaine des attributs

Le premier problème est que Dom n'a été considéré jusqu'ici qu'en extension, alors qu'il ne peut être exprimé qu'en intension. Or le seul moyen que nous avons pour définir un ensemble, ce sont les classes. En première approximation, $Dom(\alpha, P)$ peut donc être exprimé par le *type* de l'attribut, et par ses cardinalités admissibles :

$$type : \mathcal{X} \times \mathcal{P} \longrightarrow \mathcal{X} \qquad card : \mathcal{X} \times \mathcal{P} \longrightarrow 2^{\mathbf{N}}$$

et l'ensemble des valeurs admissibles est défini par :

$$Dom(\alpha, P) \stackrel{\text{def}}{=} \{A \subseteq Ext(type(\alpha, P)) \mid |A| \in card(\alpha, P)\}. \tag{4.1}$$

Les contraintes sur l'inclusion des domaines se traduisent alors sur *card* et *type* par le double diagramme suivant :

$$\begin{tikzcd}
2^{\mathbf{N}} \arrow[d, "\subseteq"'] & \mathcal{X} \arrow[l, "card_P"'] \arrow[r, "type_P"] \arrow[d, "\prec"'] & \mathcal{X} \arrow[d, "\prec"] \\
2^{\mathbf{N}} & \mathcal{X} \arrow[l, "card_P"] \arrow[r, "type_P"'] & \mathcal{X}
\end{tikzcd}$$

Les fonctions $type_P$ et $card_P$ sont monotones.

$\mathcal{X}$ doit donc contenir toutes les classes explicitement introduites, mais aussi toutes celles qui servent à typer les attributs. Ce typage des attributs a l'inconvénient d'introduire, au moins potentiellement, des circularités dans les descriptions, mais c'est inévitable : il se contente d'en faire apparaître le risque (cf. paragraphe 4.5.3, page 88).

Pour la cardinalité, il existe des systèmes qui n'autorisent que les 2 parties $\{1\}$ (ou $\{0\ 1\}$) et $\mathbf{N}$, pour des attributs respectivement *mono-valué* ou *multi-valué*, voire que la première, comme les ψ-termes de [Aït-Kaci et Nasr, 1986; Aït-Kaci, 1991a] (cf. paragraphe 7.3.1, page 170). Les systèmes plus expressifs se limitent en général à des intervalles, avec des cardinalités minimum et maximum : c'est le cas assez général des LT— avec leurs constructeurs `atleast` et `atmost` —, et de certains langages de *frames*, avec des facettes `card-min` et `card-max`. Rien n'empêche, si ce n'est la complexité, de généraliser, par exemple aux unions d'intervalles [Capponi, 1995; Gensel, 1995].

Avec cette formalisation du domaine des attributs, la relation d'ordre sur les intensions (formule 2.13) se réécrit intensionnellement :

$$Int(\alpha) \sqsupseteq Int(\beta) \overset{\text{def}}{\Longleftrightarrow} \begin{cases} \mathcal{P}_\alpha \supseteq \mathcal{P}_\beta \\ \forall P \in \mathcal{P}_\beta : \begin{cases} type(\alpha, P) \preceq type(\beta, P) \\ card(\alpha, P) \subseteq card(\beta, P) \end{cases} \end{cases} \tag{4.2}$$

Chercher à remplacer $\preceq$ par $\trianglelefteq$ dans cette définition serait oublier que cette dernière relation n'est pas connue directement[1].

Conflits de type

En revanche, en toute généralité, l'intersection ($\sqcap$) et l'union ($\sqcup$) des intensions (2.16 et 2.17) ne se définissent pas concrètement en fonction de *type* et *card*, pas plus que l'héritage des domaines par intersection sur les super-classes (2.23). Pour que l'intersection (resp. union) des domaines s'exprime par une classe, il faut un demi-treillis inférieur (resp. supérieur) qui fait de $type_P$ un morphisme par rapport à $\wedge$ (resp. $\vee$) :

$$type_P(\alpha \wedge \beta) = type_P(\alpha) \wedge type_P(\beta) \tag{4.3}$$

1. En revanche, l'utilisation de $\trianglelefteq_a$ est possible, mais est-elle souhaitable ? On pourrait aussi considérer que c'est une façon d'affirmer $\vartriangleleft_a$:

$$\alpha \prec \beta \Longrightarrow type_P(\alpha) \trianglelefteq_a type_P(\beta)$$

Voir par exemple le chapitre 5.

Cette limitation est la justification principale d'un treillis ou, au moins, de l'existence d'une borne inférieure dans tous les cas d'héritage multiple [Castagna, 1996b].

Soit $type_P(\alpha) \wedge type_P(\beta)$ est défini, soit, pour tout *minorant maximal* γ de α et β ($\alpha \wedge \beta$ si défini), il faut expliciter $type_P(\gamma)$ comme sous-classe commune (minorant pas forcément maximal) de $type_P(\alpha)$ et de $type_P(\beta)$, si l'on veut pouvoir caractériser précisément et dans les termes du langage le domaine d'un attribut, en particulier pour vérifier l'hypothèse de *minimalité des domaines* (cf. paragraphe 4.3.1, page 76). Cela revient à résoudre à la main les conflits d'héritage sur *type*, alors qu'ils se résolvent automatiquement sur *Dom* (avant qu'il ne soit réellement spécifié) ou sur *card*.

Si l'on n'a pas ce besoin, dont la justification n'est pas claire, ces conflits de type ne sont pas gênants : il suffit juste que les classes α et β ne soient pas incompatibles, et donc $type_P(\alpha)$ et $type_P(\beta)$ non plus. Dans ce cas, $type_P$ n'est plus qu'une fonction partielle sur $Def(P)$, définie uniquement sur les classes qui l'explicitent : pour les autres, il faut se contenter de la définition de *Dom* (2.23). La nuance est sans objet pour $card_P$ qui peut se définir sans problème par intersection sur les super-classes, comme Dom_P dans (2.23) lorsqu'il n'est pas explicite. Notons que rien n'oblige à expliciter en même temps $type_P$ et $card_P$.

NOTE *4.1* *Monotonie de* $type_P$

La monotonie de $type_P$ est plus une analogie quelque peu arbitraire qu'une propriété nécessaire : pour $card_P$, la monotonie suit des mêmes arguments que pour Dom_P, en particulier l'hypothèse de minimalité des domaines. En revanche, pour $type_P$, les arguments sur les conflits de type exposés ci-dessus peuvent aussi faire admettre que $type_P$ n'est pas monotone : si $\alpha \prec \beta$, P doit prendre ses valeurs pour les instances de α dans les instances communes à $type_P(\alpha)$ et $type_P(\beta)$, par exemple dans les instances des sous-classes communes, mais cela n'impose pas que $type_P(\alpha)$ soit sous-classe de $type_P(\beta)$. Dans certains cas, cela peut être le seul moyen d'exprimer exactement le domaine voulu, sauf à introduire une classe pour ce seul besoin.

Pour résoudre ces questions de monotonie de $type_P$ aussi bien que de conflits de type, on pourrait remplacer *type* par une fonction *types* à valeur dans $2^{\mathcal{X}}$, et plus particulièrement dans $\mathcal{X}^*$. La monotonie de $types_P$ serait relative à l'ordre sur les antichaînes (ou sur les idéaux). Ce serait une généralisation parfaite à la multi-instanciation et au treillis des idéaux sous-jacent (cf. paragraphe 3.2.1, page 41).

On remarquera que la sémantique qui suit, pas plus que les règles d'inférences des chapitre 5 et 6, ne fait aucune hypothèse sur la monotonie de $card_P$ ou $type_P$: au contraire, c'est la monotonie de la « partie utile » de ces fonctions qui se déduit de la sémantique ou des règles. La monotonie n'est pas nécessaire pour le domaine exprimé puisque la sémantique l'assure pour le domaine effectif. Il faut donc considérer que la monotonie de $type_P$ n'est qu'une convention simplificatrice, utile pour rationaliser intuitivement les SC, mais superflue puisqu'impliquée en pratique par la sémantique du SC.

NOTE *4.2* *Domaines et valeurs*

Cette définition du domaine est insuffisante sur au moins un point : elle ne permet pas de fixer la valeur d'un attribut dans une classe comme peut le faire la *facette* `valeur` dans les langages de *frames*, autrement qu'en introduisant la classe de cette seule instance. Il faudrait donc disposer d'équivalent de cette facette `valeur`, aussi bien que des constructeurs `one-of` et `fills` de certaines LT comme CLASSIC (cf. annexe B.1, page 191), dont l'effet respectif est de donner un domaine en extension, ou d'imposer une ou plusieurs valeurs. La facette `valeur` correspond au constructeur `fills` doublé d'une restriction de cardinalité pour interdire des valeurs suplémentaires. Dans tous les cas, ces constructions ont l'inconvénient de mélanger les composantes intensionelle et extensionnelle (cf. paragraphe 2.2.2, page 28), mais c'est un peu inévitable. Nebel (1990) ne les considère pas.

CLASSIC, de son côté, limite la prise en compte des instances dans la description des classes : ni les valeurs des rôles, ni les classes de ces instances ne sont considérées dans la subsomption. C'est le prix a payer pour que la composante extensionnelle reste une *extension conservative* de la composante intensionnelle.

NOTE *4.3* *Théorie des types : covariance, contravariance ou invariance ?*

Les approches des objets de type *théorie des types* [Gunter et Mitchell, 1994; Abadi et Cardelli, 1996; Castagna, 1996b] concluent à l'impossibilité de la *covariance* du type d'un attribut lors d'une *redéfinition* : l'*invariance* serait au contraire nécessaire, de même que la *contravariance* pour le type des paramètres « secondaires » des *méthodes* ou *fonctions génériques* (ceux qui ne servent pas à la sélection de la méthode). La nécessité de l'invariance provient de ce que l'accès en lecture à un attribut doit être covariant, et l'accès en écriture contravariant. Pour la représentation des connaissances, c'est une limitation insoutenable dont il faut espérer qu'elle est liée à une limitation actuelle de ces théories : il faudrait sinon considérer qu'un SC ne peut pas être sûr du point de vue des types.

Bien entendu, cela n'a d'importance que si l'on cherche à écrire des programmes (que l'on souhaite sûrs du point de vue des types, *type safe*) dont les structures de données sont les classes d'un SC. Cette limite de la théorie des types conduit néanmoins à l'éventualité d'un compromis entre l'*expressivité* des langages et la *sûreté* des programmes qui n'est pas sans rappeler le fameux compromis *expressivité-complétude-complexité* des LT [Levesque et Brachman, 1987; Attardi, 1991] (cf. paragraphe 5.1.2, page 101).

NOTE *4.4* *Domaines et prédicats*

On peut aussi s'interroger sur la possibilité de définir les domaines des attributs à l'aide d'un prédicat — du point de vue de l'implémentation, une fonction booléenne, la fonction caractéristique de l'ensemble — ce qui reviendrait, en vertu de l'identité « domaine = classe », à autoriser de définir les classes par un prédicat, ce qui est la porte ouverte à toutes les imprédicativités (cf. paragraphe 2.4.2, page 32), à moins de restreindre ces « définitions » prédicatives à un usage non définitionnel (cf. paragraphe 3.4.2, page 49).

Les langages de *frames* ont toujours autorisé l'usage de ces prédicats, suivis en cela par certaines LT comme CLASSIC avec le constructeur `test` : dans tous les cas, ces prédicats ne peuvent être utilisés qu'au niveau extensionnel pour imposer des contraintes à des instances. Au niveau intensionnel, l'indécidabilité de la comparaison de fonctions imposerait la déclaration explicite et subjective d'une éventuelle subsomption.

NOTE *4.5* *Réification des collections*

Cette définition du domaine des attributs par des spécification et restriction de type et de cardinalité évite d'avoir à réifier les collections, comme le font les langages d'objets classiques quand ils manipulent des types non atomiques (listes, ensembles, tableaux, etc.) Cette réflexion amène, en passant de la réification des ensembles à la réification des classes, à poser le problème de la réflexivité, c'est-à-dire de la représentation réflexive des classes comme des instances de leur propre système classificatoire, au moyen de méta-classes, mais c'est une tout autre histoire (cf. paragraphe 2.4.2, page 32).

NOTE *4.6* *Décidabilité de la cardinalité*

Nous avons conservé un caractère abstrait à l'expression de la cardinalité : il faudra bien supposer que cette cardinalité reste décidable et de complexité réduite pour tous les problèmes qui seront nécessaires : appartenance, disjonction, inclusion, maximum, etc. dont les chapitres 5 et 6 donneront quelques exemples d'utilisation. On pourra donc choisir des intervalles ou unions d'intervalles, mais on évitera de définir $card_P$ par « les nombres premiers », ou « les nombres qui ne vérifient pas le dernier théorème de Fermat ».

Plus généralement, on évitera de tomber dans l'indécidabilité du dixième problème de Hilbert et on choisira pour codomaine de $card$ une partie de $\mathbf{N}$ fermée par intersection (les intervalles par exemple), éventuellement par union aussi (les unions d'intervalles alors). Si ce n'est pas le cas, on sera confronté avec $card_P$ aux mêmes hésitations qu'avec $type_P$. On peut enfin envisager d'intégrer cette expression des cardinalités comme des classes du SC, dans le cas où il intègre les types de base.

NOTE *4.7* *Notation*

Dans la suite, l'usage de Dom sera ambivalent : il s'agira soit d'une abréviation de la définition (4.1), soit d'un usage intuitif et incorrect ou insuffisant, prélude à une reformulation plus précise.

4.1.2 L'hypothèse d'indépendance des attributs

Si cette définition du domaine des attributs n'est pas tellement limitative, c'est la notion même de domaine — dépendant uniquement de la classe et de l'attribut, mais pas de l'objet lui-même et de ses autres attributs — qui limite sérieusement l'expressivité du SC. En particulier, l'hypothèse de l'indépendance des attributs est implicite : on suppose que toute combinaison de valeurs de deux attributs, c'est-à-dire le produit cartésien de leurs domaines, est admissible. En toute généralité des domaines modélisés, ce n'est pas le cas, et de nombreuses contraintes entre les valeurs des attributs ont été imaginées.

Hiérarchie d'attributs

Les LT proposent par exemple deux façons de lier les valeurs de deux rôles P et Q, pas seulement dans une même classe mais dans un même objet :

$$\forall o \in Ext(\alpha) : P(o) \subseteq Q(o).$$

Cette relation peut être globale, c'est-à-dire valable pour tout $\alpha \in Def(P)$, P est alors un *sous-rôle* de Q — noté $P \preceq Q$ —, ou locale à une classe α donnée, et à ses sous-classes : il s'agit alors de *role-value-map* — noté $P_\alpha \preceq Q_\alpha$.

L'utilisation de la même notation $\prec$ que pour les classes est la porte ouverte à toutes les généralisations : on peut définir des rôles par des restrictions de domaine et co-domaine. On en déduit bien sûr des équivalents de $\trianglelefteq$ et $\trianglerighteq$ pour les rôles, des définitions en extension (les `enfants` sont la réunion des `fils` et des `filles`), etc. Plus généralement, les attributs eux-mêmes forment un SC.

Les restrictions de valeur et de cardinalité peuvent aussi se combiner dans une même expression, pour faire l'économie d'un sous-rôle explicite, comme avec le constructeur `c-atleast` — `(c-atleast 2 enfants Femme)`, pour « avoir au moins deux filles » —, ou la version plus restreinte de la quantification existentielle qualifiée du langage $\mathcal{ALE}$ ou du constructeur `c-some`[2] : `(c-some enfants Femme)`, pour « avoir au moins une fille ». De façon générale, les sous-rôles peuvent être introduits plus formellement par l'intersection de rôles primitifs comme le fait Nebel (1990) avec le constructeur `and-role` ou Donini *et al.* (1994) avec le langage $\mathcal{ALR}$. Un sous-rôle peut aussi être *défini*, comme une classe, par une restriction de valeur (`range`).

Nebel (1990) montre comme il devient difficile de décider de $\sqsupseteq$ sur les super-rôles lorsque leurs domaines peuvent être contraints implicitement par des sous-rôles : il en déduit en particulier qu'un système contraint à la complétude devrait se passer de ces sous-rôles.

Chemins d'attributs, inverse

D'autres contraintes sur les attributs peuvent être basées sur les opérations élémentaires sur les relations : composition, inverse, etc.

- il est ainsi possible de contraindre des chemins d'attributs, ou *role chains*, formés par le constructeur `compose`, qui s'interprètent comme des compositions de relations ou de fonctions : on définit, par exemple, les `grands-parents` comme les `parents` des `parents`. Dans certains langages, ces chemins d'attributs peuvent être restreints aux attributs mono-valués[3]. Leur usage sur des attributs multi-valués augmenterait considérablement la complexité.

  Les contraintes sur les attributs composés contraignent implicitement, de façon potentiellement complexe, les attributs du chemin, en particulier le premier ;

- on peut aussi définir et restreindre le rôle `inverse` ;

- les ψ-termes de [Aït-Kaci et Nasr, 1986; Aït-Kaci, 1991a], de même que le constructeur `same-as` de CLASSIC [Borgida et Patel-Schneider, 1994], permettent

2. À ne pas confondre avec la quantification existentielle non qualifiée : `(some x)` équivaut à `(atleast 1 x)`, alors que `(c-some x Y)` équivaut à `(c-atleast 1 x Y)`.

3. Autre restriction : en général, ces chemins d'attributs n'apparaissent qu'au niveau intensionnel (classes) ; au niveau extensionnel (instances), il faut expliciter un objet sur tous les attributs du chemin, ce qui signifie « choisir » une valeur et non pas simplement supposer son existence. Mais si un attribut est défini par un chemin, ou contraint par `same-as`, les contraintes sur l'attribut provoquent des hypothèses existentielles sur le chemin.

d'imposer des contraintes d'identité entre les valeurs de différents attributs (ou chemins d'attributs) : cette possibilité est en général restreinte aux attributs monovalués [4] ;

la classification dans les *graphes conceptuels* doit certainement poser des problèmes du même ordre que les ψ-termes (cf. 7.5.2) ;

- on pourrait vouloir lier les cardinalités de différents rôles, de telle sorte que $\forall \alpha : card(\alpha, P) \subseteq card(\alpha, Q)$, ou par des sortes de *role-cardinality-map* faisant par exemple que $\forall o \in Ext(\alpha) : |P(o)| \leq |Q(o)|$: on porte généralement moins de chapeaux que de têtes, moins de chaussures que de pieds, et moins de bagues que de doigts, et on a, sauf polyandrie, moins de `gendres` que de `filles`.

- ou réaliser le quantificateur universel MA de [Woods et Schmolze, 1992] pour que la valeur de l'attribut P contienne exactement toutes les instances de $type(\alpha, P)$, et ce pour toutes les instances o de α : la valeur de P serait alors la même pour toutes les instances de α, et pourrait être implémentée par une *variable de classe* des LPO.

NOTE *4.8* *Réification des relations*

POWER-CLASSES propose une notion de *relation* entre classes qui correspond à deux attributs symétriques dont les valeurs sont maintenues cohérentes et qui peut se spécialiser dans des sous-relations [ILO, 1996; ILO, 1995]. Chaque sous-relation correspond effectivement à une définition en intension, par un prédicat LISP (c'est-à-dire ILOGTALK [ILO, 1995]), à partir de la sur-relation. Les relations, leurs inverses et éventuellement des sous-relations sont présentes dans de nombreux langages depuis SRL2 [Wright *et al.*, 1984]. Si certains systèmes comme YAFOOL traitent ces relations comme des attributs particuliers, les limitant ainsi au cas binaire, l'originalité de POWER-CLASSES est d'introduire la relation en tant que telle, hors toute introduction de classe, ce qui permettrait de traiter des relations n-aires, même si, en pratique, le système ne le permet pas.

Contraintes sur les attributs

Il faudrait donc étendre le cadre formel d'expression des attributs pour permettre d'y inclure ces différents traits. La difficulté serait de le faire avec la plus grande généralité possible. Les contraintes (au sens des CSP) offrent sans doute cette généralité : se pose alors le problème de la généralité du langage de contraintes. Pour ce qui est des rapports entre SC et contraintes, ils sont naturels, ce qui explique que la déduction doive utiliser des techniques d'énumération exhaustive, ce que les LT essaient d'éviter, soit par l'incomplétude de leurs algorithmes, soit dans la spécification des types de base [Baader et Hanschke, 1991]. Voir aussi la note 10 sur le problème du *zèbre*. Certaines hypothèses comme le principe d'exclusion de Pauli (cf. paragraphe 3.4.2, page 51) se traduisent par des contraintes implicites dont la prise en compte semble difficile.

Par ailleurs, tout cela mène à des difficultés de définition (c'est-à-dire de calcul) des relations intensionnelles comme $\sqsupseteq$ et $\trianglerighteq$ qui paraissent inextricables. Nous conserverons

4. Le problème est alors polynomial, alors qu'il devient indécidable dès que l'on passe aux attributs multivalués [Borgida, 1992a]

sur ce point notre position semi-formelle habituelle : la formalisation fera constamment cette hypothèse implicite d'indépendance, quitte à ce que la discussion non formelle s'en dégage à l'occasion.

4.2 Sémantique et interprétation des domaines

La sémantique [...] est de nos jours si compliquée
et le nom qui la distingue tellement ambivalent
qu'il convient d'appliquer l'analyse sémantique au terme même de « sémantique » [...].
Adam Schaff, Préface à *Introduction à la sémantique*, 1960 [Schaff, 1960]

Le cadre extensionnel présenté plus haut visait à offrir une sémantique intuitive des systèmes classificatoires. Nous proposons maintenant un cadre d'interprétation plus formel.

4.2.1 Le choix d'une sémantique

Dans le cadre général de ce que l'on peut appeler *sémantique dénotationnelle*, dans lequel nous nous sommes placés pour donner une interprétation intuitive des SC, il existe plusieurs façons de considérer plus formellement le domaine d'interprétation du système :

- la *théorie des modèles* offre un cadre général pour établir une sémantique formelle ensembliste : le but est en particulier de déterminer les inférences *valides*, c'est-à-dire valables pour tous les *modèles* du système considéré, d'un ensemble de descriptions ;

- il est quelquefois possible de construire explicitement un domaine unique qui vaut pour tous les domaines possibles, à la façon du *domaine de Herbrand* du calcul des prédicats. Il faut alors disposer d'un équivalent du *théorème de Herbrand* qui assure que ce qui vaut pour ce domaine est effectivement valide. C'est par exemple la démarche de [Nebel, 1990], pour des terminologies sans circuit, ou celle de [Dionne *et al.*, 1993; Coupey et Fouqueré, 1994]. Dans ce cas, la sémantique peut être qualifiée de *constructive* puisque le domaine est explicitement construit, et le domaine d'*universel*, puisqu'il est solution d'un *problème universel* [5]. Dans ce cas, ce domaine universel, d'essence sémantique, est construit syntaxiquement ce qui a tendance a unifier en apparence les deux points de vue sémantique et syntaxique.

- certains auteurs partent du principe que l'objectif de la représentation des connaissances est de représenter une partie idéalisée du monde réel, et considèrent qu'il existe un *domaine modélisé*, c'est-à-dire un modèle privilégié, qui est censé détenir la sémantique du SC. C'est par exemple la position d'Euzenat (1994). Les propositions qui sont vraies dans ce modèle particulier sans être valides traduisent

5. C'est-à-dire unique à un isomorphisme près.

alors une incapacité du système de représentation (ou/et de ses utilisateurs) à représenter fidèlement le domaine modélisé [6] et lui seul.

À cette *incomplétude intensionnelle* s'ajoute une *incomplétude extensionnelle* liée au fait que l'*état du monde* — c'est-à-dire l'état courant du *domaine modélisé* — n'est pas complètement connu : ce sera l'objet principal du chapitre 6.

- l'ensemble des instances effectivement définies au niveau assertionnel ne peut évidemment pas être considéré comme un domaine satisfaisant à moins de pouvoir assurer de la complétude de la description du monde qu'elles réalisent. C'est le cas des *modèles canoniques* de Nebel, qui peuvent être considérés comme leur propre modèle.

En fait, un SC étant d'abord un système générique, il ne peut pas s'appuyer sur l'existence d'un domaine modélisé pour la réalisation de ses algorithmes qui doivent forcément être corrects, c'est-à-dire correspondre à des inférences valides. Bref, la prise en compte du domaine modélisé en tant que modèle privilégié ne paraît possible que dans le choix d'un SC muni de propriétés spécifiques, ou dans des extensions *ad hoc* du SC.

Il existe d'autres techniques pour établir une sémantique d'un système formel, mais nous nous restreignons ici à une sémantique de type dénotationnel [7] compte-tenu de l'objectif de représentation des connaissances des systèmes considérés : en effet, seule une sémantique de ce type paraît à même d'offrir le caractère intuitif nécessaire à sa déclarativité (cf. [Carré *et al.*, 1995]). C'est aussi la seule qui paraisse justifier l'usage du mot « sémantique » dont le sens usuel est « système de signification ». Nous verrons dans les chapitres suivants que le versant déduction des SC offre une *sémantique opérationnelle* qui peut, dans certains cas, caractériser avec autant d'intuition ce que calcule effectivement le SC.

NOTE *4.9* *Modèle et modélisation*

Il ne faut pas confondre la notion de *modèle* au sens de la théorie des modèles, avec la notion homonyme au sens de *modélisation* ou de *modèle réduit.* Berthier (1994) présente avec bonheur les notions de modèle et de *représentation.* Pris dans le sens de modélisation, un modèle est « une formalisation de certains aspects de son objet » (le monde réel) : c'est ce modèle réduit qui est le *domaine modélisé* [8]. Les deux notions de modèle s'unifient néanmoins ainsi : ce modèle du monde réel n'est [9] qu'un des modèles (au sens de la théorie des modèles) de la représentation qui en est faite.

---

6. C'est une incomplétude *syntaxique* analogue à l'incomplétude de l'arithmétique qui fait l'objet du deuxième théorème de Gödel : il existe des formules qui ne sont ni valides (c'est-à-dire démontrables compte-tenu de la complétude *sémantique*), ni la négation de formules valides. Ces formules sont donc vraies dans certains modèles, fausses dans d'autres (sur ces distinguo fondamentaux de complétude, voir par exemple [Gochet et Gribomont, 1991]). Être en si bonne compagnie ne rend pas forcément cette situation enviable ou souhaitable. En particulier, cela n'empêche pas de chercher, 1° à compléter le système formel par l'introduction de nouveaux axiomes (ce dont Gödel a aussi montré le caractère illusoire pour l'arithmétique), 2° à étudier les capacités générales de déduction du SC considéré, c'est-à-dire les formules valides.

7. Dans le sens le plus général et intuitif du terme, et donc sans référence particulière à la sémantique dénotationnelle des langages de programmation [Mosses, 1990].

8. Qu'il faudrait plutôt appeler le *domaine représenté*, ou alors c'est le domaine modélisé qui est constitué par « certains aspects du monde réel ».

9. À un isomorphisme près, voire un simple morphisme.

4.2.2 Une sémantique « directe » en théorie des modèles

Récemment, la tendance était de sacrifier la simplicité sur l'autel de la théorie des modèles.
W.V. Quine, « Mathématose » in [Quine, 1993]

Notre niveau d'abstraction ne nous permettant pas une sémantique constructive, nous nous plaçons donc dans le cadre le plus général de la théorie des modèles. Précisons qu'il ne s'agit pas d'une sémantique complète, mais d'un schéma général où ne figurent que les équations communes à tous les SC, qui correspondent aux diverses contraintes introduites au chapitre 2. Le cadre général et les notations sont empruntés, avec quelques variantes, à [Nebel, 1990].

Modèle d'un sc

Une *structure sémantique* d'un SC est un quadruplet $(\Delta, \mathcal{E}_{\mathcal{X}}, \mathcal{E}_{\mathcal{P}}, \mathcal{E}_{\mathcal{O}})$ formé par

- un *domaine* arbitraire Δ,
- une fonction d'interprétation des classes $\mathcal{E}_{\mathcal{X}} : \mathcal{X} \longrightarrow 2^{\Delta}$,
- une fonction d'interprétation des attributs $\mathcal{E}_{\mathcal{P}} : \mathcal{P} \longrightarrow (\Delta \to 2^{\Delta})$,
- et une fonction d'interprétation des objets, injective, $\mathcal{E}_{\mathcal{O}} : \mathcal{O} \longrightarrow \Delta$.

$\mathcal{X}$ et $\mathcal{O}$ sont les ensembles de classes et d'instances introduits au chapitre 2 dans la description des composantes intensionnelle et extensionnelle.

La tendance actuelle est à la simplification de ces notations, en considérant une seule fonction, notée $\mathcal{I}$ (en général en exposant : $.^{\mathcal{I}}$)[10], réunion disjointe des trois fonctions précédentes, donc élément de $(\mathcal{X} \to 2^{\Delta}) \cup (\mathcal{P} \to (\Delta \to 2^{\Delta})) \cup (\mathcal{O} \to \Delta)$. Chacune de ces trois fonctions est donc la restriction de $\mathcal{I}$ à un domaine spécifique :

$$\mathcal{E}_{\mathcal{X}} = \mathcal{I}/\mathcal{X} \tag{4.4}$$

$$\mathcal{E}_{\mathcal{P}} = \mathcal{I}/\mathcal{P} \tag{4.5}$$

$$\mathcal{E}_{\mathcal{O}} = \mathcal{I}/\mathcal{O} \tag{4.6}$$

On réduira ainsi une structure sémantique à la paire $(\Delta, \mathcal{I})$.

Un *modèle* $\mathcal{M}$ d'un système classificatoire est une structure sémantique $(\Delta, \mathcal{I})$ qui vérifie un ensemble de contraintes que nous appellerons, par commodité, *axiome* :

$$\bot^{\mathcal{I}} = \emptyset \qquad \top^{\mathcal{I}} = \Delta \tag{4.7}$$

$$\alpha \trianglelefteq_a \beta \Longrightarrow \alpha^{\mathcal{I}} \subseteq \beta^{\mathcal{I}} \tag{4.8}$$

$$\alpha \nabla_a \beta \Longrightarrow \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} = \emptyset \tag{4.9}$$

10. La sémantique dénotationnelle des langages de programmation [Mosses, 1990] va même jusqu'à remplacer la notation fonctionnelle explicite par la notation $[\![.]\!]$. On remarque aussi qu'une unique fonction d'interprétation serait impossible dans le cas des systèmes réflexifs dont les classes sont elles-mêmes des objets ($\mathcal{X} \subset \mathcal{O}$).

$$(4.10) \qquad o \in \alpha^{\mathcal{I}}, P \in \mathcal{P}_\alpha \Longrightarrow \begin{cases} P^{\mathcal{I}}(o) \subseteq type(\alpha, P)^{\mathcal{I}} \\ |P^{\mathcal{I}}(o)| \in card(\alpha, P) \end{cases}$$

$$(4.11) \qquad o \notin \bigcup_{P \in \mathcal{P}_\alpha} \alpha^{\mathcal{I}} \Longrightarrow P^{\mathcal{I}}(o) \text{ pas défini}$$

$$(4.12) \qquad (\alpha\ o) \in \mathcal{W} \Longrightarrow o^{\mathcal{I}} \in \alpha^{\mathcal{I}}$$

$$(4.13) \qquad (P\ o_1\ o_2) \in \mathcal{W} \Longrightarrow {o_2}^{\mathcal{I}} \in P^{\mathcal{I}}({o_1}^{\mathcal{I}})$$

Les deux dernières formules se réécrivent dans un « style objet » comme suit :

$$(4.12') \qquad \alpha \in Isa_{\mathcal{W}}(o) \Longrightarrow o^{\mathcal{I}} \in \alpha^{\mathcal{I}}$$

$$(4.13') \qquad o_2 \in P_{\mathcal{W}}(o_1) \Longrightarrow {o_2}^{\mathcal{I}} \in P^{\mathcal{I}}({o_1}^{\mathcal{I}})$$

$o^{\mathcal{I}}$ sera dit une *interprétation* de o, et o une *représentation* de $o^{\mathcal{I}}$.

Le double diagramme « pseudo commutatif » suivant visualise bien l'interprétation des classes et des attributs, avec $E^{\mathcal{I}^*} \stackrel{\text{def}}{=} \{x^{\mathcal{I}} \mid x \in E\}$:

$$\begin{tikzcd}
\mathcal{X} \arrow[d, "\mathcal{I}"'] & \mathcal{O} \arrow[l, "Isa"'] \arrow[r, "P"] \arrow[d, "\mathcal{I}"'] & 2^{\mathcal{O}} \arrow[d, "\mathcal{I}^*"] \\
2^{\Delta} & \Delta \arrow[l, "\ni"] \arrow[r, "P^{\mathcal{I}}"'] & 2^{\Delta}
\end{tikzcd}$$

Il montre en particulier comment Isa s'interprète par l'appartenance ensembliste [Brachman, 1983].

Commentaires

> De vraies questions sur l'identité peuvent surgir lorsque nous avons deux manières de nous référer à quelque chose et que nous pouvons à bon droit nous demander si c'est à chaque fois de la même chose qu'il s'agit.
>
> W.V. Quine, « Identité » in [Quine, 1993]

Soulignons d'abord que cette sémantique adopte un point de vue essentiellement *aristotélicien* (cf. note 7.5, page 177) sur les classes et les instances : la classe ne sert qu'à contraindre ses instances et il n'est pas nécessaire de contraindre la classe, en particulier pour imposer une monotonie du type ou du domaine des attributs (cf. note 4.1, page 57). De plus, seule la sémantique descriptive (les conditions nécessaires) a été prise en compte : nous verrons au paragraphe 4.5.1 les diverses façons de prendre en compte les classes définies.

- l'axiome (4.8) vaut pour $\trianglelefteq_a$, plus généralement que pour $\preceq$ (mais pas pour $\trianglelefteq$ qui n'est pas encore défini) ;

- l'axiome (4.9) ne vaut et n'est nécessaire que pour ∇_a (et non pas ∇, qui n'est pas encore défini) ;

- les axiomes (4.10-4.11) assurent qu'un attribut s'interprète par une fonction totale sur l'union des interprétations des classes qui introduisent l'attribut :

  (4.14) $$\mathrm{dom}(P^{\mathcal{I}}) = \bigcup_{\alpha \in \overline{Def}(P)} \alpha^{\mathcal{I}}$$

  L'axiome (4.10) décrit les conditions d'appartenance au domaine des attributs dans la formulation restreinte que nous avons adoptée et qu'il faudrait adapter aux formulations plus générales. Compte-tenu de la discussion sur le caractère partiel de $type_P$, il est plus juste de décomposer cette règle en deux règles concernant *type* et *card*, valables uniquement pour les classes qui les explicitent :

  (4.10-1) $$o \in \alpha^{\mathcal{I}}, \alpha \in \mathrm{dom}(type_P) \Longrightarrow P^{\mathcal{I}}(o) \subseteq type_P(\alpha)^{\mathcal{I}}$$

  (4.10-2) $$o \in \alpha^{\mathcal{I}}, \alpha \in \mathrm{dom}(card_P) \Longrightarrow |P^{\mathcal{I}}(o)| \in card_P(\alpha)$$

  En ce qui concerne l'axiome (4.11), sa présence est discutable et sera discutée (cf. paragraphe 4.3.3, page 81).

NOTE *4.10* *Structure sémantique, modèle ou interprétation*

Les notions de *structure sémantique*, *modèle* et *interprétation* sont voisines et les termes souvent synonymes ou interchangeables d'un auteur à un autre : nous avons pris le parti de désigner par *structure sémantique* un « type », constitué d'un domaine et des quelques fonctions associées à ce domaine, par *modèle* une structure sémantique vérifiant des propriétés particulières, et par *interprétation* les fonctions (ou leurs images) qui associent aux structures du langage des éléments du domaine.

NOTE *4.11* *Hypothèse du nom unique*

L'injectivité de $\mathcal{E}_{\mathcal{O}}$ correspond à l'hypothèse du nom unique (*unique name assumption*) admise par de nombreuses LT ([Nebel, 1990] par ex.) mais que d'autres, comme [Baader et Hanschke, 1991], refusent. Dans une interprétation injective, l'élément de $\mathcal{O}$ est un *nom* non ambigu.

Si ce n'est plus injectif, l'élément de $\mathcal{O}$ est assimilable à une constante de Skolem. La non injectivité entraîne la possibilité de la *coréférence* au niveau extensionnel qui a été étudiée dans la mouvance des techniques de *point de vue* [Ferber et Volle, 1988; Volle, 1989; Ferber, 1989].

L'injectivité peut aussi se comprendre comme un critère d'*adéquation* du SC au modèle [Hautamäki, 1986].

Subsomption, incompatibilité, incohérence et inconsistance

On peut maintenant définir formellement diverses notions extensionnelles dont la présentation n'a été qu'intuitive :

- les relations de subsomption et d'incompatibilité extensionnelle $\trianglelefteq$ et ∇ se définissent en symétrisant les axiomes (4.8) et (4.9) et en les quantifiant universellement sur tous les modèles du SC :

  (4.15) $$\alpha \trianglelefteq \beta \overset{\mathrm{def}}{\Longleftrightarrow} \forall \mathcal{M} : \alpha^{\mathcal{I}} \subseteq \beta^{\mathcal{I}}$$

  (4.16) $$\alpha \nabla \beta \overset{\mathrm{def}}{\Longleftrightarrow} \forall \mathcal{M} : \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} = \emptyset$$

- une classe *incohérente* est une classe dont l'extension est vide dans tous les modèles : c'est donc une classe subsumée extensionnellement par $\perp$;
- une description du monde est *inconsistante* si elle n'a aucun modèle.

Les notions d'incohérence et d'inconsistance, qui sont relativement synonymes dans le langage commun, sont donc clairement distinguées. Ces deux notions sont reliées formellement, par le fait que toute instance d'une classe incohérente rend la description du monde inconsistante, et moralement, par la connotation d'erreur qu'elles induisent. Mais, sur ce dernier point, leur différence est importante : la portée d'une incohérence est locale, propre à une classe, ce qui n'invalide pas l'ensemble des connaissances. En revanche, une inconsistance a une portée globale et elle prive de sens la totalité des connaissances. Dans la suite, nous utiliserons toujours ces deux termes dans leur sens formel, en évitant tout usage « commun ».

NOTE *4.12* *Propriété intensionnelle ou extensionnelle ?*

Les notions de subsomption et incompatibilité extensionnelles, ainsi que d'incohérence, sont définies à partir des modèles d'une description du monde (composante extensionnelle) bien qu'elles ne concernent apparemment que la composante intensionnelle. Dans les bons cas de figure, la composante extensionnelle est une *extension conservative* de la composante intensionnelle ce qui assure que ces notions ne dépendent pas de la description du monde [Nebel, 1990].

Inversement, l'inconsistance est une notion purement extensionnelle, qui ne peut pas qualifier (toujours dans les bons cas) une composante intensionnelle. L'inconsistance de la composante intensionnelle se caractérise néanmoins simplement : elle survient quand aucune description du monde ne peut être consistante. Aucune instance n'est alors possible et toutes les classes sont incohérentes.

Objet inconsistant ou paradoxal

On peut chercher à localiser dans un objet la cause d'une inconsistance. C'est un exercice un peu vain car les causes sont multiples, mais on peut au moins les circonscrire. Ainsi, les causes de l'inconsistance de $\mathcal{W}$ sont constituées du sous-ensemble de descriptions élémentaires

$$\textit{inconsist}(\mathcal{W}) \stackrel{\text{def}}{=} \{d \in \mathcal{W} \mid \exists \mathcal{W}' \subset \mathcal{W} : \mathcal{W}' \text{ consistant } \& \ \mathcal{W}' \cup \{d\} \text{ inconsistant}\} \tag{4.17}$$

Un objet o est *inconsistant dans* $\mathcal{W}$ s'il est l'objet de l'une de ces descriptions, c'est-à-dire s'il existe $(\alpha\ o)$ ou $(P\ o\ o')$ dans $\textit{inconsist}(\mathcal{W})$. On verra plus loin que la définition d'une version intensionnelle de l'inconsistance d'un objet est beaucoup plus facile et précise (cf. paragraphe 6.3.1, page 129). On pourrait penser qu'un objet est inconsistant ssi la description de l'un de ses attributs est en contradiction avec ses classes. Cette dernière propriété est en fait assez difficile à définir : on dira que $P_{\mathcal{W}}(o)$ est en contradiction avec $Isa_{\mathcal{W}}(o)$ si, soit la cardinalité est incompatible :

$$|P_{\mathcal{W}}(o)| > \max \bigcap_{\beta \in Isa_{\mathcal{W}}(o) \cap Def(P)} card(\beta, P)$$

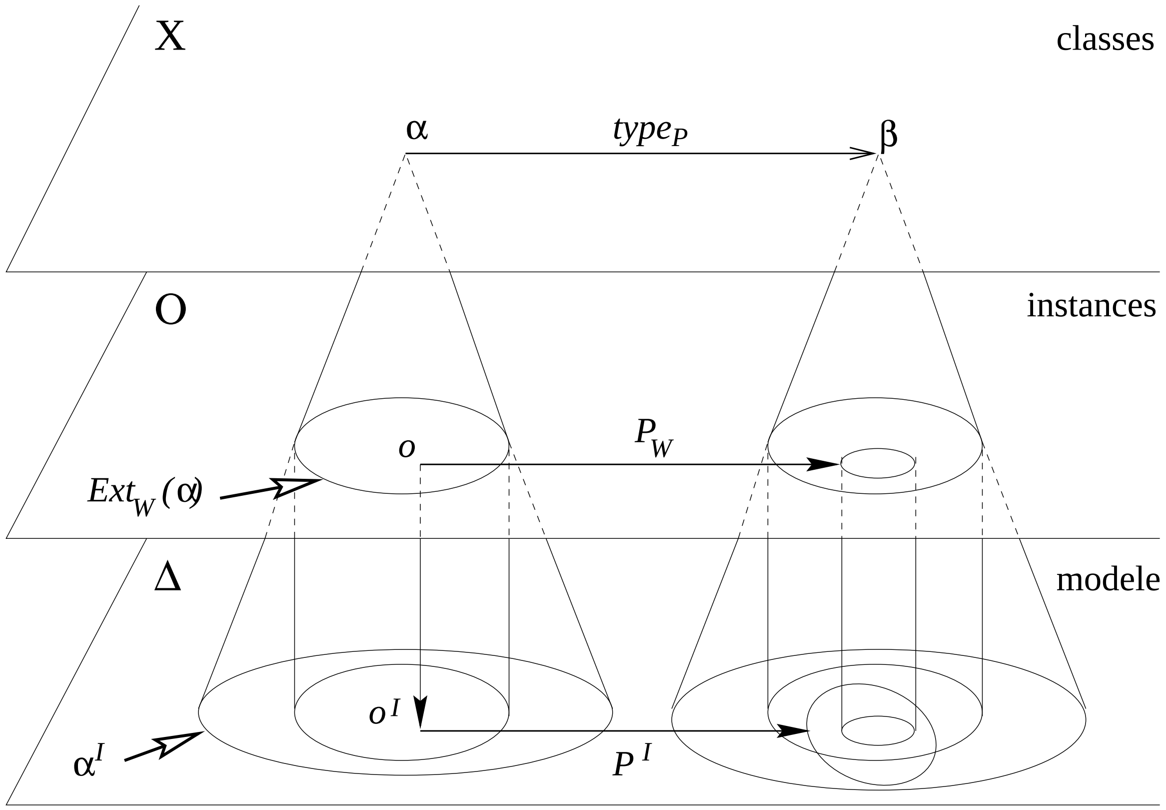

FIG. 4.1 – *Composantes intensionnelle et extensionnelle, et interprétation.*

soit le type d'une des valeurs est incompatible :

$$(4.18)\quad \exists o' \in P_{\mathcal{W}}(o) : o' \text{ inconsistant dans } \mathcal{W} \cup \{(type(\beta, P)\ o') \mid \beta \in Isa_{\mathcal{W}}(o) \cap Def(P)\}$$

Comme $\mathcal{W}$ est déjà inconsistant, il n'est pas possible de se reposer sur l'absence de modèle dans lequel o' serait une instance de $type(\beta, P)$. Il faut donc faire de o' un objet inconsistant dans la description du monde (éventuellement $\mathcal{W}$ elle-même : il suffit que l'une des valeurs soit un objet inconsistant dans la description du monde courante) où il est une instance de cette classe.

Malgré cette définition très générale, l'équivalence est fausse : de nombreux exemples d'objets paradoxaux l'ont montré. Mais cela permet de donner une définition abstraite des objets paradoxaux : un objet est *paradoxal* s'il est inconsistant alors que tous ses attributs sont compatibles avec ses classes[11].

11. Cette définition n'arrive pas en fait à cerner suffisamment le paradoxe : ainsi, dans l'exemple de l'`Equipe-moderne` (figure 5.2) rajouter `(Man MARY)` entraîne une inconsistance, ce qui rend `MARY` inconsistant(e), mais cette inconsistance ne présente aucun caractère paradoxal.

La notion de conséquence

On arrive alors classiquement à la notion de conséquence valide d'une description du monde :

- la structure sémantique $\mathcal{M}$ satisfait la description d, noté $\models_{\mathcal{M}} d$, qui correspond aux versions symétriques des axiomes (4.12-4.13) ci-dessus ;

$$\models_{\mathcal{M}} (\alpha\ o) \iff o^{\mathcal{I}} \in \alpha^{\mathcal{I}} \tag{4.19}$$

$$\models_{\mathcal{M}} (P\ o_1\ o_2) \iff {o_2}^{\mathcal{I}} \in P^{\mathcal{I}}({o_1}^{\mathcal{I}}) \tag{4.20}$$

  La notation s'étend à un ensemble de descriptions : $\models_{\mathcal{M}} \mathcal{W}$ exprime ainsi le fait que $\mathcal{M}$ est un modèle de $\mathcal{W}$.

- tous les modèles de $\mathcal{W}$ satisfont d, noté $\mathcal{W} \models d$: d est une *conséquence valide* de $\mathcal{W}$.

$$\mathcal{W} \models d \iff \forall \mathcal{M} : \ \models_{\mathcal{M}} d \tag{4.21}$$

  Si $\mathcal{W}$ n'a pas de modèle — $\mathcal{W}$ est *inconsistant* —, toute description en est une conséquence valide, comme en logique classique.

On utilisera la notation $\not\models$ pour la négation des notions précédentes.

Une description du monde $\mathcal{W}$ est *fermée* par $\models$ ssi $\mathcal{W} = \{d \mid \mathcal{W} \models d\}$.

En pratique, la sémantique d'un SC se projette sur la description du monde pour donner quelques propriétés que doit vérifier toute description du monde pour ne pas être inconsistante, par exemple :

$$\alpha \trianglelefteq_a \beta \Longrightarrow Ext_{\mathcal{W}}(\alpha) \subseteq Ext_{\mathcal{W}}(\beta)$$
$$o \in Ext_{\mathcal{W}}(\alpha)\ \ \&\ \ P \in \mathcal{P}_\alpha \Longrightarrow |P_{\mathcal{W}}(o)| \leq \max(card(\alpha, P))$$

Mais les propriétés de la description du monde sont finalement assez limitées et la sémantique se traduit principalement par le fait que, intuitivement, la description du monde peut être complétée pour donner un modèle. En fait, ce n'est vrai que si le SC a la propriété du *modèle fini* : en effet une description du monde est finie mais ce n'est pas forcément le cas d'un modèle et l'existence d'un modèle fini est une propriété éventuelle d'un SC. Lorsqu'une description du monde est son propre modèle, elle est fermée par $\models$. Mais il n'y a pas de raison que la fermeture de $\mathcal{W}$ par $\models$ soit un modèle de $\mathcal{W}$: ce serait sinon le seul modèle minimal de $\mathcal{W}$.

NOTE *4.13* *Modèle canonique et domaine modélisé*

Un *modèle canonique* est un modèle $\mathcal{M}$ dont toutes les descriptions qu'il satisfait sont des conséquences valides [Nebel, 1990] :

$$\models_{\mathcal{M}} d \iff \mathcal{W} \models d \tag{4.22}$$

Un *modèle canonique* est un *unique modèle minimal* (la réciproque est fausse) : mais toute description du monde, même parfaitement cohérente, n'a pas forcément de modèle canonique. L'existence d'un modèle canonique rend la théorie *catégorique*. Un *modèle minimal* peut être considéré comme *une* solution du système classificatoire, le problème pouvant alors être de trouver une solution, toutes les solutions, etc., comme dans les CSP. Intuitivement, tout modèle peut être obtenu en étendant un modèle minimal. Inversement, de tout modèle $\mathcal{M}$ *fini* de $\mathcal{W}$, se déduit naturellement une description du monde $\mathcal{W}_{\mathcal{M}}$ contenant $\mathcal{W}$, dont $\mathcal{M}$ est un modèle minimal, voire canonique. C'est ce qui permet d'identifier une description à un modèle, en parlant par exemple des modèles (ou descriptions du monde) qui sont leur propre modèle. Un modèle, c'est-à-dire la « signification » du système classificatoire, peut donc être vu comme une description du monde complète.

Lorsque l'on prétend s'intéresser au *domaine modélisé* (cf. note 6), on raisonne ainsi dans un modèle particulier. Le seul problème est que ce modèle $\mathcal{M}$, présumé unique, n'est connu que par une description du monde partielle $\mathcal{W}$, ce qui exclut *a priori* toute inférence valide reposant sur $\models_{\mathcal{M}}$ et non sur $\models$, à moins que la théorie soit catégorique, ce qui est peu probable.

NOTE *4.14* *Conséquence intensionnelle*

Dans tout ce qui précède, il faut sous-entendre une composante intensionnelle $\mathcal{H}$ donnée. Il faudrait donc modifier les notations, soit pour indexer $\models$ par $\mathcal{H}$ ($\models_{\mathcal{H}}$) soit pour rajouter $\mathcal{H}$ en partie gauche :

$$\mathcal{H}; \mathcal{W} \models d$$

Cette dernière solution a l'avantage de permettre une extension immédiate de la notation à la notion de conséquence dans la composante intensionnelle, la description d pouvant être alors de la forme $\alpha \trianglelefteq \beta$ ou $\alpha \nabla \beta$.

4.2.3 Retour sur les no(ta)tions Ext, Isa, $Attr$ et P

La sémantique qui précède a vidé de toute signification formelle les notations intuitives comme $Isa(o)$, $Ext(\alpha)$, $Attr(o)$ ou $P(o)$. Il faut en fait choisir, pour toutes ces notions, entre trois versions distinctes, suivant que l'on considère un modèle, tous les modèles ou la description du monde. Ainsi, pour les classes de o :

$$Isa_{\mathcal{M}}(o) \stackrel{\text{def}}{=} \{\alpha \mid \ \models_{\mathcal{M}} (\alpha\ o)\} \tag{4.23}$$

$$Isa_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{\alpha \mid \exists \beta \trianglelefteq_a \alpha : (\beta\ o) \in \mathcal{W}\} \tag{4.24}$$

$$Isa^{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{\alpha \mid \mathcal{W} \models (\alpha\ o)\} = \bigcap_{\mathcal{M}} Isa_{\mathcal{M}}(o) \tag{4.25}$$

suivant qu'il s'agit des classes de o dans un modèle $\mathcal{M}$ donné, des classes de o dans la description du monde $\mathcal{W}$[12], ou enfin des classes de o dans tous les modèles de $\mathcal{W}$ (figure 4.2).

Les images de ces trois fonctions sont des idéaux et vérifient, pour tout modèle $\mathcal{M}$ de $\mathcal{W}$:

$$\underline{Isa}_{\mathcal{M}}(o) \preceq_a \underline{Isa}^{\mathcal{W}}(o) \preceq_a \underline{Isa}_{\mathcal{W}}(o) \preceq_a \{\top\} \tag{4.26}$$

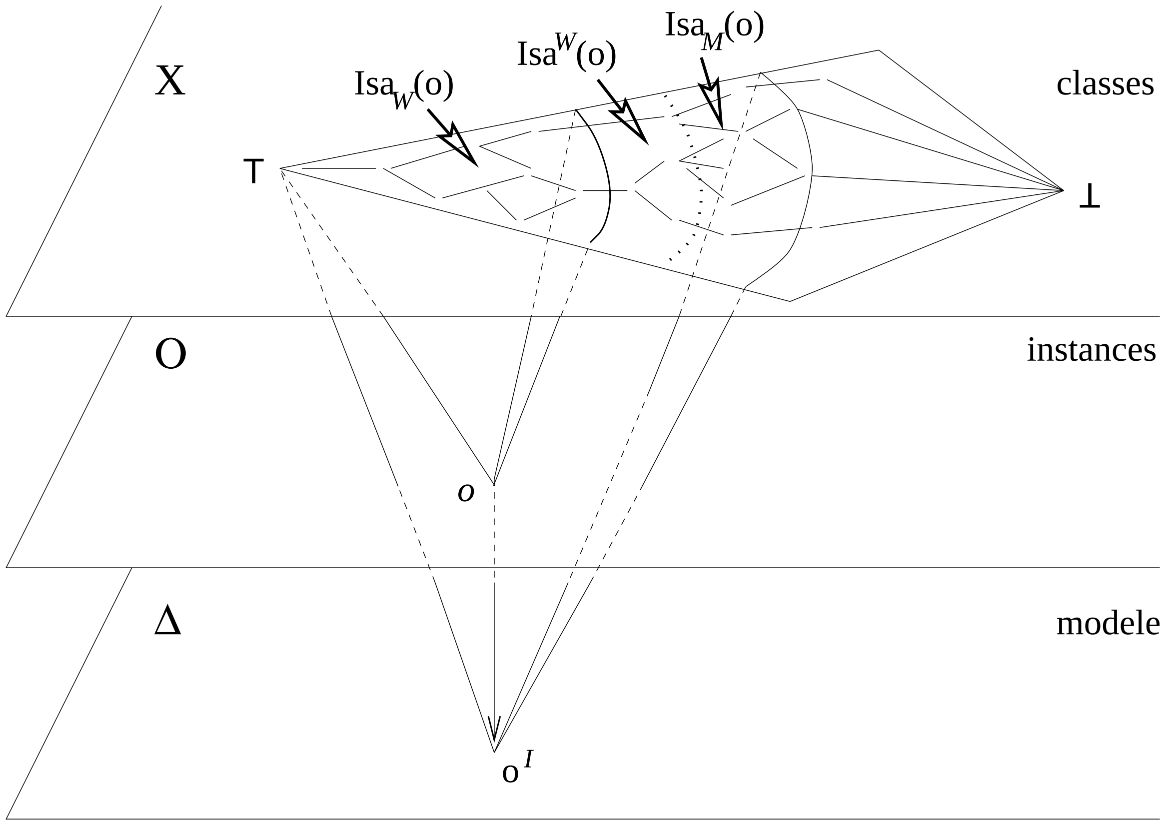

FIG. 4.2 – *Les classes d'un objet, pour une description du monde, un modèle ou, entre les deux, tous les modèles.*

Pour $Attr$, la définition est tout à fait analogue à celle de Isa :

(4.27) $$Attr_{\mathcal{M}}(o) \stackrel{\text{def}}{=} \bigcup_{\alpha \in Isa_{\mathcal{M}}(o)} \mathcal{P}_\alpha$$

(4.28) $$Attr^{\mathcal{W}}(o) \stackrel{\text{def}}{=} \bigcup_{\alpha \in Isa^{\mathcal{W}}(o)} \mathcal{P}_\alpha = \bigcap_{\mathcal{M}} Attr_{\mathcal{M}}(o)$$

En revanche, pour Ext et P, si des équivalents sont possibles, ils sont limités par le fait que tout le domaine n'est pas représenté, et que ce domaine est *a priori* illimité, au sens où l'ensemble des objets représentés n'est pas borné. Ainsi, si $P_{\mathcal{M}}(o)$ désigne la valeur de l'attribut P de l'objet o dans le modèle $\mathcal{M}$,

(4.29) $$P_{\mathcal{M}}(o) \stackrel{\text{def}}{=} \{o' \in \mathcal{O} \mid \ \models_{\mathcal{M}} (P\ o\ o')\} = [P^{\mathcal{I}}(o^{\mathcal{I}})]^{-\mathcal{I}}$$

(4.30) $$P^{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{o' \in \mathcal{O} \mid \mathcal{W} \models (P\ o\ o')\} = \bigcap_{\mathcal{M}} P_{\mathcal{M}}(o)$$

Seuls figurent, dans $P_{\mathcal{M}}(o)$ ou $P^{\mathcal{W}}(o)$, les éléments du domaine effectivement représentés. De même, pour les fonctions $Ext_{\mathcal{M}}$, $Ext^{\mathcal{W}}$ et $Ext_{\mathcal{W}}$ qui se définissent symétriquement à Isa :

(4.31) $$Ext_{\mathcal{M}}(\alpha) \stackrel{\text{def}}{=} \{o \in \mathcal{O} \mid \ \models_{\mathcal{M}} (\alpha\ o)\} = [\alpha^{\mathcal{I}}]^{-\mathcal{I}}$$

12. Notons que cette nouvelle définition tient compte des passerelles.

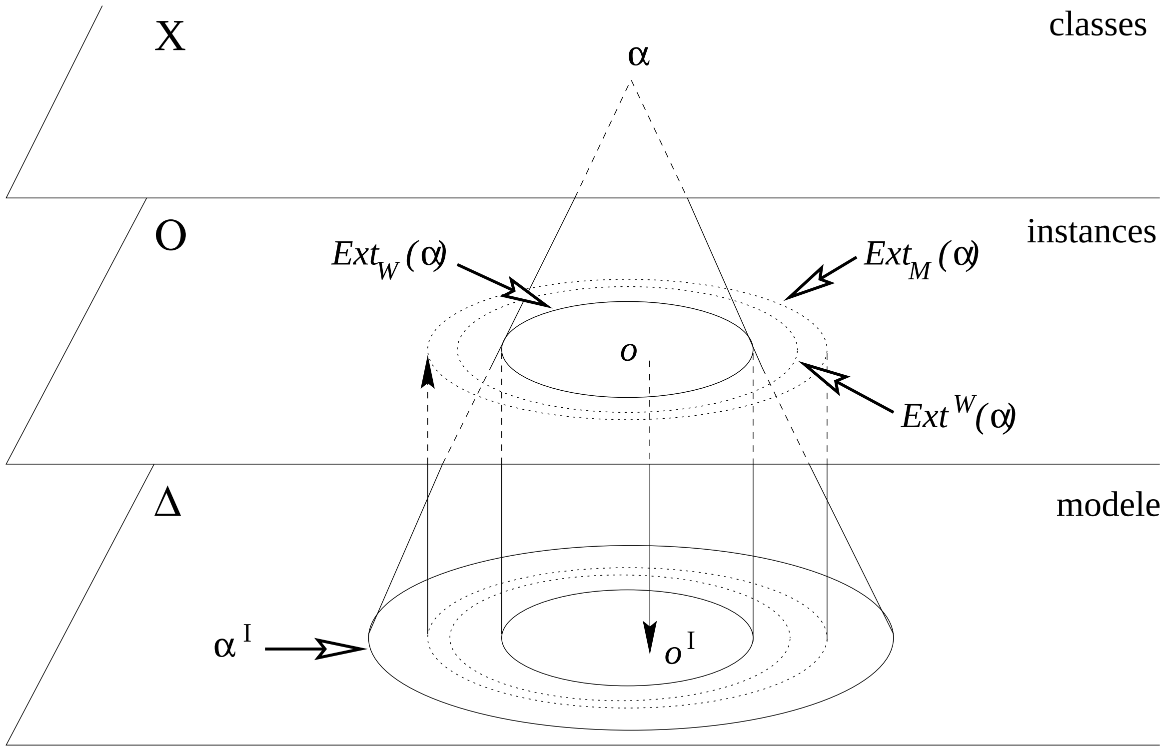

FIG. 4.3 – *Les instances d'une classe, pour une description du monde, un modèle ou, entre les deux, tous les modèles.*

$$Ext_{\mathcal{W}}(\alpha) \stackrel{\text{def}}{=} \{o \in \mathcal{O} \mid (\beta\ o) \in \mathcal{W}\ \&\ \alpha \trianglelefteq_a \beta\} \tag{4.32}$$

$$Ext^{\mathcal{W}}(\alpha) \stackrel{\text{def}}{=} \{o \in \mathcal{O} \mid \mathcal{W} \models (\alpha\ o)\} = \bigcap_{\mathcal{M}} Ext_{\mathcal{M}}(o) \tag{4.33}$$

NOTE *4.15* *Autres no(ta)tions*

On peut enfin introduire les fonctions $Card_{\mathcal{W}}$, $Card_{\mathcal{W}}$ et $Card^{\mathcal{M}}$ pour désigner respectivement le cardinal de la valeur de l'attribut P de l'objet o, dans $\mathcal{W}$, le minimum de ce cardinal dans tous les modèles, ou enfin ce cardinal dans le modèle $\mathcal{M}$:

$$Card_{\mathcal{W}}(P,o) \stackrel{\text{def}}{=} |P_{\mathcal{W}}(o)| \tag{4.34}$$

$$Card_{\mathcal{M}}(P,o) \stackrel{\text{def}}{=} |P^{\mathcal{I}}(o^{\mathcal{I}})| \geq |P_{\mathcal{M}}(o)| \tag{4.35}$$

$$Card^{\mathcal{W}}(P,o) \stackrel{\text{def}}{=} \min\{Card_{\mathcal{M}}(P,o) \mid \mathcal{M}\} \geq |P^{\mathcal{W}}(o)| \tag{4.36}$$

Ces divers ensembles de fonctions entretiennent les rapports naturels suivants :

$$Ext_{\mathcal{W}}(\alpha) \subseteq Ext^{\mathcal{W}}(\alpha) \subseteq Ext_{\mathcal{M}}(\alpha) \tag{4.37}$$

$$P_{\mathcal{W}}(o) \subseteq P^{\mathcal{W}}(o) \subseteq P_{\mathcal{M}}(o) \tag{4.38}$$

$$Attr_{\mathcal{W}}(o) \subseteq Attr^{\mathcal{W}}(o) \subseteq Attr_{\mathcal{M}}(o) \tag{4.39}$$

$$Card_{\mathcal{W}}(P,o) \leq Card^{\mathcal{W}}(P,o) \leq Card_{\mathcal{M}}(P,o) \tag{4.40}$$

Pour finir, on peut introduire les notations absolues $Isa_{\forall\mathcal{M}}(o)$, $Ext_{\forall\mathcal{M}}(\alpha)$, $P_{\forall\mathcal{M}}(o)$, $Card_{\forall\mathcal{M}}(P,o)$ et $Attr_{\forall\mathcal{M}}(o)$ utilisable lorsque la valeur correspondante est la même dans tous les modèles :

$$\forall\mathcal{M}: \begin{cases} P_{\forall\mathcal{M}}(o) = P_{\mathcal{M}}(o) \\ Isa_{\forall\mathcal{M}}(o) = Isa_{\mathcal{M}}(o) \\ Attr_{\forall\mathcal{M}}(o) = Attr_{\mathcal{M}}(o) \\ Card_{\forall\mathcal{M}}(P,o) = Card_{\mathcal{M}}(P,o) \\ Ext_{\forall\mathcal{M}}(\alpha) = Ext_{\mathcal{M}}(\alpha) \end{cases} \tag{4.41}$$

Notons que ces notations pour P et Ext sont assez délicates à interpréter puisque la valeur d'un attribut ou l'extension d'une classe dans un modèle ne sont pas définies au niveau de la représentation (dans le SC), mais uniquement par leur interprétation, dans le domaine (figure 4.3) : certains des éléments du domaine ne sont pas représentés. C'est la raison pour laquelle on a pu les définir en fonction de l'inverse de $\mathcal{I}$, $-\mathcal{I}$, qui peut être considérée comme la fonction de *représentation*[13]. Bien sûr, il n'est pas possible de comparer deux éléments de domaines différents, s'ils ne sont pas tous deux représentés : c'est leur représentation que l'on compare. Une notation comme $P_{\mathcal{W}}(o) = P_{\forall\mathcal{M}}(o)$ ne dit rien des éléments non représentés et ne signifie donc pas que :

$$\forall\mathcal{M}, P^{\mathcal{I}}(o^{\mathcal{I}}) = [P_{\mathcal{W}}(o)]^{\mathcal{I}^*}$$

ce qui correspondrait à une parfaite commutation du diagramme du paragraphe 4.2.2, mais simplement qu'aucun autre élément actuellement représenté ne fait partie de la valeur dans aucun modèle. Plus généralement, on s'en sort en identifiant les modèles finis à des descriptions du monde : tout élément du domaine est alors représenté.

4.2.4 sc particuliers

À ce cadre sémantique général peuvent s'ajouter les spécificités des diverses familles de SC, ou leurs propriétés particulières.

Propriétés spécifiques du sc

Les propriétés structurelles du système classificatoire telles que l'exhaustivité, l'exclusivité ou l'univocité, peuvent être relatives à la description du monde (ou à l'ensemble $\mathcal{O}$ des instances affirmées à un moment donné) ou à un modèle du SC. On pourrait donc appeler *modèle exhaustif* (resp. *univoque, exclusif*) un modèle du SC qui vérifie les diverses propriétés en question, un SC ayant la propriété si et seulement si tous ses modèles l'ont. Les règles suivantes s'ajouteront donc, respectivement pour l'exhaustivité, l'univocité ou l'exclusivité :

$$\emptyset \neq \alpha^{\mathcal{I}} \subseteq \beta^{\mathcal{I}} \Longrightarrow \beta^{\mathcal{I}} = \bigcup_{\gamma^{\mathcal{I}} \subset \beta^{\mathcal{I}}} \gamma^{\mathcal{I}} \tag{4.42}$$

$$o \in \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \Longrightarrow \exists\gamma : o \in \gamma^{\mathcal{I}} \subseteq \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \tag{4.43}$$

13. Noter que dans $(\alpha^{\mathcal{I}})^{-\mathcal{I}}$, l'usage uniforme de $\mathcal{I}$ masque le fait qu'il ne s'agit pas, dans ses 2 occurrences, de la même fonction : c'est en fait $\mathcal{E}_{\mathcal{O}}^{-1}(\mathcal{E}_{\mathcal{X}}(\alpha))$.

$$(4.44) \qquad \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \neq \emptyset \Longrightarrow \alpha^{\mathcal{I}} \subseteq \beta^{\mathcal{I}} \text{ ou } \beta^{\mathcal{I}} \subseteq \alpha^{\mathcal{I}}$$

On qualifie de même les descriptions du monde, en remplaçant $\mathcal{I}$ par $Ext_{\mathcal{W}}$.

Il faut noter que l'exclusivité dans un modèle implique l'exclusivité dans la description du monde, ce qui n'est pas le cas pour les deux autres propriétés (cf. le caractère *provisoire* ou *définitif* des propriétés spécifiques, au paragraphe 3.3). Le point à souligner est qu'il s'agit d'abord de propriétés du modèle, avant d'être, éventuellement, des propriétés du SC valable pour un, des ou tous les modèles.

Lorsqu'un SC est *exhaustif* (resp. *univoque, exclusif*), il est équivalent de rajouter l'axiome correspondant 4.42 (resp. 4.43, 4.44) ou de ne considérer que les modèles exhaustifs (resp. univoques, exclusifs).

NOTE *4.16* *Variantes*

Les axiomes (4.42-4.44) reposent implicitement sur la subsomption extensionnelle La formulation suivante, qui repose explicitement sur la spécialisation, est sans doute plus fidèle :

$$(4.45) \qquad \emptyset \neq \alpha^{\mathcal{I}} \;\&\; \alpha \prec \beta \Longrightarrow \beta^{\mathcal{I}} = \bigcup_{\gamma \prec \beta} \gamma^{\mathcal{I}}$$

$$(4.46) \qquad o \in \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \Longrightarrow \exists \gamma : o \in \gamma^{\mathcal{I}} \;\&\; \gamma \preceq \alpha \;\&\; \gamma \preceq \beta$$

$$(4.47) \qquad \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \neq \emptyset \Longrightarrow \alpha \preceq \beta \text{ ou } \beta \preceq \alpha$$

Ce n'est pas forcément équivalent.

N.B. Le fait que l'on puisse trouver plusieurs formalisations non équivalentes d'une propriété « intuitive » n'a rien de répréhensible, bien au contraire : c'est la preuve de la nécessité d'une sémantique formelle. La différence de formalisation repose, en l'occurrence, sur la possibilité de formuler les propriétés à partir de notions intuitives (la spécialisation) ou formelles (la subsomption). Bien sûr, la formulation intuitive ne peut pas se servir des notions formelles, mais cela ne signifie pas que la « bonne » formulation s'obtient en remplaçant les notions intuitives par les notions formelles correspondantes.

Le cas particulier des lpo

Pour un LPO ne sachant pas faire migrer ses instances, il faut aussi la réciproque des axiomes 4.12 ou 4.12' :

$$(4.12'') \qquad \alpha \in Isa_{\mathcal{W}}(o) \iff o^{\mathcal{I}} \in \alpha^{\mathcal{I}}$$

l'appartenance à une classe doit alors toujours être affirmée ; c'est la condition *sine qua non* pour accepter C++ ou SMALLTALK dans le club des SC.

De plus, si l'on ne veut pas traiter des valeurs d'attribut partiellement connues (c'est-à-dire si la valeur est totalement connue ou totalement inconnue, comme dans beaucoup de systèmes de RCO ou en programmation logique, ce qui correspond à une hypothèse du *monde clos* sur la valeur des attributs) il faut symétriser les axiomes 4.13 et 4.13', ce qui donne :

$$(4.13'') \qquad [P_{\mathcal{W}}(o)]^{\mathcal{I}^*} = P^{\mathcal{I}}(o^{\mathcal{I}})$$

Les types de base

On n'a pas traité le cas où le système classificatoire inclut les types de base, par exemple les entiers. Dans ce cas, il faut faire l'hypothèse qu'une partie des objets, par exemple les instances de la classe `entier`, s'interprètent dans $\mathbf{N}$. La classe `entier` est primitive et n'a aucun attribut (ou rôle), ce qui ne pose pas de problème particulier. Dans tous les cas, le traitement des types de base n'est pas si simple que ça, dans les LT tout au moins.

Dans CLASSIC (cf. annexe B.1, page 191), le SC restreint aux types est en particulier *exclusif* (donc *univoque*) et « infiniment » non *exhaustif*, au sens où chaque type a une infinité d'instances propres (ceci pour éviter toute tentation de résolution combinatoire [14].

Dans [Baader et Hanschke, 1991], les domaines sont restreints par la décidabilité de certain problème de satisfaisabilité, ce qui oblige les auteurs à invoquer Tarski pour accepter le domaine des nombres réels, ou le dixième problème de Hilbert (équations diophantiennes) pour refuser les entiers rationnels. Dans tous les cas, les sous-domaines sont fournis une fois pour toutes avec le domaine, sous la forme de prédicats unaires.

Il est remarquable que, de leur côté, les systèmes de RCO acceptent les types de base avec une totale naïveté [15] dont on ne sait si elle est justifiée. Il faudrait regarder si la classification d'instances seule (à l'exclusion de la subsomption) autorise cette naïveté.

La sémantique des lt

Nebel (1990) distingue la sémantique des classes de la sémantique des instances, en séparant les formalismes terminologique $\mathcal{TF}$ (axiomes 4.8–4.11) et assertionnel $\mathcal{AF}$ (axiomes 4.12–4.13) : il ne semble pas que cela change quoi que ce soit, mais cela lui permet, quand même, de montrer que l'ensemble $\mathcal{TF}+\mathcal{AF}$ est une *extension conservative* de $\mathcal{TF}$: une description du monde $\mathcal{W}$ ne permet pas de déduire des relations de subsomption non déductibles de la terminologie seule. En d'autres termes, rajouter des instances n'a pas d'effet sur les classes [16].

En revanche, la sémantique des LT est beaucoup plus précise que notre schéma général, parce qu'elles offrent des *constructeurs* des concepts qui permettent d'assurer des conditions nécessaires et suffisantes, donc des doubles implications. On peut décrire les constructeurs de *conjonction* (`and`), de restriction de valeur (`all`) et de cardinalité (`atleast` et `atmost`) ainsi :

$$(\texttt{and}\ \alpha\ \beta)^{\mathcal{I}} = \alpha^{\mathcal{I}} \cap \beta^{\mathcal{I}} \tag{4.48}$$

$$(\texttt{all}\ P\ \beta)^{\mathcal{I}} = \{o \in \Delta \mid P^{\mathcal{I}}(o) \subseteq \beta^{\mathcal{I}}\} \tag{4.49}$$

14. Ce choix démontre une certaine perversité : la complétude est acquise au prix de l'impossibilité d'exprimer des choses simples qui pourrait pourtant donner lieu à un algorithme complet, mais trop complexe. Au premier rang des expressions souhaitables mais interdites, les intervalles d'entiers, qui sont pourtant des prototypes parfaits de domaines : l'`age` d'un `Homme` appartient à l'intervalle `[0 120]`, celui d'un `Enfant` à `[0 17]`.

15. À l'exception du travail de Capponi (1995), dont je ne vois pas comment il s'insère là-dedans.

16. Ce n'est plus le cas dès que la composante intensionnelle contient des constructeurs comme `fills` ou `one-of` qui font référence à la composante extensionnelle, comme en CLASSIC (cf. paragraphe 2.2.2, page 28).

$$(\texttt{atleast}\ P\ n)^{\mathcal{I}} = \{o \in \Delta \mid |P^{\mathcal{I}}(o)| \geq n\} \tag{4.50}$$

$$(\texttt{atmost}\ P\ n)^{\mathcal{I}} = \{o \in \Delta \mid |P^{\mathcal{I}}(o)| \leq n\} \tag{4.51}$$

Ces nouvelles équations permettent, d'une certaine manière, de remplacer l'inclusion de l'axiome (4.10) par une égalité, `all` exprimant *type*, `atleast` et `atmost`, *card*, et `and` construisant une classe définie par agrégation de ses différents attributs et super-classes.

Enfin, comme nous le discuterons plus loin (cf. paragraphe 4.3.3, page 81), la sémantique usuelle des LT ne suppose pas l'axiome (4.11).

Voir aussi, en annexe B.5, la sémantique de tous les constructeurs des différents langages. Le problème plus général de la définition des classes sera abordé au paragraphe 4.5.

4.3 Quelques hypothèses

Le domaine des interprétations aussi bien que le domaine des attributs peuvent donner lieu à diverses hypothèses plus ou moins intuitives ou réalistes.

4.3.1 Domaine universel et minimalité des domaines

La construction d'un domaine universel apparaît intuitivement aisée, même si le problème se complique rapidement. Il suffirait de considérer la combinatoire de toutes les valeurs d'attribut admissibles, quelque chose comme :

$$\begin{aligned} Ext(\alpha) &\simeq \prod_{P \in \mathcal{P}_\alpha} Dom(\alpha, P) \\ &= \prod_{P \in \mathcal{P}_\alpha} \{A \subseteq Ext(type(\alpha, P)) \mid |A| \in card(\alpha, P)\} \end{aligned} \tag{4.52}$$

C'est cette expression intuitive du domaine universel qui est mise en œuvre dans les différentes figures pour donner une représentation géométrique des intensions. Cette intuition des produits cartésiens est à la base des treillis des intensions, de $\mathcal{INT}$ ou des produits de SC de Euzenat (1994).

Une définition de ce genre n'est correcte en toute généralité qu'en l'absence de circuit dans la définition des classes [Nebel, 1990], et en faisant bien entendu l'hypothèse implicite de l'indépendance des attributs (cf. paragraphe 4.1.2, page 59) ainsi que de l'identité des indiscernables (cf. paragraphe 3.4.2, page 51)[17].

17. Le domaine Δ se définirait ainsi comme un point fixe, quelque chose comme $\Delta = \prod_{P \in \mathcal{P}} \Delta$, ou, de façon constructive, comme la limite d'une suite croissante de domaines [Coupey et Fouqueré, 1994; Coupey et Fouqueré, 1995] : bien entendu, encore faut-il que ce point fixe ou cette limite existe (vérifier par une équation aux cardinaux par exemple, pour ne pas réinventer « l'ensemble de tous les ensembles », cf. paragraphe 2.4.2, page 32). D'un point de vue sémantique, ce modèle universel doit se comprendre comme un unique modèle maximal assurant que toute inférence d'un certain type ($\trianglelefteq$ et ∇ par exemple) vraie dans ce modèle est valide. C'est donc un modèle canonique au sens de Nebel.

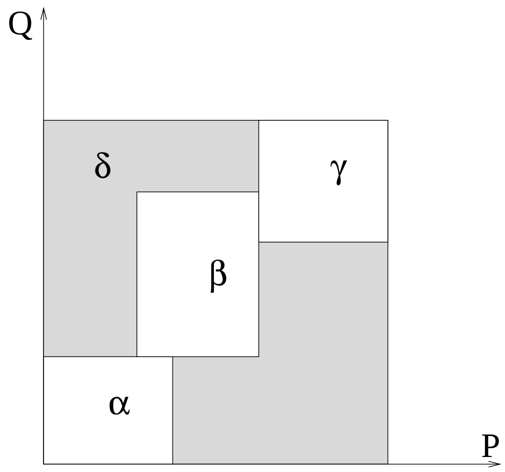

FIG. 4.4 – *L'intension d'une classe exhaustive δ avec deux attributs* `P` *et* `Q` *n'est pas forcément recouverte par celles de ses trois sous-classes α, β et γ. Les instances de δ non recouvertes sont des objets paradoxaux.*

L'inclusion au moins est vérifiée mais ce domaine universel peut contredire certaines propriétés comme l'exhaustivité d'une classe : si α n'a pas d'instance propre — elle est définie en extension par ses sous-classes, cf. paragraphe 3.4.2, page 52 —, son extension peut ne pas être définie par ce produit cartésien. Il s'ensuit alors que certaines valeurs du domaine universel, c'est-à-dire certaines instances de α, *a priori* bien formées, sont en fait impossibles : bien que respectant toutes les contraintes de la classe, elles n'ont d'interprétation dans aucun modèle. Ce sont des *objets paradoxaux*. La figure 4.4 montre comment le domaine d'une classe δ munie de deux attributs `P` et `Q` n'est pas recouvert par les domaines de ses trois sous-classes α, β et γ, pour peu que δ soit *localement exhaustive*.

Bien sûr, cela ne signifie pas qu'un domaine universel n'existe pas en cas d'exhaustivité, mais seulement qu'il ne se construit pas par de simples produits cartésiens. Dans le cas de l'exhaustivité, le domaine universel se construit par la réunion des produits cartésiens sur les puits de $\preceq$. Si ces puits sont deux à deux intensionnellement incompatibles, le domaine universel est parfaitement défini (les circularités mises à part). En fait, c'est sans doute le seul cas où la construction du domaine universel est sans problème : sinon, dès que $\alpha \prec \beta$ et que $\mathcal{P}_\alpha \supset \mathcal{P}_\beta$, le (une partie du) domaine universel de α se projette forcément naturellement sur celui de β ce qui pose le problème de la distinction entre les instances de ces deux classes qui ont les mêmes valeurs d'attributs pour tous les attributs de β :

$$o_1 \in Ext(\alpha), o_2 \in Ext(\beta) - Ext(\alpha), \forall P \in \mathcal{P}_\beta : P(o_1) = P(o_2)$$

C'est un argument en faveur de l'exhaustivité : o_2 n'existerait pas. C'est aussi un argument de [Ghelli et Orsini, 1991] pour proposer une sémantique à base de *relation d'équivalence partielle* (PER). Dans une telle sémantique, un élément d'un type s'interprète par une classe d'équivalence de la PER, et un sous-type, soit abandonne quelques classes d'équivalence (d'où le « partiel »), soit raffine la relation en partitionnant quelques classes (soit les deux) [Bruce et Longo, 1990]. Cette formalisation ne vérifie bien sûr pas l'inclusion des extensions (2.3).

Hypothèse de minimalité des domaines

L'*hypothèse de minimalité des domaines* est plus faible que la construction d'un domaine universel par produits cartésiens : elle consiste simplement à poser que toute valeur d'un domaine est possible, au sens où il peut exister un objet qui la réalise :

$$\forall \alpha \in \mathcal{X}, \forall P \in \mathcal{P}_\alpha, \forall x \in Dom(\alpha, P), \exists o \in Ext(\alpha) : P(o) = x \tag{4.53}$$

ou, plus simplement :

$$Dom(\alpha, P) = \{P(o) \mid o \in Ext(\alpha)\}. \tag{4.54}$$

En faisant abstraction de la façon dont les domaines sont exprimés, cette hypothèse ne fait perdre aucune généralité, puisqu'il suffirait sinon d'enlever x du domaine de P dans α.

Bien entendu, si le domaine D est exprimable, ce n'est sans doute plus le cas pour $D - \{x\}$. Il faut donc reformuler cette hypothèse de minimalité de la façon suivante : $Dom(\alpha, P)$ est le plus petit domaine exprimable qui contienne $\{P(o) \mid o \in Ext(\alpha)\}$. Sans tenir compte des cardinalités, cela signifie que $type(\alpha, P)$ est la classe la plus spécialisée dont l'extension contient toutes les valeurs admissibles de P. On a vu que cela pouvait poser des problèmes de conflit de type si la hiérarchie n'a pas une structure de treillis extensionnel.

Cette hypothèse revient à considérer que les domaines effectif et exprimé coïncident. Elle a ainsi l'avantage d'expliquer (en l'impliquant) l'inclusion des domaines (formule 2.13) et la résolution des conflits (formule 2.23) imposées par l'héritage (cf. paragraphe 2.1.3, page 17).

NOTE *4.17 Minimalité dans les modèles*

Ici, il s'agit d'une propriété du SC quantifiée existentiellement sur les modèles : toutes les valeurs ne peuvent pas être réalisées dans tous les modèles, il suffit d'un seul. L'hypothèse de minimalité des domaines traduit le fait que, en toute généralité, c'est-à-dire sous réserve d'impossibilité spécifique (due par exemple à une incompatibilité explicitement affirmée entre des sous-classes de la classe considérée), pour tout modèle $\mathcal{M}$ d'un SC, pour toute valeur v de $Dom(\alpha, P)$ dans ce modèle, il existe un modèle $\mathcal{M}'$ contenant [18] $\mathcal{M}$ qui réalise v pour une instance de α (ouf!) :

$$\forall \alpha \in \mathcal{X}, \forall P \in \mathcal{P}_\alpha, \forall \mathcal{M}, \forall v \in [type(\alpha, P)]^{\mathcal{I}}$$

18. On définira l'inclusion de modèles de la façon suivante. $\mathcal{M} = (\Delta, \mathcal{I})$ est inclus dans $\mathcal{M}' = (\Delta', \mathcal{I}')$ ssi $\exists h : \Delta \longrightarrow \Delta'$ injective et que, pour toute entité du SC :

$$\forall o \in \mathcal{O} : h(o^{\mathcal{I}}) = o^{\mathcal{I}'}$$
$$\forall \alpha \in \mathcal{X} : h(\alpha^{\mathcal{I}}) \subseteq \alpha^{\mathcal{I}'}$$
$$\forall o \in \mathcal{O}, \forall P \in Attr_{\mathcal{M}}(o) : h(P^{\mathcal{I}}(o^{\mathcal{I}})) \subseteq P^{\mathcal{I}'}(o^{\mathcal{I}'})$$

La troisième clause de cette définition est une contrainte d'inclusion qui porte sur les objets représentés, et non pas sur tous les objets du domaine. Pour avoir réellement une relation d'ordre sur les modèles, indépendamment de la description du monde, il faut la remplacer par :

$$\forall P \in \mathcal{P}, \forall o \in \mathrm{dom}(P^{\mathcal{I}}) : h(P^{\mathcal{I}}(o)) \subseteq P^{\mathcal{I}'}(h(o))$$

On obtient alors la définition de [Nebel, 1990].

il existe un modèle $\mathcal{M}'$ contenant $\mathcal{M}$ tel que

$$\exists o \in \alpha^{\mathcal{I}'}, v \in P^{\mathcal{I}'}(o).$$

Cette formulation ne tient pas compte des cardinalités, et il faudrait l'adapter en conséquence.

NOTE *4.18* *Relativité de la minimalité*
L'hypothèse de minimalité des domaines paraît difficilement tenable lorsque les moyens d'expression mis à la disposition de l'utilisateur sont confrontés à « sa » sémantique spécifique : ainsi, dans aucun système à ma connaissance, il n'est possible de se restreindre à des cardinalités paires, impaires ou premières, ce qui interdit de spécifier qu'un `Camion` a un nombre pair de `Roues`, compris entre 4 et 1000. Un utilisateur peut bien sûr ne s'intéresser qu'aux interprétations dans lesquelles un `Camion` a un nombre pair de `Roues`. Mais cela n'empêche pas l'existence de modèles où les camions en ont un nombre impair, et cela peut même interdire que les « bons » modèles — par exemple les modèles minimaux — puissent jamais respecter la parité (par exemple si seul un nombre impair de roues est connu) : la parité est une propriété *subjective* que le SC ne sait pas maintenir, à cause d'une *incomplétude syntaxique*.

C'est l'occasion de revenir sur l'*hypothèse de discrimination des intensions* émise au paragraphe 3.4.2 : s'il y a deux classes (*de facto* incompatibles) pour les `véhicules` à nombre de `roues` respectivement pair et impair, mais que les cardinalités ne sont pas exprimables autrement que par des intervalles, cette hypothèse s'applique exactement. Notons aussi qu'il ne servirait à rien de rajouter un attribut `nombre-de-roues`, dont la dépendance avec l'attribut `roues` n'arrangerait rien.

4.3.2 Maximalité des attributs et des intensions

En toute généralité, il y a des objets, des classes et des attributs (figure 2.1), et il est légitime de se demander ce qui est premier, de la poule ou de l'œuf. Une première question concerne le rapport entre l'intension des classes ($\mathcal{P}_\alpha$) et ce que l'on pourrait appeler l'intension de ses instances, c'est-à-dire $Attr(o)$. L'hypothèse de minimalité des domaines posait le même type de question aux domaines des attributs.

En fait, on a bien l'inclusion

$$\mathcal{P}_\alpha \subseteq \bigcap_{o \in Ext(\alpha)} Attr(o) \tag{4.55}$$

mais rien ne permet d'affirmer l'égalité. Dit autrement, l'intension d'un concept est formé des attributs qui le décrivent (ou le définissent), et non pas des attributs que toutes ses instances possèdent. On appellera *hypothèse de maximalité des intensions* l'hypothèse de l'égalité dans une version sémantique de l'inclusion précédente : elle signifie en fait que l'intersection sur tous les modèles de l'intersection de l'ensemble des attributs des instances de α est exactement $\mathcal{P}_\alpha$. Il n'existe pas d'attribut qui n'appartienne pas à $\mathcal{P}_\alpha$, tout en appartenant à toutes les instances de α dans tous les modèles :

$$\nexists P \in \mathcal{P} - \mathcal{P}_\alpha : \forall \mathcal{M}, \forall o \in \alpha^{\mathcal{I}} : P \in Attr_{\mathcal{M}}(o)$$

De même que l'hypothèse de minimalité des domaines implique l'inclusion des domaines (2.11), l'hypothèse de maximalité des attributs implique l'inclusion des ensembles d'attributs (2.7). Ces deux hypothèses forment conjointement une hypothèse de *maximalité des intensions* qui justifie en totalité l'héritage des attributs et des domaines. Elles ont aussi une conséquence simplificatrice importante sur les rapports entre $\trianglelefteq$ et $\trianglerighteq$:

Proposition. Sous les hypothèses de *minimalité des domaines* et de *maximalité des attributs*, $\alpha \trianglelefteq \beta \Longrightarrow \alpha \trianglerighteq \beta$ (cas 3.5).

Preuve. Supposons que $\alpha \trianglelefteq \beta$, c'est-à-dire que $Ext(\alpha) \subseteq Ext(\beta)$. Toute instance de α étant alors une instance de β, elle possède bien tous les attributs de β : donc $\mathcal{P}_\alpha \supseteq \mathcal{P}_\beta$ (hypothèse de maximalité des attributs). La suite vient aisément de l'hypothèse de minimalité des domaines : toute valeur de $Dom(\alpha, P)$ est réalisable par une instance de α, donc la valeur est aussi dans $Dom(\beta, P)$.

La maximalité des intensions est en fait une version faible de la définition des classes : si une classe est définie, tout objet qui vérifie son intension en est forcément instance (sauf si d'autres raisons rendent cet objet impossible, c'est-à-dire *paradoxal*).

Attributs supplémentaires et instanciation

Pourtant, cette hypothèse n'a rien de naturel : il est très facile de la mettre en défaut, même si c'est artificiel. La question est de savoir ce qui peut bien causer une inégalité, et ce qu'elle peut bien signifier. Supposons qu'il existe $P \in (\bigcap_{o \in Ext(\alpha)} Attr(o)) - \mathcal{P}_\alpha$. Trois cas se présentent :

- si α est exhaustive et que toutes ses sous-classes directes (ou indirectes en cas d'exhaustivité) introduisent P : α n'a alors pas d'instance propre et toutes ses instances indirectes ont l'attribut P ;

- si la passerelle $\alpha \triangleleft_a \beta$ a été posée explicitement et que $P \in \mathcal{P}_\beta$, y a-t-il des raisons pour ne pas avoir accepté, lors de l'introduction de α, que $\mathcal{P}_\alpha \supseteq \mathcal{P}_\beta$? Que cela contrarie les propriétés formelles du SC : dans le cas de TROPES[19], que certains attributs de β ne fassent pas partie du *point de vue* de α. Plus généralement, si l'on considère que $\mathcal{P}_\alpha$ exprime une CNS, alors que $\mathcal{P}_\beta$ n'exprimerait que des CN (pour α) : cf. paragraphe 4.5.5, page 91.

- si toute instance (propre) de α est aussi instance d'une classe $\beta \in Def(P)$ (multi-instanciation), sans que l'extension de α soit intégralement incluse dans celle de l'une de ces classes β.

Ces différents cas peuvent être simultanés.

Le second cas pose un problème lors de l'*instanciation*, ou plus généralement lors du rattachement de l'instance à une classe : il faut que l'instanciation tienne compte des passerelles.

19. Le cas de TROPES est un peu particulier : toutes les classes sont censées avoir le même ensemble d'attributs, mais, en fait, ce sont les instances (d'un *concept* au sens de TROPES) qui ont toutes les mêmes attributs, chaque classe (ou *point de vue*) n'en possèdant/décrivant qu'une partie.

Dans les deux autres cas, il serait possible d'introduire une sous-classe γ de α pour introduire P : on ne voit pas bien pourquoi, dans le premier cas, γ différerait de α et, dans le troisième, comme $\alpha \trianglelefteq \gamma$, pourquoi l'on n'aurait pas une passerelle, ce qui ramènerait au second cas.

NOTE *4.19*

On pourrait sans doute inverser la définition des intensions, en posant $Attr(o)$ comme primitif (défini à partir des domaines des $P^{\mathcal{I}}$), et en déduire $\mathcal{P}_\alpha$ comme intersection sur tous les modèles. La classification automatique de l'analyse de données, qui part des objets pour les regrouper en classes, se fonderait sans doute ainsi. C'est aussi l'approche des treillis de Galois.

4.3.3 Sur le statut ontologique des classes

Explication des attributs

Le cadre formel que nous avons proposé repose implicitement sur une hypothèse d'*explication des attributs*. En effet, par l'axiome (4.11), nous avons supposé que *tous* les attributs d'un objet du domaine sont *expliqués* par leur appartenance à l'intension de l'une des classes de l'objet[20].

Cette hypothèse pourrait être étendue à la description du monde en imposant que tout les attributs affirmés pour un objet dans la description du monde soient expliqués par les classes affirmées de l'objet. Cela s'obtiendrait en ajoutant à la définition (2.26) de $Attr_{\mathcal{W}}(o)$ la contrainte suivante :

$$Attr_{\mathcal{W}}(o) \subseteq \bigcup_{\alpha \in \underline{Isa}_{\mathcal{W}}(o)} \mathcal{P}_\alpha \tag{4.56}$$

La définition (2.26) pourrait alors être remplacée par

$$Attr_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \bigcup_{\alpha \in \underline{Isa}_{\mathcal{W}}(o)} \mathcal{P}_\alpha \tag{4.57}$$

puisque les attributs appartenant au membre de droite mais pas au membre de gauche appartiennent bien à l'objet mais n'ont pas encore de valeur connue. Si tout attribut a une unique classe de définition ($|\overline{Def}(P)| = 1$), ce qui est défendable, en particulier pour résoudre les problèmes d'*héritage de nom* [Ducournau *et al.*, 1995], alors l'affirmation de tout attribut équivaut à l'affirmation de l'appartenance à la classe de définition de l'attribut.

On a donc deux définitions possibles des descriptions du monde.

Cette hypothèse d'explication des attributs revient à faire l'hypothèse d'un *monopole ontologique des classes* : dans les systèmes d'objets usuels, l'introduction d'une classe, qu'elle soit définie ou primitive, a en plus une signification ontologique. Que ce soit dans la description du monde ou dans la sémantique, les objets sont déterminés, dans leurs attributs, par leurs classes explicitement introduites.

20. Chercher à relativiser l'axiome (4.11) en ne l'imposant qu'aux éléments du domaine effectivement représentés n'améliorerait rien : le domaine ne serait plus uniformément représentable.

Le système TROPES (cf. paragraphe 7.2.3, page 168) est une exception : Euzenat (1993b) revendique pour ses classes un statut purement taxinomique qui ne nous paraît guère souhaitable.

NOTE *4.20* *Laxisme des* frames

Les langages de *frames* ont souvent eu une position assez ambigüe sur ce point puisque, contrairement aux langages d'objets classiques, il leur arrive de permettre l'ajout d'attribut sur une instance, sans que ce nouvel attribut soit astreint à la moindre déclaration préalable et, partant, à la moindre interprétation, comme dans les LT, mais sans le cadre sémantique des LT.

[Dekker, 1994] discute le problème de cohérence qui apparaît lorsque l'inclusion (4.56) n'est pas vérifiée. En fin de classification, il peut apparaître que $Attr_{\mathcal{W}}(o)$ n'est pas inclus dans $Attr(o)$, ce qui est un peu fort : il reste des *attributs inexpliqués*. Il faudrait donc remettre en cause le fait que

$$Attr(o) \subseteq \bigcup_{\alpha \in \underline{Isa}(o)} \mathcal{P}_\alpha. \tag{4.58}$$

Quelle serait alors la signification des attributs supplémentaires ?

NOTE *4.21*

Rappelons que $Isa_{\mathcal{W}}(o)$ est défini par $\trianglelefteq_a$, la fermeture transitive de $\triangleleft_a$ et de $\prec$ (et non de $\prec$ seul) : il s'ensuit que $Attr_{\mathcal{W}}(o)$ contient aussi les attributs hérités par des passerelles.

Explication des attributs dans les LT

Cette hypothèse d'explication des attributs n'est pourtant pas la position prise par les LT, qui autorisent, aussi bien dans la description du monde que dans la sémantique, des attributs non expliqués par les classes de l'objet.

Plusieurs compromis permettraient de résoudre cette différence :

- relativiser la définition de $Attr_{\mathcal{W}}(o)$ ainsi que l'axiome (4.11) ;
- sous-entendre un espace de définition global des attributs,
- introduire des classes virtuelles implicites.

Pour le premier point, il suffit de ne pas considérer l'axiome (4.11) et de définir $Attr_{\mathcal{W}}$ par la synthèse des deux définitions précédentes, ce qui revient à conserver la définition d'origine (2.26) :

$$Attr_{\mathcal{W}}(o) \stackrel{\text{def}}{=} \{P \mid (P\ o\ v) \in \mathcal{W}\} \cup \bigcup_{\alpha \in \underline{Isa}_{\mathcal{W}}(o)} \mathcal{P}_\alpha \tag{4.59}$$

Le deuxième point considère que l'espace des attributs n'est pas limité, qu'ils sont partout définis, sans aucune restriction de domaine :

$$\forall P \in \mathcal{P} : Def(P) = \mathcal{X}$$

$$\forall \alpha \in \mathcal{X} : \mathcal{P}_\alpha = \mathcal{P} \tag{4.60}$$
$$\forall P \in \mathcal{P}, \forall \alpha \in \mathcal{X} : \begin{cases} type(\alpha, P) = \top \\ card(\alpha, P) = \mathbf{N} \end{cases}$$

En l'absence d'introduction explicite de P dans l'intension de α, $Dom(\alpha, P) = 2^\Delta$, alors que, dans ce cas, pour les systèmes d'objets, $Dom(\alpha, P)$ n'est pas défini.

La troisième approche consiste à associer à chaque attribut P une classe définie virtuelle α_P qui introduit l'attribut, avec

$$\begin{aligned} Super(\alpha_P) &= \{\top\} & \qquad type(\alpha_P, P) &= \top \\ \mathcal{P}_{\alpha_P} &= \{P\} & card(\alpha_P, P) &= \mathbf{N} \\ (\alpha_P)^\mathcal{I} &= \mathrm{dom}(P^\mathcal{I}) \end{aligned} \tag{4.61}$$

toute classe introduisant P étant alors implicitement sous-classe de α_P et tout objet ayant une description pour P étant implicitement instance de α_P. Cette manière de voir a l'avantage de permettre d'interdire un attribut dans une classe β, en posant $\beta \nabla_a \alpha_P$ [21].

Aucune de ces approches n'est véritablement satisfaisante : leur relative équivalence montre bien que le statut ontologique des classes est finalement très faible dans la sémantique usuelle des LT : seuls comptent les classes et rôles primitifs. Le problème n'est pas l'introduction de ces classes α_P virtuelles, mais le fait que la sémantique puisse leur permettre d'avoir des instances propres.

Ainsi, dans sa version la plus simple, la composante intensionnelle des LT ne contient aucune définition, mais uniquement un ensemble de concepts et de rôles, atomiques aussi bien du point de vue syntaxique (ce sont des noms) que du point de vue sémantique (ils sont primitifs : leur interprétation est irréductible à celle de toute autre entité). Une *terminologie*, c'est-à-dire un ensemble de concepts définis à partir de ces notions primitives, n'est au premier ordre que du sucre syntaxique. Remplacer un concept défini non circulaire par sa description ne change rien du point de vue sémantique, bien que le coût de cette expansion soit élevé, exponentiel dans le pire des cas [Nebel, 1990], et qu'il vaille donc mieux s'en passer [Paramavisam et Plaisted, 1996]. Cependant, Nebel (1990) constate que l'introduction d'une classe définie a une signification au second ordre : « ce concept m'intéresse ». En particulier, le résultat de la classification d'une instance doit retourner un ensemble de ces classes explicitement introduites, et non pas n'importe quelle antichaîne du treillis virtuel.

Il est tentant d'essayer de rendre compte, au premier ordre, de cette signification secondaire en l'incluant dans la sémantique des LT. C'est une direction plus prometteuse dans la mesure où elle offre un pont entre les LT et les systèmes d'objets usuels. Il faut donc étudier la sémantique des LT en faisant l'hypothèse d'explication des attributs par les classes explicitement introduites, donc en conservant l'axiome (4.11) et la définition (4.59) pour $Attr_\mathcal{W}$. L'axiome (4.11) ne serait pas sans effet sur la sémantique résultante : si $(P\ o\ o')$ est affirmé, o appartient à l'extension d'au moins une classe qui introduit P.

21. Mais (4.60) est sans doute plus fidèle aux LT dans la mesure où aucune différence n'est faite entre une cardinalité 0 et l'absence de définition. (4.61) est sans doute plus expressive dans la mesure où, justement, elle fait cette différence.

4.4 L'incomplétude extensionnelle

L'explication des attributs mise à part, il n'y a *a priori* aucune contrainte sur ce qui est connu ou inconnu dans une description du monde : seule l'existence de certains objets est connue, parmi ceux-ci seul une partie de leurs classes et de leurs attributs sont connus, et pour chacun de ces attributs, seul une partie des valeurs est connue. C'est une incomplétude extensionnelle.

La sémantique des systèmes d'objets proposée ici fait l'hypothèse du *monde ouvert*, comme le font les LT, mais à l'opposé de l'hypothèse habituelle du *monde clos* de la programmation logique, comme par exemple [Kifer *et al.*, 1993] (cf. paragraphe 7.3.2, page 171).

Il faut quand même supposer que cette description incomplète du monde représente partiellement un état du monde « complet » et qu'il est donc possible de « compléter » la description du monde pour diminuer cette incomplétude. On peut considérer une suite croissante (au sens ensembliste) de descriptions consistantes du monde $\mathcal{W}_i, 1 \leq i \leq n$: le fait que les descriptions soient consistantes et la suite croissante permet de supposer qu'elles décrivent toutes le même monde. Il est probable que pour toute théorie à peu près bien formée, les modèles de $\mathcal{W}_i$ sont des modèles de $\mathcal{W}_j$, $\forall j < i$. C'est en fait une définition de monotonie.

Le problème est que le SC est utilisé, à un moment donné, pour représenter, non pas n'importe quel modèle, mais un certain *état du monde* réel (ou du domaine modélisé[22]) qui n'est connu qu'incomplètement au travers de la description du monde $\mathcal{W}$.

Si l'on assimile cet état du monde réel à un (sous-ensemble de)[23] modèle(s) de $\mathcal{W}$, toute suite croissante de descriptions $\{\mathcal{W}_i\}_{i \geq 0}$, avec $\mathcal{W} = \mathcal{W}_0$, n'est pas réalisable : seules celles qui conservent ceux que l'on appellera les *modèles du problème* sont possibles.

La difficulté réside dans le fait que ces modèles du problème ne sont pas connus, pas plus du système que de l'utilisateur : c'est d'ailleurs le (ou une partie du) problème à résoudre.

Le raisonnement valide que le système peut mettre en œuvre ne suffit bien sûr pas pour atteindre les modèles du problème. Or, que peut-il faire de plus ? Si l'on exclut un raisonnement hypothétique ou non monotone, rien, si ce n'est déléguer au monde extérieur (l'utilisateur) le soin de compléter ses connaissances pour de nouvelles inférences. Dans ce cadre, l'un des rôles du SC pourrait être de présenter à l'utilisateur des hypothèses plus plausibles que d'autres.

Bien sûr, cette problématique peut se compliquer de diverses façons, par exemple en considérant que le SC ne représente pas le monde de façon correcte, mais un état partiellement inconsistant des connaissances sur le monde : c'est l'interprétation de

22. On pourrait réserver le terme de *domaine modélisé* pour la composante intensionnelle, et se servir de *état du monde* pour la composante extensionnelle.

23. Le nombre de modèles du problème dépend du genre de problème traité : pour un diagnostic, il n'y a vraisemblablement qu'un modèle, à moins de considérer des hypothèses alternatives. En revanche, pour un problème de conception ou de satisfaction de contraintes (CSP), il peut y avoir plusieurs solutions alternatives. Enfin, s'il y a plusieurs modèles, on peut chercher à les calculer tous (diagnostic) ou un seul (conception).

[Patel-Schneider, 1989b], motivée par le besoin de trouver une sémantique plus faible avec de meilleures propriétés calculatoires. Cette interprétation pose divers problèmes, d'abord par son caractère non intuitif, ensuite par le déplacement qu'elle paraît provoquer de la représentation du monde à la représentation d'un agent (intelligent ?) qui se représente le monde.

NOTE *4.22* *Variables de décision et variables d'environnement*

Cette distinction entre diagnostic et conception est grossière, un problème pouvant être un mélange des deux.

Les CSP cherchent une solution en distinguant des variables de décision pour lesquelles il peut exister plusieurs solutions possibles, et des variables d'environnement dont la valeur, sans doute inconnue, est dictée par un état du monde [Fargier *et al.*, 1996a; Fargier *et al.*, 1996b].

4.5 Conditions nécessaires et/ou suffisantes

CLASSE, D. *Klasse* ; E. *Class* ; I. *Classe.*
A. LOGIQUE. Ensemble d'objets défini par le fait que ces objets possèdent tous et possèdent seuls un ou plusieurs caractères communs.

André Lalande, Vocabulaire technique et critique de la philosophie, 1921

4.5.1 Classes descriptives ou définies

Revenons sur la distinction entre classes définies et classes primitives (cf. paragraphe 3.4.2, page 49). La sémantique du paragraphe 4.2.2 autorise le rattachement d'une instance à une classe uniquement lorsque l'instance est valeur d'un attribut dont le type est n'est pas plus général que celui de l'instance (axiomes 4.13 et 4.10). En fait, pour pouvoir classer une instance d'après la valeur de ses attributs, il faut symétriser (4.10) d'une manière ou d'une autre.

Plus formellement, on peut d'abord définir le fait qu'un élément o du domaine vérifie l'intension de α *dans un modèle* $\mathcal{M}$, noté $o \sqsubseteq_{\mathcal{M}} \alpha$, par :

$$\forall P \in \mathcal{P}_\alpha, \begin{cases} P^\mathcal{I}(o) \subseteq type(\alpha, P)^\mathcal{I} \\ |P^\mathcal{I}(o)| \in card(\alpha, P) \end{cases} \tag{4.62}$$

Alors, une classe définie autorise la réciproque globale de l'axiome (4.10), dans tous les modèles :

$$\begin{aligned} \alpha^\mathcal{I} &= \{o \in \Delta \mid o \sqsubseteq_{\mathcal{M}} \alpha\} \\ &= \big\{o \in \Delta \mid \forall P \in \mathcal{P}_\alpha, P^\mathcal{I}(o) \subseteq type(\alpha, P)^\mathcal{I} \ \&\ |P^\mathcal{I}(o)| \in card(\alpha, P)\big\} \\ &= \bigcap_{P \in \mathcal{P}_\alpha} \big\{o \in \Delta \mid P^\mathcal{I}(o) \subseteq type(\alpha, P)^\mathcal{I} \ \&\ |P^\mathcal{I}(o)| \in card(\alpha, P)\big\} \\ &= \bigcap_{P \in \mathcal{P}_\alpha} \big(\{o \in \Delta \mid P^\mathcal{I}(o) \subseteq type(\alpha, P)^\mathcal{I}\} \cap \{o \in \Delta \mid |P^\mathcal{I}(o)| \in card(\alpha, P)\}\big) \end{aligned} \tag{4.63}$$

Bref, on constate sans étonnement excessif que la sémantique d'une classe définie est exactement celle d'un concept défini des LT (cf. paragraphe 4.2.4, page 75, axiomes 4.48-4.51), formé à partir des constructeurs de conjonction et de restriction de valeur et de cardinalité `and`, `all`, `atleast` et `atmost` du langage $\mathcal{TF}$ de Nebel.

Le but de la classification d'instances est de déterminer les classes à l'extension desquelles un objet o appartient, c'est-à-dire $Isa(o)$: seules les classes définies permettent de conclure à une telle appartenance pour toutes leurs instances. En toute généralité, c'est-à-dire en cas de classes seulement descriptives, la classification est un problème *inductif* dont la solution ne peut pas être toujours logiquement correcte.

NOTE *4.23*

Il faut souligner que $\sqsubseteq_{\mathcal{M}}$ concerne les éléments du domaine et non pas leurs représentants dans la description du monde. Pour qu'un objet o de $\mathcal{W}$ satisfasse l'intension d'une classe (notion non définie), il faut par exemple que $o^{\mathcal{I}} \sqsubseteq_{\mathcal{M}} \alpha$, dans au moins un modèle $\mathcal{M}$. Mais ce serait une définition bâtarde, mi-extensionnelle, mi-intensionnelle : le paragraphe 6.3.3 cherchera à en donner une définition purement intensionnelle.

NOTE *4.24* *Interprétations d'Euzenat*

[Euzenat, 1993a] propose pour ses SC deux fonctions d'interprétation simultanées, l'interprétation réelle I_R et l'interprétation abstraite I_A. La première correspond à $\mathcal{I}$ (plus précisément à $\mathcal{E}_{\mathcal{X}}$). La seconde s'obtient à partir de $\sqsubseteq_{\mathcal{M}}$:

$$I_A(\alpha) = \{o \in \Delta \mid o \sqsubseteq_{\mathcal{M}} Int(\alpha)\}.$$

Bien sûr, $I_R(\alpha) \subseteq I_A(\alpha)$. Pour que la correspondance esquissée ici soit exacte, il faut sans doute que la définition de $\sqsubseteq_{\mathcal{M}}$ soit récursive, en y remplaçant l'interprétation réelle ($\mathcal{I}$ ou I_R) par l'abstraite (I_A) :

$$\forall P \in \mathcal{P}_\alpha, \begin{cases} P^{\mathcal{I}}(o) \subseteq I_A(type(\alpha, P)) \\ |P^{\mathcal{I}}(o)| \in card(\alpha, P) \end{cases}$$

NOTE *4.25* *Définition et subsomption*

Si la définition d'une classe fait de son intension une CNS d'appartenance pour les instances, elle en fait bien entendu aussi une condition suffisante de subsomption :

$$\alpha \text{ définie } \ \& \ \ \beta \trianglerighteq \alpha \Longrightarrow \beta \trianglelefteq \alpha$$

Bien sûr, seule α, la classe subsumante, a besoin d'être définie.

4.5.2 Définition absolue ou relative

En fait, il existe deux notions, subtilement différentes, de classe définie. Celle qui précède est *absolue* : elle s'exprime entièrement par référence aux attributs de la classe, sans aucune référence à ses super-classes. Soit $\alpha \prec \beta$: si α est définie dans l'absolu alors o peut être reconnu comme une instance de α sans avoir été préalablement reconnu comme une instance de β, même si β est primitive. On a vu plus compliqué, avec une classe α *définie en extension* par ses sous-classes $\alpha_i, i \in [1\ n]$, elles-mêmes *définies dans l'absolu* :

$$\alpha^{\mathcal{I}} = \bigcup_{i \in [1\ n]} \alpha_i{}^{\mathcal{I}} \tag{4.64}$$

α est alors une classe définie par des conditions nécessaires et suffisantes, qui ne s'expriment pas directement par son intension : en particulier, il peut exister des objets qui vérifient l'intension de α, sans en être instance : ce sont des objets impossibles, ou *paradoxaux*, dont l'existence traduit une inconsistance de la description du monde[24].

L'opérateur `cover` des LT correspond à cette définition en extension, dont l'*exhaustivité* offre l'opportunité.

En revanche, si α est *définie relativement à* β (ou plus généralement à ses super-classes directes, et indirectement à ses super-classes primitives), la condition nécessaire et suffisante ne porte plus sur la totalité des attributs de α mais uniquement sur ceux qui diffèrent de ceux de β (ce que l'on a noté $\mathcal{P}^*_\alpha$ et $Int^*(\alpha)$ au paragraphe 2.1.5), l'affirmation explicite de l'appartenance à β remplaçant la part de la CNS portant sur $Int(\beta)$:

$$\alpha^{\mathcal{I}} = \bigcap \begin{cases} \bigcap_{\beta \in Super(\alpha)} \beta^{\mathcal{I}} \\ \bigcap_{P \in \mathcal{P}^*_\alpha} \{o \in \Delta \mid P^{\mathcal{I}}(o) \subseteq type(\alpha, P)^{\mathcal{I}}\} \\ \bigcap_{P \in \mathcal{P}^*_\alpha} \{o \in \Delta \mid |P^{\mathcal{I}}(o)| \in card(\alpha, P)\} \end{cases} \tag{4.65}$$

C'est le cas habituel des LT lorsque β est un concept *primitif* : lorsque β est défini, il suffit de remplacer récursivement son extension par l'une des deux formules suivant que la définition est relative ou absolue. S'il n'y a aucune super-classe indirecte primitive, les deux notions sont équivalentes.

Cette distinction entre définitions absolue et relative recouvre complètement la distinction faite entre *intension* et *description* : la définiiton absolue (resp. relative) est une définition par l'intension (resp. la description) (cf. paragraphe 2.1.5, page 22).

Ainsi, la classe `champignon vénéneux` (`cv`) est clairement primitive, même si `amanite phalloïde` (`ap`) est parfaitement et absolument définie. Peut-être faut-il considérer que la sous-classe définie n'est pas une sous-classe de la classe primitive, mais simplement que son extension est incluse dans celle de cette dernière. Il faudrait donc remplacer `ap` $\prec$ `cv` par `ap` $\lhd_a$ `cv`, ce qui permettrait certainement de faire une classification des champignons du *point de vue* `gastronomique` aussi bien que `toxique`.

24. De façon générale, quelle que soit leur origine, les objets paradoxaux traduisent la non localité des contraintes qui pèsent sur les instances d'une classe : l'objet est bien formé par rapport aux classes auxquelles il est rattaché, mais en fait, il ne peut être rattaché à aucune classe définitive dans aucun modèle. Bien que cette situation soit sans doute un peu inévitable, il s'agit là d'un argument contre la déclarativité des systèmes en cause.

4.5.3 Définitions et circularités

Formellement, mais en toute généralité, une classe α est définie si, pour tout modèle $\mathcal{M}$, son extension $\alpha^{\mathcal{I}}$ est définie comme une fonction d'un ensemble d'attributs et des extensions d'un ensemble de classes :

$$\alpha^{\mathcal{I}} = f_\alpha(P_1, .., P_n, \beta_1^{\mathcal{I}}, .., \beta_m^{\mathcal{I}}) \tag{4.66}$$

On construit alors une suite de fonctions $f_\alpha^i, i \geq 0$, avec $f_\alpha = f_\alpha^0$, telle que f_α^{i+1} résulte de la substitution dans f_α^i des extensions de toutes les classes β_j définies par leur définition par f_{β_j}. Une *circularité* dans la définition intervient lorsqu'il existe i tel que :

$$\alpha^{\mathcal{I}} = f_\alpha^i(..., \alpha^{\mathcal{I}}, ...) \tag{4.67}$$

ou bien, de façon un peu plus générale, si la suite f_α^i n'est pas stationnaire[25]. Pour les définitions circulaires, seules les extensions des classes β_j dont la définition n'est pas circulaire (donc différentes de α) seront substituées.

La définition d'une classe α se caractérise alors suivant f_α^∞ :

- si f_α^∞ n'est fonction que d'attributs, c'est une *définition purement intensionnelle* ;
- si f_α^∞ est fonction des extensions d'un ensemble de classes primitives ou circulaires différentes de α : c'est une *définition partiellement extensionnelle* ;
- si f_α^∞ est fonction de $\alpha^{\mathcal{I}}$, c'est une définition circulaire, et l'extension est un point fixe de la fonction[26] ; les autres arguments peuvent être extensionnels ou pas.

La fonction f_α dépend de l'expressivité du langage : les formules 4.63, 4.64 et 4.65 en sont trois exemples. Le problème est que f_α (et *a fortiori* f_α^∞) ne s'exprime pas forcément dans les termes des intensions du SC considéré

Les circularités ne sont pas forcément là où on les attend : ainsi, `Humain` se définit (extensionnellement) comme la réunion de `Homme` et de `Femme`, qui à leur tour se définissent comme des `Humains` de `sexe` respectivement `masculin` ou `feminin`.

NOTE *4.26* *Sémantique constructive*

Les fonctions $f_\alpha^\infty, \alpha \in \mathcal{X}$, permettent d'envisager une *sémantique constructive*, voire un *domaine universel*.

Dans les LT, il correspond à chaque *constructeur* `cons` une fonction $f_{\texttt{cons}}$ particulière, telle que si $\alpha = \texttt{cons}(P_1, ..., P_n, \beta_1, ..., \beta_m)$, alors $f_\alpha = f_{\texttt{cons}}$.

f_α^∞ correspond à l'*expansion* des définitions étudiée dans les LT par [Nebel, 1990] (procédure **EXP**) et dans les GC par [Leclère, 1995], dans les deux cas pour des définitions non circulaires. D'un point de vue opératoire, l'expansion est parfaitement contestable : c'est une solution de facilité lorsqu'elle est possible, c'est-à-dire en l'absence de circularités (quoique l'on puisse expanser jusqu'aux circularités), mais particulièrement inefficace, puisque la taille des définitions expansées peut augmenter exponentiellement [Nebel, 1990]. [Paramavisam et Plaisted, 1996] jugent cette expansion naïve.

25. α peut alors être défini sans circularité à partir d'une classe β_j circulaire.
26. Mais lequel ? C'est bien le problème. Voir les solutions de [Nebel, 1990].

NOTE *4.27* *Circularités extensionnelles*

Nous n'examinons ici que les circularités au niveau intensionnel. Au niveau extensionnel, elles posent peu de problèmes — quoique nous les retrouverons un peu plus loin — contrairement à PROLOG où elles nécessitent l'*occurs check* à l'unification et ont conduit à la théorie des *arbres infinis* qui ne sont qu'une vision arborescente des graphes.

NOTE *4.28* *Définition d'attributs*

Cette discussion s'étendrait aussi au cas des attributs définis et, comme les attributs interviennent dans la définition des classes par leur interprétation, il faudrait donc substituer aussi les extensions des attributs dans leur définition. Mais, comme il l'a déjà été dit, nous ne considérerons pas plus les définitions circulaires.

Descriptions circulaires

On n'a considéré les circularités que dans le cas des définitions, pas dans le cas des classes primitives dont l'intension ou la description peuvent elles-aussi contenir des circularités. Curieusement, les exemples de Nebel portent d'ailleurs sur des concepts habituellement primitifs dont il cherche une définition : `Hommes`, `Chevaux` et `Centaures`, dont les `ascendants` et `descendants` sont respectivement des `Hommes`, des `Chevaux` et des `Centaures`[27]. Or, une définition circulaire pose un problème d'*imprédicativité*, parce qu'elle peut ne pas avoir de point fixe (paradoxe de Cantor-Russel) ou parce qu'elle en a plus d'un. Si la classe est primitive, ce n'est plus un problème puisque, pour un domaine donné, une classe primitive n'a pas *une* interprétation précise : il suffit qu'il en existe une qui satisfasse la contrainte de la description circulaire.

Une difficulté particulière entraînée par les circularités peut être le caractère infini du domaine : ainsi, introduire la classe `Humain`, même primitive, comme ayant exactement deux `parents`, `Humains` eux-aussi, impose des extensions infinies[28]. Pour représenter une humanité finie, il faut se reposer sur les diverses mythologies et distinguer, parmi les `Humains`, les `Humains-ordinaires`, dont les 2 `parents` sont des `Humains` et les `enfants` des `Humains-ordinaires`, des `Demi-Dieux` et autres `Adam` et `Eve`, dont les `parents` sont moins ordinaires. Bien entendu, ces modèles infinis mettent en échec la technique de modèles finis de [Paramavisam et Plaisted, 1996].

4.5.4 Sous-ensemble(s) suffisant(s) d'attributs nécessaires

Le problème des attributs inconnus n'est pas si grave si tous les attributs ne sont pas nécessaires pour la reconnaissance de l'objet, c'est-à-dire s'ils ne font pas tous partie des conditions nécessaires et suffisantes d'appartenance à la classe. On pourrait parler d'attributs *essentiels* ou *accidentels*, mais cela prête un peu à confusion, car ils restent tous nécessaires.

27. On remarque que ces exemples nécessiteraient quelque forme de *polymorphisme paramétrique* : leur isomorphisme montre bien qu'il ne peut s'agir de réelle définition.

28. À condition d'interdire les circuits de la relation, par exemple en ajoutant des contraintes sur l'`âge` — par exemple $\forall$ `(X : Mortel) (age.X < age.parents.X)` [Pelenc et Ducournau, 1996] —, mais cela ne fait plus partie du cadre général des SC.

[Woods, 1991] propose de distinguer des définitions alternatives exprimant chacune des conditions suffisantes différentes — par exemple, un `triangle` est un `[polygone à 3 côtés]` et un `[polygone à 3 angles]` — dont la réunion est nécessaire. Il propose inversement des définitions disjonctives dont aucune n'est contenue par les conditions nécessaires.

Le but de ce paragraphe est de s'interroger sur ce que pourrait être un sous-ensemble d'attributs suffisant [29] pour reconnaître une instance d'une classe. Imaginons un ensemble $\mathcal{P}'_\alpha$, naturellement inclus dans $\mathcal{P}_\alpha$ et représentant les attributs suffisants de la classe α.

- d'abord, la dualité entre caractère suffisant absolu et relatif paraît inévitable ;
- pour une classe α définie, $\mathcal{P}'_\alpha = \mathcal{P}_\alpha$.
- que faire si $\forall P \in \mathcal{P}'_\alpha, P(o) \in Dom(\alpha, P)$ (donc $o \in Ext(\alpha)$) mais $\exists Q \in \mathcal{P}_\alpha - \mathcal{P}'_\alpha$, tel que $Q(o) \notin Dom(\alpha, Q)$ [30] ? C'est un problème un peu similaire à ce qui peut se passer si l'objet a des attributs connus qui n'appartiennent pas à ses classes connues. C'est l'attribut lui-même qui peut avoir un caractère suffisant, et non pas la propriété, c'est-à-dire son éventuelle restriction de domaine, dont la nécessité entraîne la suffisance si l'attribut l'est.
- La monotonie des $\mathcal{P}'_\alpha$ (comme celle des $\mathcal{P}_\alpha$) est-elle nécessaire ? Quel serait le sens de $\alpha \prec \beta$ et $\mathcal{P}'_\alpha$ ne contient pas $\mathcal{P}'_\beta$? On en déduit en fait qu'une restriction de domaine peut suffire à reconnaître un β. On peut donc généraliser avec des intensions partielles suffisantes : $Int'(\alpha) = (\mathcal{P}'_\alpha, Dom'_\alpha)$. Il n'y a pas moyen de déduire en toute généralité une relation entre α et β, mais si $\mathcal{P}'_\alpha \subseteq \mathcal{P}_\beta$, alors $Int'(\alpha)$ est suffisant pour β.

Bref, cette notion de sous-ensemble d'attributs suffisants pose divers problèmes. Une solution naturelle et plus simple (car n'imposant aucune notion nouvelle) consiste à introduire explicitement la classe α' *définie* par les CNS de α, en explicitant le fait qu'elle est subsumée par α. Alors, $\alpha \prec \alpha'$ et $\alpha' \lhd_a \alpha$ (figure 4.5, gauche). Bien entendu, cela introduit des circuits dans $\lhd_a$, mais ils n'ont jamais été explicitement exclus. Mais il est alors difficile de savoir si α' est un sous-ensemble suffisant de α, ou α un sur-ensemble nécessaire de α'.

NOTE *4.29* *CN et CNS en* FROME *ou* OBJLOG+

[Dekker, 1994] propose une distinction analogue entre, d'un côté des attributs nécessairement présents ou valués, qui sont suffisants pour la classification, donc des CNS, et, de l'autre, des attributs déduits qui ne donnent que des conditions nécessaires, non suffisantes. Cela revient donc à l'ajout d'un sous-ensemble $\mathcal{P}'^{*}_\alpha$ d'attributs seulement nécessaires définis dans α, sans la notion de passerelle permettant de transformer, par

29. Par abus de langage, on parlera souvent d'un attribut suffisant, mais il s'agit d'un attribut nécessaire appartenant au sous-ensemble suffisant. Cet abus devient réellement abusif dès que l'on a plusieurs ensembles suffisants alternatifs, comme le prône [Woods, 1991; Woods et Schmolze, 1992] : un attribut peut appartenir à une CNS, mais pas à une autre.

30. En toute logique, ce n'est pas bien grave : $\mathcal{W}$ est inconsistant et entraîne n'importe quoi.

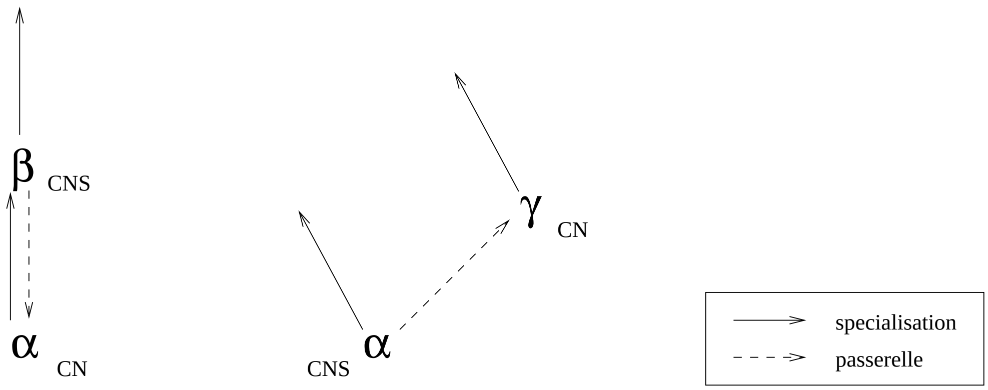

FIG. 4.5 – *Deux façons d'isoler les CNS de la classe α. À gauche, introduire une super-classe définie β dont l'intension est la CNS de α, avec $\beta \lhd_a \alpha$. À droite, introduire α par sa CNS, comme une classe définie, avec une passerelle $\alpha \lhd_a \gamma$ vers une classe γ (sans doute primitive) dont l'intension complète les CN de α.*

héritage, des CNS en simples CN. S'y ajoute la simple présence de l'attribut, ce qui paraît plus difficile à traduire (à part un `(atleast 0)`). Le statut de ces CNS dans les sous-classes, c'est-à-dire de leur héritage, n'est pas clair.

La distinction CN/CNS proposée par [Gonzalez-Gomez, 1996] consiste en une CNS *absolue* (et non pas *relative* aux super-classes) en termes de couples attributs-valeurs (et non pas attributs-domaines : cf. paragraphe 6.5.2, page 157). Rétrospectivement, les CNS de [Dekker, 1994] présentent un caractère absolu certain.

4.5.5 Sur-ensemble(s) d'attributs nécessaires

Une solution alternative consiste à considérer un sur-ensemble d'attributs nécessaires.

La composante assertionnelle des LT — la *ABox*, dans une acception plus large que celle de [Nebel, 1990], que nous avons adoptée ici — propose souvent des règles qui peuvent être assimilées à des conditions nécessaires mais non suffisantes : dans ce cas, au lieu d'avoir un sous-ensemble suffisant d'attributs, il y a un sur-ensemble nécessaire et la monotonie des attributs suffisants est bien assurée. C'est aussi le cas des *passerelles*, de TROPES ou plus généralement des assertions explicites de la relation $\lhd_a$ (cf. paragraphe 2.1.5, page 22).

Revenons donc d'abord sur la possibilité constatée de l'existence et/ou du besoin d'une relation de subsomption extensionnelle explicitement affirmée ($\lhd_a$). Quelle différence y a-t-il entre $\preceq$ et $\lhd_a$? Essentiellement que $\preceq$ a un caractère nécessaire lié à son aspect définitionnel (même si la classe est primitive : « définitionnel » est pris au sens de l'introduction), et implique un héritage des propriétés définitionnelles (l'intension) au niveau des classes, alors que $\lhd_a$ est une assertion complètement contingente qui n'implique qu'un héritage fortuit de propriétés, au niveau des instances[31].

31. La contingence ne concerne que la classification, c'est-à-dire la reconnaissance d'un objet : si tous les objets de [`forme ronde`] sont de [`couleur bleue`], il n'est cependant pas nécessaire de connaître la `couleur` pour reconnaître un `rond`.

Si $\alpha \lhd_a \beta$ et $\mathcal{P}_\beta - \mathcal{P}_\alpha \neq \emptyset$, comment une instance o de α, que l'on peut imaginer créer comme on crée des objets dans un LPO, comment cette instance va-t-elle pouvoir avoir les attributs de β que α ne connaît pas? L'intérêt pour la classification est évident — il n'est plus nécessaire de connaître les attributs de β pour classer o comme un α — mais il reste un problème lors de l'*instanciation*[32].

Une solution peut être de revenir à la distinction entre attributs nécessaires et suffisants, et attributs seulement nécessaires : $\preceq$ rend les attributs hérités suffisants, alors que $\lhd_a$ les rend seulement nécessaires.

En fait, on a vu que l'introduction des passerelles obligeait à modifier quelques définitions en remplaçant $\prec$ par $\lhd_a$. Inversement, elle oblige aussi à dédoubler quelques notions comme l'intension. Nous nous contentons ici de donner un sens à ce dédoublement.

Considérons alors dans chaque classe α les ensembles d'attributs suivants :

- $\mathcal{P}_\alpha$ est l'ensemble suffisant d'attributs, la CNS ;
- $\mathcal{P}'_\alpha$ est l'ensemble des attributs nécessaires[33] : $\mathcal{P}'_\alpha \supseteq \mathcal{P}_\alpha$;
- $\mathcal{P}^*_\alpha$ est l'ensemble des attributs nécessaires et suffisants définis dans α et n'appartenant pas à ses super-classes (par $\prec$).

Alors :

$$\mathcal{P}_\alpha = \mathcal{P}^*_\alpha \cup \bigcup_{\alpha \prec \beta} \mathcal{P}_\beta \qquad \mathcal{P}'_\alpha = \mathcal{P}_\alpha \cup \bigcup_{\alpha \lhd_a \beta} \mathcal{P}'_\beta. \tag{4.68}$$

$\mathcal{P}'_\alpha - \mathcal{P}_\alpha$ représente les attributs hérités, mais ne servant pas pour reconnaître un α. S'y ajouterait sans difficulté l'ensemble $\mathcal{P}'^*_\alpha$ des attributs seulement nécessaires définis dans α. L'introduction de $\mathcal{P}'_\alpha$ amène à définir son symétrique $Def'(P)$ de façon analogue à Def (formule 2.8).

Cette solution a l'avantage d'expliquer les attributs seulement nécessaires d'une instance d'une classe en résolvant le problème de l'*instanciation* (plus généralement, du rattachement d'une instance à une classe) : l'instanciation et la compatibilité d'un objet avec α (cf. paragraphe 6.3.1, page 129) font appel à $\mathcal{P}'_\alpha$, la classification à $\mathcal{P}_\alpha$. Il faut noter que les deux solutions permettent d'associer à une classe des conditions suffisantes distinctes, avec plusieurs passerelles concluant sur la classe.

NOTE *4.30* *À faire*

L'interprétation des passerelles par un héritage de propriétés nécessaires est une relative évidence : elle oblige néanmoins à revoir l'ensemble des définitions et formules, qu'elles précèdent ou qu'elles suivent, pour voir quelles notions sont à dédoubler et quelles autres à modifier. *Dans tous les cas, les considérations qui précèdent sont des*

Cette distinction entre l'héritage pour les classes et l'héritage pour les instances correspond exactement aux règles de CLASSIC, qui sont appliquées individuellement aux instances.

32. Que TROPES évite en considérant un ensemble uniforme d'attributs : seuls les domaines varient. Mais c'est un artifice (cf. paragraphe 2.4.1, page 31).

33. NB. $\mathcal{P}$ et $\mathcal{P}'$ ont été inversés par rapport au paragraphe précédent.

conséquences naturelles de la sémantique formelle du SC *telle qu'elle a été esquissée précédemment, aussi bien que de la sémantique opérationnelle établie par le système de déduction esquissé dans les chapitres 5 et 6. Il s'agit donc principalement de mettre la formalisation intuitive en rapport avec la sémantique formelle du* SC*. Cependant, dans notre démarche aristotélicienne (cf. note 7.5, page 177), la sémantique ne doit véritablement s'appliquer qu'aux instances et il n'est pas nécessaire d'expliciter les propriétés seulement nécessaires des classes ($\mathcal{P}'_\alpha$) : il suffit que les instances les vérifient. C'est la démarche suivie pour les règles de* CLASSIC*. En revanche, les propriétés suffisantes doivent bien être isolées si l'on veut pouvoir s'en servir pour reconnaître, c'est-à-dire classer, une instance de la classe.*

Dans le chapitre 6 sur la classification d'instances, savoir si l'ensemble des attributs d'une classe α, ou l'ensemble des classes d'un attribut P, tiennent compte des passerelles ($\mathcal{P}'_\alpha$ et $Def'(P)$) ou pas ($\mathcal{P}_\alpha$ et $Def(P)$) n'est pas clair. Pour la compatibilité, il faudrait les considérer. Pour la vérification, c'est moins clair. Mais il faut noter que la distinction CN/CNS par le biais des passerelles fournirait un cadre beaucoup plus restreint que celui que nous analysons ici.

NOTE *4.31* *Hypothèse aristotélicienne, maximalité des attributs et passerelles*

L'hypothèse aristotéliciennes consistant à faire résider dans les instances la réalité de la sémantique du SC semble conduire naturellement à considérer les seuls attributs effectifs, ce qui mènerait à l'hypothèse de maximalité des attributs et à :

$$\mathcal{P}_\alpha = \bigcap_{o \in Ext(\alpha)} Attr(o) \qquad (4.69)$$

Cela rendrait impossible l'usage que nous proposons ici pour les passerelles.

Mais il ne s'agit que d'un malentendu : l'hypothèse aristotélicienne n'empêche pas de décrire des classes, en leur associant une intension qui n'est pas forcément maximale.

4.5.6 Critique

L'approche de [Dekker, 1994] est critiquée — à juste titre et la critique vaut aussi, voire encore plus, pour [Gonzalez-Gomez, 1996] —, par [Girard, 1995], parce que cela revient à distinguer un problème, avec des attributs d'entrée et de sortie : la classification opère sur les attributs suffisants (entrée) et l'héritage en déduit des attributs nécessaires (sortie). Cela permet de reconnaître des `champignons` d'après leur écologie et leur morphologie, et d'en déduire leurs propriétés gastronomiques et[34]/ou toxiques. Mais cela ne permet pas de reconnaître l'agent toxique, d'après les causes de la mort, dans le cas où un *bug* dans la représentation n'a pu empêcher une issue fatale[35].

Bien sûr, le postulat qui consiste à représenter la connaissance d'une façon indépendante de son utilisation fait preuve d'un angélisme discutable. Vaut-il mieux ne rien faire plutôt que de ne faire que ça ?

Remarquons quand même que, s'il existe une définition morphologico-écologique complète de l'`amanite phalloïde`, il est probable que ce devrait être la seule classe

34. Ne pas oublier le `gyromitre`, « comestible mais parfois mortel » (*sic*).
35. À moins qu'il n'y ait eu préméditation.

possible, ce qui devrait donc la rendre *sûre* dans le contexte d'un SC manifestement *exhaustif* (voir plus loin). En ce sens, le modèle TROPES devrait pouvoir être utilisé pour représenter des connaissances dont l'exploitation pourrait se faire en spécifiant au cas par cas le problème, par la désignation des points de vue d'entrée et de sortie.

[Fikes et Kehler, 1985] propose une application des langages de *frames* à la classification en se servant de *règles de production* pour exprimer les conditions suffisantes d'appartenance à une classe. Il est remarquable de constater que le seul exemple qu'il donne s'exprimerait parfaitement en termes d'attributs et de domaines, donc dans des termes, et un « style », purement « objet ».

NOTE *4.32* *Objets et règles*
Depuis leurs débuts, *frames*, LT et RCO ont cherché du côté des règles des compléments d'expressivité ou de sémantique. Une partie de ces règles s'exprime dans un style objet et entre, *de facto*, dans le cadre des SC. Seul le reste peut justifier leur introduction, mais encore faut-il le justifier. C'est la question que l'on peut poser à CLASSIC, ou à CARIN qui intègre le calcul des prédicats réduit aux clauses de Horn à $\mathcal{ALCNR}$ et en démontre la décidabilité [Levy et Rousset, 1995] : à quoi ça sert qui ne s'exprimerait pas dans un style objet? En ce qui concerne CLASSIC, les règles semblent bien n'exprimer que des relations extensionnelles: subsomption (passerelles) ou incompatibilité.

4.5.7 Les composants primitifs

Dans sa procédure de normalisation des LT, [Nebel, 1990] transforme les concepts primitifs — qui peuvent être eux-mêmes sous-classes d'autres concepts, définis ou primitifs — en *composants primitifs* : l'intérêt est de rendre atomique le caractère primitif, et de le séparer de l'intension (et de la description) de la classe considérée. Les langages de la famille $\mathcal{AL}$ [Donini *et al.*, 1994] ont suivi une approche similaire, le composant primitif étant alors appelé *concept atomique*. Dans les deux cas, la négation ne peut porter que sur tel concept primitif atomique (constructeur `a-not`), sauf à étendre le langage à une négation généralisée (constructeur `not`), ce qui donne $\mathcal{ALC}$. [Nebel, 1990] montre qu'un modèle est caractérisé de façon unique par l'interprétation des concepts et rôles primitifs.

Cette mise à l'écart du caractère primitif des classes présente un intérêt calculatoire évident sans limiter l'expressivité des langages ni alourdir l'expression. En effet, en pratique, l'introduction d'un concept primitif non atomique revient simplement à introduire d'une part, implicitement, le composant primitif correspondant, qui ne sert qu'à l'implémentation, d'autre part, explicitement, une classe définie à partir de ce composant primitif.

Comme le composant primitif ne fait pas partie du langage d'expression de la description du monde, son introduction ne devrait pas entraîner de problèmes analogues à celui de l'explication des attributs avec les classes virtuelles α_P (cf. paragraphe 4.3.3, page 81). En particulier, imposer que ces composants primitifs n'aient pas d'instance propre ne semble pas devoir modifier la sémantique du SC. *Mais c'est à vérifier.*

NOTE *4.33* *Propriétés spécifiques et classes implicites*
Le seul probème est que l'introduction de ces composants primitifs peut contrarier

les propriétés spécifiques du sc, mais cela signifie simplement que ces propriétés spécifiques ne concernent que les classes explicitement introduites, et pas les classes plus ou moins virtuelles introduites implicitement pour des raisons d'implémentation ou de sémantique.

Mais, comme le remarque [Nebel, 1990], l'introduction d'une classe est une assertion de second ordre qui ne concerne pas ces classes virtuelles qui sont un peu de « deuxième classe », et les propriétés spécifiques ne s'appliquent qu'aux classes de « première classe » (cf. paragraphe 7.1.1, page 162).

NOTE *4.34* *Composant primitif et attribut*

Le composant primitif reste cependant un trait hétérogène : si l'on souhaite plus d'uniformité, on pourrait faire du composant primitif de la classe α un attribut *est-un-*α, à domaine quelconque, que possède toute instance de la classe. Cet attribut constitue, à lui seul, une CNS d'appartenance.

Si cet attribut était booléen, il exprimerait déjà beaucoup plus, par exemple la négation de l'appartenance à une classe : on retrouverait alors le *méta-héritage* proposé dans [Ducournau et Habib, 1991; Ducournau *et al.*, 1994].

Cependant, faire passer le caractère primitif du statut de la classe au statut d'attribut a l'inconvénient de le faire passer de la description à l'intension : il devient en particulier impossible d'introduire des définitions absolues, au moins sur la base de l'intégralité de l'intension. Par ailleurs, la seule façon convaincante que nous avons trouvée pour représenter les CS consiste à introduire une classe avec la CS comme intension : la boucle est bouclée, le composant primitif se retrouvant défini par un attribut qui le paraphrase.

Cette voie ressemble donc à une impasse, et on choisira plutôt la solution esquissée précédemment, avec un composant primitif implicite,

4.6 Évolution conservative des sc

L'essentiel du présent travail se restreint à une interprétation statique des sc hors toute évolution. Sans aller jusqu'à prendre en considération la *révision* non monotone d'un sc comme [Nebel, 1990; Crampé et Euzenat, 1996], il est nécessaire de se poser la question de ce que peut signifier la *conservation* de la sémantique d'un sc lors de l'évolution à caractère *monotone* comme la *réorganisation* [Dicky *et al.*, 1994; Dicky *et al.*, 1996] ou l'extension (que l'on souhaite *conservative*) par ajout de nouvelles classes ou de nouveaux attributs (cf. paragraphe 7.2.1, page 165).

Au niveau extensionnel, l'évolution monotone la plus simple consiste à rajouter à une description du monde consistante $\mathcal{W}$ une description d telle que $\mathcal{W}' = \mathcal{W} \cup \{d\}$ soit consistant. L'ensemble des modèles de $\mathcal{W}'$ est alors un sous-ensemble des modèles de $\mathcal{W}$. L'évolution monotone de la description du monde sera discutée au chapitre 5.

Au niveau intensionnel, on a vu qu'il semblait possible de compléter un sc quelconque pour lui donner certaines propriétés spécifiques (exhaustivité et univocité, cf.

paragraphe 3.1.3, page 39), sans que sa sémantique soit modifiée[36]. Qu'a-t-on exactement voulu dire par là ?

Une réorganisation naturelle consiste à faire le *quotient* du SC par les circuits du préordre $\trianglelefteq$. L'intension des classe résultantes s'obtient par union ($\sqcup$) des intensions des classes de même extension. La relation de spécialisation entre deux classes quotient se déduit de l'existence d'une relation entre les classes correspondantes dans le SC initial.

Plus généralement, l'introduction d'une nouvelle classe définie ne change rien à la sémantique du SC si son effet se réduit à délimiter une nouvelle extension dans le domaine. Cela peut poser un certain nombre de contraintes :

- la classe doit être introduite au bon endroit dans la hiérarchie, ce qui impose une classification automatique des classes, pour que ce soit une propriété *objective* ;
- la nouvelle classe ne doit pas introduire de nouveaux attributs et se contente de restreindre les domaines des attributs hérités ;
- il ne faut pas que la nouvelle classe introduise d'hypothèse existentielle, en contredisant l'incompatibilité : en cas d'univocité, elle ne doit pas être sous-classe de deux classes sans sous-classe commune ;
- toujours en cas d'univocité, l'introduction de deux sous-classes d'une même classe impose d'introduire leur sous-classe commune, sans intension propre particulière.

Enfin, la *catégorisation*, c'est-à-dire la construction de la hiérarchie, pose aussi le problème du statut des propriétés spécifiques pendant cette construction [Leclère, 1995].

36. Ce qui justifie que l'on dénie à ces propriétés un quelconque caractère sémantique : une propriété est sémantique lorsqu'elle a un effet sur les modèles possibles d'un SC.

Chapitre 5

La déduction dans les SC

Si en réalité pure on a toujours raison de ne pas avoir tort,
en réalité altérée on a souvent tort d'avoir raison.
Pierre Dac, *Pensées*, 1972

Nous avons présenté une sémantique des SC et il faut bien lui associer un système de déduction pour en faire une logique. Pendant longtemps, l'étude des SC (essentiellement des LT) s'est cantonnée aux propriétés calculatoires (décidabilité et complexité) des problèmes, réduisant la déduction elle-même à un algorithme dont on discute, au mieux, la complexité et la complétude. Les approches les plus récentes se sont néanmoins tournées vers des techniques relevant plus de la théorie de la déduction.

La déduction dans les SC a été jusqu'à maintenant abordée de différentes façons :

- la première approche, naïve, a consisté à étudier des algorithmes basés sur la normalisation des termes [Nebel, 1990] ;
- [Borgida et Patel-Schneider, 1994] utilisent un algorithme basé sur une transformation des termes en une forme canonique dite « *description graph* » ;
- [Baader et Hollunder, 1991; Schmidt-Schauß et Smolka, 1991; Buchheit *et al.*, 1993; Donini *et al.*, 1994] traduisent le SC en un système de contraintes (analogue aux CSP ?) ; c'est une technique dérivée de la méthode des *tableaux sémantiques* de la logique du premier ordre [Gochet et Gribomont, 1991] : elle consiste à générer de façon non déterministe des systèmes de contraintes en appliquant un certain nombre de règles, jusqu'à obtenir soit un motif d'inconsistance ce qui fait revenir en arrière à la dernière application d'une règle non déterministe, soit un système de contraintes complet et consistant qui prouve que le système est consistant. Le système de contraintes n'est jamais explicitement résolu au sens des CSP.
- [Paramavisam et Plaisted, 1996] utilisent des techniques générales de démonstration de théorèmes, basées sur la recherche exhaustive d'un modèle fini ;
- [Borgida, 1992b] introduit formellement un système de déduction qu'il emprunte à Gentzen par l'intermédiaire de la *sémantique naturelle* de [Kahn, 1987].

5.1 La déduction abstraite

De façon abstraite, la déduction dans un SC va consister à définir un opérateur de dérivation, noté $\vdash$, analogue méta-syntaxique de l'opérateur de conséquence $\models$, mais opérant au plan syntaxique (intensionnel) et non plus sémantique (extensionnel) :

$$\mathcal{H}; \mathcal{W} \vdash d$$

se lit « de $\mathcal{H}$ et $\mathcal{W}$ on peut dériver d ». Cet opérateur peut être défini aussi bien implicitement, au travers d'un algorithme ou d'un système de contraintes, qu'explicitement, par des règles d'inférence. Lorsque la composante extensionnelle n'intervient pas dans l'inférence, on pourra omettre $\mathcal{W}$. Dans chaque « jugement » $\mathcal{H}; \mathcal{W} \vdash d$, les seules descriptions qui peuvent figurer à gauche du $\vdash$ ne peuvent pas figurer à droite, parce qu'elles ne peuvent pas se déduire : c'est le cas de $\prec$, ∇_a et $\lhd_a$ par exemple, ou des conditions de cardinalité.

L'introduction de $\vdash$ conduit à distinguer fermement ce qui est *syntaxique* et peut figurer à gauche ou à droite du symbole $\vdash$, de ce qui est *méta-syntaxique*, car il contient le symbole $\vdash$ ou est défini par une expression qui le contient.

En suivant l'approche logique classique de Gentzen, reprise par Borgida, cet opérateur est défini par un ensemble de règles d'inférence de la forme suivante :

$$[\text{nom}] \qquad \frac{\mathcal{H}_1; \mathcal{W}_1 \vdash d_1 \qquad \ldots \qquad \mathcal{H}_n; \mathcal{W}_n \vdash d_n}{\mathcal{H}; \mathcal{W} \vdash d} \qquad \textit{contexte}$$

dans laquelle le « contexte » représente des conditions d'utilisation de la règle qui doivent s'exprimer directement en fonction des différentes variables apparaissant dans la règle, sans être le résultat d'aucune inférence[1] et sans variable libre bien entendu. De même, des prémisses $\mathcal{H}; \mathcal{W} \vdash d$ peuvent être regroupées et indicées sous la forme $\{\mathcal{H}_i; \mathcal{W}_i \vdash d_i\}_{i \in I}$, où l'ensemble I peut dépendre de l'instanciation des différentes variables de la règle et de l'état courant de la déduction. Dans cette notation, il faudra distinguer le cas où l'ensemble I regroupe toutes les descriptions d_i — I, en général l'intervalle $[1, n]$ est alors « défini » dans l'état courant de la déduction — et le cas où il ne s'agit que d'un ensemble de descriptions d_i ou d'un ensemble préexistant : pour cela, on positionnera $i \in I$ respectivement en exposant ou en indice. Il faut donc bien distinguer, dans la composante intensionnelle essentiellement, ce qui est donné au départ de ce qui est calculé ou inféré dans la déduction. D'autre part, comme en pratique $\mathcal{H}_i = \mathcal{H}$ et $\mathcal{W}_i = \mathcal{W}$, on pourra omettre $\mathcal{H}$ et $\mathcal{W}$ des règles.

Comme nous nous plaçons dans un cadre exclusivement monotone[2], les prémisses des règles ne peuvent pas porter sur des faits qui pourraient être remis en cause ultérieurement : en particulier, il est exclu d'y faire figurer en prémisse (numérateur) la négation de la déductibilité d'un fait. L'opérateur de déduction doit donc vérifier la propriété :

$$[\text{monotonie}] \qquad \frac{\Gamma \vdash d}{\Gamma, \Delta \vdash d}$$

1. Est-ce si sûr ? Cf. les règles [$\exists$-exhaust] et [$\exists$-univoc].

2. Cette monotonie de la déduction ne doit pas être confondue avec la monotonie de l'évolution de la description du monde.

Si une description est dérivable de l'ensemble d'assertions Γ, elle est dérivable de tout ensemble d'assertions contenant Γ.

Enfin, comme pour la relation de conséquence valide, cette notation va permettre de traiter les dérivations aussi bien dans la composante intensionnelle que dans la composante extensionnelle. Deux problèmes privilégiés s'identifient facilement :

- au niveau intensionnel, la *subsomption* et ses variantes (l'*incompatibilité*, l'*incohérence* et la *classification de classe*),
- au niveau extensionnel, la *vérification d'instance* (*instance checking*) et ses variantes (l'*inconsistance* et la *classification d'instance*).

Ces différents problèmes sont plus ou moins équivalents [Nebel, 1990; Donini *et al.*, 1994] : nous discuterons plus loin de cette équivalence.

On notera que le *filtrage* est un mélange de ces deux catégories de problèmes puisqu'il est extensionnel mais qu'il passe par une classification de classe (le *filtre*).

5.1.1 Exemples de règles d'inférence

Les règles qui suivent sont des exemples de règles possibles, simples et générales : comme pour la sémantique du chapitre 4, elles n'ont aucune prétention à la complétude, mais prétendent à la généralité, sauf exception signalée. Les premières concernent la composante intensionnelle, introduisent la subsomption et explicitent le fait que c'est un préordre :

[$\top$-intro] $\dfrac{}{\vdash \alpha \trianglelefteq \top}$

[$\bot$-intro] $\dfrac{}{\vdash \bot \trianglelefteq \alpha}$

[$\trianglelefteq$-réflex] $\dfrac{}{\vdash \alpha \trianglelefteq \alpha}$

[$\trianglelefteq / \prec$] $\dfrac{}{\alpha \prec \beta \vdash \alpha \trianglelefteq \beta}$

[$\trianglelefteq / \lhd_a$] $\dfrac{}{\alpha \lhd_a \beta \vdash \alpha \trianglelefteq \beta}$

[$\trianglelefteq$-transit] $\dfrac{\mathcal{H} \vdash \alpha \trianglelefteq \beta \qquad \mathcal{H} \vdash \beta \trianglelefteq \gamma}{\mathcal{H} \vdash \alpha \trianglelefteq \gamma}$

puis introduisent l'incompatibilité, d'abord dans le cas binaire :

[∇-intro] $\dfrac{}{\alpha \nabla_a \beta \vdash \alpha \nabla \beta}$

[$\nabla / card$] $\dfrac{}{card_P(\alpha) \cap card_P(\beta) = \emptyset \vdash \alpha \nabla \beta}$

[∇-sym] $\dfrac{\mathcal{H} \vdash \alpha \nabla \beta}{\mathcal{H} \vdash \beta \nabla \alpha}$

[$\nabla / type$] $\dfrac{\mathcal{H} \vdash type_P(\alpha) \nabla type_P(\beta)}{\mathcal{H} \vdash \alpha \nabla \beta}$

[∇-hérit] $\dfrac{\mathcal{H} \vdash \alpha \nabla \beta \qquad \mathcal{H} \vdash \gamma \trianglelefteq \alpha}{\mathcal{H} \vdash \gamma \nabla \beta}$

[∇-incohér] $\dfrac{\mathcal{H} \vdash \alpha \nabla \beta \quad \mathcal{H} \vdash \gamma \trianglelefteq \alpha \quad \mathcal{H} \vdash \gamma \trianglelefteq \beta}{\mathcal{H} \vdash \gamma \trianglelefteq \bot}$

Les suivantes s'appliquent à la composante extensionnelle et donnent les deux seules inférences de classification valables pour tous les SC (cf. chapitre 6) :

[⊴-ferm] $\dfrac{\mathcal{H}; \mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H} \vdash \alpha \trianglelefteq \beta}{\mathcal{H}; \mathcal{W} \vdash (\beta\ o)}$

[*type*-infér] $\dfrac{\mathcal{H}; \mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H}; \mathcal{W} \vdash (P\ o\ o')}{\mathcal{H}; \mathcal{W} \vdash (type_P(\alpha)\ o')} \qquad \alpha \in \mathrm{dom}(type_P)$

[⊴-ferm] correspond exactement à la *règle de subsomption* de la théorie des types (cf. note 2.2). Les propriétés spécifiques du SC induisent aussi des inférences particulières, par exemple :

[∇-exclusiv] $\dfrac{}{\alpha \prec \gamma, \beta \prec \gamma \vdash \alpha \nabla \beta} \qquad \alpha \neq \beta$

[∧-univoc] $\dfrac{\mathcal{H}; \mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H}; \mathcal{W} \vdash (\beta\ o)}{\mathcal{H}; \mathcal{W} \vdash (\alpha \wedge \beta\ \ o)} \qquad \alpha \neq \beta$

La règle [∇-exclusiv] ne vaut bien sûr que si $\prec$ ne désigne que le graphe de couverture de la relation de spécialisation. Le traitement n-aire de l'incompatibilité pourrait donner les règles suivantes :

[∇-intro∗] $\dfrac{\mathcal{H} \vdash \nabla_a(A)}{\mathcal{H} \vdash \nabla(A)} \qquad A \in \mathcal{X}^*$

[∇/*type*∗] $\dfrac{\mathcal{H} \vdash \nabla\{type_P(\alpha)\}_{\alpha \in A}}{\mathcal{H} \vdash \nabla(A)} \qquad A \subseteq \mathrm{dom}(type_P)$

[∇/*card*∗] $\dfrac{}{\bigcap_{\beta \in A} card_P(\beta) = \emptyset \vdash \nabla(A)} \qquad A \subseteq Def(P)$

[∇-hérit∗] $\dfrac{\mathcal{H} \vdash \nabla(B) \qquad \mathcal{H} \vdash A \trianglelefteq^* B}{\mathcal{H} \vdash \nabla(A)}$

[∇-incohér∗] $\dfrac{\mathcal{H} \vdash \nabla(B) \qquad \{\mathcal{H} \vdash \alpha \trianglelefteq \beta\}_{\beta \in B}}{\mathcal{H} \vdash \alpha \trianglelefteq \bot}$

en définissant l'ordre sur les antichaînes $\trianglelefteq^*$ comme suit, avec $A, B, C, D \subseteq \mathcal{X}$:

[⊴∗-réflex] $\dfrac{}{\mathcal{H} \vdash A \trianglelefteq^* A}$

[⊴∗-transit] $\dfrac{\mathcal{H} \vdash A \trianglelefteq^* B \qquad \mathcal{H} \vdash B \trianglelefteq^* C}{\mathcal{H} \vdash A \trianglelefteq^* C}$

[⊴∗/⊴] $\dfrac{\mathcal{H} \vdash \alpha \trianglelefteq \beta}{\mathcal{H} \vdash \{\alpha\} \trianglelefteq^* \{\beta\}}$

[⊴∗/ ⊂] $\dfrac{\mathcal{H} \vdash A \trianglelefteq^* B}{\mathcal{H} \vdash A \cup C \trianglelefteq^* B}$

[$\trianglelefteq^*/\cup$] $$\frac{\mathcal{H} \vdash A \trianglelefteq^* B \qquad \mathcal{H} \vdash C \trianglelefteq^* D}{\mathcal{H} \vdash A \cup C \trianglelefteq^* B \cup D}$$

Vérifier que les règles sur $\trianglelefteq^*$ sont bien suffisantes. L'inconsistance peut aussi se déduire :

[incoher-inconsist] $$\frac{\mathcal{H}; \mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H} \vdash \alpha \trianglelefteq \bot}{\mathcal{H}; \mathcal{W} \vdash \text{inconsistance}}$$

[*card*-inconsist] $$\frac{\{\vdash (\alpha_i\ o)\}_{i=1,n} \qquad \{\vdash (P\ o\ o_j)\}_{j=1,m}}{\vdash \text{inconsistance}} \qquad m > \max \bigcap_i card_P(\alpha_i)$$

[∇-inconsist] $$\frac{\{\vdash (\alpha\ o)\}_{\alpha \in A} \qquad \vdash \nabla(A)}{\vdash \text{inconsistance}}$$

[*type*-inconsist]
$$\frac{\{\vdash (\alpha_i\ o)\}_{i=1,n} \quad \vdash (P\ o\ o') \quad \{\vdash (\beta_j\ o')\}_{j=1,m} \quad \nabla(\beta_1, ..., \beta_m, type(\alpha_1), ..., type(\alpha_n))}{\vdash \text{inconsistance}}$$

La règle [*type*-inconsist] subsume la précédente [∇-inconsist].

5.1.2 Correction et complétudes

Les rapports entre les versants syntaxique et sémantique, entre déduction et sémantique, s'analysent traditionnellement en termes de correction et de complétude.

Correction

La correction de la déduction correspond à l'implication suivante :

$$\mathcal{H}; \mathcal{W} \vdash d \Longrightarrow \mathcal{H}; \mathcal{W} \models d \tag{5.1}$$

qui exprime que tout se qui se déduit est une conséquence valide.

La correction ne fait pas l'objet de longs débats, sinon de longues preuves : elle est exigée, ni plus, ni moins.

Histoires de complétude

En revanche, la complétude pose plus de problèmes : d'une part, par réalisme, elle n'est que souhaitée, d'autre part, elle se présente sous divers aspects. La déduction se heurte en effet aux difficultés soulevées par plusieurs catégories d'incomplétude :

- les objets eux-mêmes sont incomplets : leurs attributs et les valeurs de ces attributs peuvent être partiellement inconnus (incomplétude extensionnelle) ; on verra plus loin que cette incomplétude peut s'expliquer de deux façons différentes : méconnaissance de l'*état du monde* ou liberté de choix ;

- le processus de déduction (de classification par exemple) est incomplet : il se fait surtout, mais pas seulement, syntaxiquement, au niveau des intensions, par des comparaisons *a priori* incomplètes (incomplétude sémantique) ; la complétude sémantique correspondrait à l'implication inverse de la correction (5.1) :

  $$\mathcal{H}; \mathcal{W} \models d \Longrightarrow \mathcal{H}; \mathcal{W} \vdash d \tag{5.2}$$

qui exprime le fait que toute conséquence valide se déduit.

- le langage du SC est incomplet : il ne permet pas d'exprimer exactement le domaine modélisé (incomplétude syntaxique) ; en particulier :
    - le langage ne permet pas de décrire exactement les classes ou les objets (incomplétude descriptive) ;
    - les descriptions des classes peuvent être ou ne pas être des CNS d'appartenance pour les objets (incomplétude définitionnelle).

Décidabilité, complétude sémantique et complexité

Pour ce qui est de la complétude sémantique, c'est-à-dire de la capacité d'un algorithme de faire toutes les inférences correspondant à des conséquences valides, il faut encore que le problème de déduction considéré soit décidable, c'est-à-dire qu'il existe un tel algorithme.

Or, il a été montré que, dans de nombreuses LT, le calcul de la subsomption extensionnelle n'est que semi-décidable [Patel-Schneider, 1989a]. Et lorsqu'il est décidable, il peut être trop complexe pour que les algorithmes complets soient utilisables [Nebel, 1990], ce qui amène à envisager un compromis *expressivité-complétude-complexité* [Levesque et Brachman, 1987; Attardi, 1991], voire à introduire un quatrième terme dans le compromis, la sémantique, comme le propose [Patel-Schneider, 1989b].

Pour un SC donné, dont l'expressivité et la sémantique sont fixées, ce compromis se réduit à un compromis *complétude-complexité*. Il en résulte qu'en pratique la déduction est ou peut être incomplète et qu'une approche générale doit prendre en compte l'éventualité d'une incomplétude.

NOTE *5.1* *Complétudes classiques*

La logique distingue classiquement complétude sémantique et complétude syntaxique (cette dernière avec des versions syntaxique et sémantique[3]), ce qui explique les deux théorèmes de Gödel concluant respectivement à la complétude ou à l'incomplétude de l'arithmétique (voir [Gochet et Gribomont, 1991] p.318sq). L'usage de complétude extensionnelle s'explique bien, puisqu'elle concerne la composante de même nom. Les classes primitives offrent un exemple abstrait d'incomplétude définitionnelle. Pour l'incomplétude descriptive, on pourra revenir sur l'exemple des `véhicules` au nombre pair de `roues`, dans un langage qui ne sait (comme tous ceux que l'on connaît) exprimer les cardinalités que par des intervalles.

Dans le cas de la complétude sémantique, la logique distingue aussi des complétudes faible ou forte, suivant qu'elles s'appliquent uniquement à des tautologies (complétude faible) ou à toutes les conséquences (complétude forte) [Gochet et Gribomont, 1991]. Cette distinction ne semble pas pertinente pour les SC.

En revanche, il pourrait être souhaitable de distinguer, toujours dans la complétude sémantique, l'inconsistance de la description du monde : il ne semble en effet pas sou-

3. Ces deux versions « ne sont pas synonymes mais sont coextensives » : nous avons par exemple présenté le théorème d'incomplétude de Gödel suivant la seconde, alors qu'il l'est généralement suivant la première, l'équivalence des deux venant du théorème de ... complétude (sémantique) du même.

haitable d'être vraiment complet en cas d'inconsistance. Une complétude « maîtrisée » permettrait au contraire de limiter les dégâts en cas d'inconsistance[4].

Cas particulier du cas général

Un général, dégénéré
Boris Vian

Le cadre général des SC caractérise un SC particulier, celui qui est défini exactement par l'ensemble des axiomes généraux. On peut aussi y rajouter l'univocité pour se rapprocher des systèmes d'objets. *Programme : exhiber toutes les règles de déduction des* SC *généraux, avec ou sans univocité, et en montrer la correction et la complétude. Essayer avec différentes cardinalités : monovalué, mono ou multivalué, intervalles, etc.*

5.2 Histoires de subsomption

Le calcul de la subsomption extensionnelle ($\trianglelefteq$) est, au niveau des classes, l'un des premiers buts des systèmes classificatoires, au moins pour ceux qui le permettent et en particulier pour les LT.

Ce calcul présente deux des trois catégories d'incomplétude :

- l'incomplétude syntaxique liée à la fois aux spécificités du domaine représenté — qui fait que l'équivalence entre `[polygone à trois côtés]` et `[polygone à trois angles]` doit être affirmée explicitement — et à la présence de classes primitives ;

- l'incomplétude sémantique liée à la calculabilité de la relation à partir des données du problème, essentiellement l'intension des classes, et donc à partir de $\trianglerighteq$.

L'incomplétude extensionnelle n'est pas en cause, au moins lorsque les composantes intensionnelle et extensionnelle sont bien séparées. En CLASSIC où ce n'est pas le cas, avec les constructeurs `one-of` et `fills`, la subsomption est figée à partir de l'état initial des instances impliquées dans la description des classes. Cela revient bien à éliminer (ou à refuser de considérer) toute incomplétude extensionnelle.

5.2.1 Subsomptions théoriques et calculées

On peut donc distinguer, outre la subsomption affirmée $\trianglelefteq_a$, une subsomption extensionnelle théorique $\trianglelefteq_{th}$, identique à $\trianglelefteq$, et une subsomption calculée $\trianglelefteq_c$, définie (méta-syntaxiquement) par

$$\alpha \trianglelefteq_c \beta \iff \vdash \alpha \trianglelefteq \beta$$

4. Tout ça est à voir à la lumière de la logique classique.

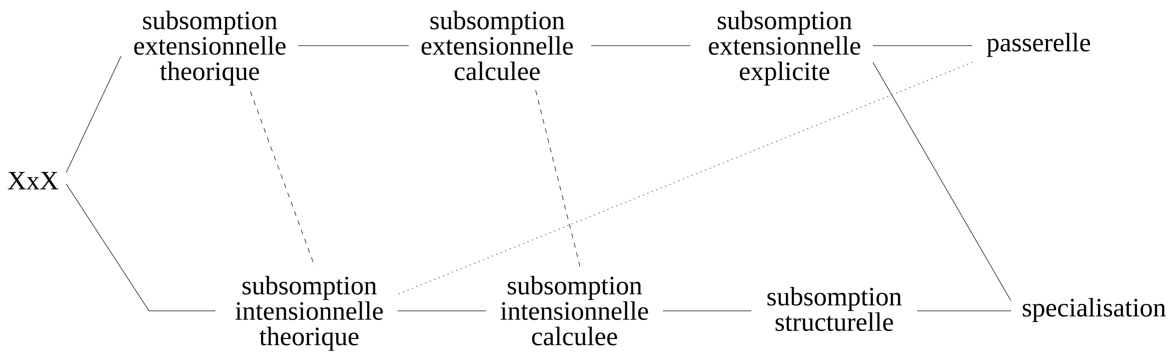

FIG. 5.1 – *Les diverses sortes de subsomption, et leurs relations d'inclusion de droite à gauche. Les inclusions en tireté ne valent généralement que pour des classes définies. La relation en pointillé est une disjonction.*

avec :

$$\preceq \;\subseteq\; \trianglelefteq_a \;\subseteq\; \trianglelefteq_c \;\subseteq\; \trianglelefteq_{th} = \trianglelefteq$$

Pour ce qui est des intensions, on peut faire le même genre de distinction : nous avons défini une relation $\trianglerighteq$ de subsomption structurelle (ou intensionnelle) entre classes. Cette relation, assez aisément calculable, ne recouvre pas forcément toutes les inclusions d'intension possibles, c'est-à-dire toutes les implications possibles entre descriptions. [Nebel, 1990] montre des cas — à base de sous-rôles, ce qui sort donc de notre formalisme général — calculables, mais avec une certaine complexité : ce sont ces cas qui rendent le problème *co-NP-difficile*, ce qui lui fait exclure leur traitement des algorithmes.

Du côté intensionnel, on obtiendrait donc une subsomption intensionnelle théorique $\trianglerighteq_{th}$, et une subsomption intensionnelle calculée $\trianglerighteq_c$, avec :

$$\preceq \;\subseteq\; \trianglerighteq_c \;\subseteq\; \trianglerighteq_{th} \,.$$

mais c'est un peu vain, puisque cela revient à comparer des systèmes de déduction, dont un idéal qui est particulièrement abstrait. Quel rapport peut-on trouver entre toutes ces subsomptions ? Sans doute que

$$\trianglelefteq_c \;=\; (\trianglelefteq_a \;\cup\; \trianglerighteq_c)^*$$

(où $*$ en exposant représente la fermeture transitive), à condition de se placer dans un cadre de classes définies. Il est bien sûr préférable que les subsomptions calculées, $\trianglelefteq_c$ et $\trianglerighteq_c$, vérifient les propriétés attendues des préordres, en particulier la transitivité, ce que [Nebel, 1990] vérifie pour son algorithme `TSUB` et que nous avons exprimé par les règles [$\trianglelefteq$-réflex] et [$\trianglelefteq$-transit].

Dans la pratique, seule $\trianglelefteq_c$ est accessible et, quand nous parlons par exemple de quotienter $\mathcal{X}$ par les circuits de $\trianglelefteq$, il faut comprendre $\trianglelefteq_c$. La complétude sémantique de la subsomption se caractérise ainsi par l'identité des subsomptions théorique et calculée.

Une discussion similaire peut s'appliquer à la relation d'incompatibilité, en considérant ses variantes extensionnelle, intensionnelle et affirmée. Les problèmes d'incomplétude se reposeront à propos des propriétés de complétude ou de fermeture des attributs, objets et description du monde (cf. paragraphe 6.3.2, page 132).

Bien sûr, toute cette discussion reste abstraite, puisqu'elle doit avoir lieu concrètement, c'est-à-dire relativement non seulement à un SC donné mais à un système de déduction particulier, et que la complétude reste possible dans de nombreux cas, comme le montrent [Dionne *et al.*, 1993] pour ??? ou [Borgida et Patel-Schneider, 1994] pour CLASSIC.

5.2.2 Subsomptions intensionnelle ou structurelle

Si le terme de subsomption extensionnelle ne pose pas de problème de définition particulier, ce n'est pas le cas des *subsomption intensionnelle* ou *subsomption structurelle* qui sont assez fréquents dans la littérature. [Patel-Schneider, 1989b] définit (informellement) la *subsomption structurelle* comme le résultat d'une mise en correspondance terme à terme des opérandes des conjonctions définissant les deux termes (classes) à comparer. Elle revient donc, pour nous, à comparer les classes α et β attribut par attribut, pour vérifier :

$$\forall P \in \mathcal{P}_\beta \begin{cases} P \in \mathcal{P}_\alpha \\ Dom(\alpha, P) \subseteq Dom(\beta, P) \end{cases} \tag{5.3}$$

Cette subsomption ne peut être complète qu'en cas d'indépendance des attributs : il faudrait l'étendre pour les sous-rôles, comme suit :

$$\forall P \preceq Q \in \mathcal{P}_\beta, Dom(\alpha, P) \subseteq Dom(\beta, Q),$$

mais la complétude nécessiterait d'examiner des combinaisons de sous-rôles, ainsi que l'incompatibilité de leurs domaines. [Nebel, 1990] montre bien que c'est là une cause principale de la complexite.

D'une certaine manière, cette subsomption structurelle est la borne inférieure de $\trianglerighteq_c$, et son calcul est aisé. C'est d'ailleurs la définition concrète exacte qui a été donnée de l'inclusion ($\sqsupseteq$) des intensions (2.13).

Vouloir donner un sens plus général à la *subsomption intensionnelle*, pour désigner une subsomption résultant de la comparaison aussi « complète » que possible des intensions, paraît plus difficile. Cette subsomption intensionnelle ne se réduirait pas à la subsomption structurelle, puisqu'il est nécessaire de comparer des combinaisons d'attributs pour tenir compte de leur éventuelle dépendance. Elle serait donc coincée entre le marteau de la subsomption structurelle et les enclumes de la subsomption extensionnelle d'un côté, et de la subsomption calculée, c'est-à-dire des algorithmes, de l'autre. Une définition abstraite qui ne se réduit ni à l'une ni à l'autre pourrait être basée sur une stricte comparaison des deux termes (classes et/ou instance) sans y faire intervenir aucun tiers, c'est-à-dire « toutes choses égales par ailleurs ».

Dans les deux cas, il est licite de parler d'*une* subsomption intensionnelle (ou structurelle), mais plus ambigu de parler de *la* subsomption intensionnelle (ou structurelle), à moins d'en avoir défini *une* auparavant. En ce sens, la subsomption intensionnelle définie au chapitre 2 par le préordre $\trianglerighteq$ n'est qu'une version minimale ou abstraite.

5.2.3 Subsomption et vérification d'instances

Schématiquement, on peut dire que la vérification d'instance est à la classification d'instance ce que la subsomption est à la classification de classe.

En effet, l'une ou l'autre de ces classifications repose sur l'itération de la comparaison d'une instance ou d'une classe avec les classes successives de la hiérarchie. Bien sûr, cette analogie ne sort pas du cadre d'une classification « intensionnelle », basée exclusivement sur la comparaison de deux descriptions (cf. paragraphe 5.2.2, page 105). Dans cette analogie, la différence repose essentiellement sur l'incomplétude extensionnelle des objets, dont l'impact est beaucoup plus fort que l'incomplétude définitionnelle des classes.

Traditionnellement, les LT traitaient souvent le problème de la vérification d'instance (un objet est-il instance d'une classe?) comme un cas particulier du problème de la subsomption (une classe est-elle subsumée par une autre classe?). Cette approche a souvent été justifiée de façon plus ou moins *ad hoc* et est sans doute justifiable pour certains langages[5].

Classification extensionnelle

Mais [Donini *et al.*, 1994] montre que cette réduction de la vérification d'instances à une subsomption intensionnelle est en fait insuffisante dans de nombreux cas.

Pour montrer que les sous-rôles sont une cause importante de complexité [Nebel, 1990] donnait déjà un exemple, celui de l'`Equipe-moderne` (figure 5.2), qui montrait clairement qu'une vérification d'instance strictement intensionnelle, par comparaison de deux termes, ne peut pas être complète : dans l'exemple, `MARY` (un `Humain`) est classé(e) comme une `Femme`, par défaut, parce qu'aucun autre `membre` ne peut jouer ce rôle, ce sont tous des `Hommes` :

$$\mathcal{H};\mathcal{W} \models \texttt{(Woman MARY)}$$

Ainsi, avec les sous-rôles et éventuellement des fonctionnalités plus simples, la vérification d'instance, donc la classification, nécessitent des techniques énumératives à la CSP, ce qui explique sans doute les techniques de systèmes de contraintes utilisées dans $\mathcal{ALC}$ et $\mathcal{ALCNR}$ [Schmidt-Schauß et Smolka, 1991; Buchheit *et al.*, 1993; Donini *et al.*, 1994][6].

5. Ce qui est un peu ironique si l'on songe que cet amalgame, même s'il n'est que technique, correspond à celui fait par les prototypes, dont [Brachman, 1983; Brachman, 1985] a été l'un des critiques les plus virulents. L'usage philosophique justifie pourtant cet amalgame : le Lalande (1926) considère que « subsomption » s'utilise aussi bien pour la relation classe-instance que pour la relation classe-classe.

6. Cet exemple montre aussi qu'il faut pouvoir faire des inférences non triviales sur la valeur d'un attribut, puisque

$$\mathcal{H};\mathcal{W} \models \texttt{(leader TEAM-A MARY)}$$

```
Human       ≤̇ Anything
Man         ≤̇ Human
Woman       ≤̇ Human
(disjoint Man Woman)
Set         ≤̇ Anything
member      ≤̇ anyrelation
Team        ≐ (and Set
                  (all member Human)
                  (atleast 2 member))
leader      ≤̇ member
Modern-Team ≐ (and Team
                  (atmost 3 member)
                  (atleast 1 leader)
                  (all leader Woman))
```

```
(Modern-team TEAM-A)
(Man         DICK)
(Man         HARRY)
(member      TEAM-A DICK)
(member      TEAM-A HARRY)
(member      TEAM-A MARY)
```

FIG. 5.2 – *L'exemple de l'*`Equipe-moderne` *de [Nebel, 1990] :* `MARY` *est une* `Woman` *parce que tous les autres* `member` *de* `TEAM-A` *sont des* `Man`*, que le rôle* `member` *est* fermé, *qu'il y a au moins un* `leader` *et que les* `leader` *sont des* `Woman`*. On a donc* $\mathcal{H};\mathcal{W} \models$ `(leader TEAM-A MARY)` *et, par conséquent,* $\mathcal{H};\mathcal{W} \models$ `(Woman MARY)`*.*

Si l'on ne met pas en œuvre ces techniques énumératives, la déduction restera sans doute correcte dans le sens formel énoncé plus haut, mais son incomplétude peut donner lieu à une incorrection majeure : ne pas détecter certaines inconsistances. Ainsi dans l'exemple de l'`Equipe-moderne` on pourrait rajouter que `MARY` est un `Homme` — `(Man MARY)` dans la figure 5.2 — sans que l'inconsistance soit détectée.

De plus, ce qui est en cause ici est la complétude de la comparaison de deux descriptions de classe et d'instance respectivement, donc la possibilité de réaliser une vérification « intensionnelle » d'instance, comme on a pu parler plus haut de subsomption intensionnelle. Dans l'exemple, la classification n'est pas purement intensionnelle dans la mesure où elle repose sur l'examen « extensionnel » de tous les autres candidats. L'énumération extensionnelle nécessaire dans ces cas conduit à des techniques à la CSP que Nebel refuse pour leur complexité : les CSP sont NP-complets. L'analogie entre subsomption et vérification d'instance permet de retourner la question : ne peut-il y avoir des cas où la subsomption, bien que décidable, ne peut pas se calculer de façon complète par une simple comparaison des descriptions des deux classes, « toutes choses égales par ailleurs » ?

Une subsomption (ou une vérification d'instance) intensionnelle complète offrirait, pas sa « localité », le comble de la déclarativité et de l'intuition [Carré *et al.*, 1995]. L'expressivité nécessaire des langages impose d'en faire son deuil, mais il faut alors accepter, comme le suggère [Attardi, 1991], d'être capable de résoudre des *puzzles* compliqués et difficiles, aussi bien pour l'homme que pour la machine, en acceptant aussi de recourir à des techniques extensionnelles et d'y mettre le temps, éventuellement.

Implications entre vérification d'instance et subsomption

[Borgida, 1992a] s'insurgeait contre la volonté de faire de la *TBox* le siège principal du raisonnement dans les LT. En réalité, comme nous l'avons souligné dans la présentation de la sémantique en adoptant une démarche aristotélicienne (cf. note 7.5, page 177), la sémantique est faite pour les objets et n'est qu'exprimée par les classes. Toutes les contraintes qui peuvent peser sur les classes — complétude de la subsomption, monotonie des domaines, types ou cardinalités des attributs, etc. — ne sont que secondaires du moment que le principal est assuré : vérification des domaines, types et cardinalités des attributs, complétude de la classification, etc.

La subsomption peut entraîner la classification, par la règle [$\trianglelefteq$-ferm], mais l'inverse est impossible, la règle réciproque

$$[\trianglelefteq\forall] \qquad \frac{\{\vdash (\alpha\ o_i)\}^{i=1,n} \qquad \{\vdash (\beta\ o_i)\}_{i=1,n}}{\vdash \alpha \trianglelefteq \beta}$$

étant tout à fait incorrecte, sauf peut-être si la description du monde est fermée (dans un sens particulièrement fort, cf. paragraphe 6.3.2, page 132), mais ce serait donner à la composante extensionnelle un pouvoir exagéré sur la composante intensionnelle, puisque la première ne serait plus une *extension conservative* de la seconde.

La règle [$\trianglelefteq$-ferm] du paragraphe 5.1.1 a pour conséquence de lier les inférences de classification et de subsomption : cela ne permet cependant pas de lier leurs complétudes respectives comme le montre CLASSIC dont les règles sont prises en compte par la classification mais pas par la subsomption ou l'incompatibilité [Borgida et Patel-Schneider, 1994]. Cette relation entre les deux problèmes fait néanmoins suspecter que, dans le cas général, la classification n'a de chance d'être complète que si la subsomption l'est, ce qui renvoie à toute la littérature consacrée à la décidabilité et à la complexité de la subsomption [Nebel, 1990; Patel-Schneider, 1989a] et qui correspond en fait aux résultats de [Donini *et al.*, 1994] [7].

De nombreuses règles de classification peuvent ainsi apparaître comme redondantes s'il existe une règle correspondante de subsomption. Avant d'éliminer ces redondances, il faut quand même bien vérifier que c'en est bien, les problèmes de subsomption et de classification n'étant pas forcément traités avec la même complétude : en d'autres termes, il faut s'assurer que la règle de subsomption est effectivement là, et qu'elle est bien équivalente à la règle de classification. Ce serait par exemple une grave erreur de supprimer les règles de classification corresondant aux « règles » de CLASSIC, sous prétexte qu'elles sont redondantes avec la subsomption ! Toute cuistrerie pédantesque mise à part, la classification est à la subsomption ce que l'ontogénèse est à la phylogénèse.

5.2.4 Déduction naturelle

[Borgida, 1992b] innove en proposant un système de déduction des LT, dans le style de la déduction naturelle de Gentzen, en particulier tel qu'il est repris par la théorie

7. On aurait pu penser que la vérification d'instance pourrait être plus facile dans son versant intensionnel et plus dure dans son versant extensionnel, ce qui aurait pu laisser une incertitude.

des types dans la *sémantique naturelle*[8] de [Kahn, 1987], puis dans la plupart des approches de théorie des types appliquées aux objets [Abadi et Cardelli, 1996]. Les règles d'inférence ainsi obtenues représentent les divers cas qu'un algorithme doit prendre en compte (TSUB de [Nebel, 1990] par exemple), et cette approche est censée aboutir aux mêmes résultats que la sémantique opérationnelle explicitée par ces algorithmes.

Cette approche semble intéressante mais limitée :

- elle offre une sémantique opérationnelle assez déclarative et intuitive ;
- elle ne paraît pas pouvoir exprimer une subsomption autre qu'intensionnelle, voire structurelle, en tout cas de façon complète : cela va bien en cas d'indépendance des attributs, ou pour traiter les cas simples de dépendance (un sous-rôle, un super-rôle) mais il semble douteux qu'elle soit encore adéquate pour décrire les cas complexes.

La question serait ainsi de savoir si cette technique ne bute pas elle-aussi sur ce qui provoque la complexité du calcul de la subsomption, et sur ce que Nebel refuse de traiter. L'intérêt résulterait de cette convergence de complexité : complexité du problème et complexité d'expression des règles d'inférence. Plus précisément, dans le cas où le problème est décidable, est-il possible de l'exprimer de façon complète par un ensemble de règles de ce type ?

On retombe ainsi sur la difficulté exhibée au paragraphe précédent : dans les situations plus complexes, la subsomption structurelle n'est certainement pas complète, mais la technique de déduction à la Gentzen peut-elle l'être plus ? Encore plus précisément, les règles peuvent-elles exprimer autre chose qu'une subsomption intensionnelle, voire structurelle ?

Il est en tout cas remarquable que [Borgida et Patel-Schneider, 1994] proposent un algorithme complet basé sur une technique de « graphes de description », tout en donnant pour objectif à la technique proposée par [Borgida, 1992b], et comme ce dernier d'ailleurs, de caractériser les inférences effectivement réalisées par le système dans le cas où il n'est pas complet. Il s'agit alors de fournir ce que les auteurs appellent une *proof-theoretic semantics* qui est plus de l'ordre de la sémantique opérationnelle que d'une « vraie » *model-theoretic semantics*.

Finalement, cette technique est un moyen de caractériser les inférences effectivement réalisées par les algorithmes, ce qui peut avoir deux objectifs :

- fournir une formalisation intermédiaire pour aider à la correction des algorithmes ;
- fournir une sémantique alternative comme support de l'intuition de l'utilisateur.

Mais le premier point n'a d'intérêt que si les règles sont « complètes » vis-à-vis des algorithmes, c'est-à-dire si les deux systèmes d'inférences sont effectivement équiva-

8. Il est piquant de constater ce passage au sémantique d'une technique de déduction, d'essence syntaxique donc. Le passage par la sémantique naturelle paraît cependant superflu : la sémantique naturelle donne des règles d'inférence sur les types, alors que Borgida énonce les règles d'inférence d'une logique. Il aurait pu les prendre directement chez Gentzen.

lents[9]. Et le deuxième point n'a d'intérêt que si le système de déduction n'est pas sémantiquement complet, sinon autant se reposer sur la sémantique dénotationnelle. On retombe alors sur une tentative analogue à celle de [Patel-Schneider, 1989b] pour résoudre par la sémantique le compromis *expressivité-complétude-complexité*.

5.2.5 Comparaison locale ou énumération exhaustive

Les diverses techniques utilisées pour la déduction dans les SC, qu'il s'agisse de calcul de subsomption ou classification, se divisent ainsi en deux grandes familles :

- les techniques locales de comparaison de deux termes, « toutes choses égales par ailleurs », dont la complexité est relativement bonne (quoique les sous-rôles puissent déjà notablement compliquer les choses) mais dont la complétude ne saurait résister à une expressivité un tant soit peu poussée [Nebel, 1990; Borgida, 1992b; Borgida et Patel-Schneider, 1994] ;
- les techniques exhaustives et non locales, qu'elles soient à base de contraintes [Schmidt-Schauß et Smolka, 1991; Buchheit *et al.*, 1993], ou de logique propositionnelle [Paramavisam et Plaisted, 1996].

La répartition des diverses techniques dans ces deux classes mérite un examen un peu plus poussé.

Deux catégories de sémantiques correspondent à cette dichotomie : d'un côté, une sémantique opérationnelle partielle, *proof-theoretic*, même si très déclarative et intuitive, et basée sur un ensemble de règles d'inférences à la Gentzen ; de l'autre côté, une sémantique dénotationnelle complète, *model-theoretic*, mais définissant un problème complexe, voire indécidable, avec des effets combinatoires de type *puzzle* finalement très peu intuitifs.

NOTE *5.2* SC *et* CSP

La comparaison avec les CSP[10] n'est pas que pure analogie. Elle repose sur le constat suivant : lorsque P est un sous-rôle de Q, chaque valeur $Q(o)$ joue le rôle de *domaine* (au sens des CSP) pour la *variable* (toujours au sens des CSP) $P(o)$.

9. On peut néanmoins imaginer une architecture logicielle dans laquelle une première couche d'algorithmes serait caractérisée par les règles, une deuxième couche prenant en charge les inférences non caractérisées, par exemple extensionnelles et complexes. Ce serait aussi un moyen de se sortir du compromis *expressivité-complétude-complexité*.

10. Après réflexion, il apparaît que le problème du *zèbre* s'exprime bien en logique terminologique, à condition d'avoir des rôles inverses. A vérifier, je ne suis plus sûr de moi : je n'ai pas noté la preuve (elle ne tenait pas dans la marge !) On rapprochera aussi les contraintes des CSP de la technique des *constraints systems* utilisée par [Buchheit *et al.*, 1993] comme technique de preuve.

Chapitre 6

La classification d'instances

Comment pourrait-on classer les verbes qui suivent : cataloguer, classer, classifier, découper, énumérer, grouper, hiérarchiser, lister, numéroter, ordonnancer, ordonner, ranger, regrouper, répartir ?

Georges Perec, *Penser/Classer*, 1982[1].

La classification d'instances, constitue le problème le plus typique offert par les SC. Il consiste à reconnaître un objet comme instance d'une classe, et nous allons considérer ce problème isolément, même s'il est un peu artificiel de l'isoler du problème plus général de la déduction dans un SC : l'exemple de l'`Equipe-moderne` a en effet montré que des inférences non triviales peuvent permettre de calculer la valeur d'un attribut, ce qui peut mener à une classification.

Plusieurs raisons justifient de traiter spécifiquement la classification :

- c'est la base d'un raisonnement spécifique, étudié en tant que tel ;
- certains systèmes ne pratiquent que la classification, ou ne pratiquent comme inférences que les inférences de classification, à l'exclusion donc des inférences nécessaires à la solution de l'`Equipe-moderne` ;
- certains systèmes ont proposé une extension du problème de la classification dont il n'existe aucun analogue pour la déduction elle-même dans toute sa généralité.

6.1 Introduction à la classification d'instances

En toute généralité, le terme de « classification » s'applique aussi bien à la classification de classes (insertion d'une nouvelle classe dans la hiérarchie) qu'à la classification d'instances (rattachement d'une instance à sa ou ses classes). Nous ne traitons ici que le deuxième problème.

1. Dans la citation de Perec, remplacer le quatrième mot par l'un des « verbes qui suivent ». Les essayer tous, à haute voix.

6.1.1 Le raisonnement par classification

Le « raisonnement par classification » est souvent présenté — dans [Napoli, 1991; Napoli, 1992] par exemple, à la suite de [Lipkis, 1982] — comme une recherche descendante, partant de la classe la plus générale ($\top$) pour calculer d'abord les *subsumants les plus spécifiques* (SPS), puis les *subsumés les plus généraux* (SPG), de l'entité à classer. Le deuxième point ne concerne que la classification de classe.

Cette vision idéale est d'une part souvent contredite, en pratique, par des approches qui diffèrent du point de vue algorithmique (mais pas du point de vue conceptuel), ainsi de la théorie du *focus* de Ellis (1993) [Guinaldo, 1995; Guinaldo, 1996].

De plus, pour la classification d'instance, cette vue purement intensionnelle de la classification est insuffisante puisque nous avons vu plusieurs types d'inférences plus extensionnelles (cf. paragraphe 5.2.3, page 106). Pire, cette vision partielle de la classification ne s'applique qu'aux classes définies : elle est complètement inadéquate pour les classes primitives.

Il s'agit là, néanmoins, d'une bonne approximation du processus principal et on supposera donc donné un *objet à classer*.

NOTE *6.1* *Non objet à classer*

[Gonzalez-Gomez, 1996] est un cas extrême, mais typique, pour lequel l'objet à classer n'est même pas un objet, mais une simple liste attribut-valeur, ce qui fait que $Isa_{\mathcal{W}}(o) = \emptyset$. C'est difficilement justifiable dans un cadre objet où le statut ontologique des classes est fort.

6.1.2 Classification et déduction

Jusqu'ici, sans faire d'hypothèse particulière sur les propriétés du SC, le seul raisonnement réellement supporté par le SC est l'héritage, c'est-à-dire déduire de l'appartenance d'un objet à une classe les valeurs admissibles de ses attributs.

La classification d'instances n'est qu'un cas particulier du problème de la déduction dans les systèmes classificatoires tel qu'on l'a vu dans sa généralité au chapitre 5, mais restreint à la composante extensionnelle : étant donné un système classificatoire et une description du monde $\mathcal{W}$,

1. déterminer toute les formules valides, c'est-à-dire $\{d \mid \mathcal{H}; \mathcal{W} \models d\}$, la fermeture de $\mathcal{W}$ par $\models$;

2. ou plus généralement encore, déterminer tous les modèles minimaux du SC.

La classification d'instance va préciser le problème en ne considérant que des descriptions de la forme $(\alpha\ o)$. Étant donné un *objet à classer* o, partiellement connu par ses classes ($Isa_{\mathcal{W}}(o)$), ses attributs ($Attr_{\mathcal{W}}(o)$), et les valeurs de ces derniers ($P_{\mathcal{W}}(o)$), il va s'agir plus précisément de

3. déterminer la meilleure approximation valide $Isa^{\mathcal{W}}(o)$ de $Isa(o)$, qui représente les classes de o dans tous les modèles ; par rapport à l'énoncé (1), on ne s'intérese plus à n'importe quelle description et il s'agit de déterminer l'ensemble $\{\beta \mid \mathcal{H}; \mathcal{W} \models (\beta\ o)\}$;

4. plus généralement, déterminer les (des) classes dont o *peut*, ou *ne peut pas*, être l'instance, dans un sens à préciser ;

Il y a une différence fondamentale entre l'énoncé 1 (ou 3) et l'énoncé 4. Le premier décrit le problème de tout système déductif : que peut-on déduire avec certitude qui s'énonce dans les termes du formalisme? La solution abstraite du problème consiste à remplacer l'opérateur de conséquence $\models$ par un opérateur de dérivation $\vdash$.

En revanche, le 4, comme le 2, ne permet pas d'exprimer ses conclusions dans le formalisme et nécessite un méta-langage. On peut en fait le décliner extensionnellement ou intensionnellement, mais aussi relativement au domaine modélisé.

Jusque là, sans hypothèse particulière sur la définition des classes ou sur des propriétés spécifiques du SC, donc en ne considérant que les axiomes (4.8-4.13), le seul raisonnement réellement supporté par le SC est l'héritage, c'est-à-dire déduire de l'appartenance d'un objet à une classe les valeurs admissibles de ses attributs. En fait, dans le cas général, deux règles de classifications non triviales sont possibles, que nous rappeleons ici :

$$[\trianglelefteq\text{-ferm}] \qquad \frac{\mathcal{H};\mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H} \vdash \alpha \trianglelefteq \beta}{\mathcal{H};\mathcal{W} \vdash (\beta\ o)}$$

$$[\mathit{type}\text{-infér}] \qquad \frac{\mathcal{H};\mathcal{W} \vdash (\alpha\ o) \qquad \mathcal{H};\mathcal{W} \vdash (P\ o\ o')}{\mathcal{H};\mathcal{W} \vdash (\mathit{type}_P(\alpha)\ o')} \qquad \alpha \in \mathrm{dom}(\mathit{type}_P)$$

La première de ces deux règles dit que si o est instance d'une classe α subsumée extensionnellement par β, il est aussi instance de β : c'est la définition de la subsomption extensionnelle. La seconde dit que si o' est valeur d'un attribut P d'un objet o instance de α, alors o' est instance de $\mathit{type}_P(\alpha)$: c'est la version active de la contrainte exprimée par le type des attributs.

Ce sont les deux seules règles générales de classification dans un SC quelconque. En effet, parmi les axiomes (4.7-4.13), seul l'axiome (4.10) permet de conclureà l'appartenance d'un objet à l'extension d'une classe. Allié à l'axiome (4.13), on obtient la deuxième règle : toute valeur d'un attribut d'un objet *est* instance des types de l'attribut dans les classes de l'objet. C'est une règle d'inférence de type qui pourrait avantageusement remplacer la vérification de type constituée par l'axiome (4.10) que pratiquent tous les systèmes d'objets : ce n'est qu'une question d'interprétation de la copule (*est*) dans la phrase précédente, par la forme passive *doit être* ou par la forme active *devient*. C'est, au sens strict, une coercition. Cette règle d'inférence a la particularité d'être la seule à permettre le franchissement de la barrière des classes primitives, que ce soit pour les logiques terminologiques ou la représentation des connaissances par objets, dans la mesure où $\mathit{type}(\alpha, P)$ peut être une classe aussi bien primitive que définie.

Cela vaut bien sûr autant pour les logiques terminologiques que pour la représentation des connaissances par objets : les premières peuvent, *quand même*, classer dans des classes primitives, tandis que la seconde peut, *quand même*, classer automatiquemnet. Mais cela reste marginal, pour les deux.

Il est ainsi notable que, dans toute la généralité de la sémantique des SC, les seules déductions possibles sont

- soit relativement triviales comme les précédentes ; tout SC, aussi général soit-il, permet (ou devrait permettre) de faire de la classification automatique d'instances, au moins sur les valeurs ([*type*-infér]), à la seule condition qu'il permette une migration des instances, c'est-à-dire une inférence de type, et pas seulement une vérification de type. Si l'inconsistance logique est assimilée à une erreur de type, cette inférence de type n'est qu'une généralisation, finalement assez élémentaire et naturelle, de la vérification de type.

- soit, si le formalisme comporte un équivalent des sous-rôles, trop compliquées pour être envisagées par Nebel, comme déduire une valeur pour un sous-rôle du fait que son super-rôle est fermé et que la valeur considérée est la seule admissible.

Si C++ ne pratique évidemment pas cette deuxième inférence, il faut bien constater que SHIRKA non plus, malgré la simplicité de l'inférence. Les raisons tiennent autant à l'architecture logicielle du système — la vérification de type lors de l'affectation fait partie du noyau du langage sur lequel la classification se greffe de façon assez extérieure — qu'à une philosophie implicite pour laquelle une affectation ne peut pas être la cause d'une classification : si le type est incorrect, ce doit être une erreur de l'utilisateur. Pourtant la notion d'erreur se ramène ici à celle d'inconsistance de la description du monde : si la valeur de l'attribut n'est pas du bon type mais est « classable » dans ce type, cette classification est valide et devrait être faite automatiquement.

NOTE *6.2* *Erreur, inconsistance ou inférence*

Le remplacement de la notion d'erreur par la notion d'inconsistance n'est pas si simple que cela. D'une part, la notion d'inconsistance s'applique mal à la composante intensionnelle seule, d'autre part, des questions d'interaction et d'ergonomie entrent en jeu. Faut-il par exemple traiter le caractère monotone de $type_P$ comme une propriété imposée et sanctionnée par une erreur, ou comme une règle d'inférence :

[$type_P$-monotone] $$\frac{\mathcal{H} \vdash \alpha \trianglelefteq \beta}{\mathcal{H} \vdash type_P(\alpha) \trianglelefteq type_P(\beta)}$$

ou encore, n'imposer aucune contrainte sur les types, la contrainte sur les instances — la seule qui compte réellement — étant maintenue par la sémantique du SC ? Nous avons suivi ce dernier point de vue dans les règles que nous avons explicitées.

Ces distinctions correspondent plus ou moins aux notions de *contraintes d'intégrité* ou de *règles déductives* faites par les BD déductives.

6.1.3 Classification valide ou incertaine

Les diverses incomplétudes, qu'elles soient sémantiques, extensionnelles ou définitionnelles, sont autant d'obstacles à la classification.

En l'absence simultanée d'incomplétude extensionnelle — lorsque tout ce qu'il faut connaître est connu — et d'incomplétude définitionnelle — lorsque les classes considérées sont définies —, une classification automatique est valide. C'est l'approche suivie par les LT. En revanche dès que l'une ou l'autre de ces deux formes d'incomplétude apparaît, la classification automatique n'est possible que dans les cas particuliers vus plus haut.

Entre ces deux extrêmes, lorsqu'il y a une incomplétude définitionnelle mais pas extensionnelle, il est envisageable de considérer une classification incertaine, qui, n'étant pas valide, ne peut pas être automatique et doit être interactive. Cette approche interactive consiste à demander à l'utilisateur la caution que la sémantique ne peut fournir : si l'on ne peut pas déterminer automatiquement les classes de l'objet à classer dans tous les modèles, on se retourne vers l'utilisateur pour qu'il choisisse certains modèles. C'est l'approche suivie par SHIRKA, dont nous allons montrer qu'elle peut se généraliser pour englober la classification automatique des LT.

Cette classification incertaine va se décliner sous des formes successives extensionnelle puis intensionnelle, cette dernière de façon abstraite ou concrète, entraînant des questions de correction et de complétude sémantique.

6.2 La classification incertaine

Mal nommer les choses, c'est ajouter aux malheurs du monde
Albert Camus

L'idée de départ est de ne plus se contenter de calculer ce qui est vérifié par tous les modèles, mais de s'intéresser aussi à ce qui n'est vérifié que par certains modèles. Il va s'agir ensuite de reproduire dans la classification les deux versants sémantique et syntaxique, extensionnel et intensionnel, de notre cadre abstrait.

6.2.1 Classification incertaine : version extensionnelle

Une approche sémantique de la classification conduit à définir plusieurs catégories de classes relativement à l'objet à classer o. Une classe β peut être :

valide – c'est une classe de l'objet o dans tous les modèles : $\forall \mathcal{M}, \ \models_{\mathcal{M}} (\beta\ o)$, qui équivaut à $\mathcal{H}; \mathcal{W} \models (\beta\ o)$; l'ensemble des classes valides est $Isa^{\mathcal{W}}(o)$;

satisfaisable – c'est une classe de o dans au moins un modèle : $\exists \mathcal{M}, \ \models_{\mathcal{M}} (\beta\ o)$; l'intérêt de ces classes satisfaisables[2] réside dans le fait que l'utilisateur (ou tout programme extérieur au SC) peut rattacher d'autorité un objet à une classe satisfaisable sans risque d'erreur : il existe un modèle (une solution) contenant ce rattachement, et, par ce geste, l'utilisateur choisit ce (ou ces) modèle(s) ;

insatisfaisable – ce n'est une classe de o dans aucun modèle : $\forall \mathcal{M}, \ \not\models_{\mathcal{M}} (\beta\ o)$;

falsifiable – ce n'est pas une classe de o dans au moins un modèle : $\exists \mathcal{M}, \ \not\models_{\mathcal{M}} (\beta\ o)$.

2. Dans l'absolu, une classe *satisfaisable* (resp. *insatisfaisable*) serait une classe *cohérente* (resp. *incohérente*) : il faut donc souligner que l'usage que l'on fait de ce terme est relatif à l'*objet à classer*.

Si la description du monde est consistante, les classes valides sont satisfaisables et les classes insatisfaisables sont falsifiables. En revanche, en cas d'inconsistance, toutes les classes sont à la fois valides et insatisfaisables. Dans tous les cas, valide et falsifiable d'un côté, satisfaisable et insatisfaisable de l'autre, forment des couples de complémentaires. La classification de o est donc caractérisée par le couple valide/insatisfaisable que l'on peut étendre en une partition en rajoutant les classes satisfaisables non valides (ou satisfaisables et falsifiables, ou encore falsifiables non insatisfaisables). Dans la suite nous utiliserons peu « falsifiable ».

Dans les logiques terminologiques, la classification d'instance consiste à calculer des classes valides, tout ou partie suivant la complétude sémantique de la classification, sans s'intéresser aux classes satisfaisables ou insatisfaisables. Dans une approche plus interactive qui est celle de la représentation des connaissances par objets, l'objectif est de soumettre à l'utilisateur des classes satisfaisables en lui demandant s'il veut y rattacher l'objet : un tel rattachement revient en fait à éliminer tous les modèles dans lesquels l'objet n'est pas instance de la classe.

Notons que c'est la relation entre une instance et une classe qui est ainsi qualifiée de valide, satisfaisable ou insatisfaisable : on pourra donc parler, tout aussi bien, d'instances valides, satisfaisables ou insatisfaisables, relativement à une classe. Formellement, on peut introduire des prédicats Val, Sat, $InSat$ et $Fals$ aussi bien que des ensembles de classes, voire des ensembles d'instances : profitant du caractère non réflexif de nos SC (cf. paragraphe 2.4.2, page 32), on gardera la même notation pour les trois avec la convention d'équivalence

$$\alpha \in Val(o) \iff Val(\alpha, o) \iff o \in Val(\alpha)$$

de même que pour Sat, $InSat$ et $Fals$. On remarquera aussi que les quatre ensembles $Val(o)$, $Sat(o)$, $InSat(o)$ et $Fals(o)$ sont des idéaux, non seulement de $(\mathcal{X}, \preceq)$ (ou de la relation duale) mais aussi de la subsomption extensionnelle : ils sont donc caractérisés par les minimaux des deux premiers et les maximaux des deux derniers, ce qui est utile d'un point de vue calculatoire. Malheureusement, tous ces ensembles sont aussi difficiles à calculer les uns que les autres.

Les classes solution

Comme seuls les modèles du problème intéressent réellement un utilisateur, il faudrait définir les ensembles de classes correspondants, obtenus par une quantification sur ce sous-ensemble de modèles :

- les classes *solution valide* sont les classes de l'objet dans tous les modèles du problème ;
- les classes *solution* sont les classes de l'objet dans un modèle du problème ;
- les classes *non solution* ne sont les classes de l'objet dans aucun modèle du problème.

Cette définition est un peu vaine puisqu'il n'y a aucun moyen d'appréhender formellement ces ensembles de classes, mais c'est un moyen de rappeler les limites de cette

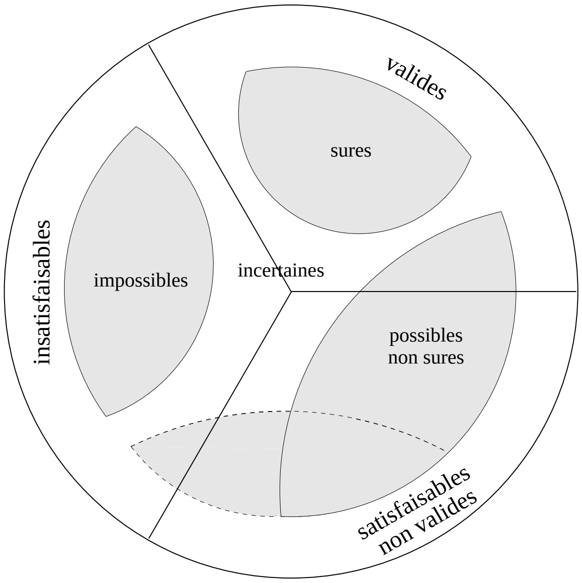

FIG. 6.1 – *Partition des classes par les classifications incertaines extensionnelle et intensionnelle abstraite*

approche. Aussi, le choix d'une classe satisfaisable est bien sans erreur vis-à-vis du système de déduction (il ne peut pas introduire d'inconsistance immédiate) mais il peut être en désaccord avec l'état du monde, ce qui se traduirait ultérieurement par une contradiction (soit une inconsistance interne, soit un désaccord entre le système et ce qu'il représente), si ce n'est pas une classe solution. Dans tous les cas, cette contradiction n'est pas appréhendable, *au moment du choix*, dans les catégories sémantiques du SC.

6.2.2 Classification incertaine : version intensionnelle abstraite

Bien entendu, tout énoncé au niveau des modèles doit se traduire, *in fine*, dans le système de déduction, au niveau de l'intension, ce qui doit conduire à déterminer de nouvelles catégories de classes associées à l'objet à classer o. C'est là que les difficultés commencent pour les systèmes de représentation des connaissances par objets comme SHIRKA, TROPES ou FROME dont la classification repose sur la détermination de différentes catégories de classes correspondant plus ou moins aux catégories de classes définies ici.

Une première façon de faire est abstraite et va consister à remplacer méta-syntaxiquement l'opérateur de conséquence $\models$ par celui de dérivation $\vdash$. Cela marche très bien pour les classes valides. Ainsi, une classe β pourra être

sûre – c'est une classe dont la validité est effectivement déduite : $\mathcal{H}; \mathcal{W} \vdash (\beta\ o)$.

Malheureusement, ce remplacement automatique de $\models$ par $\vdash$ n'est pas possible, en l'état, pour les deux autres catégories. Une solution consiste à introduire les prédicats

Val, Sat, $InSat$ et $Fals$ dans le système de déduction, en les transformant de prédicats méta-syntaxiques en des prédicats syntaxiques[3]. Une classe β pourra alors être :

sûre – c'est une classe dont la validité est effectivement déduite :
$\mathcal{H}; \mathcal{W} \vdash Val(\beta, o)$.

possible – c'est une classe dont la satisfaisabilité est effectivement déduite :
$\mathcal{H}; \mathcal{W} \vdash Sat(\beta, o)$.

impossible – c'est une classe dont l'insatisfaisabilité est effectivement déduite :
$\mathcal{H}; \mathcal{W} \vdash InSat(\beta, o)$.

falsifiée – c'est une classe dont la falsifiabilité est effectivement déduite :
$\mathcal{H}; \mathcal{W} \vdash Fals(\beta, o)$.

Bien sûr le fait de se servir des prédicats Sat et $InSat$ à droite de l'opérateur de dérivation signifie que l'on peut aussi les utiliser à gauche : le langage de la description du monde s'enrichit donc de ces deux prédicats (pour Val, c'est déjà implicite puisque $Val(\alpha, o)$ équivaut à $(\alpha\ o)$). Il devient ainsi possible de faire un *rattachement négatif* en disant, par $InSat(\alpha, o)$ qu'un objet n'est pas instance d'une classe : sémantiquement, cela revient à éliminer tous les modèles dans lesquels l'objet est instance de la classe. En revanche, affirmer, par $Sat(\alpha, o)$, qu'un objet « peut être instance d'une classe » n'est pas très clair sémantiquement, puisque cela ne peut pas se traduire par l'élimination de certains modèles : Sat est une propriété momentanée de la description du monde, qui ne sera pas forcément conservée lors de son évolution monotone. On conviendra donc que Sat et $InSat$ peuvent tous deux figurer à gauche de $\vdash$, mais que seul $InSat$ peut être affirmé.

La possibilité d'affirmer $InSat$ revient à avoir dans la composante extensionnelle une négation généralisée sur les classes (sans l'avoir forcément dans la composante intensionnelle : en effet, pouvoir affirmer ou déduire que deux classes sont incompatibles ne signifie pas que l'on définit implicitement la négation d'une classe). On pourra donc avoir des résultats de classification sans le résultat de subsomption implicite qui lui correspond, à cause de l'absence de la négation dans la composante intensionnelle, mais avec un résultat d'incompatibilité.

De la même manière que pour la version extensionnelle, on introduit les prédicats et/ou ensembles Sur, Pos et $ImPos$, qui vérifient, si la déduction est correcte :

$$Sur(o) \subseteq Val(o) \qquad Pos(o) \subseteq Sat(o) \qquad ImPos(o) \subseteq InSat(o)$$

et si la description du monde est consistante,

$$Sur(o) \subseteq Pos(o)$$

Comme pour la classification extensionnelle, les ensembles de classes sûres, possibles ou impossibles sont des idéaux de la relation de spécialisation (ou de sa duale), mais pas forcément des subsomptions extensionnelle ou intensionnelle.

3. Le détail de cette assomption sera pudiquement passé sous silence.

Enfin, $Pos(o)$ et $ImPos(o)$ sont toujours disjoints, comme $Sat(o)$ et $InSat(o)$, mais pas forcément complémentaires : il existe des classes qui ne sont ni possibles, ni impossibles, qu'on peut qualifier d'*incertaines*, et pour lesquelles on introduit le prédicat et/ou ensemble $InCert$ (figure 6.1). Pour finir, nous appellerons *envisageable* une classe qui n'est pas *impossible*, qui est donc *possible* ou *incertaine*.

Soulignons que cela n'a aucun sens de faire figurer ces prédicats dans les règles de déduction car ils sont définitivement méta-syntaxiques : $Sur(\beta, o)$ équivaut à $\vdash Val(\beta, o)$.

6.2.3 Classification incertaine : version structurelle

Comment concrétiser ces définitions abstraites ? Intuitivement et approximativement, la comparaison des descriptions d'un objet et d'une classe conduit à considérer trois catégories de classes :

- les classes *impossibles*, dont o viole au moins une des contraintes :

$$\exists P \in \mathcal{P}_\beta \cap Attr_\mathcal{W}(o) : P_\mathcal{W}(o) \notin Dom_P(\beta) \tag{6.1}$$

- les classes *possibles*, dont o ne viole aucune contrainte :

$$\forall P \in \mathcal{P}_\beta \cap Attr_\mathcal{W}(o) : P_\mathcal{W}(o) \in Dom_P(\beta) \tag{6.2}$$

- les classes *sûres*, dont o vérifie toutes les contraintes :

$$\forall P \in \mathcal{P}_\beta : P_\mathcal{W}(o) \in Dom_P(\beta) \tag{6.3}$$

  Ce sont des classes possibles vis-à-vis desquelles l'objet est fermé, dans un sens qui sera discuté plus loin.

Dans tous les cas, il s'agit d'une classification intensionnelle, voire structurelle, au sens de la subsomption structurelle discutée au paragraphe 5.2.2. Le choix du vocabulaire étant à la fois délicat (comme le lecteur doit déjà s'en douter et comme la suite le confirmera) et limité, pour ne pas introduire de nouveaux adjectifs, on préfixera les trois adjectifs précédents de l'adverbe *structurellement*. Comme d'habitude, on introduit les prédicats et/ou ensembles $SPos$, $SImPos$ et $SSur$. Comme pour la classification extensionnelle ou intensionnelle, les ensembles de classes structurellement sûres, possibles ou impossibles sont des idéaux de la relation de spécialisation (ou de sa duale), mais aussi de la subsomption intensionnelle (mais pas de la subsomption extensionnelle).

Dans ces définitions (6.1-6.3), les comparaisons entre la valeur et le domaine de l'attribut sont doublement approximatives : d'une part, la relation entre les valeurs de P et $type(\beta, P)$ peuvent elles-mêmes être qualifiées de structurellement sûre, possible ou impossible, ou de tout autre qualificatif ; d'autre part, la cardinalité peut n'être que provisoire et toutes les valeurs pas encore connues. De plus, la formulation utilisée laisse entendre qu'il s'agit d'une comparaison effective entre la valeur et le domaine : en réalité, il peut aussi s'agir d'une comparaison entre le domaine de l'attribut pour l'objet, tel qu'il résulte de ses classes connues, avec le domaine de l'attribut dans β : il peut en effet exister $\alpha \in Isa_{\mathcal{W}}(o)$ tel que $Dom(\alpha, P) \subseteq Dom(\beta, P)$, ce qui permet de conclure malgré l'incomplétude extensionnelle. Inversement, les classes de o déjà connues peuvent être incompatibles avec β (au sens de ∇, parce que $Dom(\alpha, P) \cap Dom(\beta, P) = \emptyset$), ce qui assure l'impossibilité. Enfin, les comparaisons varient notablement entre (6.2) et (6.1) d'un côté et (6.3) de l'autre : dans un cas il s'agit d'éliminer ce qui est évidemment impossible alors que dans l'autre cas il s'agit de sélectionner ce qui est définitivement sûr.

On verra plus loin comment préciser techniquement ces comparaisons avec diverses notions de vérification et de compatibilité mais, dans la suite immédiate, nous ferons l'hypothèse angélique d'une définition « parfaite ».

Voyons d'abord où nous souhaitons et jusqu'où nous pouvons espérer en arriver : dans l'idéal, nous aimerions bien qu'une classe *structurellement impossible* (resp. *possible*, *sûre*) soit *impossible* (resp. *possible*, *sûre*), c'est-à-dire *insatisfaisable* (resp. *satisfaisable*, *valide*). Plus précisément, au stade d'abstraction où nous en sommes, y a-t-il une définition structurelle qui vérifie ces souhaits ?

On va voir que seul le premier point va combler notre attente : les classes structurellements impossibles sont évidemment insatisfaisables. Pour les autres, il va falloir se rabattre sur des objectifs moins ambitieux :

- pour une classe *structurellement sûre*, deux options sont envisageables :
  - la version forte en fait une classe *valide* lorsqu'elle est *définie* :

    (6.4) $$SSur(\alpha, o) \ \ \& \ \ \alpha \text{ défini} \Longrightarrow Val(\alpha, o)$$

    Ce serait en fait la définition abstraite des classes structurellements sûres : ce sont les classes qui sont valides quand elles sont définies. Cette équation vient utilement compléter la propriété d'idéal de $SSur$:

    (6.5) $$\alpha \trianglerighteq \beta \Longrightarrow \big(\forall o : SSur(\alpha, o) \Rightarrow SSur(\beta, o)\big)$$

  - la version faible (que l'on qualifiera de structurellement *probable*) en fait une classe privilégiée pour un rattachement interactif : la consistance du rattachement n'est pas garantie, mais elle est probable ;
- pour une classe *structurellement possible*, elle n'est qu'*incertaine* et, tant qu'elle n'est pas prouvée *insatisfaisable*, elle peut servir
  - soit pour des inférences très spécifiques, en jouant uniquement sur l'élimination des classes structurellement impossibles ;
  - soit comme « second choix » pour une classification interactive.

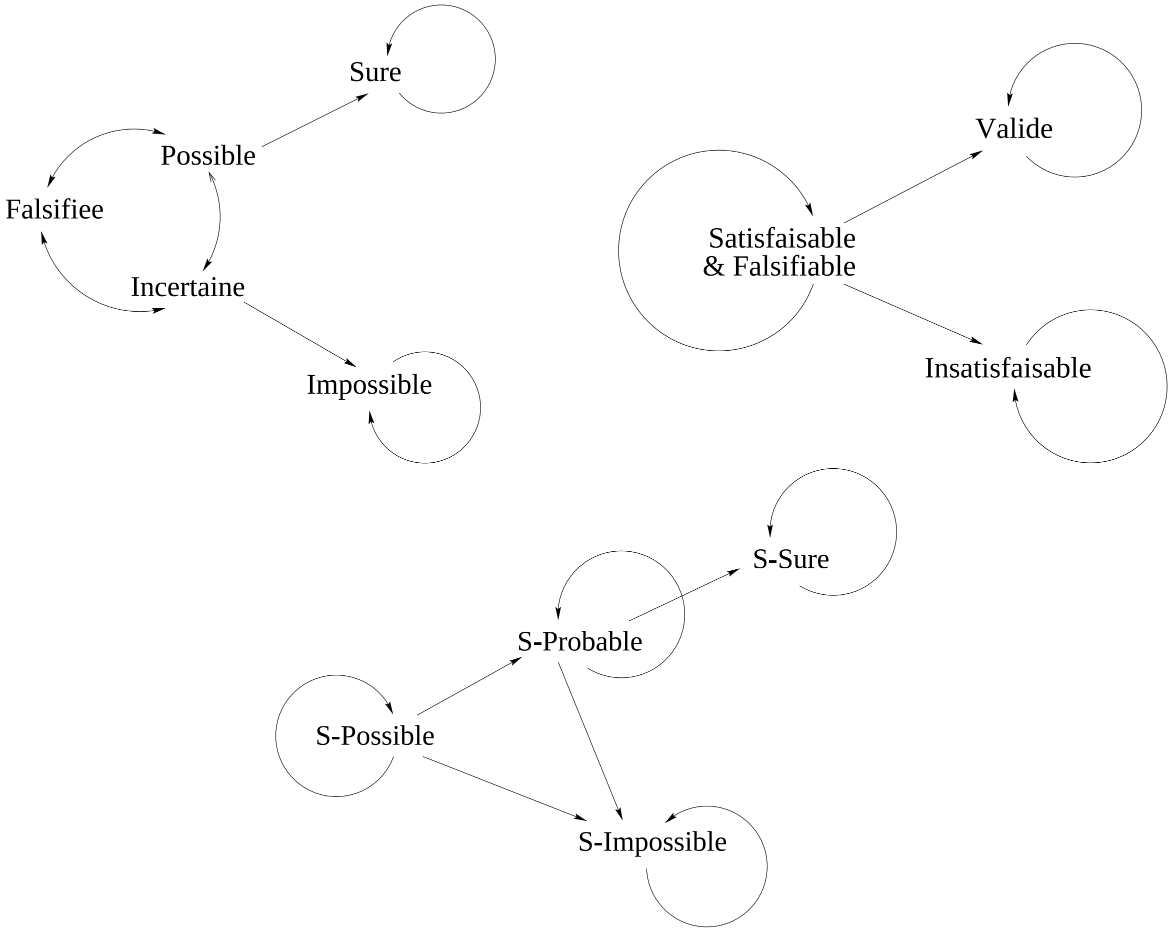

FIG. 6.2 – *Évolution temporelle des satisfaisabilité et possibilité : ajouter de l'information peut aussi bien éliminer les modèles qui falsifient ou vérifient la classe, que permettre de nouvelles inférences en sens inverse : en classification intensionnelle, toutes les transitions sont possibles.*

Évolution temporelle

L'évolution dans le temps des qualificatifs, au fur et à mesure de l'évolution monotone de la description du monde, est un bon moyen de les appréhender intuitivement (figure 6.2).

La version extensionnelle est la plus simple : une classe est valide, insatisfaisable ou satisfaisable et falsifiable. L'ajout monotone d'information ne peut qu'éliminer des modèles : la seule transition possible consiste à faire passer une classe de la troisième catégorie dans l'une des deux premières (figure 6.2, haut droite).

Dans la version intensionnelle abstraite, la déduction est monotone mais l'ajout d'information peut aussi entraîner de nouvelles inférences : toutes les transitions sont donc *a priori* possibles tant qu'une classe n'est pas sûre ou impossible (figure 6.2, haut gauche).

Pour la version structurelle, nous avons quelque latitude puisqu'elle n'a été définie qu'approximativement. Il est clair qu'une classe structurellement impossible le restera : le point crucial est d'obtenir la même stabilité pour les classes structurellement sûres, par analogie avec les deux autres versions (figure 6.2, bas), analogie justifiée par le fait que c'est une condition nécessaire de correction de ces modalités structurelles. Aucune nouvelle information ne peut contredire la sûreté structurelle : l'objet doit être *fermé*. Il est en particulier impossible d'ajouter de nouvelle assertion de valeur ($P\ o\ o'$) pour aucun attribut $P \in P_\beta$ sans aboutir à une inconsistance (*sauf si la cardinalité n'est pas limitée et que le type est imposé*).

Si une classe est structurellement possible dans la description du monde $\mathcal{W} = \mathcal{W}_k$, elle l'était aussi dans toutes les descriptions du monde passées, $\mathcal{W}_i, i < k$, avec $\mathcal{W}_i \subseteq \mathcal{W}_j$, si $i < j$, au moins dans toutes celles où l'objet à classer o était connu. Mais une classe structurellement possible peut devenir impossible dans les descriptions du monde à venir. En revanche, si une classe est structurellement sûre, elle ne l'a pas toujours été dans le passé (elle n'a été que possible), et elle le restera dans le futur.

6.2.4 Correction et complétude

Reste à savoir dans quelle mesure l'interprétation souhaitée de ces qualificatifs de *structurellement sûr* et *possible* est correcte. Une classe structurellement sûre est-elle valide, satisfaisable ou seulement envisageable ? Une classe structurellement possible est-elle satisfaisable ? Est-elle même envisageable ?

Notons d'abord que la question de la correction ne présente pas vraiment de difficulté :

- pour la définition abstraite des classes sûres, possibles, impossibles et incertaines, qui est correcte par construction dès lors que l'opérateur de dérivation est présumé correct ;
- pour les classes *structurellement impossibles* qui sont clairement insatisfaisables ;
- en corollaire, une classe satisfaisable est forcément structurellement possible, (de même qu'une classe incertaine ou structurellement sûre) ; si jamais la réciproque était vraie, il y aurait alors identité entre les notions structurelles et extensionnelles.

La complétude est en revanche plus difficile à atteindre. En particulier, il n'est pas nécessaire pour une classe insatisfaisable d'être structurellement impossible, pas plus que pour une classe valide d'être structurellement sûre : l'incomplétude extensionnelle en dispense.

Rappelons qu'il n'y a pas de définition concrète et unique des classes possibles, sûres et impossibles. Ce ne sont que des cas d'espèces : chaque SC spécifique en donne une définition concrète particulière, éventuellement implicite (c'est-à-dire purement opérationnelle ou algorithmique) par son système de déduction. Et un même SC peut présenter plusieurs systèmes de déduction différents dans leurs résultats, suivant la façon de résoudre le compromis complétude-complexité.

Les notions extensionnelles abstraites comme les classes valides, satisfaisables ou insatisfaisable se définissent entièrement d'une manière indépendante des spécificités du SC : on peut considérer qu'il s'agit d'une définition concrète d'une notion abstraite. En

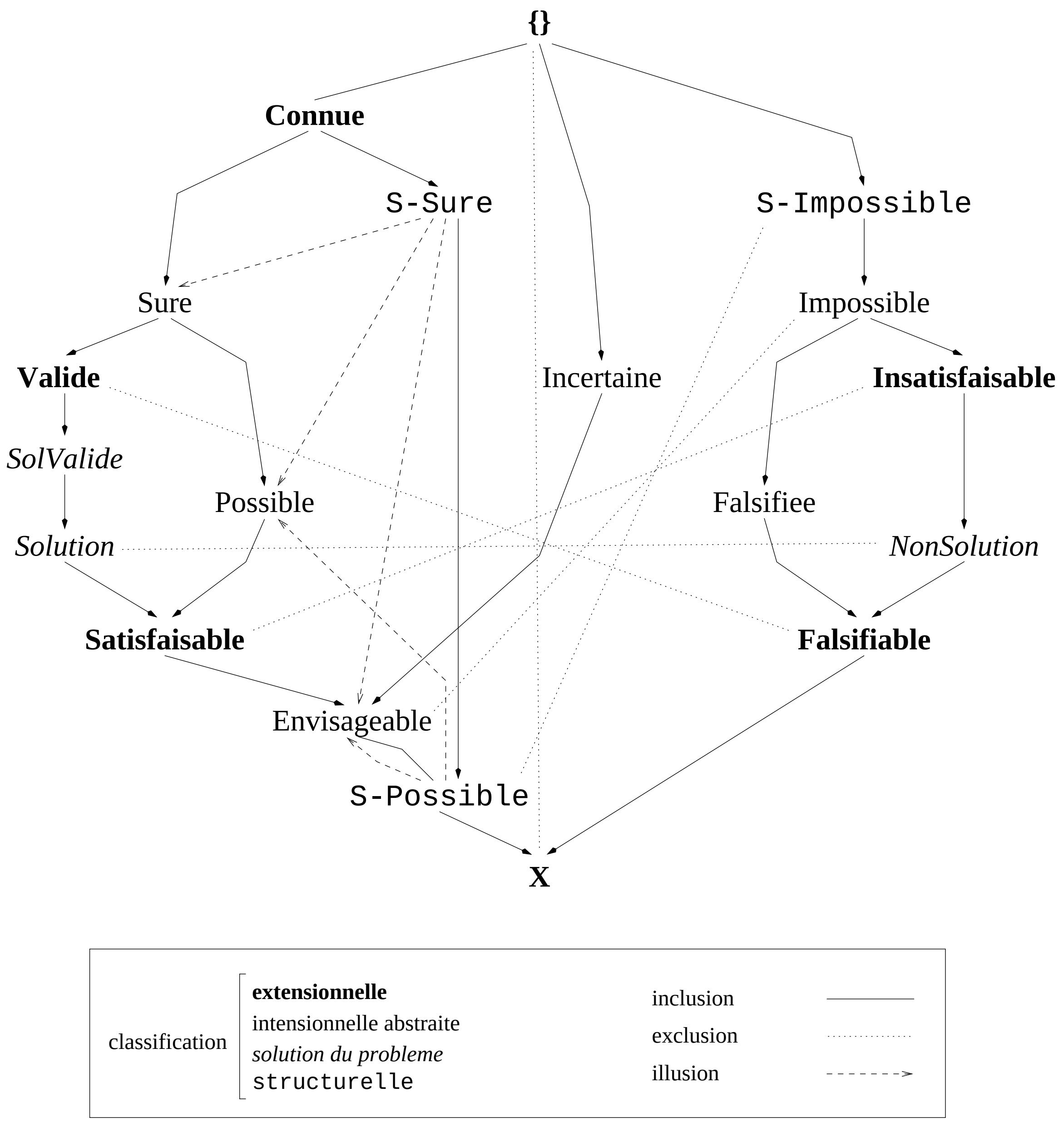

FIG. 6.3 – *Relations d'inclusion et d'exclusion entre les divers ensembles de classes résultant des classifications incertaines extensionnelle et intensionnelle abstraite et structurelle, pour une description du monde consistante.*

revanche, les notions intensionnelles abstraites ne peuvent se définir concrètement que partiellement : s'il a été possible de définir intuitivement les classes structurellement possibles et sûres, ce n'est qu'une approximation concrète de notions abstraites, et il faut bien juger de la correction de cette approximation. L'un des problèmes soulevés par [Borgida, 1992b; Borgida et Patel-Schneider, 1994], est que les algorithmes, lorsqu'ils sont incomplets, ne permettent pas toujours de caractériser ce qu'ils calculent.

Deux démarches inverses sont ainsi à mettre en œuvre :

- caractériser le système de déduction effectif, par exemple avec des règles d'inférence (cf. paragraphe 5.1.1, page 99), pour donner une définition concrète des catégories de classes effectivement calculées ;
- étudier la correction de définitions générales comme celles de la version structurelle.

Sans anticiper sur la suite, soulignons que le seul intérêt de la classification structurelle réside dans l'incomplétude sémantique, que cette incomplétude soit causée par une indécidabilité ou par la trop grande complexité de la classification extensionnelle.

Les classes valides dans une classification non structurelle

Pour qu'une classe définie soit valide, il suffit que la classe soit structurellement sûre (dans sa version forte (6.4)) pour l'objet à classer. Par ailleurs, deux catégories d'inférence permettent de déduire qu'une classe, aussi bien primitive que définie, est *valide*, même en cas d'incomplétude extensionnelle de l'objet :

- d'abord, l'inférence de type, présentée comme élémentaire, qui permet de classer un objet lorsqu'il est valeur d'un attribut (cf. règle [*type*-infér]) ;
- ensuite, certaines constructions des logiques terminologiques comme la quantification existentielle qualifiée du langage $\mathcal{ALE}$, l'intersection de rôles de $\mathcal{ALR}$ [Donini *et al.*, 1994] ou les *sous-rôles* de [Nebel, 1990] : Nebel donne en particulier l'exemple de l'`Equipe-moderne` dans lequel un objet est rattaché à une classe pour la seule raison que c'est le seul qui puisse remplir un certain rôle (figure 5.2). Notons que l'inférence due à ces constructions porte sur la valuation d'un attribut, la classification de la valeur s'en déduisant par [*type*-infér].
- enfin, les éventuelles propriétés structurelles spécifiques du SC :
    - si le SC est exhaustif et qu'il n'y a qu'un puits envisageable, c'est une classe *valide*[4]. Plus généralement : les super-classes communes à tous les puits envisageables sont valides et si une classe valide n'a qu'une seule sous-classe directe envisageable, cette sous-classe est valide. Inversement, toujours en cas d'exhaustivité, si une classe (non puits) envisageable n'a aucune sous-classe puits envisageable, elle est en fait insatisfaisable[5].

---

4. Que le puits soit satisfaisable ou pas, il est valide : mais s'il est insatisfaisable, la description du monde est inconsistante.

Il y a cependant un point délicat : dans un SC permettant ce genre d'inférence, il y a vraisemblablement des objets *paradoxaux*, au sens où toute combinaison de valeurs d'attributs du domaine d'une classe n'a pas forcément d'interprétation (cf. paragraphe 3.4.2, page 52). C'est donc toujours une éventualité que l'objet considéré, surtout s'il est incomplet, soit en fait paradoxal. Le problème des objets paradoxaux vient du fait que l'utilisateur peut faire un mauvais choix et rendre paradoxal un objet, la classe de l'objet ne le protégeant pas, par définition, d'un tel paradoxe (alors que la classe de l'objet le protège d'une inconsistance non paradoxale).

5. Il semble que le modèle SHOOD [Nguyen et Rieu, 1989; Rieu *et al.*, 1992] prenne comme sémantique du *possible* celle de *sous-classe minimale envisageable*, ce qui exclut forcément toutes les classes envisageables localement exhaustives.

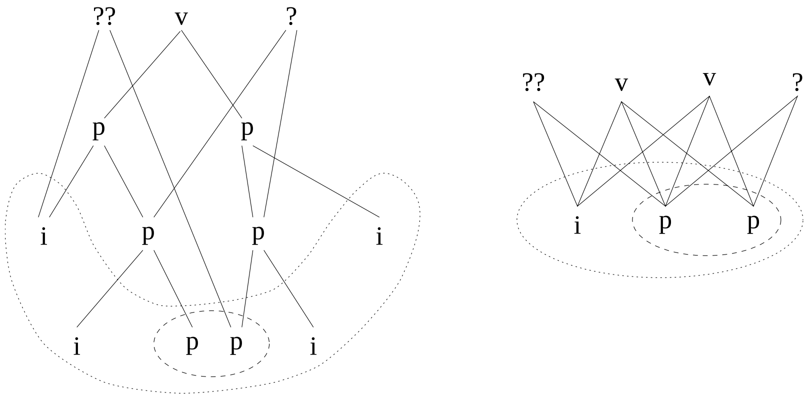

FIG. 6.4 – *À gauche en cas d'exhaustivité, à droite en cas d'univocité, les classes marquées ? sont valides, car super-classes communes du sous-ensemble de classes envisageables (ellipse en tireté) d'un ensemble de classes (pointillé) dont l'une au moins doit être satisfaite. On ne peut rien conclure, en revanche, pour les classes marquées ??.*

- si le SC est univoque, lorsqu'elle existe, la *borne inférieure* de tout ensemble de classes valides est une classe valide :

(6.6) $$(\alpha \wedge \beta)^{\mathcal{I}} = \alpha^{\mathcal{I}} \wedge \beta^{\mathcal{I}}$$

  Plus généralement, si cet ensemble de classes a un seul *minorant maximal* envisageable, ce minorant maximal est valide ; plus généralement encore, les super-classes communes à tous les minorants maximaux envisageables sont valides.

Les deux généralisations présentent la même structure : lorsqu'une hypothèse existentielle de validité porte sur un ensemble de classes (dans chaque modèle, l'objet est instance d'une de ces classes[6]), les super-classes communes à toutes celles qui sont envisageables sont valides (figure 6.4)[7]. Les contrapposées sont analogues : l'absence d'une classe envisageable dans l'ensemble considéré entraîne une inconsistance. Si une classe valide non puits n'a pas de sous-classe envisageable (exhaustivité), ou si deux classes valides n'ont pas de minorant maximal envisageable (univocité), la description du monde est inconsistante.

L'exhaustivité avait déjà montré son tropisme descendant dont le pouvoir inférentiel est quasiment aussi fort que celui des classes définies mais on remarque que l'univocité offre elle-aussi un tel tropisme, plus faible. Dans ces deux cas, la déduction ne repose plus sur une comparaison « locale » de deux termes, mais il est nécessaire d'énumérer tous les candidats, comme dans l'exemple de l'`Equipe-moderne`.

6. Ce qui ne signifie pas que l'une au moins est valide. Il faut éviter de commuter les deux quantifications. Il s'agit de $\forall \mathcal{M}, \exists \alpha \in A : \ \models_{\mathcal{M}} (\alpha\ o)$ et non pas de $\exists \alpha \in A, \forall \mathcal{M} : \ \models_{\mathcal{M}} (\alpha\ o)$ qui est équivalent à $\exists \alpha \in A : \mathcal{W} \models (\alpha\ o)$.

7. On remarquera que l'hypothèse existentielle duale — parmi un ensemble d'objets, l'un au moins est instance de la classe — ne semble permettre aucune autre inférence que celle que l'exemple de `Equipe-moderne` a illustrée, à peine généralisée en jouant sur la cardinalité du sous-rôle (cf. figure 5.2, page 107)

- l'hypothèse d'explication des attributs peut induire la même hypothèse existentielle : l'objet doit appartenir à l'une des classes qui introduisent l'attribut (cf. la règle [expliq-P]).

Dans ces divers cas, la déduction ne repose plus sur une comparaison « locale » de deux termes, mais il faut énumérer tous les candidats. En particulier, une classe peut être valide sans qu'il soit nécessaire qu'elle soit structurellement sûre : il suffit qu'elle ne soit pas structurellement impossible (il y aurait sinon une inconsistance).

En corollaire, si une classe structurellement possible mais pas sûre a de grandes chances d'être falsifiable (il suffit d'affecter à un attribut une valeur en dehors du domaine de l'attribut dans la classe), ce n'est pourtant pas certain : la classe pourrait être valide et toute tentative de falsification mènerait à une inconsistance.

Les classes sûres sont constituées de ces sous-ensembles de classes valides, effectivement calculables. Le détail de ces classes sûres est relativement secondaire : ce qui compte est qu'il en existe et qu'elles n'ont même pas besoin d'être structurellement sûres. Il y a toujours moyen de faire un peu de classification automatique, même avec des classes primitives. On a vu comment caractériser les classes valides et sûres par les opérateurs de conséquence et de dérivation : on voit alors bien pourquoi ces ensembles de classes n'ont pas de définition précise : tout dépend du système de déduction que l'on se donne, de la façon dont on a tranché dans le *compromis complétude-complexité*, etc.

Les classes satisfaisables

Dans le cadre d'une classification interactive, il est évidemment essentiel de pouvoir garantir à l'utilisateur que le rattachement d'une instance à une classe est *possible*, au sens où il mène avec certitude à une solution, et pas seulement qu'il est *possible* que ce rattachement mène à une solution.

Une classe *structurellement sûre* est-elle bien *satisfaisable* ? En toute généralité, vu l'aspect « structurel » des définitions, il est à craindre que non : il suffit d'introduire un élément non structurel comme l'incompatibilité ou les sous-rôles pour construire un contre-exemple.

Supposons que o ait deux classes α et β structurellement possibles (voire sûres) et que ces deux classes soient incompatibles ($\alpha \nabla_a \beta$), le tout sans que la description du monde soit inconsistante. En l'absence d'autre contrainte, ces deux classes sont à la fois satisfaisables et falsifiables : chacune est la classe de o dans certains modèles, mais pas dans d'autres à cause de l'incompatibilité. Mais rien n'empêche — c'est-à-dire que cela ne conduit pas forcément à une inconsistance — que o soit la valeur d'un attribut d'un objet dont le type est β :

$$(P\ o'\ o), (\gamma\ o') \in \mathcal{W} \text{ et } type(\gamma, P) = \beta$$

ce qui rendrait β valide (par [$type$-infér]) et α insatisfaisable. Bien sûr, la validité de β peut avoir une cause plus complexe, par exemple se déduire par les inférences spécifiques de l'exhaustivité ou de l'univocité, voire des sous-rôles. On voit bien pourquoi une classe structurellement possible ou sûre, ne peut pas être considérée comme satisfaisable avec certitude : rien n'empêche qu'une inférence quelconque ne provoque l'incompatibilité de l'objet avec la classe[8]. Bref, en toute généralité, les classes satisfaisables paraissent réellement inaccessibles si l'on ne prend pas en compte explicitement la relation d'incompatibilité. Ainsi il faudrait inclure dans la définition de *structurellement sûre* : une classe dont aucune classe incompatible n'est structurellement possible. Mais cette intervention d'un tiers fait sortir cette notion d'un cadre structurel strict au sens du paragraphe 5.2.2. Est-ce que cela suffirait à résoudre le cas des sous-rôles ? Sans doute pas.

On a le même genre de problème lorsque l'on rajoute des contraintes au SC comme le fait [Gensel, 1995] : comme calculer l'existence d'une solution (consistance globale) est aussi difficile que le calcul d'une solution, on se limite à une consistance locale (ex. consistance d'arc) qui fait que le possible traduit plus un espoir qu'une réalité !

Les sous-rôles, qui ne sont que des cas particuliers de contraintes, illustrent bien ces difficultés : l'exemple de l'`Equipe-moderne` de [Nebel, 1990] est une instance parfaite de l'exemple abstrait précédent. `Homme` et `Femme` sont tous deux des classes structurellement possibles (voire sûres) incompatibles, mais l'une est valide, l'autre insatisfaisable. Et le calcul de la validité de la classe est trop complexe pour que Nebel accepte d'incorporer ce genre d'inférence dans ses algorithmes.

Complétude sémantique

Pour avoir des chances d'obtenir l'interprétation idéale de ces notions structurelles, il faudrait donc les définir,

- en tenant compte de l'incompatibilité ;
- à partir d'une description du monde $\mathcal{W}$ fermée par $\models$, c'est-à-dire contenant toutes ses conséquences valides : $\mathcal{W} = \{d \mid \mathcal{W} \models d\}$.

Alors seulement peut-on espérer que, dans la description du monde résultante, toute classe structurellement possible ou sûre sans être incompatible est satisfaisable, et peut être considérée, arbitrairement, comme une solution.

Mais il faut bien voir ce que cela va impliquer :

- si les classes structurellement possibles sont satisfaisables, elles sont donc possibles puisque manifestement calculables, ce qui conduit à l'équivalence d'une part, des classes satisfaisables, possibles, envisageables et structurellement possibles, d'autre part, par complément, des classes insatisfaisables, impossibles et structurellement impossibles (figure 6.3) ; en corollaire, il n'y a pas de classe incertaine ; cela signifie donc que l'on a obtenu la complétude sémantique pour les prédicats Sat et $InSat$.
- pour que les classes structurellement sûres soient satisfaisables, il faut assurer la complétude sémantique, au moins pour les classes valides, mais sans doute aussi

8. On comprend mieux le refus par Nebel d'admettre que l'incompatibilité puisse porter sur des classes définies : si deux classes définies sont incompatibles, ce ne sont des CNS que modulo la consistance du système. C'est un peu désagréable, mais pas vraiment gênant.

sur les classes satisfaisables et insatisfaisables [9]. Or, si l'on a la complétude sur les classes (in)satisfaisables, la correction des classes structurellement possibles et sûres suit trivialement : il suffit d'inclure la satisfaisabilité dans leur définition. Plus sérieusement, les définitions structurelles n'ont de réel intérêt qu'en cas d'incomplétude : dès que la déduction est complète pour tous les prédicats Val, Sat et $InSat$, il devient en effet inutile de chercher à caractériser ce qui est calculé.

- vouloir faire des classes structurellement sûres des classes valides est encore plus prétentieux : cela signifie qu'un tropisme descendant force le rattachement à une classe dès lors que l'objet satisfait définitivement toutes les propriétés de la classe. Cela revient en fait à considérer que les classes sont définies.

Bien sûr, si la correction s'avère délicate, la complétude n'était même pas revendiquée jusqu'ici, entraînant la coexistence de classes caractérisées au niveau sémantique (valide, satisfaisable, insatisfaisable) et au niveau syntaxique (sûre, possible, impossible, etc.). Or, en un retournement inattendu, la correction de l'interprétation souhaitée des notions structurelles les plus syntaxiques semble nécessiter finalement la complétude sémantique : on ne garderait donc plus que les classes valides, satisfaisables et insatisfaisables (identiques respectivement aux classes structurellement sûres, possibles et impossibles). Rien n'empêche bien sûr, pour un SC particulier, d'essayer de relever ce défi de la complétude, mais le cas général semble désespéré comme le montre toute la littérature consacrée à la décidabilité et à la complexité de la subsomption [Nebel, 1990; Patel-Schneider, 1989a] ou de la vérification d'instance [Donini *et al.*, 1994] (cf. paragraphe 5.2.3, page 108). Mais, de toute façon, cette complétude enlèverait tout intérêt pratique aux définitions structurelles.

Version dynamique

Finalement, les classes *structurellement possibles* et *impossibles* représentent la meilleure (par sa simplicité) approximation des classes *envisageables* et *impossibles*, respectivement par excès ou par défaut. Quant au caractère priviligié des classes *structurellement sûres*, il résulte uniquement de la maîtrise de l'incomplétude extensionnelle : vis-à-vis de la classe, l'objet à classer est parfaitement connu. Aucun retournement n'est à craindre d'un attribut dont le dévoilement viendrait contredire l'attachement de l'objet à une classe. Mais le retournement peut très bien venir d'ailleurs !

L'erreur est de chercher une définition statique : les définitions statiques sont abstraites et non structurelles, et les définitions structurelles ne peuvent être utilisées que dynamiquement.

- pour les classes *structurellement impossibles* ou *sûres*, il s'agit d'une propriété monotone, qui peut donc servir à faire des inférences de façon « normale » ;

- pour les classes *structurellement possibles*, il s'agit d'une propriété non monotone qui ne peut servir à faire des inférences que sous des formes bien particulières,

9. Considérons deux uniques puits structurellement possibles, dont un insatisfaisable : l'autre est donc valide (On suppose ici que la seule façon de prouver la validité de l'un passe par l'insatisfaisabilité de l'autre).

comme la forme existentielle présentée plus haut : si l'une au moins des classes d'un ensemble doit être satisfaite, toutes les super-classes communes des classes possibles de cet ensemble sont valides.

On verra plus loin comment concrétiser ces définitions avec des règles d'inférence.

6.3 La vérification d'instance

L'objectif de ce paragraphe est de concrétiser et de préciser les définitions approximatives de la version structurelle de la classification incertaine. Il s'agit essentiellement de concrétiser la notion de vérification d'une classe par un objet.

Mais, du fait de l'incomplétude des objets, la question de savoir s'ils vérifient ou pas l'intension d'une classe se réduit d'abord souvent à savoir s'ils sont compatibles ou non avec cette classe, dans un sens analogue à celui de l'incompatibilité entre classes (∇). Ensuite, lorsque les objets ne sont plus si incomplets que ça, il s'agit de savoir s'ils sont réellement fermés relativement à cette classe, dans le sens où aucune nouvelle information ne peut contredire la vérification de la classe par l'objet, sauf à introduire une inconsistance. ce que nous exprimerons avec une notion de fermeture.

6.3.1 Notions de compatibilité

Bien sûr, pour que la description du monde ne soit pas inconsistante, les valeurs connues des attributs doivent être consistantes avec les domaines :

$$\forall \alpha \in \underline{Isa}_{\mathcal{W}}(o), \forall P \in Attr_{\mathcal{W}}(o) \cap \mathcal{P}_{\alpha} : P_{\mathcal{W}}(o) \in Dom_P(\alpha) \tag{6.7}$$

Ce paragraphe va préciser la signification de cette dernière appartenance " c'est encore une formule naïve, comme dans (6.2). D'une part, la valeur peut être étendue (en cardinalité) ou précisée (en type) pour appartenir au domaine, d'autre part, elle permet d'inférer l'appartenance de chacune des valeurs à l'extension d'une classe par la règle [*type*-infér].

La notion de compatibilité d'un objet et d'une classe recouvre intuitivement le fait que l'objet peut être instance de la classe vu leurs descriptions respectives. On a vu les difficultés sémantiques qui se cachent derrière ce naïf « peut être ». Aussi il s'agit ici d'une pure notion syntaxique que nous cherchons à définir, une compatibilité par excès, acceptant de répondre « oui » pour des classes insatisfaisables, mais en refusant de répondre « non » pour des classes satisfaisables.

Retour sur la compatibilité intensionnelle

Il faut d'abord revenir sur l'expression de la compatibilité (ou incompatibilité) intensionnelle entre classe : cette notion n'a en effet été définie que relativement à un domaine des attributs abstrait, *Dom* (cf. paragraphe 2.1.4, page 21). Or le fait d'exprimer les domaines par des classes (*type*) conduit à des circularités.

Un ensemble de classes $X \subset \mathcal{X}$ — dont on peut supposer qu'il s'agit d'une antichaîne de $(\mathcal{X}, \preceq)$, quitte à en prendre les minimaux — sera dit *intensionnellement compatible* ssi les conditions suivantes sont réalisées :

1.1. il n'est pas dominé dans $(\mathcal{X}^*, \preceq_b)$ [10] par un élément de ∇_a considéré comme une partie quelconque de $\mathcal{X}$;

1.2. pour tout $P \in \bigcup_{\beta \in X} \mathcal{P}_\beta$, l'ensemble de classes X est compatible pour l'attribut P.

Un ensemble de classes X est *compatible pour un attribut* P ssi

1.2.1. les cardinalités sont compatibles :

$$\bigcap_{\beta \in Def(P) \cap X} card_P(\beta) \neq \emptyset$$

1.2.2. les types sont compatibles, c'est-à-dire que l'ensemble de classes

$$\{type_P(\beta) \mid \beta \in Def(P) \cap X\}$$

est intensionnellement compatible[11].

Moyennant cette définition dont la circularité sera supposée bien se résoudre[12], on peut passer à la compatibilité d'un objet avec une classe.

L'*univocité* du SC simplifie grandement la définition de la compatibilité d'un ensemble de classes : un ensemble de classes X est intensionnellement compatible ssi il a une sous-classe commune intensionnellement cohérente, c'est-à-dire ssi

$$\exists \alpha \in \mathcal{X} : \begin{cases} Int(\alpha) \neq Int(\bot) \\ \forall \beta \in X : \alpha \preceq \beta \end{cases} \tag{6.8}$$

Compatibilité d'un objet et d'une classe

Pour ce qui concerne la classe, deux notions différentes doivent être prises en compte, l'*intension* ou la *description* :

2.1. l'objet o est *compatible avec l'intension de la classe* α ssi pour tout $P \in Attr_W(o) \cap \mathcal{P}_\alpha$, $P_\mathcal{W}(o)$ est compatible avec α.

10. Il faudrait en fait considérer l'ordre sur les antichaînes induit par $\trianglelefteq_a$ ou mieux par $\trianglelefteq_c$, et non par $\prec$. L'efficacité du calcul pour la comparaison des antichaînes, qui semble difficile, est à surveiller de près. Une première solution : marquer X et ses super-classes, puis rechercher dans ∇_a un élément dont tous les éléments sont marqués.

11. Cette formule n'est correcte que si $type_P$ est une fonction totale sur $Def(P)$. Comme l'a montré la discussion du paragraphe 4.1.1, ce n'est pas le cas et il faut plutôt formuler cet ensemble ainsi :

$$\{type_P(\beta) \mid \beta \in \mathrm{dom}(type_P) \cap Super^*(X)\}$$

où $Super^*(X)$ dénote l'ensemble des classes et super-classes de X. Les deux styles sont en revanche possibles pour $card_P$.

12. Dès qu'une récursion infinie s'amorce, c'est-à-dire dès que l'on cherche à calculer quelque chose que l'on est déjà en train de calculer, il suffit de retourner `true` : la complétude n'est pas recherchée ici.

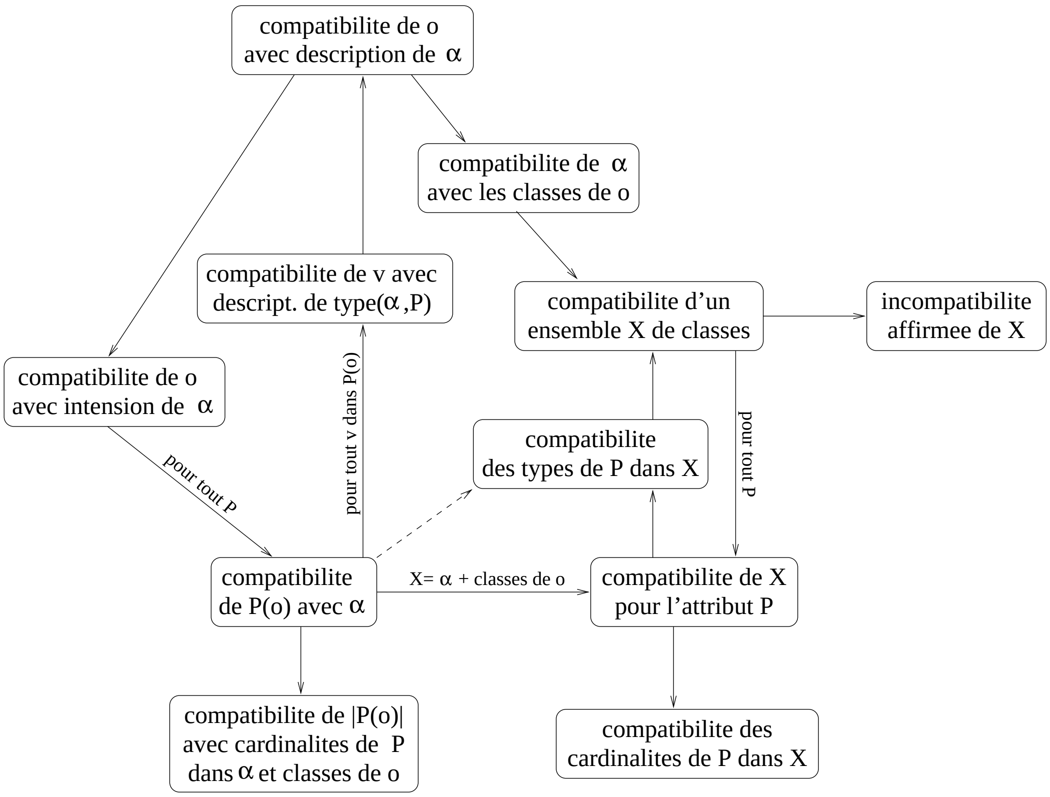

FIG. 6.5 – *Interaction des diverses compatibilités.*

2.2. l'objet o est *compatible avec la description de la classe* α ssi les deux conditions suivantes sont vérifiées :

2.2.1. o est compatible avec l'intension de α ;

2.2.2. l'ensemble de classes $\{\alpha\} \cup Isa_{\mathcal{W}}(o)$ est intensionnellement compatible ;

Ces deux notions se ramènent donc à la définition de la compatibilité d'un attribut d'un objet avec une classe.

Soit $P \in Attr_{\mathcal{W}}(o) \cap \mathcal{P}_\alpha$: $P_W(o)$ est *compatible avec* α ssi les trois conditions suivantes sont vérifiées :

3.1. compatibilité de type et cardinalité : l'ensemble de classes $\{\alpha\} \cup Isa_{\mathcal{W}}(o)$ est compatible pour l'attribut P ;

3.2. compatibilité des cardinalités avec celle de o[13] :

$$|P_{\mathcal{W}}(o)| \leq \max\Big(\bigcap_{\beta \in \{\alpha\} \cup \underline{Isa}_{\mathcal{W}}(o) \cap Def(P)} card(\beta, P)\Big)$$

13. La compatibilité des cardinalités mérite un petit commentaire. La version suivante est incorrecte si les cardinalités s'expriment autrement que par des intervalles :

$$\forall \beta \in \{\alpha\} \cup \underline{Isa}_{\mathcal{W}}(o) \cap Def(P) : |P_{\mathcal{W}}(o)| \leq card(\beta, P)$$

Il suffit d'imaginer deux classes β pour lesquelles $type_P$ serait l'ensemble des entiers respectivement pairs et impairs, mais le problème se pose déjà avec des unions d'intervalles.

3.3. compatibilité de valeur : pour tout $v \in P_{\mathcal{W}}(o)$, v est compatible avec (la description de ?)[14] $type(\alpha, P)$.

La récursivité de la définition de la compatibilité avec une classe conduit à l'alternative suivante : soit utiliser récursivement la seule compatibilité avec la description, soit que chaque compatibilité s'utilise elle-même récursivement. Si la compatibilité avec la description n'est pas concernée par ce flou, la compatibilité avec l'intension perd un peu plus de sa crédibilité.

D'un point de vue efficacité, on remarque que la compatibilité des cardinalités de P pour $\{\alpha\} \cup \underline{Isa}_{\mathcal{W}}(o)$ est calculée deux fois, une fois directement (3.2), une autre fois indirectement (3.1) : on peut donc remplacer (3.1) par une simple compatibilité des types en court-circuitant la compatibilité des classes. Par ailleurs, dans le cas de la compatibilité avec la description, cette compatibilité des classes de (3.1) aura déjà été vérifiée en (2.2.2).

On dira enfin qu'un objet est *consistant* s'il est compatible avec la description de chacune de ses classes connues : c'est une condition nécessaire pour que la description du monde soit consistante, mais elle est loin d'être suffisante. Un objet *paradoxal* est un objet consistant dont l'existence rend toute description du monde inconsistante[15].

Version récursive

Pour la compatibilité avec l'intension d'une classe, une définition récursive est certainement plus opératoire : o est compatible avec l'intension de α ssi o est compatible avec l'intension de toute super-classe $\beta \in Super(\alpha)$ et si o est compatible avec l'intension propre de α, $Int^*(\alpha)$: pour tout $P \in Attr_W(o) \cap \mathcal{P}^*_\alpha$, $P_{\mathcal{W}}(o)$ est compatible avec α. Pour l'efficacité de l'algorithme, on prendra garde de n'oublier le cas où o est connu comme une instance de α : la compatibilité entre un objet et ses classes va de soi, sous peine d'inconsistance.

Pour la compatibilité avec la description, il n'est pas suffisant de remplacer partout « intension » par « description » : il faut encore rajouter la contrainte de non incompatibilité sur l'ensemble de classes.

6.3.2 Degrés de fermeture

La complétude extensionnelle d'un objet est une notion beaucoup moins claire qu'il n'y paraît de prime abord. On introduit une notion de *fermeture* pour en rendre compte.

La fermeture d'un objet peut se qualifier de différentes manières, soit dans l'absolu, soit relativement à une classe et/ou à un attribut. Elle peut enfin être *actuelle*, c'est-à-dire explicite dans la description du monde, ou *déduite* de cette description du monde. Il faut donc distinguer :

14. C'est ce genre d'interrogation qui revèle la complexité de formulation du problème : qu'il y ait deux formulations différentes, passe encore ! Mais que l'incertitude demeure sur la formulation qu'il faut utiliser récursivement, voilà qui passe les bornes !

15. Il s'agit ici des versions intensionnelles des définitions extensionnelles correspondantes du paragraphe 4.2.2.

- une *fermeture actuelle* : toute nouvelle description est soit déjà incluse dans la description du monde, soit rend cette dernière inconsistante ; soit $d \in \mathcal{W}$, soit $\{d\} \cup \mathcal{W}$ est inconsistant ; c'est la version de la fermeture par Nebel.
- une *fermeture valide* : toute nouvelle description est soit une conséquence valide de la description du monde, soit rend cette dernière inconsistante ; soit $\mathcal{W} \models d$, soit $\{d\} \cup \mathcal{W}$ est inconsistant : c'est l'*hypothèse du monde clos* (ou *closed world assumption*) ; c'est la définition de [Borgida et Patel-Schneider, 1994], mais elle est moins directement exploitable que la précédente, comme le note [Borgida, 1992b] qui ne donne des règles que pour la fermeture actuelle.
- une *fermeture déduite* : toute nouvelle information soit se déduit de la description du monde, soit rend cette dernière inconsistante ; c'est une version intermédiaire, intensionnelle qui s'obtient comme toujours en remplaçant l'opérateur de conséquence par l'opérateur de dérivation : soit $\mathcal{W} \vdash d$, soit $\{d\} \cup \mathcal{W}$ est inconsistant.

Les fermetures actuelle et valide (resp. déduite) sont liées par le fait que la fermeture par $\models$ (resp. $\vdash$) d'une fermeture valide (resp. déduite) est une fermeture actuelle. Enfin, les fermetures valide et déduite se confondent dès lors que la déduction est complète. Ce sont donc des notions relativement redondantes.

NOTE *6.3* *Fermé, fermer ou fermez ?*

On utilise le mot « fermeture » dans deux sens : celui de l'opération (fermeture par $\models$ ou $\vdash$) ou celui de l'état d'une description fermée. Ce n'est pas très correct.

À ces deux sens, il faut en ajouter un troisième, celui qui consiste à décréter la fermeture d'un attribut, d'un objet ou d'une description du monde en introduisant un opérateur de fermeture dans la description du monde, ce qui revient à éliminer tous les modèles qui empêcherait l'état de fermeture souhaité.

Entre les fermetures valide et déduite, il reste un peu de place, par exemple pour exprimer que l'on ne veut pas rajouter d'objets supplémentaires. Considérons l'exemple de l'`Equipe-moderne` de Nebel (cf. figure 5.2, page 107), et supposons qu'un certain nombre de `membres`, tous des `Hommes`, sont connus. Il est possible d'affirmer que l'attribut `membre` est fermé, en voulant dire par là que l'on ne considère que les modèles qui minimisent le nombre de valeurs, en rajoutant l'unique `Femme` nécessaire pour tenir le rôle de `chef`. Mais ce n'est pas exprimable dans les termes de la description du monde, puisque de nombreuses `Femmes` différentes peuvent tenir ce rôle.

Fermeture d'un attribut

En posant dans ce qui suit $|P_{\mathcal{W}}(o)| = n$, on pourra d'abord dire de l'attribut P de l'objet o — en attribuant abusivement la propriété à $P_{\mathcal{W}}(o)$ — que :

- il est *fermé* ssi soit il a été déclaré fermé par une assertion $closed(o, P)$, soit

$$n = \max\Big(\bigcap_{\beta \in Isa_{\mathcal{W}}(o) \cap Def(P)} card(\beta, P)\Big).$$

Il s'agit d'une fermeture actuelle : toute nouvelle valeur d'attribut (non encore affirmée dans $\mathcal{W}$) rend la description du monde inconsistante. Dans ce cas, $\forall \mathcal{M}$: $P_{\mathcal{W}}(o)^{\mathcal{I}} = P_{\mathcal{M}}(o)$.

fermeture	$P(o)$	$Isa(o)$	o
actuelle	$(P\ o\ v) \in \mathcal{W}$	$(\alpha\ o) \in \mathcal{W}$	$\forall P \in Attr_{\mathcal{W}}(o)$
valide	$\mathcal{W} \models (P\ o\ v)$	$\mathcal{W} \models (\alpha\ o)$	$\forall P \in Attr^{\mathcal{W}}(o)$

FIG. 6.6 – *Fermetures actuelle et valide, de P, Isa et o.*

- La fermeture valide correspondante ne se définit que de façon abstraite : pour toute valeur o', soit $\mathcal{W} \models (P\ o\ o')$, soit $\mathcal{W} \cup \{(P\ o\ o')\}$ est inconsistant. Dans ce cas, $\forall \mathcal{M} : P^{\mathcal{W}}(o)^{\mathcal{I}} = P_{\mathcal{M}}(o)$

NOTE *6.4*
De nombreux SC comme SHIRKA font une hypothèse du monde clos sur les attributs : un attribut est soit totalement inconnu — ce qui impose d'introduire une valeur spéciale `unknown`, pour distinguer $P_{\mathcal{W}}(o) =$ `unknown` et $P_{\mathcal{W}}(o) = \emptyset$ —, soit actuellement fermé : cela simplifie toutes les notions de compatibilité, fermeture ou vérification. C'est en particulier le cas de tous les systèmes qui ne disposent pas d'attributs multi-valués.

Fermeture d'un objet

Ces notions se déclinent alors pour les objets, dans différentes combinaisons : d'une part, elles s'appliquent à $Isa(o)$, aux attributs de $Attr_{\mathcal{W}}(o)$, de $Attr^{\mathcal{W}}(o)$ ou de $\mathcal{P}_\alpha$, d'autre part, elles sont actuelles ou valides.

- un objet o est *classé* s'il y a fermeture actuelle (resp. valide) de $\mathcal{W}$ vis-à-vis de toute description $(\alpha\ o)$:

$$(6.9) \qquad \forall \alpha \in \mathcal{X} : \begin{cases} (\alpha\ o) \in \mathcal{W} \text{ (resp. } \mathcal{W} \models (\alpha\ o)) \\ \text{ou bien } \mathcal{W} \cup \{(\alpha\ o)\} \text{ est inconsistant} \end{cases}$$

  Dans un SC univoque, une condition suffisante, très forte, pour qu'un objet soit classé est qu'il soit instance d'un puits de la hiérarchie.

- un objet o est *fermé relativement à α* ssi $\forall P \in \mathcal{P}_\alpha$, $P_{\mathcal{W}}(o)$ est fermé. Cela suppose bien sûr que tous les attributs de la classe soient connus de l'objet : $\mathcal{P}_\alpha \subseteq Attr_{\mathcal{W}}(o)$. Les deux variantes des fermetures actuelle ou valide sont possibles.

- un objet o est *fermé* s'il y a fermeture actuelle (resp. valide) de $\mathcal{W}$ vis-à-vis de tous les attributs $P \in Attr_{\mathcal{W}}(o)$ (resp. $Attr^{\mathcal{W}}(o)$)[16]. Il n'y a pas de relation simple entre les fermetures actuelle et valide d'un objet : la fermeture actuelle n'implique pas forcément la fermeture valide. Cependant, un objet actuellement fermé et classé est validement fermé et classé.

16. Utiliser $Attr^{\mathcal{W}}(o)$ pour la fermeture actuelle n'aurait aucun sens puisque $(P\ o\ v) \in \mathcal{W} \Longrightarrow P \in Attr_{\mathcal{W}}(o)$. Pour la fermeture valide en revanche, les deux cas diffèrent et sont envisageables.

Dans le dernier cas, il s'agit d'une fermeture provisoire puisque le rattachement de l'objet à de nouvelles classes peut apporter de nouveaux attributs.

Bien sûr, la fermeture considérée jusqu'ici ne tient pas compte du fait que les valeurs d'un attribut puissent ne pas être elles-mêmes fermées. La notion de *fermeture forte* permet de combler cette lacune :

- $P_{\mathcal{W}}(o)$ est *fortement fermé* ssi $P_{\mathcal{W}}(o)$ est *fermé* et si $\forall v \in P_{\mathcal{W}}(o)$, v est *fortement fermé relativement à* $type(\alpha, P)$.
- o est *fortement fermé relativement à* α ssi $\forall P \in \mathcal{P}_\alpha$, $P_{\mathcal{W}}(o)$ est fortement fermé. Cela suppose bien sûr que o soit fermé relativement à α.
- un objet o est alors *fortement fermé* si tous ses attributs sont fortement fermés.

Un objet actuellement (resp. validement) fermé est actuellement (resp. validement) fermé relativement à toutes ses classes connues ($Isa_{\mathcal{W}}(o)$) (resp. valides ($Isa^{\mathcal{W}}(o)$)). Le fait qu'un objet soit (fortement) fermé n'implique pas qu'il le soit par rapport à d'autres classes que les siennes ($Isa_{\mathcal{W}}(o)$). Enfin un objet est actuellement (resp. validement) *définitivement fermé* s'il est actuellement (resp. validement) fermé et que $Attr_{\mathcal{W}}(o) = Attr_{\forall\mathcal{M}}(o)$ (resp. $Attr^{\mathcal{W}}(o) = Attr_{\forall\mathcal{M}}(o)$).

En gros, aucune information ne peut être ajoutée à un objet fortement fermé autrement qu'en rattachant à une nouvelle classe, soit l'objet lui-même, soit l'un des objets avec lesquels il est en relation, directe ou indirecte. En revanche, la valeur d'un attribut d'un objet simplement fermé peut être modifiée, non pas en ajoutant une valeur élémentaire, mais en en modifiant une qui n'est pas fermée.

Une fermeture encore plus forte imposerait aux valeurs d'être en plus classées ! Devant la combinatoire qui se présente, la difficulté est de choisir les notions pertinentes.

Toutes ces notions de fermeture sont bien compliquées et il est nécessaire de revenir à leur emploi particulier pour déterminer celles qui sont utiles. La fermeture va essentiellement servir à dire si un objet peut encore respecter, ou ne plus respecter une restriction de valeur ou de cardinalité. Il faut donc savoir si la cardinal d'une valeur d'attribut peut augmenter pour atteindre une certaine valeur, ou au contraire ne pas augmenter pour ne pas dépasser cette valeur et, si chacune des valeurs peut être spécialisée pour atteindre une certaine classe.

Fermeture d'une description du monde

Ces notions de fermeture s'étendent des objets aux descriptions du monde, par une quantification universelle sur les objets connus ($o \in \mathcal{O}$). La question est bien sûr de savoir s'il faut tenir compte du classement des objets ou pas ?

Proposons la définition suivante : $\mathcal{W}$ est *actuellement* (resp. *validement*) *fermé* ssi tous les objets de $\mathcal{O}$ sont *actuellement* (resp. *validement*) fermés et classés. En particulier, tous les objets de $\mathcal{O}$ sont *actuellement* (resp. *validement*) fortement fermés[17] : tous les objets étant fermés, ils sont forcément fortement fermés.

17. Il faut remarquer que dans la définition précédente, la restriction aux objets déjà connus ne porte que sur les tuples de typage et les objets porteurs d'attribut, pas sur les valeurs de ces attributs : il reste donc possible d'introduire de nouveaux objets, mais uniquement comme valeur.

La fermeture actuelle implique la fermeture par $\models$, et la fermeture par $\models$ d'une fermeture valide est actuellement fermée.

Or, en règle générale, même pour une description du monde fermée par $\models$, une nouvelle description est vraie dans certains modèles, fausse dans d'autres. Une description du monde *actuellement* ou *validement fermée* est en fait assimilable à un modèle minimal : sa fermeture par $\models$ est isomorphe à son unique modèle minimal, donc canonique.

Notons enfin que dans la définition de ces notions, déjà si fortes, il était nécessaire de limiter la fermeture aux objets déjà affirmés dans $\mathcal{O}$: *a priori*, les SC ne permettent pas de contraindre l'ensemble des objets considérés. La possibilité de rajouter un nouvel objet interdirait toute fermeture. En effet, s'il était possible de fermer $\mathcal{O}$, n'importe quoi pourrait subsumer n'importe quoi d'autre, sur la base de la règle [$\trianglelefteq\forall$].

Expression de la fermeture

La fermeture, qui peut s'appliquer à une description du monde, un objet ou à un attribut d'objet, peut être introduite dans le langage au moyen de quatre prédicats d'arité respective 0, 1 ou 2, pour lesquels nous utiliserons uniformément le nom *closed* : *closed* pour la fermeture de la description du monde, $closed(o)$ pour la fermeture d'un objet, $closed(o, \alpha)$ pour la fermeture d'un objet vis-à-vis d'une classe, et $closed(o, P)$ pour la fermeture d'un attribut d'un objet.

Dans tous les cas, l'expression de la fermeture est simple : restriction de cardinalité ou simple affirmation, mais ses effets ne sont pas forcément très prévisibles car la déclaration d'une fermeture peut entraîner une inconsistance ou avoir des effets très peu intuitifs, surtout en cas de fermeture valide.

Règles de fermeture

Les règles suivantes expriment la fermeture actuelle :

[*closed*-attribut] $$\frac{\{\vdash (P\ o\ o_i)\}^{i=1,n} \qquad \vdash (\alpha\ o)}{\vdash closed(o, P)} \qquad \begin{cases} o_i \neq o_j, i \neq j \\ n = \max(card_P(\alpha)) \end{cases}$$

[*closed*-objet] $$\frac{\vdash closed(o) \qquad \vdash (\alpha\ o)}{\vdash closed(o, \alpha)}$$

[*closed*-objet-classe1] $$\frac{\{\vdash closed(o, P)\}_{P \in \mathcal{P}_\alpha}}{\vdash closed(o, \alpha)}$$

[*closed*-objet-classe2] $$\frac{\vdash closed(o, \alpha)}{\vdash closed(o, P)} \qquad P \in \mathcal{P}_\alpha$$

[*closed*-monde] $$\frac{\vdash closed}{\vdash closed(o)}$$

[*InSat/closed*] $$\frac{\{\vdash (P\ o\ o_i)\}^{i=1,n} \qquad \vdash closed(o, P)}{\vdash InSat(\alpha, o)} \qquad n \notin (card_P(\alpha))$$

Ces règles sont très incomplètes : en particulier, elles n'expriment pas le fait qu'un objet est classé et la règle [*closed*-objet] n'a pas de symétrique. Dans [*InSat/closed*], suivant la convention adoptée au début du chapitre 5, n représente bien $|P_{\mathcal{W}}(o)|$ (ou plus exactement toutes les valeurs déduites) et non pas un sous-ensemble. Pour la fermeture actuelle, il faudrait en fait que les $(P\ o\ o_i)$ soient à gauche du $\vdash$. En fait, l'affirmation de $closed(o, P)$ (mais pas la déduction par [*closed*-attribut]) n'empêche même pas de rajouter de nouvelles valeurs à $P(o)$, (par affirmation ou déduction). Rajouter le cardinal en argument du prédicat, avec $closed(o, P, n)$, ne changerait rien puisque suivant l'ordre d'application des règles, on obtiendrait une fermeture actuelle ou déduite. En fait, il faudrait pouvoir exprimer que $closed(o, P)$ est affirmé dans la fermeture de $\mathcal{W}$ par $\vdash$ et qu'il lui est associé le cardinal courant de $P(o)$.

Dans les deux cas, [Borgida, 1992b] propose une solution, sans réussir à convaincre vraiment. La difficulté vient de l'hypothèse du monde clos, qui conduit Borgida a une négation par l'échec dans le plus pur style PROLOG.

L'usage de $\mathcal{P}_\alpha$ dans les règles [*closed*-objet-classe1] et [*closed*-objet-classe2] mérite aussi un commentaire : $\mathcal{P}_\alpha$ ne peut pas être lui-même le résultat d'inférence car cela rendrait la première règle non monotone. Il faut donc supposer que $\mathcal{P}_\alpha$ se calcule statiquement, à partir de l'introduction des classes, de $\prec$ et $\lhd_a$, mais pas de $\unlhd$. Ce n'est pas réellement limitatif puisque $\unlhd$ se déduit lui-même certainement de $\mathcal{P}_\alpha$, comme le montreront les règles concernant les classes définies, [$\unlhd$-def-abs] et [$\unlhd$-def-rel].

6.3.3 Degrés de vérification

L'étape suivante consiste à mesurer la façon dont un objet vérifie l'*intension* ou la *description* d'une classe (cf. paragraphe 2.1.5, page 22). Si la compatibilité repose sur l'absence d'information négative, la vérification exige la présence de toutes les informations positives.

Nous avons déjà vu deux notions de vérification de l'intension d'une classe : au paragraphe 4.5.1, $\sqsubseteq_{\mathcal{M}}$ représentait la vérification de l'intension d'une classe par un élément du domaine ; au paragraphe 3.4.2, $\sqsubseteq$ représentait une notion intuitive de vérification de l'intension d'une classe par un objet. C'est d'une version rigoureuse de la seconde dont nous avons besoin, mais elle est beaucoup moins simple à définir que la première.

Vérification définitive

On cherche une *vérification définitive* au sens où, si $o \sqsubseteq \alpha$ dans une description du monde W_0, cela restera vrai dans toutes les descriptions $\mathcal{W}_i$ issues de $\mathcal{W}_0$. Comme pour les notions de compatibilité ou de fermeture, la vérification se décline pour les objets et pour les attributs d'objet. On peut de plus distinguer une *vérification effective*, due à l'état actuel de l'objet, d'une *vérification nécessaire* due aux classes de l'objet.

D'abord, on peut dire de l'attribut $P \in Attr_{\mathcal{W}}(o) \cap \mathcal{P}_\alpha$ de o que :

- il *vérifie* la classe α *par héritage* ssi

$$\Big(\bigcap_{\beta \in Isa_{\mathcal{W}}(o) \cap Def(P)} Dom_P(\beta)\Big) \subseteq Dom_P(\alpha) \tag{6.10}$$

qui se décompose en deux équations analogues sur $card_P$ et $type_P$ (dont la seconde n'est pas si simple que cela à formuler).

- il *vérifie effectivement* la classe α ssi, il est *fermé* et $P(o) \in Dom(\alpha, P)$ (pris au sens strict : $\forall v \in P_{\mathcal{W}}(o) : type(\alpha, P) \in Isa_{\mathcal{W}}(v)$).
- il vérifie définitivement la classe α, s'il la vérifie effectivement ou par héritage. Soit la valeur est complètement connue, soit elle *doit* vérifier la classe compte-tenu des classes déjà connues.

Les notions équivalentes, globales pour un objet o, suivent naturellement, qui se dédoublent en vérification de l'intension ou de la description :

- *o vérifie l'intension de la classe α par héritage* (resp. *effectivement*, *définitivement*) ssi $\forall P \in \mathcal{P}_\alpha, P_{\mathcal{W}}(o)$ vérifie α *par héritage* (resp. *effectivement*, *définitivement*).
- *o vérifie la description de la classe α par héritage* (resp. *effectivement*, *définitivement*) ssi o est instance des super-classes de α ($Super(\alpha) \subseteq Isa_{\mathcal{W}}(o)$) et *$o$ vérifie l'intension de la classe α par héritage* (resp. *effectivement*, *définitivement*).

La vérification par héritage correspond à une relation de subsomption intensionnelle entre la classe de o (en cas de mono-instanciation) et α. La vérification définitive de l'intension (resp. la description) d'une classe α constitue une condition suffisante pour reconnaître un objet comme instance de α dans le cas où c'est une classe définie absolue (resp. relative).

NOTE *6.5* *Vérification par héritage et effective*

La séparation complète est difficile : on peut en effet tenir compte du cardinal courant de $P_{\mathcal{W}}(o)$. L'équation « analogue » pour les cardinalités ne sera pas si analogue que ça :

$$|P_{\mathcal{W}}(o)| > \max(card_P(\alpha) - \bigcap_{\beta \in Isa_W(o) \cap Def(P)} card_P(\beta)) \tag{6.11}$$

tient au mieux compte de l'état actuel de l'objet. Il n'y a rien d'équivalent pour les types, puisque les types des valeurs connues n'ont aucune influence sur ceux des valeurs encore inconnues.

Vérification apparente

On peut se demander si, dans la dernière définition, il ne serait pas suffisant que o soit compatible avec la description de α (comme la vérification avec l'intension implique manifestement la compatibilité avec l'intension, seule la compatibilité des classes serait réellement exigée dans cette dernière condition). Si l'on vise une définition qui pourrait servir de condition suffisante pour une définition de classe relative, ce serait manifestement insuffisant.

Si la *vérification définitive de la description* nous intéresse, c'est en ce qu'elle constitue une condition suffisante pour faire d'une classe définie une classe valide. Mais, d'un

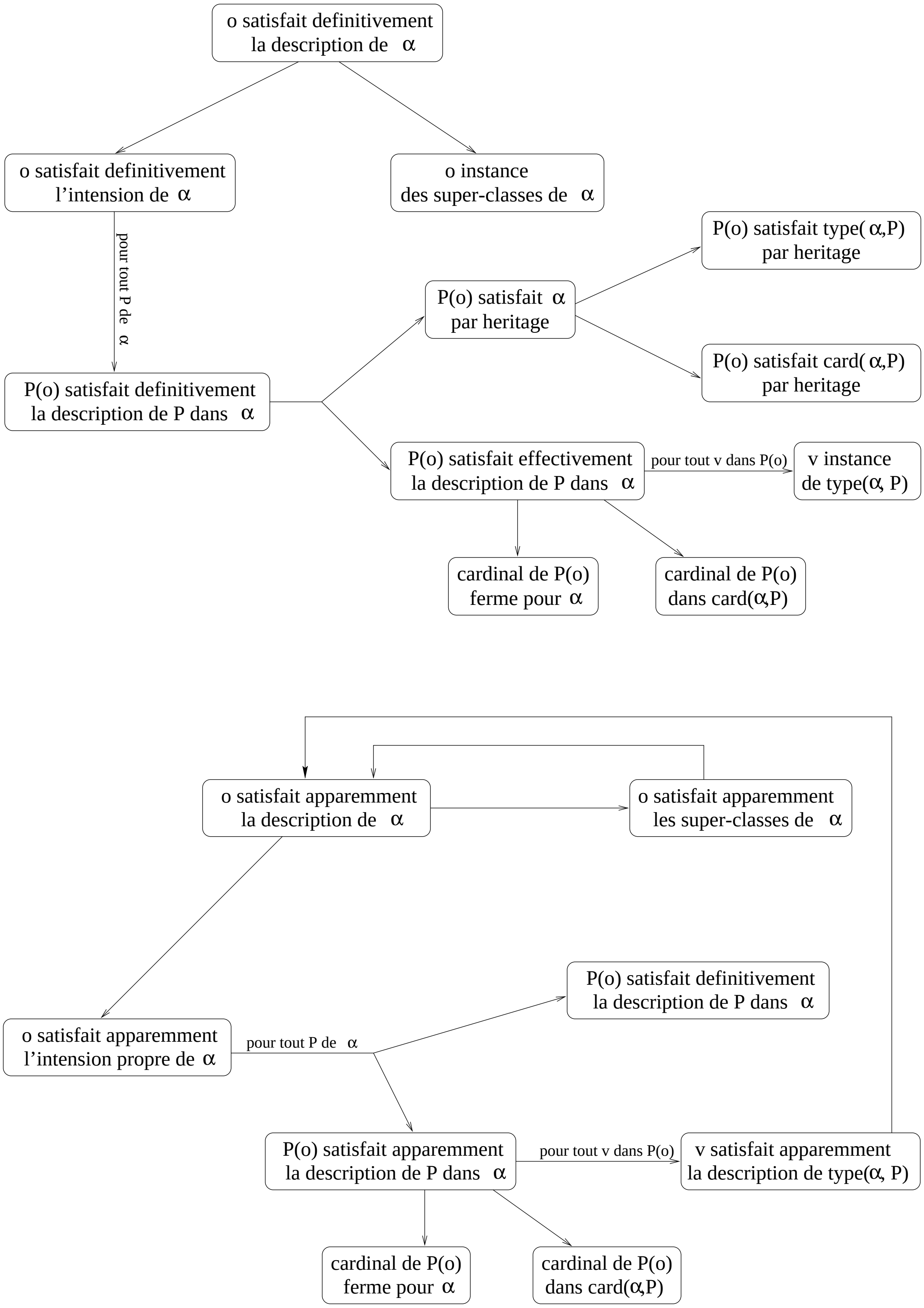

Fig. 6.7 – *Vérifications définitive et apparente*

autre côté, on recherche aussi une propriété structurelle voisine qui permettrait d'affirmer que la classe est « un peu plus possible » qu'une classe structurellement possible, ce qui en ferait un meilleur candidat pour une classification interactive. Ce sera la *vérification apparente* :

- *o vérifie apparemment la description* de α, noté $o \sqsubseteq_a \alpha$, ssi o vérifie apparemment la decsription de toutes les super-classes de α et $\forall P \in \mathcal{P}_\alpha$, $P_\mathcal{W}(o)$ vérifie apparemment la description de α.

- $P_\mathcal{W}(o)$ *vérifie apparemment la description* de α ssi $P_\mathcal{W}(o)$, soit vérifie définitivement α, soit est fermé pour α et $n \in card(\alpha, P)$ et $\forall v \in P_\mathcal{W}(o)$, v vérifie apparemment la description de $type(\alpha, P)$.

La compatibilité de α avec les classes de o ne figure pas dans la définition précédente car ce n'est pas une propriété monotone.

La poursuite de cet assouplissement passerait par l'abandon de la fermeture, pour une notion de *fermabilité* : l'objet n'est pas fermé mais sa fermeture n'entraînerait pas d'inconsistance.

NOTE *6.6* *Version récursive*

Pour toute les propriétés relatives à la description d'une classe α, une définition plus opératoire consiste à les définir récursivement relativement aux super-classes directes $Super(\alpha)$ et à l'intension propre $Int^*(\alpha)$: la propriété est vraie de α si elle vraie de toute super-classe dans $Super(\alpha)$ et si sa condition portant sur les attributs de la classe est vraie pour tout attribut de $\mathcal{P}^*_\alpha$. Si c'est possible pour la vérification apparente, ça ne l'est plus pour la vérification définitive (figure 6.7).

NOTE *6.7* *Intension ou description ?*

Les diverses notions de compatibilité ou de vérification se comprennent bien lorsqu'elles sont relatives à la *description* d'une classe, mais leur intérêt est moins clair lorsqu'elles sont relatives à son *intension*. À quoi bon ces dernières ? Voici déjà deux réponses, en attendant mieux :

- une classe peut être définie dans l'absolu (formule 4.63), ce qui impose des comparaisons avec l'intension de la classe, mais l'objet doit quand même être compatible avec la description de la classe, sous peine d'inconsistance [18] : cela ramène donc à une comparaison avec la description, mais au lieu qu'elle soit booléenne (`vrai` / `faux`), elle est ternaire : `vrai` / `faux` / `inconsistant` ;
- pour [Gonzalez-Gomez, 1996], l'objet à classer n'en est pas un, mais une simple collection de couples attribut-valeur : ce pseudo objet n'ayant pas de classe, les deux notions sont équivalentes.

En l'état de nos arguments, seules les notions relatives à la description sont réellement nécessaires.

18. De façon générale, la distinction entre CN et CNS donne la possibilité d'inconsistances et d'objets paradoxaux, qui vérifient les CNS mais pas les CN.

6.3.4 Classification intensionnelle : définitions concrètes

Nous pouvons maintenant proposer des définitions précises pour la classification structurelle — bien que ces définitions ne soient plus en fait totalement structurelles — en remplaçant la formule intuitive $P_{\mathcal{W}}(o) \in Dom_P(\beta)$ par sa traduction en termes de vérification, de compatibilité et de fermeture :

- une classe β est *structurellement possible* pour o ssi o est compatible avec la description de β, elle est impossible sinon ;
- une classe β est *structurellement sûre* pour o ssi o vérifie définitivement la description de β. La vérification définitive peut s'interpréter par le fait que seuls une inférence extensionnelle ou un typage explicite de l'objet par une classe déclarée incompatible peut rendre la classe impossible.
- une classe β est *structurellement probable* pour o ssi o vérifie apparemment la description de β. On définit comme d'habitude le prédicat/ensemble $SProb$.

Justifions ces définitions :

- la correction de la définition de structurellement impossible est évidente : si o est incompatible avec la description de β, β est clairement insatisfaisable ; inversement, cette définition semble être structurellement maximale : on ne voit pas bien ce que l'on pourrait lui rajouter. La correction de la définition de structurellement possible suit par complément.

  Pour une classe seulement structurellement possible, l'objet peut être tellement incomplet (incomplétude extensionnelle) qu'il est possible, voire probable[19], que la classe soit satisfaisable mais hautement improbable qu'elle soit solution : il y a peu de chance que l'état du monde corresponde au rattachement de l'objet à la classe.

- Pour la correction de la définition de structurellement sûre, il faut vérifier la propriété (6.4), et pour cela, il suffit de vérifier (*est-ce si sûr ? faire une démonstration un peu plus formelle*) que si $SSur(\alpha, o)$ tient dans la description du monde $\mathcal{W}$, elle tient aussi dans toute description du monde consistante contenant $\mathcal{W}$.

  pour une classe structurellement sûre, l'objet est suffisamment complet pour que la classe soit probablement satisfaisable et pour que l'utilisateur puisse décider si la classe est solution ou non, réduisant ainsi l'incomplétude définitionnelle qui demeure ; en fait, le rattachement de l'objet à la classe ne devrait pas dépendre de l'état du monde (extensionnel), mais du domaine modélisé (intensionnel) ; l'hypothèse de discrimination des intensions fait qu'il est toujours possible de décider si un objet peut (doit) être rattaché, ou pas, à une classe β lorsque cette classe est structurellement sûre (bien que certains attributs de l'objet, ou des objets valeurs de ses attributs, ne soient pas effectivement connus à cause de la vérification par héritage).

19. Plus un objet est incomplet, plus il est possible. Plus un objet est complet, plus il est probable (tant qu'il est possible).

- enfin, une classe qui n'est que structurellement probable, même si elle est définie, ne peut pas être rattachée automatiquement à l'objet : elle peut en revanche être soumise pour approbation à l'utilisateur, qui peut décider de ce rattachement. L'intérêt de ces classes est que l'objet n'est en relation (du point de vue de la classe), qu'avec des objets eux-mêmes structurellement probables (pour le type des attributs de la classe). En particulier, en cas de circularité, il n'est pas possible d'avoir des objets structurellement sûrs, mais seulement structurellement probables.

6.3.5 Règles d'inférence de classification incertaine

Ces définitions n'ont en réalité de sens que dans leur utilisation par un système de déduction, au moins pour la possibilité structurelle qui n'est pas une propriété stable.

Pour les classes impossibles

Il est facile d'exhiber des règles d'inférence pour les classes impossibles :

$$[InSat/\nabla] \qquad \frac{\vdash (\beta\ o) \qquad \vdash \alpha\nabla\beta}{\vdash InSat(\alpha, o)}$$

$$[InSat/card] \qquad \frac{\{\vdash (P\ o\ o_i)\}_{i=1,n}}{\vdash InSat(\alpha, o)} \qquad n > \max(card_P(\alpha))$$

$$[InSat/type] \qquad \frac{\vdash (P\ o\ o') \qquad \vdash InSat(type_P(\alpha), o')}{\vdash InSat(\alpha, o)}$$

ce qui conduit à deux règles simples pour l'inconsistance et l'incohérence :

$$[\text{incohérence}] \qquad \frac{\vdash \alpha \trianglelefteq \bot}{\vdash InSat(\alpha, o)}$$

$$[\text{inconsistance}] \qquad \frac{\vdash (\alpha\ o) \qquad \vdash InSat(\alpha, o)}{\text{inconsistance}}$$

sans compter la règle [$InSat/closed$] vue plus haut. Il n'est pas nécessaire de rajouter une règle précisant que toute sous-classe d'une classe impossible est une classe impossible : c'est une conséquence des 3 règles qui précèdent. On remarque qu'il ne s'agit déjà plus d'une impossibilité structurelle à cause de [$InSat/\nabla$]. Ces trois premières règles (mais pas [$InSat/closed$]) doivent être reformulées pour étendre ∇ au cas n-aire ainsi que pour faire apparaître explicitement l'intersection des cardinalités (cf. note 13).

$$[InSat/\nabla*] \qquad \frac{\{\vdash (\beta_i\ o)\}_{i=1,n} \qquad \vdash \nabla\{\alpha, \beta_1, ..., \beta_n\}}{\vdash InSat(\alpha, o)}$$

$$[InSat/card*] \qquad \frac{\{\vdash (P\ o\ o_j)\}_{j=1,n} \quad \{\vdash (\beta_i\ o)\}_{i=1,m}}{\vdash InSat(\beta_0, o)} \qquad n > \max \bigcap_{i=0,m} card_P(\beta_i)$$

[$InSat/type*$]

$$\frac{\vdash (P\ o\ o') \quad \{\vdash (\beta_i\ o)\}_{i=1,m} \quad \vdash \nabla\{type_P(\alpha), type_P(\beta_1), ..., type_P(\beta_m)\}}{\vdash InSat(\alpha, o)}$$

$$\alpha, \beta_i \in dom(type_P)$$

Pour [$InSat/type*$], il faut en fait appliquer ∇ à tous les $type_P$ des classes de $Sur\{\beta_i\}_i$ (cf. note 11), mais c'est inutile de l'expliciter dans une règle car ce n'est qu'un cas d'application de [$InSat/type*$].

Soulignons qu'en ce qui concerne $card_P$, un test d'appartenance unaire (à une classe unique), comme dans [$InSat/closed$], suffit mais qu'une comparaison avec le max doit être n-aire, comme dans [$InSat/card*$], car :

$$\max \bigcap_i card_P(\beta_i) \leq \min_i(\max(card_P(\beta_i)))$$

l'égalité n'étant vraie (en toute généralité) que pour les intervalles (exemple des cardinalités paire et impaire, dont les max sont infinis, mais l'intersection vide). $type_P$ pose un peu le même problème. La règle

[$InSat/closed*$]

$$\frac{\{\vdash (P\ o\ o_i)\}^{i=1,n} \quad \vdash closed(o, P) \quad \{\vdash (\beta_i\ o)\}_{i=1,m}}{\vdash InSat(\beta_0, o)} \quad n \notin \bigcap_{i=0,m} card_P(\beta_i)$$

est ainsi inutile : soit $n \notin card_P(\beta_0)$ et la règle [$InSat/closed$] suffit soit n n'appartient pas à l'intersection et [$InSat/closed$] ne s'applique pas pour β_0 : dans ce cas elle s'applique pour un $\beta_i, i > 0$ et il y a une inconsistance.

Pour les classes définies

La propriété *structurellement sûre* est monotone, ce qui permet de déduire la validité des classes définies :

[Val-def-abs] $$\frac{\{\vdash SSur(o, P, \alpha)\}_{P \in \mathcal{P}_\alpha}}{\vdash (\alpha\ o)} \quad \alpha \text{ définie absolue}$$

[Val-def-rel] $$\frac{\{\vdash (\beta\ o)\}_{\beta \in Super(\alpha)} \quad \{\vdash SSur(o, P, \alpha)\}_{P \in \mathcal{P}^*_\alpha}}{\vdash (\alpha\ o)} \quad \alpha \text{ définie relative}$$

On remarque que ces règles sont des règles de classification, pas de subsomption. On peut donc les remplacer par les règles de subsomption suivantes :

[$\trianglelefteq$-def-abs]

$$\frac{\{\vdash card_P(\alpha) \subseteq card_P(\beta) \quad \vdash type_P(\alpha) \trianglelefteq type_P(\beta)\}_{P \in \mathcal{P}_\beta}}{\vdash \alpha \trianglelefteq \beta}$$

$$\beta \text{ définie absolue}$$

[$\trianglelefteq$-def-rel]

$$\frac{\{\vdash \alpha \trianglelefteq \gamma\}_{\gamma \in Super(\beta)} \quad \{\vdash card_P(\alpha) \subseteq card_P(\beta) \quad \vdash type_P(\alpha) \trianglelefteq type_P(\beta)\}_{P \in \mathcal{P}^*_\beta}}{\vdash \alpha \trianglelefteq \beta}$$

$$\beta \text{ définie relative}$$

Le prédicat $SSur(o, P, \alpha)$ reproduit tous les cas de la vérification définitive (figure 6.7) :

[$SSur$-hérit]

$$\frac{\{\vdash (\beta_i\ o)\}_{i=1,n} \quad \{\vdash (P\ o\ o_j)\}_{j=1,m} \quad \vdash \mathit{type}_P(\beta) \trianglelefteq \mathit{type}_P(\alpha)}{\vdash SSur(o, P, \alpha)}$$

$$\beta \in \{\beta_i\}_i \text{ et } m > \max(\mathit{card}_P(\alpha) - \bigcap_{i=1,n} \mathit{card}_P(\beta_j))$$

[$SSur$-actuel]

$$\frac{\vdash \mathit{closed}(o, P) \quad \{\vdash (P\ o\ o_j)\}^{j=1,m} \quad \{\vdash (\mathit{type}_P(\alpha)\ o_j)\}_{j=1,m}}{\vdash SSur(o, P, \alpha)}$$

$$m \in \mathit{card}_P(\alpha)$$

On remarque que [$SSur$-hérit] suit, en ce qui concerne les cardinalités, la formule (6.11), mais, en ce qui concerne les types, il ne sait pas gérer les conflits, pas plus que [$\trianglelefteq$-def-abs] ou [$\trianglelefteq$-def-rel] : dans la suite, on étendra le système de déduction à la conjonction définie d'intensions, avec le constructeur `and`.

Pour les propriétés spécifiques

Toutes les règles qui précèdent complètent les règles du chapitre 5 pour donner un système de déduction calculant des classes valides et insatisfaisables, ainsi que des inconsistances.

Il faudrait bien trouver aussi un moyen d'exprimer les inférences valides qui s'expriment à partir de classes seulement possibles.

Mais s'il est relativement facile, pour les classes sûres ou seulement possibles, d'exprimer des règles correspondant à des définitions structurelles, cela ne donne comme on l'a vu, aucune garantie de correction et ce serait des règles par nature non monotones, de même que toutes les règles qui utiliseraient en prémisse la négation de l'insatisfaisabilité, ou plus exactement la négation de la déductibilité de l'insatisfaisabilité, ce qui serait une façon commode d'introduire les classes incertaines. Les règles suivantes qui pourraient exprimer l'incertitude, l'insatisfaisabilité ou la satisfaisabilité :

[$InCert$-intro] $$\frac{\mathcal{H}; \mathcal{W} \nvdash InSat(\alpha, o) \qquad \mathcal{H}; \mathcal{W} \nvdash (\alpha\ o)}{\mathcal{H}; \mathcal{W} \vdash InCert(\alpha, o)}$$

[Sat-intro] $$\frac{\mathcal{H}; \mathcal{W}, (\alpha\ o) \nvdash \text{ inconsistance}}{\mathcal{H}; \mathcal{W} \vdash Sat(\alpha, o)}$$

[backtrack] $$\frac{\mathcal{H}; \mathcal{W}, (\alpha\ o) \vdash \text{ inconsistance}}{\mathcal{H}; \mathcal{W} \vdash InSat(\alpha, o)}$$

sont problématiques à des degrés divers. La première doit être exclue parce qu'elle n'est pas monotone. La seconde n'est correcte qu'en cas de complétude de l'inférence de l'inconsistance, ce qui suppose que cette dernière ne repose pas, par exemple, sur la complétude de l'inférence de Sat. De plus, la propriété Sat n'est pas conservée lors de l'évolution monotone de la description du monde : elle doit donc être abandonnée lors de l'affirmation de toute nouvelle description. La troisième est la plus claire : il n'est même pas utile de rajouter la condition ou prémisse portant sur la consistance courante ($\mathcal{H}; \mathcal{W} \not\vdash$ inconsistance), dans la mesure où si la description du monde est inconsistante, aucune inférence n'est incorrecte. Il est pourtant possible d'utiliser le fait que quelque-chose n'a pas été encore déduit, à condition que cela reste monotone. On pourra ainsi commencer par définir un prédicat correspondant à l'existence d'une classe vérifiee dans un ensemble :

[$\exists sat$-inconsist] $$\frac{\vdash \exists sat(A, o) \qquad \{\vdash InSat(\alpha, o)\}_{\alpha \in A}}{\vdash \text{inconsistance}}$$

[$\exists sat/\trianglelefteq$] $$\frac{\vdash \exists sat(A, o) \qquad \{\vdash \alpha \trianglelefteq \beta\}_{\alpha \in \{\gamma \in A \ | \ \not\vdash InSat(\gamma, o)\}}}{\vdash (\beta \ o)}$$

Bien que peu intuitive à première vue, cette règle est correcte car elle reste monotone malgré la négation de la dérivabilité de $InSat$: la négation doit être comprise « dans l'état courant de la dérivation ». La complétude du calcul de $InSat$ n'est *a fortiori* pas requise. Si de nouvelles instances de $InSat$ sont déduites, cela fera autant de γ en moins, et donc autant de β en plus (tant que la consistance est préservée). On utilise alors ce nouveau prédicat pour l'univocité et l'exhaustivité :

[$\exists$-exhaust] $$\frac{\vdash (\alpha \ o)}{\vdash \exists sat(A, o)} \qquad A = \text{ ensemble des sous-classes puits de } \alpha$$

[$\exists$-univoc] $$\frac{\{\vdash (\alpha_i \ o)\}_{i=1,n}}{\vdash \exists sat(A, o)} \qquad A = \text{ensemble des minorants max. des } \alpha_i$$

mais aussi pour tenir compte de l'explication des attributs (cf. paragraphe 4.3.3, page 81) :

[expliq-P] $$\frac{\vdash (P \ o \ o')}{\vdash \exists sat(\overline{Def}(P), o)}$$

NOTE *6.8* *Règles d'inférence de la conjonction*

Pour résoudre les problèmes de non monotonie ou de conflits de $type_P$, il faut introduire dans le système de déduction la conjonction.

[and-gauche] $$\frac{\vdash \alpha \trianglelefteq \beta}{\vdash \text{and}(.., \alpha, ..) \trianglelefteq \beta}$$

[and-droite] $$\frac{\{\vdash \alpha \trianglelefteq \beta_i\}_{i=1,n}}{\vdash \alpha \trianglelefteq \text{and}(\beta_1, ..., \beta_n)}$$

[and-elim] $$\frac{\vdash \alpha \trianglelefteq \text{and}(.., \beta, ..)}{\vdash \alpha \trianglelefteq \beta}$$

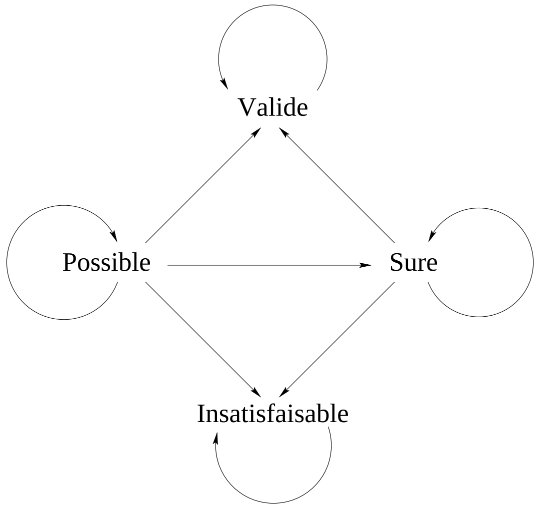

FIG. 6.8 – *Évolution temporelle des marquages*

moyennant quoi, [$SSur$-hérit] se réécrit plus précisément :

[$SSur$-hérit]

$$\frac{\{\vdash (\beta_i\ o)\}_{i=1,n} \quad \{\vdash (P\ o\ o_j)\}_{j=1,m} \quad \vdash \mathrm{and}(type_P(\beta_1),...,type_P(\beta_n)) \trianglelefteq type_P(\alpha)}{\vdash SSur(o,P,\alpha)}$$

$$m > \max(card_P(\alpha) - \bigcap_{i=1,n} card_P(\beta_j))$$

Introduire la conjonction dans le système de déduction d'un langage qui ne la possède pas peut certainement avoir des effets pervers.

6.3.6 Schéma d'algorithme

On peut extraire de ces notions cinq types de marqueurs — `valide`, `insatisfaisable`, `sûre`, `probable` ou `possible` — correspondant aux catégories de classes valide, satisfaisable, structurellement sûre, probable et possible. Par défaut, toute classe est `possible`. Elle peut devenir `valide` ou `insatisfaisable` (dans le sens que nous leur avons donné précédemment) ou `sûre`, qui correspond à structurellement sûr (figure 6.8). Le marquage `sûr` se fait par les mécanismes d'inférence généraux et n'est incompatible ni avec `valide` ni avec `insatisfaisable`. En revanche, le marquage `probable` est non monotone et doit être recalculé chaque fois, mais il ne sert pas à faire des inférences.

Un schéma d'algorithme sera donc le suivant :

1. marquer `possible` toutes les classes ;

2. marquer `valide` ou `insatisfaisable` toutes les classes dont on peut déduire l'état (en se servant éventuellement des définitions structurelles pour possible et sûr, et bien entendu d'impossible), tant que de nouvelles inférences sont faites ;

3. marquer `sûr` toutes les classes marquées `possible` qui le peuvent ;

4. si le marquage a évolué depuis le cycle précédent, revenir en 2 ;

5. sinon effacer l'éventuel ancien marquage `probable` et le recalculer ;

6. dialoguer avec l'extérieur pour essayer d'obtenir plus d'information permettant de revenir en 2.

L'essentiel est de bien distinguer le marquage définitif par `valide`, `insatisfaisable` ou `sûr` (ces 3 marques cohabitant, on peut unifier les étapes 2 et 3) du marquage provisoire par `probable` qui sous-entend « structurellement ».

6.4 Le processus de classification

La sémantique de la classification étant établie, il faut voir comment la mettre en œuvre, soit automatiquement, soit interactivement.

6.4.1 Espace de classification

Notons d'abord que la classification est circonscrite à un certain espace. S'il n'est pas déterminé explicitement, mieux que la hiérarchie toute entière, c'est l'ensemble des classes qui sont compatibles avec l'objet à classer. Dans un SC univoque, c'est tout simplement les sous-classes de la classe courante. Dans les LT, il s'agit en général de l'espace déterminé par les concepts primitifs connus de l'objet à classer : la classification automatique ne peut pas franchir la barrière des classes primitives.

Dans une démarche itérative, l'espace immédiat de la classification est constitué, pour un SC univoque, des sous-classes directes de la classe courante. En revanche, si le SC n'est plus univoque, il fait considérer les sous-classes directes de toutes les classes courantes, qui ne sont pas déjà des classes de l'objet. L'univocité réduit grandement cet espace de travail. Plus généralement, l'espace de classification de la classification incertaine est déterminé par l'étiquetage courant des classes puisque son évolution dans le temps est largement contraint (figure 6.2).

Pour TROPES, l'espace de la classification est déterminé par le *concept* de l'objet, successeur de la notion de *famille* introduite par [Procop et Pivot, 1987]. Mais qu'il s'agisse de SHIRKA ou de TROPES, l'espace est en fait restreint aux sous-classes de *la* classe courante de l'objet : c'est cette restriction qui limite tant la classification en SHIRKA. En revanche, pour FROME, l'espace de classification est ouvert à l'ensemble des sous-classes de la classe d'instanciation de l'objet (celle qui a créé l'objet).

NOTE *6.9* *Espace de classification ou d'instanciation ?*
Cette approche de la classification ne respecte pas la symétrie fondamentale qu'il y a entre les classes et les instances. certaine sinférences comme [*type*-infér], qui a un effet assez automatique hors tout espace de classification, ne sont pas prises en compte, et, encore moins, les inférences extensionnelles de l'exemple de `Equipe-moderne` : dans une inférence à la CSP, il vaudrait mieux parler d'espace d'instanciation ou de valuation.

6.4.2 Classification automatique

Seules les classes *valides*, ou plutôt leur sous-ensemble *sûr*, peuvent prétendre faire l'objet d'une *classification automatique*[20], c'est-à-dire une déduction correcte et générale de l'appartenance d'un objet à leur extension. Ce n'est donc envisageable, en général, que pour des classes définies ou pour des inférences très spécifiques, l'une d'entre elles étant néanmoins complètement générale ([*type*-infér]).

Le processus de classification peut se concevoir de différentes façons. En *chaînage arrière*, on cherchera à classer explicitement un objet précis. En *chaînage avant*, on propagera toute nouvelle information lors de son introduction dans la description du monde, pour classer tous les objets qui en dépendent. [Nebel, 1990] préconise la seconde solution et en décrit une mise en œuvre incomplète (les procédures CLASSIFY pour les classes et REALIZE pour les instances).

6.4.3 Classification interactive

Pour une classe purement descriptive et, même si $|\underline{SSur}(o)| = 1$, la seule classification correcte possible est *interactive* : suivant un protocole à définir, l'utilisateur choisit successivement des classes dans $\overline{SPos}(o)$, en privilégiant éventuellement $\overline{SSur}(o)$ ou $\overline{SProb}(o)$. La correction de cette classification interactive repose sur l'hypothèse que l'utilisateur choisit parmi des classes satisfaisables — charge au SC de les lui présenter —, et qu'il apporte le supplément de sémantique nécessaire pour combler l'incomplétude syntaxique du SC. Fondamentalement, le processus d'interaction revient à choisir certains modèles, en souhaitant qu'il s'agisse bien de modèles correspondant à l'*état du monde*. Si l'utilisateur choisit une classe qui n'est pas solution, donc qui ne correspond pas à l'état du monde courant, une contradiction ultérieure est inéluctable. Cette contradiction se traduira par une inconsistance, mais elle n'est pas appréhendable *au moment du choix*.

Il faut juste supposer que la part de classification automatique qui serait réalisable aura été réalisée auparavant : les déductions valides que le système sait effectuer — le calcul des classes sûres — l'aura été. On a vu que la correction de certaines notions incertaines dépend de la complétude de la classification automatique. Cependant il n'est pas possible, en toute généralité, de faire cette hypothèse de complétude : lors d'une étape de classification incertaine, toutes les classes sûres sont donc connues (c'est leur définition), mais il peut rester des classes valides inconnues.

Dans le cas d'un objet incomplet (relativement à la classe considérée), l'espoir de classes satisfaisables, voire solution, est vain, et l'utilisateur doit se contenter de classes possibles, avec les risques d'inconsistance que cela présente.

On peut bien sûr remplacer l'utilisateur par un autre mode de raisonnement automatique, extérieur à la représentation par objets et ayant sa propre sémantique. Dans tous les cas, la présence de circularités peut soulever des difficultés importantes, d'où sans doute la restriction de TROPES aux objets composites [Mariño, 1991;

20. Noter que l'analyse de données utilise ce terme pour désigner la construction de (la hiérarchie de) classes à partir d'un ensemble d'individus, pour laquelle nous utilisons plutôt le terme de *catégorisation*.

Mariño, 1993][21]. L'interaction avec l'utilisateur sert aussi à acquérir les valeurs manquantes.

Généralement, le processus de classification est incrémental et glouton : l'instance à classer est spécialisée de sa classe courante à une sous-classe directe, le processus s'arrêtant lorsque l'instance est « bloquée ». Il n'y a pas forcément de prévision (par exemple existence d'un puits possible pour un SC exhaustif), et il faut donc s'attendre, dans certains cas, à des remises en cause ultérieures. On a vu en particulier que la modalité possible ne traduit pas forcément une réelle satisfaisabilité, mais peut-être seulement une possibilité locale.

Il n'y a donc pas de calcul effectif des différents ensembles, mais examen successif des classes candidates, dans un ordre (approximativement du plus général au plus spécialisé) dépendant de l'état de l'objet et du dialogue avec l'utilisateur. Cet examen conduit à un étiquetage de la classe par l'une des modalités possible, impossible, etc., suivi d'une éventuelle propagation.

Cette vision de la classification comme un processus itératif descendant de classe sûre en classe sûre (classe possible devenue sûre par l'accord de l'utilisateur) est une vision tout à fait partielle. Si elle s'applique bien à SHIRKA, elle n'est pas suffisante en cas de multi-instanciation ou d'*exhaustivité* : en fait, même dans un système dont la sémantique est univoque, il faut pouvoir exprimer des connaissances incomplètes correspondant à une description du monde momentanément équivoque[22]. Il faut donc préciser : « de classes sûres en clesses sûres ».

De son côté, l'exhaustivité exige de pouvoir identifier un *puits* possible avant de descendre, et certaines notions comme la vérification *par héritage* n'ont aucun intérêt en mono-instanciation (ou *univocité*). Dans un système comme TROPES, une descente itérative est possible dans chaque point de vue, mais la vérification par héritage peut permettre de tirer profit de ce que l'on sait des autres points de vue. La sémantique du « possible » du modèle SHOOD (cf. note 5) impose aussi une vision définitive.

Par ailleurs, la classification d'un objet a des effets sur les objets dont il est valeur d'un attribut ou qui sont valeur d'un de ses attributs. Pour la classification automatique, ce n'est pas vraiment un problème, l'algorithme pouvant traiter ces diverses classifications dans l'ordre qu'il veut. En revanche, pour un processus interactif, il faut bien sûr faire toutes les inférences valides sur tous les objets potentiellement classables avant d'en arriver à une phase interactive. Le schéma d'algorithme du paragraphe 6.3.6 doit être modifié en conséquence, avec les problèmes de marquage multiple que cela peut poser. Le problème de la présentation à l'utilisateur de ces diverses classifications dépendantes est certainement difficile.

21. Une explication plus profonde est la suivante : peut-on réellement représenter les relations entre objets comme attributs de ceux-ci lorsque l'on prétend représenter des connaissances en toute indépendance vis-à-vis de leur éventuelle exploitation ? Sans doute non, à moins de se réduire aux relations de composition, qui décrivent vraiment des attributs du composite, si ce n'est des composants.

22. Le problème est qu'il n'est pas possible de se limiter à une description du monde univoque : il faudrait aussi que sa fermeture par $\models$ le soit aussi. Il faudrait donc avoir une sémantique dans laquelle l'opérateur $\models$ conserve l'univocité. C'est, trivialement, possible lorsqu'il est trivial, comme en SHIRKA. C'est, moins trivialement, possible lorsque le SC est un treillis classificatoire.

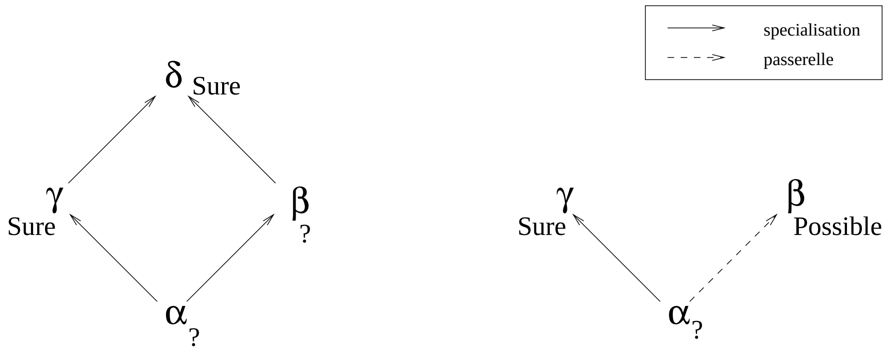

FIG. 6.9 – *La différence entre spécialisation ($\prec$, traits pleins) et subsomption ($\lhd_a$, pointillés), pour la classification : à gauche, β peut être complété comme intermédiaire dans la reconnaissance de α. À droite, β ne peut être utilisé que de façon négative et n'a donc plus d'effet à partir du moment où il est* possible.

Enfin, si l'algorithme ne sait pas proposer à l'utilisateur des classes réellement *satisfaisables* mais seulement *envisageables*, il faut considérer le risque d'inconsistance résultant d'une décision erronée de l'utilisateur et il faut donc pouvoir revenir en arrière avec les difficultés que cela peut présenter : comment reconnaître exactement la décision erronée, etc. Ce retour arrière pourrait être représenté par la règle [backtrack]. Mais tout cela nous fait sortir du cadre monotone que nous nous sommes imposé pour entrer dans le cadre des problèmes de révision [Nebel, 1990; Crampé et Euzenat, 1996].

Dans tous les cas, la classification interactive devient un problème de dialogue homme-machine qui doit respecter les règles générales de pertinence de ce dialogue, mais aussi être guidé par les propriétés formelles générales des SC et spécifiques du SC considéré : TROPES montre ainsi comment différents points de vue permettent de guider l'interaction en se focalisant sur les points de vue qui intéressent l'utilisateur (au dépens éventuellement de la complétude de la classification).

Il faut analyser les passerelles ($\lhd_a$) en termes de classification interactive. Supposons que $\alpha \prec \beta$ et que le classifieur veuille vérifier si l'objet o est ou n'est pas une instance de α (par exemple parce qu'il a déjà déterminé qu'une super-classe directe de α est sûre) : il est alors logique qu'il se pose d'abord la question pour β, quitte à demander à l'utilisateur la valeur d'un attribut de β non valué dans o (figure 6.9 gauche) [23].

En revanche, si $\alpha \lhd_a \beta$, le classifieur peut regarder s'il ne sait pas déjà que β est impossible, auquel cas α aussi, mais il paraît tout à fait illogique qu'il puisse chercher à compléter les attributs de β inconnus de o (figure 6.9 droite).

Sur la classification interactive, voir aussi [Finin, 1986], qui s'intéresse essentiellement à la *catégorisation*, c'est-à-dire à la construction de la hiérarchie.

23. Notons que SHIRKA ne le fait certainement pas parce qu'il se limite aux sous-classes de la classe courante

NOTE *6.10* *Idéaux de classes*

Tous les ensembles de classes considérés dans ce chapitre ont de très bonnes propriétés par rapport à $\preceq$: ce sont des idéaux (cf. annexe A.2.3). Soit $\alpha \prec \beta$:

- si $\alpha \in Val(o)$ (resp. $Sat(o)$, $Sur(o)$, $Pos(o)$, etc.), alors $\beta \in Val(o)$ (resp. $Sat(o)$, $Sur(o)$, $Pos(o)$, etc.) ;
- si $\beta \in InSat(o)$ (resp. $ImPos(o)$, $SImPos(o)$), alors $\alpha \in InSat(o)$ (resp. $ImPos(o)$, $SImPos(o)$).

L'inclusion de ces différents ensembles (figure 6.3) se ramène donc à un ordre sur leurs antichaînes maximales ou minimales. Ces propriétés ont un effet non négligeable sur l'efficacité des algorithmes.

Cependant, si les fonctions Sur et $ImPos$ sont superflues puisque $\underline{Sur}$ et $\overline{Impos}$ suffisent, les ensembles $SPos$ et $SSur$ doivent être considérés dans leur totalité — au moins $SPos - SSur$ et $SSur - Sur$ — puisqu'il n'est pas possible d'en rien conclure automatiquement.

De plus, les définitions extensionnelles (resp. structurelles) produisent aussi des idéaux de la subsomption extensionnelle (resp. intensionnelle).

NOTE *6.11* *Non objet à classer*

On a considéré que c'était un objet qu'il s'agissait de classer et, qu'en tant que tel, il était instance d'au moins une classe, avec les contraintes existant entre $Isa_{\mathcal{W}}(o)$ et $Attr_{\mathcal{W}}(o)$.

De son côté, [Gonzalez-Gomez, 1996] considère que l'objet à classer n'est pas un objet, mais une simple intension, ensemble de couples attribut-valeur, tous les attributs étant donc des attributs inexpliqués (cf. paragraphe 4.3.3, page 81). Il n'est donc pas possible d'avoir un processus incrémental, et il faut un étiquetage de toutes les classes, suivi du choix des « meilleures ».

NOTE *6.12* *Rattachement négatif*

Il faudrait aussi étudier le *rattachement négatif*, c'est-à-dire la possibilité pour l'utilisateur de dire qu'un objet n'est pas instance d'une des classes possibles proposées par le SC, dont on a vu qu'il est possible à partir du moment où le prédicat $InSat$ peut être aussi bien affirmé que déduit.

6.4.4 Le problème du calcul des valeurs manquantes

L'incomplétude des objets amène à chercher à les compléter par le calcul des valeurs de leurs attributs inconnus. Étant donné un tel attribut, dont l'existence est connue mais pas la valeur, c'est-à-dire $P \in Attr_{\mathcal{W}}(o)$ et $P_{\mathcal{W}}(o) = \emptyset$, il est possible de calculer ou compléter $P(o)$ si l'une des classes connues possède un moyen de le faire (par ex. un *réflexe* de type `si-besoin` des langages de *frames*, ou un *attachement procédural*). Lorsque les réflexes font partie du langage, il est nécessaire de préciser leur sémantique :

- calcul d'une *valeur par défaut* — similaire à une *facette* `defaut` des langages de *frames*—, ce qui ramène au problème plus général des valeurs par défaut, et induit une non monotonie que nous ne considérons pas ici ;

- calcul d'une valeur nécessaire pour toutes les instances : dans ce cas, si le réflexe est masqué par un autre (redéfini) dans une sous-classe, les deux doivent produire la même valeur, le second par des moyens que l'on peut présumer plus simples (ou plus rapides, précis, etc.) ;

- calcul d'une valeur nécessaire pour toutes les instances de la classe et de ses sous-classes qui ne redéfinissent pas le réflexe : dans ce cas, si l'on ne veut pas introduire de non monotonie, il n'est pas possible de se servir de ce réflexe si l'une des sous-classes qui le redéfinit est encore *possible* pour l'objet considéré[24].

Lorsque la sémantique monotone des réflexes est bien définie, ils sont assimilables à des contraintes directionnelles qui permettent de calculer un attribut inconnu en fonction des valeurs d'autres attributs, comme en GARNET [Myers *et al.*, 1992]. Ce sont bien des contraints dans la mesure où ils déterminent l'unique valeur affectable à l'attribut en fonction des autres attributs, comme si le domaine de l'attribut était un singleton fonction des autres attributs. On voit bien que l'hypothèse de l'indépendance des attributs n'est pas vérifiée pour l'attribut muni d'un réflexe, l'état de l'objet étant déterminé par les autres attributs dont le produit cartésien des domaines suffit à déterminer l'espace d'états de la classe.

Il n'est pas possible de se servir d'un tel moyen de calcul pour inférer la valeur de l'attribut, sauf éventuellement dans le dernier cas, tant qu'il est détenu par une classe seulement possible, pas plus qu'il n'est possible de calculer un attribut P dont la présence n'est pas connue, c'est-à-dire quand $P \notin \bigcup_{\alpha \in Isa_{\mathcal{W}}(o)} \mathcal{P}_\alpha$.

En revanche, il est tout à fait correct de s'en servir pour inférer l'impossibilité de la classe dans le cas où le réflexe retourne une valeur incompatible avec le domaine de l'attribut dans la classe : cela revient à faire du raisonnement hypothétique, à l'instar de la règle [backtrack][25].

Pour remédier à ce triste état de choses il a été envisagé diverses solutions :

- classification des moyens de calcul : dans ce cas, une hiérarchie parallèle de méthodes de calcul permet de calculer *la* valeur nécessaire (seconde interprétation des réflexes), le choix de la méthode étant dicté par les attributs déjà connus [Rechenmann, 1992] ; c'est ce que Rechenmann appelle le *détachement procédural*.

- classification dans un *point de vue* et passage dans un autre point de vue via des passerelles (en TROPES), ou, plus généralement, classification sur les attributs suffisants, puis héritage des attributs seulement nécessaires : ces nouveaux attributs pouvant être, ailleurs, suffisants, la classification peut se poursuivre.

24. Ce troisième point consiste en fait en une interprétation particulière de la valeur par défaut : curieusement, je n'ai pas l'impression de l'avoir jamais rencontrée (si ! dans [Girard, 1995]). Elle autorise pourtant un raisonnement parfaitement monotone. Bien sûr, le terme de défaut est alors impropre.

25. C'est une vision angélique de la programmation : si la classe se révèle impossible, le déclenchement du réflexe peut mener à des erreurs à l'exécution assimilables à des erreurs de type. Bien sûr, si l'erreur lève une exception sans laisser de trace, il n'y a pas de problème. Malheureusement, les erreurs « sans erreur », parfaitement rattrapées, sont du domaine de l'utopie même si l'on dispose du *retour en arrière* parfait, comme en SMECI [ILO, 1991] ou en CLAIRE [Caseau, 1996].

Il faut rappeler que les LT permettent de quantifier existentiellement et même quantitativement les valeurs manquantes. De son côté, FROME permet d'assurer de la présence d'un attribut, ce qui revient à affirmer l'existence d'une valeur. Dans les deux cas, cela permet de graduer le manque, alors que le calcul des valeurs manquantes est souvent envisagé dans un cas restreint d'attributs mono-valués, souvent à valeur dans les types de base. Cependant, il ne semble pas qu'un mécanisme de *réflexe* pour compléter une valeur partiellement mais incomplètement connue ait jamais été défini.

Enfin, les contraintes des CSP offrent un cadre général d'expression du calcul des valeurs manquantes [Gensel, 1995]. Sur toute cette problématique, voir [Girard, 1995].

6.5 Comparaisons

Depuis son introduction dans SHIRKA [Procop et Pivot, 1987], la classification incertaine a subi divers avatars dus à des malentendus ou à des variantes. De plus, cette approche n'est pas la seule à essayer d'élargir le champ d'application de la classification.

6.5.1 Historique de la classification incertaine

La génèse de cette idée de classification incertaine n'est pas totalement connue : a-t-elle été déjà exploitée avant SHIRKA, ou indépendamment de lui ? Il semble que non d'après [Euzenat et Rechenmann, 1995], mais on peut considérer [Granger, 1986; Granger, 1988], [Fikes et Kehler, 1985], [Finin, 1986] et KL-ONE [Brachman et Schmolze, 1985] comme des précurseurs ou inspirateurs complémentaires[26].

Sens de sûr dessous

L'historique des qualificatifs de *sûr* et *possible* et de leur interprétation nécessite quelques éclaircissements (figure 6.10).

Le problème vient des deux interprétations différentes de *sûr* : « il est *sûr* que l'objet est instance de la classe » ou bien « il est *sûr* que l'objet peut être rattaché à la classe ». Suivant l'interprétation, la classe est valide ou satisfaisable. Parallèlement à ces deux interprétations, *possible* se comprend comme « il est sûr que l'objet *peut* être rattaché à la classe » ou bien « il est *possible* que l'objet puisse être rattaché à la classe ».

En SHIRKA qui les a introduites, la signification de ces notions était claire : une classe *sûre* est une classe à laquelle il est sûr que l'objet peut être rattaché, une classe *satisfaisable* donc. En revanche, une classe *possible* n'est qu'une classe qui n'est pas immédiatement reconnue comme *impossible* : c'est donc une classe *incertaine* ou *possible*, *structurellement possible* en fait.

Par la suite, [Mariño, 1993] interprète le qualificatif de *sûr* par *valide* — comme s'il s'agissait de classification automatique sur des classes définies —, provoquant ainsi un certain quiproquo (TROPES90). De son côté, dans FROME, [Dekker, 1994] utilise

26. [Granger et Thonnat, 1985] et [Fikes et Kehler, 1985] apparaissent (sans être cités) dans la bibliographie de [Procop et Pivot, 1987]

SC	LMO'96	TROPES95	FROME	TROPES90	SHIRKA
valide	—		sûre	sûre	
satisfaisable	—	possible			sûre
insatisfaisable	—	impossible	impossible	impossible	impossible
S. sûre & définie	sûre		sûre		
S. sûre	probable	possible		sûre	sûre
S. probable					
S. possible	possible	inconnue	possible	possible	possible
S. impossible	impossible	impossible	impossible	impossible	impossible

FIG. 6.10 – *Significations et caractérisations successives de* sûr *et* possible. *Lorsqu'une case est vide, cela traduit le fait que la modalité correspondante n'est pas prise en compte par le système considéré.*

sûr pour des classes dont les conditions suffisantes sont vérifiées, ce qui en justifie l'interprétation comme des classes valides.

Pour éliminer tout risque de confusion, [Euzenat et Rechenmann, 1995] proposent maintenant pour SHIRKA et TROPES les qualificatifs de *possible* et *inconnu*, à la place de *sûr* et *possible* [Euzenat et Rechenmann, 1995; Projet Sherpa, 1995] (TROPES95) [27].

Enfin, dans une version préliminaire de ce travail [Ducournau, 1996], les versions extensionnelle et intensionnelle de la classification incertaine étaient distinguées, mais en mélangeant les définitions abstraites et concrètes (LMO'96).

La discussion sur la correction et la complétude de la classification structurelle (cf. paragraphe 6.2.4, page 122) montre en fait que la modalité « sûre », la moins incertaine de la classification incertaine à la SHIRKA, présente beaucoup d'incertitude. Quel que soit son qualificatif, son interprétation par « satisfaisable » est correcte en SHIRKA parce que ce système ne fait curieusement aucune inférence automatique — pas plus celle de la règle [*type*-infér] (puisque les valeurs d'un attribut doivent être du bon type lors de l'affectation) que celle de l'univocité (puisque l'objet ne peut jamais être classé que dans *une* sous-classe de son unique classe minimale courante) — tout en imposant un état « correct », la fermeture par $\models$, à tout instant. Mais si l'on imagine le système idéal réalisant la sémantique latente de SHIRKA, et mettant en œuvre aussi bien la règle [*type*-infér] que l'inférence spécifique de l'univocité (la première nécessite la seconde, à moins d'imposer une structure de treillis), il faudrait alors que le calcul des classes valides, satisfaisables et insatisfaisables soit complet pour assurer la correction des définitions structurelles, ce qui leur retirerait immédiatement tout intérêt. En revanche, il est à craindre que des systèmes plus complexes comme TROPES (ou FROME, mais [Dekker, 1994] ne se sert pas de classes *structurellement sûres* non définies) ne soient mis en défaut sur ce point, surtout si l'on y rajoute des contraintes comme [Gensel, 1995].

NOTE *6.13* ***Définitions extensionnelles, intensionnelles ou structurelles?***

Plusieurs raisons expliquent les difficultés rencontrées pour connaître la définition

27. On reprochera juste à ce nouveau tryptique le faux-ami provoqué par la non complémentarité de *possible* et *impossible*, ce qui ne nous a pas empêché de les reprendre dans un sens analogue.

exacte de la classification incertaine donnée par les différents auteurs cités (figure 6.10) :

- d'abord, aucun de ces auteurs n'est suffisamment formel pour que les définitions soient réellement univoques ; il est en plus difficile de distinguer la définition des éventuelles caractérisations ;

  La définition concrète, c'est-à-dire intensionnelle, de toutes ces catégories de classes est très circulaire et la façon dont les systèmes comme SHIRKA ou TROPES, gèrent cette circularité, ou plus généralement, quelle définition exacte ils adoptent, n'est pas claire. Ces systèmes considèrent principalement [28] des attributs dont le type est un type de base, donc dont les valeurs sont non classables, car toujours déjà classées (instances de puits dans un SC univoque). [Mariño, 1993] étudie cependant la classification d'objets composites, ce qui doit permettre de donner un équivalent de la définition circulaire donnée ici, sans avoir de circuits effectifs.
- ensuite, deux discours parallèles ont toujours été tenus, à quelques pages ou quelques années d'intervalle, sans être jamais explicitement corrélés : l'un est extensionnel, l'autre intensionnel ou plutôt structurel. Ainsi, dans [Rechenmann, 1988], la définition est implicitement extensionnelle, mais la caractérisation donnée dans les exemples ou ailleurs (par exemple dans [Euzenat et Rechenmann, 1995] pour SHIRKA et TROPES ou dans [Projet Sherpa, 1995] pour TROPES seul) est explicitement structurelle.
- enfin, les successeurs ont persévéré dans l'incertitude, voire l'ont aggravée : [Napoli, 1992] présente les classes sûres de SHIRKA comme des classes valides, [Mariño, 1993] oscille entre les définitions structurelles et extensionnelles (où valide remplace satisfaisable), et [Dekker, 1994] présente la classification de SHIRKA sous une forme structurelle ;
- il n'y a en fait aucune notion de modèle sous-jacente, uniquement un *domaine modélisé* : dans tous les cas, les classes possibles sont interprétées comme inconnues et non comme satisfaisables.

Si ces diverses raisons rendent difficiles une analyse exacte, le dernier point fournit une méta-analyse à peu près complète : la non distinction des niveaux extensionnel/intensionnel ou sémantique/syntaxique est manifestement la cause principale.

Tropisme classificatoire

Les propriétés spécifiques comme l'univocité ou l'exhaustivité, ou l'hypothèse de l'explication des attributs, offrent, de la même manière que le caractère défini des classes, un tropisme descendant qui se traduit par le fait que, *dans certaines conditions*, un objet peut être classé automatiquement dans une sous-classe de sa classe courante. C'est ce tropisme descendant qui fait la puissance de la classification, mais il a une contre-partie contraignante : ces « certaines conditions » sont en général plus ou moins structurelles et elles ont une influence intensionnelle non négligeable sur la hiérarchie des classes et sur leur cohérence, c'est-à-dire sur l'existence d'instance dans

28. C'est-à-dire au travers de la littérature qui leur est consacré : dans la pratique, c'est un peu moins vrai, mais leur traitement des objets dits *complexes* est souvent relégué en fin de thèse, voire bénéficie d'un traitement bizarrement privilégié

une description du monde consistante[29]. La subsomption extensionnelle est ainsi une conséquence de la classification :

$$\big(\forall o : Val(\alpha, o) \Rightarrow Val(\beta, o)\big) \Longrightarrow \alpha \trianglelefteq \beta \tag{6.12}$$

de même que la subsomption intensionnelle se déduit, en partie, des comparaisons structurelles :

$$\big(\forall o : SSur(\alpha, o) \Rightarrow SSur(\beta, o)\big) \Longrightarrow \alpha \trianglerighteq \beta \tag{6.13}$$

ce qui symétrise (6.5). Mais ces conséquences ne peuvent pas être exploitées sous cette forme extensionnelle, puisque la déduction ne peut pas énumérer toutes les instances d'une classe. (6.13) doit en fait se traduire par des définitions concrètes de $SSur$ et $\trianglerighteq$ assurant leur équivalence. Quant à (6.12), couplée à (6.4) et (6.5), elle amène à constater que le caractère défini des classes est une cause de subsomption :

$$\alpha \trianglerighteq \beta \;\; \& \;\; \beta \text{ définie } \Longrightarrow \alpha \trianglelefteq \beta \tag{6.14}$$

Ce constat a conduit les logiques terminologiques à la classification automatique des classes dont l'effet consiste à remplacer la subsomption extensionnelle ($\trianglelefteq$) par la spécialisation ($\preceq$) dans la dernière équation.

Cette classification automatique devrait être appliquée *a fortiori* à un système à objets univoque qui proposerait des classes définies : il faut à tout prix (sous peine d'incohérence) éviter d'avoir des classes incompatibles de même intension. Or, l'univocité conduit inexorablement à l'existence de classes définies : lorsqu'elle existe, la borne inférieure de deux classes est effectivement définie à partir de ces deux classes. Ce tropisme classificatoire peut faire apparaître des objets paradoxaux, dont tous les attributs vérifient scrupuleusement les domaines de leurs classes mais dont la seule existence provoque une inconsistance : c'est le cas par exemple de deux classes définies et incompatibles dont l'union des intensions est non vide, ou bien dans le cas de l'univocité si l'intension de la borne inférieure est strictement « plus grande » que la réunion ($\sqcup$) de l'intension des deux classes.

Il semble donc que les systèmes de représentation des connaissances à la SHIRKA ne peuvent pas faire plus longtemps l'économie d'une analyse approfondie de leur tropisme classificatoire et des mécanismes du niveau intensionnel susceptibles d'en limiter les effets pervers : la classification des classes, bien sûr mais pas seulement.

Classification incertaine et lt

À première vue, les LT ne s'intéressent qu'à la classification automatique. Pourtant, la plupart des LT sont très proches de la classification incertaine. Lorsque la logique terminologique considérée sait calculer la relation d'incompatibilité, par exemple pour $\mathcal{ALC}$ [Schmidt-Schauß et Smolka, 1991; Donini *et al.*, 1994], c'est en fait implicite : appliqué aux instances cela donne déjà le qualificatif *impossible. Sûr* est fourni par la

29. En ce sens, l'interprétation de la classification incertaine de Mariño (1993) est compatible avec l'interprétation définitionnelle qu'elle fait des classes de TROPES. L'erreur ne réside donc pas là mais dans le fait qu'elle ne se donne pas les moyens d'assurer cette interprétation définitionnelle.

classification et *possible* suit, par complément. La complétude est bien sûr une autre histoire.

Le problème est que cela reste potentiel : la classification n'opère que dans la hiérarchie des classes explicitement introduites par ce que [Nebel, 1990] reconnaît comme une assertion de second ordre, ce qui n'est pas le cas pour la négation de toute les classes. En fait, les LT pourvues de la négation savent calculer, de façon plus ou moins complète, l'insatisfaisabilité, c'est-à-dire la validité de la négation, mais c'est un calcul de subsomption — il faut fournir la classe à nier — alors que la classification repose sur un parcours systématique des classes pour leur appliquer un calcul de subsomption.

NOTE *6.14* *Fonctions de test trivaluées*

On remarquera avec intérêt que les fonctions de test de CLASSIC, utilisées par le constructeur `test-c` (cf. annexe B.1, page 191), doivent fournir eux aussi une réponse trivaluée : il est donc envisageable simplement d'y faire de la classification incertaine, mais uniquement sur les concepts définis de façon procédurale. Il ne semble pas que les prédicats de type fassent la même chose.

Cependant, le sens de cette troisième valeur n'est pas clair : il faut utiliser `T` quand l'objet vérifie définitivement le test, et `?` lorsqu'il vérifie, provisoirement, le test. Mais faut-il l'utiliser aussi lorsque l'objet ne vérifie pas, provisoirement, le test ?

6.5.2 Autres généralisations de la classification

Si la classification incertaine que nous venons de présenter semble trouver son origine dans SHIRKA, d'autres tentatives ont été faites pour étendre le champ d'application de la classification à des connaissances incertaines. De plus, cette approche incertaine a déjà été appliquée à d'autres problèmes, le filtrage en particulier.

Classification avec défauts et exceptions

Malgré l'interdit posé par [Brachman, 1985], [Coupey et Fouqueré, 1994; Coupey et Fouqueré, 1995] proposent d'introduire dans les LT deux constructeurs pour les *défaut* et *exception*, avec le langage $\mathcal{AL}^{-}_{\delta\epsilon}$. Cela leur permet notamment de considérer quatre catégories de relation entre classe et instance : sûre, probable, typique et exceptionnelle, les deux premières n'ayant aucun rapport avec les nôtres. Si cette première approche introduit une non monotonie au niveau extensionnel, elle reste complètement monotone au niveau des classes, malgré ces exceptions. [Padgham et Zhang, 1993; Padgham et Nebel, 1993] proposent des extensions non monotones du même genre.

Classification imprécise et incertaine

L'introduction de données imprécises et incertaines permet d'envisager une classification pondérée par un ou plusieurs coefficients. C'est ce que font [Rossazza, 1990], [Gonzalez-Gomez, 1996] et quelques systèmes plus ou moins *ad hoc* (cf. paragraphe 7.6, page 174). Dans tous les cas, le précis étant un cas particulier de l'imprécis, et le certain un cas particulier de l'incertain, cette classification imprécise et incertaine doit se ramener, dans le cas particulier de données précises et certaines, à une sémantique aussi fondée que celle qui a été proposée ici. Ce ne semble pas être toujours le cas dans les travaux cités plus haut.

L'introduction de données imprécises conduit à s'interroger sur la distinction entre *domaine* et *valeur* : les deux sont décrits par des ensembles flous, même si la seconde peut alors être appelée *valeur floue*. Traditionnellement, les langages de *frames* permettent de distinguer un aspect domaine et un aspect valeur, ce dernier traduisant un domaine (type) singleton, dont l'unique élément s'impose alors comme unique valeur admissible. Le fait de considérer des valeurs floues comme des valeurs conduit donc à considérer n'importe quel domaine comme une valeur floue. La distinction entre les valeurs et leur *type* disparaît, le calcul sur les valeurs s'étendant naturellement au calcul sur les ensembles. On se rapproche alors des CSP, où la réduction de domaine peut être vue comme une forme limitée de résolution, ainsi que des techniques d'approximation de Aït-Kaci.

Une autre façon de voir les choses est de considérer que les valeurs sont aux domaines ce que les prototypes sont aux classes : c'est effectivement l'approche suivie par OBJLOG+, bien que le terme de prototype y soit utilisé avec une sémantique de classe.

Filtrage incertain

Le filtrage se ramène, indirectement mais doublement, à la classification. Le filtre peut être considéré comme une classe, qu'il faut d'abord classer automatiquement, et par rapport à laquelle il faut ensuite classer toutes les instances de la description du monde. C'est la démarche suivie par les LT ou [Dekker, 1993]. La première étape ne donne *a priori* lieu à aucune incertitude, mais la seconde peut se faire par classification incertaine, non pas centrée sur l'*objet à classer*, mais sur le *filtre*, c'est-à-dire la *classe à instancier*. C'est là qu'il faut parler d'instances valides, satisfaisables, insatisfaisables, possibles, impossibles, sûres, etc.

Dans un deuxième temps, on constate que la classification de la classe (du filtre) elle-même peut donner lieu à un résultat multi-valué. Par rapport au filtre ϕ, une classe α peut être :

valide : toute instance de α vérifie le filtre : $\alpha \trianglelefteq \phi$;

satisfaisable : certaines instances de α peuvent satisfaire le filtre ;

insatisfaisable : aucune instance de α ne peut satisfaire le filtre : $\alpha \nabla \phi$.

Les modalités sûre, possible et impossible introduites dans YAFOOL par [Chabre, 1988], inspiré par la lecture de [Procop et Pivot, 1987], sont les analogues intensionnels (pas forcément équivalents, comme nous l'avons vu) de ces trois modalités extensionnelles. Ce filtrage incertain n'apporte donc pas de notion nouvelle mais c'est une étape obligée pour avoir un filtrage efficace : seules les instances des classes satisfaisables non valides, ou plutôt des classes possibles non sûres, sont à comparer effectivement au filtre.

NOTE *6.15* *Filtre et classe*

Pour que le filtre puisse être une classe, il faut pouvoir représenter des domaines qui soient des valeurs, soit en disposant de constructeur comme `one-of` ou `fills`, soit en étendant la définition de Dom. Pour assurer un minimum de généralité, un filtre doit s'exprimer en effet aussi bien par des couples attribut-*valeur* que par des couples attribut-*domaine*.

Chapitre 7

Quelques systèmes classificatoires

> La classification des tropes [...] permet en effet d'énoncer des vérités fondamentales sur la nature de toute classification, car [...] elle en décrit la structure.
> *La classification des tropes est donc la classification des racines de la classification.*
> Patrick Tort, *De la double racine du principe de classification*, 1983, in [Tort, 1989].

Il faudrait caractériser tous les systèmes classificatoires connus — ceux qui rentrent dans le cadre proposé ici bien entendu, pour les autres il faudrait chercher à étendre le cadre — par rapport aux diverses propriétés énoncées plus haut. On en est loin mais voici déjà quelques pistes, ce qui complète un peu la bibliographie. L'essentiel est bien sûr de distinguer dans chaque type potentiel de SC ce qui est de l'ordre des classes, des instances et des attributs. Pour certains d'entre eux, la difficulté est de distinguer, derrière les entités explicitement introduites, le SC virtuel qui exprime pleinement les propriétés du système considéré.

Soulignons que la théorie présentée ici ne se prétend qu'un cadre général permettant d'appréhender et de comparer les diverses familles de SC. Elle ne prétend en aucun cas être *la* théorie unifiée dont tous les SC ne seraient que des cas particuliers [Borgida, 1992a].

Note *7.1*
Ce chapitre ne prétend à l'exhaustivité ni du point de vue de l'ensemble des systèmes classificatoires abordés, encore moins du point de vue de chacun. Il s'agit plus d'un pense-bête que d'un état de l'art.

7.1 Les logiques terminologiques

L'annexe B fait un certain nombre de rappels sur les logiques terminologiques. Ce paragraphe les compare au cadre général des SC. Le problème est d'abord de déterminer ce qu'est $\mathcal{X}$ (et $\preceq$) dans les LT, ce qui peut inclure :

1. l'ensemble des concepts atomiques, c'est-à-dire nommés, explicitement introduits : $\prec$ est alors la relation explicite entre concepts atomiques dans leurs définitions ;

2. l'ensemble des termes utilisés pour définir les restrictions de rôles ;
3. l'ensemble des termes explicitement utilisés dans les définitions, y compris les sous-termes : si (`and` $c_1 ... c_n$) est un concept de $\mathcal{X}$, alors les $c_1, ..., c_n$ aussi ;
4. le quotient de ces ensembles par l'équivalence de termes, c'est-à-dire par les circuits de $\trianglelefteq$;
5. la fermeture (par union des intensions ou conjonction des concepts) de cet ensemble, pour en faire un demi-treillis inférieur.

Quel que soit le choix exact de $\mathcal{X}$, $\prec$ est la relation qui lie chaque concept aux concepts atomiques dont il est une conjonction explicite :

$$\prec \;\; = \{(\alpha\ \beta) \mid \alpha \in \mathcal{X}, \beta \in \mathcal{X}_a, \alpha \doteq (\texttt{and}\ ...\beta...)\}.$$

où $\mathcal{X}_a$ représente l'ensemble de ces concepts atomiques, et en supposant les termes suffisamment normalisés (pas de `and` imbriqués).

Notons qu'une LT se définit par un ensemble de *constructeurs*, dont chacun permet l'introduction d'un concept *défini*, donc la *définition* au sens d'une CNS. À ces constructeurs s'ajoutent quelques *opérateurs* comme $\doteq$ ou `disjoint`, dont le rôle consiste soit à introduire un concept, soit à poser une contrainte sur un (ou des) concept(s). Les LT ont deux façons d'introduire le caractère primitif d'une classe : pour Nebel ou $\mathcal{ALC}$, cela se fait par un opérateur explicite, $\dot{\leq}$ au lieu de $\doteq$, alors que pour CLASSIC, il s'agit d'un *constructeur* `primitive` ou `disjoint-primitive` : ce sont alors les seuls constructeurs qui n'introduisent pas un concept défini.

(1) constitue la base, et on a vu que (2) était nécessaire pour offrir un minimum de généralité à la définition des domaines. (3) paraît inutile. En ce qui concerne le point (2), il est explicite dans l'algorithme CLASSIFY de Nebel, qui commence par créer des concepts anonymes pour toutes les restrictions de valeurs des rôles du concept à classifier, avant de les classifier récursivement. Le point (4) est lui aussi explicite dans ce même algorithme. Noter que (3), c'est à peu près $\mathcal{ATTR}$, et le produit de la fonction CUNFOLD de [Nebel, 1990], et (3+5) correspond à $\mathcal{INT}$.

En revanche, Nebel paraît faire le choix implicite de (5) en qualifiant la hiérarchie de demi-treillis. Il est plus raisonnable de penser que ce n'est qu'un demi-treillis potentiel, car le demi-treillis effectif n'a pas d'interprétation satisfaisante car il ne distingue pas les concepts pas encore définis des concepts inconsistants, c'est-à-dire de $\perp$: ce treillis actuel n'est en fait ni intensionnel, ni extensionnel. Dans les logiques terminologiques, tout se passe comme s'il y avait un treillis classificatoire des classes : c'est le treillis des idéaux dont nous avons parlé au paragraphe 3.2.1. Mais ce treillis reste virtuel.

7.1.1 Terminologie de second ordre et explication des attributs

En fait, dans sa version la plus simple, la composante intensionnelle des LT ne contient aucune définition, mais uniquement un ensemble de concepts et de rôles, atomiques aussi bien du point de vue syntaxique (ce sont des noms) que du point de vue sémantique (ils sont primitifs : leur interprétation est irréductible à celle de toute autre entité). Une *terminologie*, c'est-à-dire un ensemble de concepts définis à partir de ces notions primitives, n'est au premier ordre que du sucre syntaxique. Remplacer un concept défini non circulaire par sa description ne change rien du point de vue sémantique, bien que ce soit une expansion coûteuse, éventuellement exponentielle [Nebel, 1990], dont il vaut mieux se passer [Paramavisam et Plaisted, 1996]. Cependant, introduire un concept défini a une signification au second ordre : « ce concept m'intéresse » [Nebel, 1990]. En particulier, le résultat de la classification d'une instance doit retourner un ensemble de ces classes explicitement introduites, et non pas n'importe quelle antichaîne du treillis virtuel.

Pour faire de la classification incertaine dans les LT disposant du constructeur `not`, il faudrait ainsi faire de la négation de toute classe « de première classe » une classe « de première classe ».

Quelle que soit l'interprétation exacte que l'on fait de l'espace des attributs, les LT se signalent par une « grande quantité » de modèles : pour une description du monde contenant seulement $(P\ o)$, on retrouvera tous les modèles dans lesquels o n'est qu'instance α_P, c'est-à-dire où la présence de P dans l'objet ne s'explique par aucune des classes de l'objet. Une limitation envisageable serait de se restreindre aux modèles dans lesquels tous les attributs sont expliqués par une de ces classes « de première classe ». Cela reviendrait à donner un sens, pour les LT aussi, à l'*hypothèse de l'explication des attributs* faite au paragraphe 4.3.3. Mais dans ce cas, la sémantique du SC n'est plus indépendante de la terminologie [1]. Ce changement sémantique pourrait faire des LT, de simples langages d'attributs qu'ils sont actuellement, de véritables langages de classes dans lesquels, comme dans les systèmes d'objets, les classes (explicitement introduites) déterminent les seuls objets susceptibles d'exister.

En ce sens, les classes des LT ont une sémantique encore moins « ontologique » que celles de TROPES, l'ontologie étant cantonnée dans les concepts et rôles primitifs qui suffisent à déterminer la sémantique.

7.1.2 *ABox* et *TBox* : séparation, extension conservative

En ce sens, et malgré la séparation de principe entre *ABox* et *TBox*, une assertion concernant une instance peut entraîner la définition implicite de nouveaux concepts dans la terminologie. Mais ces nouveaux concepts ne modifient pas le treillis virtuel, ce qui garantit que la *ABox* reste bien une *extension conservative* de la *TBox*.

Il y a aussi univocité, mais ni exclusivité, ni exhaustivité, sauf dans des cas dégénérés. En ce qui concerne l'exhaustivité, il n'y a pas moyen de l'assurer localement : dans la terminologie de Nebel, il n'est pas possible de dire qu'un `Humain` est forcément un

1. C'est une évidence au niveau extensionnel. En revanche, cette modification peut très bien rester complètement conservative au niveau intensionnel : *à regarder de près*. Cette modification de sémantique a-t-elle déjà été envisagée ? Mystère. [Nebel, 1990] discute au paragraphe 3.2.6 de la signification de l'introduction des classes, avec une certaine confusion entre la sémantique elle-même et la présentation des résultats à l'utilisateur comme le fait le raisonnement à base de classification.

Notons qu'il est possible de faire cette hypothèse sans modifier la sémantique des LT : il suffit de considérer que l'introduction des classes α_P est implicitement explicite !

`Homme` ou une `Femme`, alors que (parce que) `Homme` et `Femme` sont primitifs[2]. Il faudrait pouvoir les *définir* au moyen d'un attribut comme le `sexe`, dont les deux valeurs possibles devraient faire partie de la *TBox* et non de la *ABox*, ce qui n'est pas possible dans $\mathcal{TF}$[3].

Enfin, si ce n'est pas présent dans le modèle de Nebel, beaucoup de LT proposent dans leur(s) *ABox* des règles — par ex. calcul des prédicats restreint aux clauses de Horn et aux litéraux de la *TBox*, c'est-à-dire aux prédicats correspondant aux classes et aux rôles — qui permettent d'exprimer la relation $\lhd_a$ (mais qui, pour certains, permettent aussi beaucoup plus, puisqu'une grosse partie de la *TBox* peut s'exprimer ainsi : sous-rôle, *role-value-map*, composition de rôles, etc.).

NOTE *7.2* TBox *ou* ABox *?*

La question de savoir s'il faut mettre $\lhd_a$ dans la *TBox* ou dans la *ABox* ne paraît pas claire, ou peu pertinente : ce qui est sûr, c'est que *TBox* $+\lhd_a$ n'est manifestement pas une extension conservative de la *TBox*, du moins si $\unlhd$ tient compte de $\lhd_a$, ce qui est un mimimum sémantique que CLASSIC n'assure pas (si l'on assimile les règles à des passerelles). Le fait que *ABox* soit une extension conservative de *TBox* n'est aussi assuré que par un traitement incomplet des constructeurs comme **`fills`** et `one-of` qui permettent de définir des classes à partir d'instances.

En fait, il faut comprendre la présentation des règles de CLASSIC dans la *ABox* comme le symptôme du fait que ces règles ne s'appliquent qu'aux instances et ne sont pas prises en compte dans les déductions de la *TBox*. C'est encore une fois la même hypothèse aristotélicienne qui fait que tout réside dans les instances et que les déductions au niveau des classes sotn secondaires.

7.2 Les systèmes d'objets

Nous avons une pente à parler d'objets et à penser à des objets.
W.V. Quine, *Parler d'objets*, in [Quine, 1977]

Le cadre formel proposé ici s'applique *a priori* à tous les systèmes d'objets, du moins si l'on se restreint à leur cœur : les classes, les instances et les attributs, et donc en excluant tout ce qui est de l'ordre de la programmation, en particulier les méthodes et l'envoi de message.

Cependant, comme il l'a été noté plus haut, dans un langage de programmation par objets classique, il n'y a pas de classification d'instances : l'objet est classé par construction lors de (par) sa création, et ne peut pas changer de classe. Le pouvoir déductif de ses langages est en fait nul : ils ne s'autorisent, au mieux, que des vérifications de type.

2. Nebel cite en appendice l'opérateur `cover` : `(cover Humain Homme Femme)`.

3. Cette possibilité qui est offerte par CLASSIC avec le constructeur `one-of`, permettrait aussi de se passer de la relation d'incompatibilité ∇_a, au prix, il est vrai, d'une certaine lourdeur d'expression, surtout dans les cas n-aires. Notons que la restriction de cardinalité permet le même tour de passe-passe mais encore moins déclaratif : tous les bureaucrates ont l'habitude de définir le sexe par un code (`M/F` ou `1/2`). Si l'on ne dispose pas des types de base, on peut se servir des cardinalités, au moins au niveau intensionnel.

Nous ne décrivons ici que des langages de représentation par objets qui offrent une fonctionnalité minimale de classification d'instances. Il reste cependant à en faire une formalisation précise dans le cadre proposé ici, et à en étudier les propriétés calculatoires ou déductives, comme cela a été fait pour les LT. Dans tous ces systèmes, les classes sont purement descriptives[4], ou alors leur caractère définitionnel est purement subjectif. Cf. aussi le paragraphe 6.5.1.

7.2.1 Comparaison lt-rco

Comme il l'a été noté en introduction, la différence la plus visible réside dans le caractère définitionnel des classes des LT (à l'exception des classes primitives) opposé au caractère descriptif des classes de la RCO (à l'exception de quelques tentatives qui ne sont pas entièrement convaincantes : FROME (cf. paragraphe 7.2.4, page 169) ou LORE [Caseau, 1987; Benoit *et al.*, 1986; Caseau, 1985]). Cette différence se traduit en particulier dans le vocabulaire que nous avons adopté en ce qui concerne les éléments de syntaxe de ces langages : du côté des LT, des *constructeurs* qui *définissent* des classes, du côté de la RCO et des langages de *frames*, des *facettes* qui sont des élémemts de *description* des classes. Une conséquence assez immédiate est que les premiers ne savent pas bien exprimer les classes primitives (en témoignent les constructeurs `primitive` et `disjoint-primitive` de CLASSIC) et les seconds, encore moins, les classes définies.

Il apparaît clairement que les logiques terminologiques et les systèmes à objets diffèrent sur deux points, où les premières choisissent l'*ouverture* et les seconds la *fermeture* :

- Comme on l'a vu, pour les LT, l'espace des attributs est illimité : en particulier, tout attribut peut être affirmé pour tout objet (sans qu'il soit nécessaire d'avoir classé l'objet dans une classe introduisant l'attribut comme le réclament les systèmes d'objets). On a vu que l'on pouvait sous-entendre les classes α_P qui introduisent les attributs P, mais l'univocité majoritaire dans les systèmes d'objets est un autre élément qui limite l'espace des attributs.

- Pour les LT, étant donnée une hiérarchie, tout ensemble de classes incomparables (une antichaîne) non incompatibles (que ce soit par déclaration explicite de ∇_a ou par incompatibilité intensionnelle) forme (par conjonction) une classe valide, c'est-à-dire instanciable. En revanche, pour les systèmes d'objets univoques (donc pas pour FROME), seules sont instanciables les classes explicitement déclarées.

On pourrait ainsi introduire une notion de *fermeture* au niveau de la hiérarchie de classes, pour indiquer que le domaine modélisé est représenté en totalité. Si c'est inutile pour les LT qui peuvent rester *ouvertes*, c'est ambigu pour la plupart des systèmes d'objets, surtout s'ils sont univoques : ils sont censés être à la fois fermés (les seules instances possibles sont celles des classes existantes) mais aussi ouverts à une extension

4. Les auteurs de ces systèmes revendiquent parfois le caractère purement descriptif des classes pour s'affranchir du problème de la classification automatique des classes définies : cet argument ne tient pas car il est toujours naturel, pour un utilisateur, de vouloir introduire *aussi* des classes dont la sémantique est définie.

future. Ainsi, des classes (par exemple, sans sous-classe commune, dans un système univoque) peuvent n'être que provisoirement incompatibles, la définition d'une sous-classe commune suffisant à les rendre compatibles. En gros, les classes des systèmes d'objets décrivent les seuls objets qui peuvent exister, alors que les classes des LT décrivent un sous-ensemble des objets qui peuvent exister. Ou encore, dans les systèmes d'objets, ce sont les classes qui donnnent le droit à l'existence alors que, dans les LT, ce sont les attributs.

Ainsi, pour les LT, l'introduction d'une nouvelle classe constitue une extension conservative du SC préexistant, alors que ce n'est pas le cas dans un RCO univoque où la non introduction d'une classe est une affirmation d'incompatibilité. Il semble en effet que :

- dans les deux cas, un modèle ancien (d'avant l'extension) reste un modèle nouveau (d'après l'extension) : l'extension des nouvelles classes est vide ;
- pour les LT, un modèle nouveau est une extension d'un modèle ancien ;
- en revanche, dans un système d'objets classique, un modèle nouveau peut n'être l'extension d'aucun modèle ancien.

On peut préciser ces différences dans le cas où la nouvelle classe est définie :

- pour les LT, l'ajout d'une nouvelle classe définie ne change rien : il y a équivalence exacte des modèles ; c'est ce que constate Nebel (p. 59) lorsqu'il montre que tout « assignement primitif » s'étend en une unique fonction d'interprétation (pour une terminologie sans cycle). Les définitions n'ont aucune influence sur le monde.
- en revanche, pour les RCO, ajouter une classe définie, comme sous-classe de deux classes précédemment incompatibles (par absence de sous-classe commune), va rendre à ces deux classes leur compatibilité.

Le comportement des LT correspond clairement à ce que l'on attend d'une définition : c'est une abréviation, rien de plus. En revanche, les systèmes d'objets avec mono-instanciation (univocité) sous-entendent pour chaque classe une quantification existentielle implicite : une instance peut exister. Autrement dit, sans classe, il n'y a pas d'instance.

Un autre point est à discuter, la possibilité d'introduire de nouveaux (c'est-à-dire n'appartenant pas aux super-classes) attributs dans une classe définie. Si c'est apparemment possible avec les LT, ce n'est qu'apparence si l'on songe aux classes virtuelles α_P qui introduisent chaque attribut P. Seules ces classes α_P introduisent réellement un nouvel attribut : toutes les autres se contentent d'en réduire les valeurs, par des restrictions de domaine ou de cardinalité ou toute autre contrainte. En revanche, il faudrait envisager pour les systèmes d'objets ce que pourrait être une réelle introduction d'attribut dans une classe définie : nous n'en connaissons aucun exemple, les rares systèmes d'objets à avoir introduits des CNS se contentant de « définition par sélection » comme en LORE [Caseau, 1987; Caseau, 1985; Benoit *et al.*, 1986]. De toute façon la question se pose de savoir si une telle introduction de nouveaux attributs est possible qui ne se ramène pas, d'une manière ou d'une autre aux classes virtuelles α_P des LT.

Il faut aussi noter que, pour les objets, un modèle ancien s'étend par extension du domaine : il faut rajouter de nouveaux « objets » pour tenir compte de la nouvelle classe. En revanche, pour les LT, le modèle s'étend à domaine constant.

LT et RCO sont ainsi deux mises en œuvre des définitions de l'autoritarisme et du totalitarisme. Dans le premier, tout ce qui n'est pas interdit est autorisé, qui n'est pas contre moi est avec moi. Pour le second, tout ce qui n'est pas autorisé est interdit, qui n'est pas avec moi est contre moi.

On peut ainsi considérer que les LT réfutent absolument l'*hypothèse du monde clos* qui est à la base de la programmation logique, des modèles habituels des bases de données, mais aussi de modèles comme F-LOGIC (cf. paragraphe 7.3.2, page 171), et implicitement de la plupart des modèles des systèmes d'objets. La RCO (ou plus généralement tous les systèmes d'objets qui font de la classification d'instances) pratiquent une forme restreinte d'*hypothèse du monde ouvert* : une instance y peut en général devenir instance d'une sous-classe de sa classe, guère plus. En particulier, dans un système univoque comme le sont les systèmes d'objets classiques, elle ne peut pas devenir une instance d'une classe sans rapport (c'est-à-dire sans sous-classe commune) avec la classe courante.

Les notes 12 du chapitre 2 et 15 du chapitre 4 ont bien montré que la relation d'incompatibilité est un moyen explicite de fermeture, pour les SC trop ouverts, dont l'effet consiste à éliminer certains modèles : cela revient en fait à exclure du modèle[5] certains objets du monde réel, les `escargots`, les `zombies` ou les `camping-cars`. Cette opposition entre LT et RCO peut donc se réduire par fermeture explicite des classes des LT, c'est-à-dire définition de toutes les combinaisons de sous-classes compatibles, ou bien, en sens inverse, par explicitation de toutes les incompatibilités.

À tout cela s'ajoute bien entendu le retard de la RCO en matière de logique, c'est-à-dire de sémantique et de déduction, retard que l'on cherche ici à combler, mais dont il n'est pas sûr que le comblement ne réduirait pas, définitivement, la différence entre les deux approches. Resterait alors simplement à essayer de faire profiter les LT des spécificités de la RCO, comme classification incertaine ou points de vue.

7.2.2 Shirka

SHIRKA [Rechenmann, 1988; Euzenat et Rechenmann, 1995] est un système de classification[6] qui produit 3 modalités de classification dont la plus forte est malencontreusement appelée *sûre* et non *probable*, ce qui donne l'impression qu'il s'agit d'une

5. Dans cette phrase, les deux occurrences du mot modèle n'ont pas le même sens (cf. note 4.9, page 63).

6. On croit souvent SHIRKA en héritage simple, avec une hiérarchie arborescente, donc univoque. C'est une erreur due, il est vrai, à la répugnance de l'auteur (de SHIRKA) vis-à-vis de l'héritage multiple.

classification automatique sur des classes définies[7], ce qui n'est pas le cas, ou alors il s'agit d'une propriété *subjective*. [Euzenat et Rechenmann, 1995] propose maintenant le triplet *possible*, *inconnu* et *impossible* qui est plus satisfaisant.

SHIRKA a certainement eu une influence importante, même si en fait restreinte à un petit cercle, par la volonté têtue de son auteur, Rechenmann, 1° de tout faire en style objet (de ne pas faire appel à d'autres styles[8]), 2° de mettre en exergue les fondements de la représentation des connaissances (indépendance à la fois des traitements ou problèmes et des autres objets : pas de *relation* qui ne soit de composition), 3° par l'introduction de la classification incertaine, pour pallier le caractère primitif des classes et l'incomplétude des objets.

Cependant, la fonctionnalité de classification est greffée de façon assez extérieure au système, ce qui l'empêche de faire automatiquement les deux seules inférences valides qu'il pourrait faire : [*type*-infér] et [$\exists$-univoc].

7.2.3 Tropes

TROPES [Mariño *et al.*, 1990; Mariño, 1993; Euzenat et Rechenmann, 1995; Projet Sherpa, 1995] ajoute à SHIRKA une représentation par *points de vue*. Chaque point de vue est une hiérarchie de classes, en héritage simple, (donc un SC univoque à lui tout seul) et les points de vue sont reliés par des *passerelles*. Chaque objet est instance d'exactement une classe par point de vue (*multi-instanciation*) Dans chaque point de vue la classification est similaire à celle de SHIRKA (dans [Mariño, 1993]), et [Euzenat, 1993a] lui refuse à juste titre le droit à la classification automatique. On peut considérer que TROPES constitue

- un seul système classificatoire, avec une hiérarchie unique, réunion des hiérarchies des différents points de vue ;
- un système classificatoire par point de vue ;
- un produit de systèmes classificatoires, comme le fait [Euzenat, 1994] : c'est le correspondant exact, compte tenu de la contrainte de multi-instanciation : une classe par point de vue.
- mais l'interprétation la plus satisfaisante est la suivante : chaque point de vue est un SC univoque, voire exclusif, dont l'incompatibilité est donnée de façon implicite et systématique (classes sans sous-classes communes ou incomparables suivant la propriété choisie) ; le SC global est donné par le treillis des idéaux de la réunion des points de vue, dont on enlève tous les idéaux contenant un ensemble de classes incompatibles (cf. paragraphe 3.2.1, page 41)[9].

7. Ce qui sous-entend que la modalité s'applique à la relation entre l'instance et la classe. En revanche, si on la comprend comme qualifiant l'existence d'une solution (d'un modèle) dans laquelle l'instance est rattachée à la classe, cette critique tombe.

8. Même si des tentatives avec les règles ont été faites [Rechenmann et Vignard, 1985; Euzenat et Rechenmann, 1995].

9. En fait, l'application de ce treillis des idéaux faite par [Medina, 1995] — analyse d'exécution distribuées — ressemble étonnamment à la représentation multi-points de vue de TROPES. Dans les

Les propriétés en seraient forcément différentes. Notons que considéré comme un produit de SC, TROPES est en fait univoque, ce qui relativise sa multi-instanciation. Cependant, sous réserve d'un inventaire plus précis et sans étonnement excessif puisque TROPES a été notre source d'inspiration principale avec les LT, il semble que c'est un exemple parfait de la sémantique générale que nous avons proposée, avec distinction entre $\prec$ (la relation dans les hiérarchies) et $\lhd_a$ (les passerelles), la distinction entre attributs nécessaires et suffisants y étant masquée par l'uniformité des attributs sur un *concept* au sens de TROPES.

Principal point supplémentaire non considéré ici, la structuration par point de vue et les propriétés structurelles qui en découlent.

Par ailleurs, la distinction entre *concept* et *classe* permet à [Euzenat, 1993b] de prétendre distinguer ontologie et taxinomie, ce qui n'est pas d'une limpidité totale. La seule chose de sûre est que les concepts de TROPES permettent de circonscrire à moindre frais un *espace de classification* tout en autorisant un ensemble d'attributs uniforme. Le tout semble assez *ad hoc*. Pourtant, la comparaison avec les LT est très instructive, sur ces deux points de l'espace des attributs (cf. paragraphe 4.3.3, page 82) et de l'ontologie et de la taxinomie (cf. paragraphe 7.1.1, page 162). La similitude des solutions — espace global des attributs, même si partitionné en concept pour TROPES, et sémantique non ontologique des classes — s'expliqurait de diverses façons. Notons que les classes de TROPES sont quand même plus ontologiques que ne le prétend Euzenat : comme elles sont primitives, elles participent à la sémantique du SC.

Pour faire de TROPES un système à classes définies objectif, il faudrait faire de la classification automatique des classes dans chaque point de vue — ce qui imposerait le passage à l'héritage multiple —, mais aussi entre points de vue, pour construire automatiquement certaines passerelles.

7.2.4 Frome

FROME [Dekker, 1993; Dekker, 1994] est un système de classification en héritage multiple, non univoque, qui produit 3 modalités de classification comme SHIRKA. Il est par ailleurs possible de déclarer qu'un attribut est nécessaire à la classification dans une classe donnée, — sous deux modes différents : attribut nécessairement présent, ou nécessairement valué — ce qui donne un caractère définitionnel partiel aux classes (bien que l'interprétation exacte de cette fonctionnalité ne soit pas absolument claire, au moins dans mon esprit). Traitement des homonymies d'attributs (conflits de nom de [Ducournau *et al.*, 1995]). Propriétés subjectives. Regarder de plus près les rapports entre les ensembles Sur et Pos ou $Prob$.

FROME est une extension du langage d'objets ROME, dont l'originalité repose sur une notion de *point de vue* et de *multi-instanciation* [Carré, 1989; Carré *et al.*, 1990; Carré et Dekker, 1991; Dekker et Carré, 1992]. FROME est sans doute l'un des rares systèmes de RCO non univoque.

deux cas, on a des ordres (partiels ou totaux) disjoints, dont on peut imaginer une source commune, comme en TROPES, avec des « passerelles » qui ont *grosso modo* la même sémantique que les relations d'ordre (subsomption dans un cas, succession temporelle dans l'autre).

7.3 SC et programmation logique

Il y a quelque chose de plus important que la logique, c'est l'imagination.
Alfred Hitchcock[10]

Dans les exemples qui suivent, il s'agit de programmation logique dont les termes sont, d'une manière ou d'une autre, des « objets » : la sémantique est de type PROLOG, à la Herbrand, et c'est la programmation logique qui assure la déduction.

Si ce cadre est commun, on peut y distinguer au moins deux tendances. Dans l'une, les termes sont structurés de façon complexe et récursive, comme des classes des SC : c'est le cas des ψ-termes et des structures attributives typées.

Dans l'autre approche, la structure des prédicats est réduite aux tuples de la description du monde : c'est le cas de F-LOGIC et GULOG. Cette seconde approche ressemble beaucoup à ce qui s'est fait dans le domaine des *frames* pour les intégrer à des langages de règles, la rigueur sémantique apparemment en plus. On remarquera que lorsque les prédicats PROLOG sont utilisés comme des attributs, l'indéterminisme de PROLOG conduit naturellement à les considérer comme multi-valués, ce qui rend pose quelques difficultés si l'on désire représenter des attributs mono-valués (cf. GULOG).

NOTE *7.3*

Un travail important reste à faire vis-à-vis de toutes ces approches logiques, dont les rapports avec les SC d'une part, entre elles d'autre part, ne sont pas claires.

On les distinguera clairement des SC implémentés en PROLOG, comme certains LT (BACK [Nebel, 1990]) ou langages de *frames* (OBJLOG [Dugerdil, 1988] ou OBJLOG+ [Faucher, 1991; Gonzalez-Gomez, 1996]), pour lesquels la programmation logique n'a qu'un rôle d'implémentation sans aucune influence sur la sémantique.

7.3.1 Les ψ-termes

LOGIN [Aït-Kaci et Nasr, 1986] et LIFE [Aït-Kaci, 1991a; Aït-Kaci, 1991b; Aït-Kaci et Podelski, 1993] sont deux langages de programmation logique dans lesquels les termes, c'est-à-dire les arguments des prédicats, sont des ψ-termes.

Beaucoup reste à faire pour les faire rentrer dans le moule des SC. En particulier, font-ils réellement de la classification d'instances ? L'une des différences évidentes est le caractère mono-valué des attributs, bien qu'il n'ait rien à voir avec les considérations précéentes sur le caractère multi-valué des prédicats en PROLOG puisque les attributs des ψ-termes ne sont pas des prédicats.

Une autre caractéristique évidente est la présence de variables logiques traduisant la nécessaire unification (la coréférence) de sous-termes, de façon tout à fait similaire au constructeur `same-as` de CLASSIC.

Pour ce qui est des classes, outre l'éventuelle introduction explicite de classes, on peut considérer que chaque ψ-terme apparaissant dans une clause constitue l'introduc-

10. Cité par le *tee-shirt* d'Ehoud.

tion d'une classe, la classe de tous les termes[11] qui s'unifient avec lui. Le processus de classification est alors consubstantiel à l'unification.

La question se pose aussi de savoir si l'ensemble des classes, au moins primitives, est ouvert ou fermé, au sens du paragraphe 7.2.1.

Les « structures attributives typées » (*typed feature structure*) de [Carpenter, 1992] sont au confluent de la théorie des types, des logiques terminologiques, des ψ-termes et de la programmation logique : bien que dédiées principalement au traitement du langage naturel, elles demandent à être explorées. Une structure attributive typée est très proche d'un ψ-terme : c'est un graphe orienté étiqueté — les sommets par des « types », les arcs par des attributs (*feature*) — avec une « racine » d'où tous les sommets sont atteignables.

7.3.2 F-logic

[Kifer *et al.*, 1993] propose une logique des systèmes d'objets[12]. C'est une logique *directe* — dans le sens où, comme celle que nous proposons, elle ne passe pas par une traduction dans un autre formalisme — mais sur le modèle de celle de PROLOG. F-LOGIC prend en compte un certain nombre d'éléments non considérés ici : relations n-aires (méthodes ou attributs avec arguments), masquage, réflexivité (les classes sont des éléments du domaine, avec une relation d'appartenance non nécessairement bien fondée, qui permet aussi de représenter des prototypes). En revanche, il n'est pas question du tout de classification d'instances, et les LT sont absolument absentes de la bibliographie, alors qu'il s'agit pourtant de démarches très comparables.

7.3.3 Gulog

[Dobbie et Topor, 1995] propose un langage similaire, GULOG, qui est qualifié de *deductive object-oriented system*, ce qui conviendrait très bien à F-LOGIC. Il est remarquable que cet intitulé s'applique aussi bien aux LT alors qu'aucun des auteurs — qui proviennent tous des bases de données et de la programmation logique — ne semble les connaître.

Dans GULOG, la problématique de la classification est carrément absente : la relation entre une instance et sa classe fait partie de l'environnement de typage d'un programme, et ne figure pas explicitement dans les clauses : il est donc a fortiori impossible de le déduire. D'ailleurs, les attributs — en fait généralisés par des méthodes — ont bien un domaine, mais il n'est pas spécialisable dans les sous-classes.

Notons que GULOG introduit des contraintes (trop) fortes sur l'unifiabilité des clauses concluant sur un attribut mono-valué pour assurer qu'il l'est, ce qui remédie au caractère implicitement multi-valué des prédicats de PROLOG.

11. Cette phrase utilise le mot terme dans deux sens différents : dans le premier, il s'agit d'une occurrence physique d'un terme dans le langage, c'est un terme générique, une classe. Dans le second, c'est la valeur individuelle, ou structure, qui s'unifie avec le premier, une instance.

12. Le titre est un peu trompeur : page 62, F-LOGIC est plus justement qualifié de *object-oriented logic programming system*.

En fait, un langage comme GULOG est destiné à faire de la programmation par objets, fortement typée, mais cette programmation est logique, au lieu d'être impérative ou fonctionnelle, et sa sémantique est assurée par les moyens habituels de la programmation logique.

7.4 Les systèmes classificatoires de l'analyse de données

Dans son activité de classification automatique, l'analyse de données produit des systèmes classificatoires. Quels sont-ils?

7.4.1 Les treillis de concepts

L'approche des treillis de concepts [Wille, 1982; Wille, 1992; Godin *et al.*, 1995] s'intéresse à la construction[13] du treillis de Galois (cf. paragraphe 3.3, page 44) de la relation *avoir un attribut.*

7.4.2 Les arbres de décision

Il s'agit clairement d'une version dégénérée des systèmes classificatoires.

7.5 Classification de graphes étiquetés

Le modèle objet se généralise assez naturellement à un modèle de graphe étiqueté: en fait, le modèle des SC présenté ici est à base d'arbres étiquetés, dont les sommets et les arcs sont respectivement des classes (ou des objets) et des attributs. On peut donc l'étendre, d'abord à des graphes sans circuit (c'est le cas des ψ-termes ou des LT avec `same-as`), ensuite à des graphes plus ou moins quelconques. Un objet est alors constitué d'un graphe dont les sommets et les arcs (ou arêtes) sont typés. Dans les deux cas que nous présentons brièvement ci-dessous, la subsomption structurelle $\trianglerighteq$ se ramène à un morphisme de graphe étiqueté, respectant un ordre partiel sur les étiquettes.

Notons aussi que [Borgida et Patel-Schneider, 1994] passe par un modèle de graphe étiqueté pour fonder la sémantique de CLASSIC et prouver son algorithme de subsomption.

Les structures attributives typées de [Carpenter, 1992] (cf. paragraphe 7.3, page 170) sont elles aussi présentées en termes de graphes étiquetés et de morphisme, mais le morphisme inverse la relation de sous-typage sur les sommets, et les étiquettes des arcs ne sont pas hiérarchisées.

13. Il s'agit donc de classification au sens de l'analyse de données.

7.5.1 Les molécules en chimie

Dans ce cas, les classes sont des structures moléculaires, c'est-à-dire des parties de molécules (sous-graphes partiels), dont l'extension est constituée de toutes les molécules qui contiennent la structure moléculaire [Napoli *et al.*, 1994]. Il s'agit de 1-graphes non orientés, et le morphisme est injectif, ce qui fait de $\trianglerighteq$ un ordre, identique en fait à $\trianglelefteq$, et qui peut être calculé automatiquement, sans qu'il soit besoin d'une relation explicite $\prec$, que l'on peut ainsi considérer comme vide : $\prec\ = \emptyset$, c'est bien un ordre strict, particulièrement partiel.

Cependant, comme il l'a déjà été noté plus haut, l'intension de ces classes est absolument irréductible à un ensemble d'attributs : elle réside uniquement dans la structure elle-même, le rôle des étiquettes de relations étant réduit.

Pour finir, soulignons que ce système classificatoire de molécules n'a rien de gratuit : il sert d'une part à hériter des attributs non suffisants — en particulier les transformations qui peuvent être appliquées à la molécule, suivant un héritage cumulatif non monotone [Ducournau *et al.*, 1995] —, il est d'autre part utilisé dans une application de synthèse en chimie organique [Quinqueton *et al.*, 1992; Vismara, 1995; Vismara, 1996].

7.5.2 Les graphes conceptuels (gc)

Les GC [Sowa, 1984; Sowa, 1991; Mugnier et Chein, 1992; Leclère, 1994; Leclère, 1996; Leclère, 1995] sont des multi-graphes bipartis : le morphisme n'est pas nécessairement injectif, et $\trianglerighteq$ n'est qu'un pré-ordre. Par ailleurs, la classification dans les GC peut présenter la même complexité que dans le cas général présenté ici : assertion de $\prec$, calcul de $\trianglerighteq$ et possible assertion supplémentaire de $\lhd_a$.

En fait, lorsque les graphes conceptuels sont des arbres et que les relations sont binaires, ils rentrent parfaitement dans le cadre proposé ici, car les étiquettes des relations peuvent être assimilés aux attributs. En revanche, lorsqu'ils ont des circuits, il offrent la même dépendance des attributs que le constructeur `same-as` des LT. Il faut enfin noter, que les GC (ou au moins certains de leurs thuriféraires), comme les langages de programmation par objets, se refusent (pour des raisons mal identifiées que l'on peut penser provisoires) à faire du classement d'instance. Lorsqu'ils sont pris en toute généralité, s'ils ne rentrent pas dans le cadre des SC, ils en présentent, en particulier, toute la problématique.

NOTE *7.4*

Les tenants des GC tiennent à souligner qu'ils (les GC) ne se réduisent pas à un SC, les graphes permettant autant d'exprimer ce système classificatoire que des « faits » s'appliquant aux objets du SC, la sémantique (intuitive) de ces faits étant spécifique à l'application. Dans l'interprétation la plus laxiste, chaque graphe « fait » agit comme un contexte vis-à-vis de sa propre interprétation logique : interpréter deux graphes fait comme la conjonction de leurs interprétations, ou comme l'interprétation du joint des deux graphes, n'est pas possible dans toutes les applications. Cela correspond à la remarque de Nebel à propos des concepts des LT dont l'introduction s'assimile à une notion de second ordre : « ce concept-là m'intéresse »[14]. Contrairement aux LT où ce niveau de second ordre s'ajoute simplement à la sémantique du premier ordre, il y a ici un niveau qui peut échapper à la sémantique logique. Dans l'interprétation la plus stricte des GC, la base de graphes a une sémantique de conjonction qui assure une sémantique générale et correcte. En revanche, dans les interprétations plus *ad hoc*, c'est une brèche par où s'échappe toute la sémantique des GC.

7.6 Systèmes classificatoires *ad hoc*

Historiquement, de nombreux SC ont été conçus de façon *ad hoc*, souvent en utilisant des techniques à base d'objets et de règles comme [Fikes et Kehler, 1985]. La correction de ces systèmes est bien sûr complètement *subjective*.

Des exemples typiques sont donnés par les systèmes DIVA de diagnostic d'anomalies vibratoires sur des machines tournantes [David et Krivine, 1987; David et Krivine, 1988] ou CLASSIC[15] de reconnaissance de galaxies [Granger et Thonnat, 1985; Granger, 1986; Granger, 1988; Granger, 1985]. Notons que ces deux systèmes pratiquent de la classification floue, mais qu'à chaque étape du cycle de classification, ils concluent de façon certaine sur l'appartenance de l'objet à une classe.

7.7 Le cas des prototypes

Aimez les choses à double sens, mais assurez-vous qu'elles aient un sens.
Sacha Guitry

Bien que nous les ayons explicitement exclus du champ des SC, il est tentant d'essayer de voir quelle sémantique dénotationnelle peut s'appliquer aux systèmes à prototypes[16]. Ces systèmes sont représentés aussi bien dans le champ des langages de programmation avec SELF [Chambers et Ungar, 1992] ou GARNET [Myers *et al.*, 1992], même si ce dernier revendique abusivement une optique de représentation des connaissances et une programmation déclarative.

Il semble d'abord qu'il faille voir dans les prototypes des *objets concrets*, ce que nous interpréterons par l'existence d'un domaine et d'une fonction d'interprétation, éventuellement injective, qui associe à chaque prototype un élément du domaine, ainsi que par le fait que chaque prototype est donné par un ensemble de couples attribut-*valeur*, comme un objet, et non pas de couples attribut-*domaine*, comme une classe. La relation de spécialisation $\prec$ est assurée par la *délégation*.

14. Deux exemples de contexte permettent de mieux comprendre ce problème : dans un cas, tous les faits décrivent un même état du monde : leur contexte est toujours le même, et deux graphes s'interprètent bien par la conjonction de leurs interprétations. En revanche, si chaque graphe décrit le contenu d'un livre dans une bibliothèque, tous les faits ont un contexte implicite, « il y a un livre qui dit que », qui interdit toute conjonction.

15. Rien avoir avec la LT du même nom.

16. On soulignera l'analogie entre le terme de « prototype » et celui de « proto-ensemble » utilisé au paragraphe 2.1 pour caractériser la notion de classe en théorie des ensembles.

7.7.1 Délégation et affectation

La sémantique des prototypes dépend d'abord des rapports entre la délégation et l'affectation de valeur à un attribut [17].

La sémantique usuelle du masquage s'applique à la délégation. Considérons le cas $\alpha \prec \beta$, où α, le *déléguant*, délègue à β, le *délégué*. Dans tous les cas lorsque qu'est modifié un attribut de β que ne possède pas α, ce dernier est indirectement modifié.

Inversement, lorsque c'est un attribut de α, qu'il ne possède toujours pas, qui est modifié, deux sémantiques sont possibles :

- le délégué n'est pas modifié et l'attribut est rajouté, en propre, au déléguant. C'est le cas de YAFOOL [Ducournau, 1991] ;
- le délégué est modifié lui aussi, le déléguant n'étant modifié qu'indirectement. C'est le cas de SELF et de diverses études issues de ce modèle [Mulet et Marco, 1994; Mulet, 1995; Bardou et Dony, 1995; Malenfant, 1996].

Il y a enfin une alternative à la délégation, l'inclusion (*embedding*) : les attributs de β sont copiés dans α, ce qui assure une complète indépendance des deux prototypes [Abadi et Cardelli, 1996].

7.7.2 Première interprétation

Le premier cas entraîne une première interprétation : α et β ne peuvent pas représenter la même chose, puisque l'un peut être modifié sans que le second le soit. La délégation utilise simplement le délégué comme réservoir de valeurs par défaut : il s'agit d'une copie paresseuse.

Une interprétation pourrait consister à considérer chaque prototype à la fois comme une classe et comme son unique instance. Les prototypes rentrent alors dans le cadre proposé ici, la non monotonie en plus, bien sûr sans classification d'instance (chaque instance est soudée à sa classe), mais avec une éventuelle classification de classes. On peut aussi y retrouver, à la limite, l'interprétation par des ensembles non fondés de [Kifer *et al.*, 1993] (cf. paragraphe 7.3.2, page 171).

7.7.3 Seconde interprétation

L'interprétation alternative est la suivante : α et β représentent la même chose, puisque l'on ne peut pas modifier l'un sans modifier l'autre. En particulier, la fonction d'interprétation n'est plus injective.

On en déduit que tous les déléguants au même délégué représentent la même chose : en fait, une composante connexe du graphe de délégation est (incluse dans) une classe d'équivalence de l'interprétation. Il est alors tentant de considérer, comme le font [Bardou et Dony, 1995; Malenfant, 1996] la composante connexe comme le véritable objet dont chaque prototype n'est qu'un morceau.

17. Ce qui ne signifie pas que l'on se place maintenant dans un cadre non monotone ou évolutif : nous considérons la modification dans ce qu'elle révèle de la sémantique statique sous-jacente.

Il y a néanmoins quelques difficultés : si $\alpha = [\mathtt{x} = \mathtt{0}]$, alors chaque $\beta_i = [\mathtt{y} = \mathtt{y_i}]$, avec $\beta_i \prec \alpha$, devrait s'interpréter comme un `point`, mais comme le même point, ce qui est assez troublant. Faut-il alors considérer la composante comme un ensemble de `points` ? Et que se passe-t-il alors lorsque l'on modifie l'abscisse d'un de ces points ?

Plus simplement, comment faut-il interpréter un attribut qui appartient aussi bien au déléguant qu'au délégué, avec des valeurs différentes ?

7.7.4 Les réseaux d'héritage non monotone

Les réseaux d'héritage non monotone introduits par Touretzky (1986) dans NETL constituent une version dégénérée de système classificatoire : il n'y a pas d'attributs et la différence classe-instance est assez superficielle. Comme leur nom l'indique, ces réseaux sont non monotones dans la mesure où ils s'intéressent essentiellement à essayer de trouver une interprétation de l'inconsistance ou de l'incohérence : la hiérarchie est en effet constituée de la relation de spécialisation (dénommée Isa) et de sa « négation » ($Isnota$) qui est assimilable à une relation d'incompatibilité binaire, mais cette incompatibilité n'est pas limitée aux antichaînes de la relation de spécialisation. Il est ainsi possible d'introduire une classe qui est incompatible avec l'une de ses super-classes, ou dont deux super-classes sont incompatibles, ce qui entraîne une incohérence. Mais comme une instance peut prendre la place d'une classe, une inconsistance est tout aussi possible.

Sur cette approche, voir [Horty, 1991; Thomason, 1992; Horty, 1994; Simonet, 1994].

Une autre approche de cet héritage non monotone a été proposée dans [Ducournau et Habib, 1989; Ducournau et Habib, 1991; Ducournau *et al.*, 1994].

7.7.5 Conclusion

Dominique admettait aisément une chose : qu'elle représentât un idéal.
[...] Elle platonisait à bloc.
Raymond Queneau, *Loin de Rueil*

En tout état de cause, il est difficile d'interpréter ces *objets concrets* que seraient les prototypes, autrement que comme des *objets informatiques concrets*, à comprendre comme la structure de donnée physique, et non comme sa représentation[18].

Alternativement, on pourrait dire qu'un système à prototypes n'a, par lui-même, aucune sémantique : seules ses applications en ont. Mais c'est la négation de la notion même de sémantique, c'est-à-dire de « système de signification ».

L'opposition classe-prototype ne ramène-t-elle pas, en fait, à la soit-disant querelle Platon-Aristote ? Certaines notions implémentatoires des langages de prototypes comme les *traits* de SELF révèlent un regret des classes, qui reviennent par la fenêtre quand on les chasse par la porte. À l'opposé, l'introduction de prototypes dans un

18. Pour rejoindre F. Rechenmann dans l'un de ses thèmes les plus polémiques, une classe représente peut-être son extension, mais elle représente aussi, pour ne pas dire d'abord, la structure de donnée de ses instances effectives. D'où le problème du statut des méthodes des LPO lorsque l'on recherche une intégration LPO-RCO.

langage de classes pourrait être une solution élégante et un compromis satisfaisant pour les deux partis.

NOTE *7.5* *Platon et Aristote*

Il ne serait pas superflu de demander à Platon (ou à Aristote) si son fameux *idéal* était une classe ou un prototype. Il semblerait que ce soit plutôt une classe, mais une telle exégèse n'est pas sans danger. Aristote refuse l'existence d'abstraction, mais un prototype n'en est-il pas une, lui aussi ?

Le point de vue *aristotélicien* auquel nous prétendons repose sur le fait que seules les instances comptent dans l'analyse de la sémantique ou de la déduction d'un SC donné. Les classes n'ont qu'un intérêt conceptuel d'abstraction et de factorisation. « Aristotélicien » doit donc être compris comme une étiquette commode.

7.8 Théorie des types

Une question seulement effleurée jusqu'ici est celle de la compatibilité de notre cadre formel avec celui des diverses théories des types appliquées aux objets : cette compatibilité serait nécessaire autant pour appliquer notre cadre formel aux LPO que pour envisager une intégration propre de la programmation par objets avec la représentation des connaissances par objets. Cette compatibilité est loin d'être assurée : elle bute tout de suite sur l'*invariance* du domaine des attributs qu'impose la théorie des types. La technique utilisée par [Boyland et Castagna, 1996] pour rendre sûre du point de vue des types la *covariance* dans le cas des arguments secondaires d'une méthode — quelque discutable qu'elle soit — ne semble pas pouvoir s'appliquer. Diverses solutions sont envisageables :

- [Castagna, 1996a] propose une conversion explicite qui ressemble furieusement à la classification automatique dont nous pensons qu'elle est à la base de la sémantique des SC ;
- se passer d'un typage statique, ce qui est problématique, au moins d'un point de vue théorique ;
- détecter tous les cas de risques d'erreur de type, et imposer des redéfinitions de méthodes pour éliminer tous ces risques, voire redéfinir automatiquement des méthodes dans certains cas ;
- séparer le « type » des attributs, non redéfinissable, du « domaine », covariant : les programmes pourraient alors être sûrs du point de vue des types, mais pas du point de vue des domaines.

Une autre différence réside dans l'univocité des théories des types que ces dernières doivent exprimer avec la notion de *confluence* : toutes les réductions ramènent un terme à un unique type (minimal en cas de règle de subsomption).

[Borgida, 1992b; Borgida, 1995b] compare les types des langages de programmation et les classes des LT. Il remarque bien sûr le parallèle entre vérification et coercition de type d'un côté, et classification d'instances de l'autre. Il importe dans les LT la notation de la déduction naturelle pour formaliser le système de déduction des LT et caractériser l'incomplétude de la déduction. Mais il s'occupe finalement assez peu des types dont il se contente de reprendre le formalisme logique.

La différence entre les approches sémantiques respectives des langages de programmation et des systèmes de représentation des connaissances paraît immense. Il serait intéressant de les comparer plus systématiquement.

7.9 Bases de données d'objets

Le modèle de données des bases de données d'objets peut sembler relativement éloigné du modèle de représentation que constituent les systèmes classificatoires. Pourtant, l'approche des BDO exprime très explicitement le même besoin de représentation du « monde réel ». Mais venant d'un standard de représentation très pauvre — le modèle relationnel ou le modèle entité-relation — les BDO semblent avoir adopté de façon préférentielle le point de vue sur les objets des langages de programmation par objets, comme si le diagnostic de la pauvreté de la représentation une fois porté, il devenait urgent de se plonger dans la programmation [Atkinson *et al.*, 1989; Parent et Spaccapietra, 1996].

7.10 Travaux connexes

Le présent travail n'est bien sûr pas le premier à chercher une synthèse des différentes mouvances de l'approche objet : LT, RCO, bases de données, LPO, etc., ou qui cherche à en modéliser les bases.

[Calvanese *et al.*, 1994] propose ainsi un langage de description de concepts, $\mathcal{ALUNI}$ qui prétend subsumer les approches LT, LPO, modèles de données (en l'occurrence le modèle entité-relation). Il s'agit néanmoins d'une synthèse par union, consistant à introduire dans une LT les constructeurs nécessaires à la représentation dans ce langage des éléments empruntés aux autres, et sans se soucier des différences fondamentales entre ces divers systèmes, en particulier l'ouverture-fermeture. Nous avons cherché *a contrario* à procéder par généralisation, en définissant un cadre commun permettant d'appréhender les diverses approches, sans chercher à réduire leurs différences.

D'un autre côté, [Euzenat, 1993a] est sans doute le premier à avoir introduit la notion abstraite de *système classificatoire* (SC), dans un sens encore plus général puisqu'il n'explicite pas son *critère de catégorisation*, qui correspond à l'inclusion des intensions ($\trianglerighteq$) mais qui reste abstrait, comme les intensions.

[Hautamäki, 1986] propose une structure algébrique et logique pour formaliser les *points de vue* : il passe en fait par l'intermédiaire d'une structure algébrique qui tient lieu de système classificatoire. Cette structure a la particularité de contenir à la fois les composantes intensionnelle et extensionnelles du SC, d'une façon qui rappelle assez les *prototypes*. En revanche, Hautamäki en reste à un niveau intensionnel, sans interprétation extensionnelle. De plus, Hautamäki ne se limite pas au fini, mais part d'un ensemble infini d'attributs, dont il n'impose la finitude que pour chaque classe (ou point de vue), ce qui l'amène à la notion de treillis continu [Scott, 1972].

La littérature propose plusieurs panoramas de l'espace d'états des systèmes de *frames*, incluant aussi bien RCO et LT que GC (par exemple [Karp, 1993]). Ces analyses relèvent le plus souvent du catalogue des diverses notions rencontrées plutôt que d'une synthèse raisonnée. Sur un créneau plus restreint, on trouve plusieurs comparaisons des approches objet dans le génie logiciel et l'intelligence artificielle [Nebel, 1985; Kaindl, 1994].

[Nebel, 1990] fait au paragraphe 3.4.2 une comparaison rapide avec ce qu'il appelle *attributive descriptions*.

Chapitre 8

Conclusions

> Car, pour forger, il faut un marteau, et pour avoir un marteau, il faut le fabriquer. Ce pourquoi on a besoin d'un autre marteau et d'autres outils, et pour les posséder, il faut encore d'autres instruments, et ainsi infiniment.
>
> Spinoza, *De la réforme de l'entendement*[1], 1661

CETTE TENTATIVE DE DÉFINITION ABSTRAITE des SC ramène un peu inexorablement aux LT, compte-tenu de l'équivalence entre concepts définis et primitifs qu'autorisent les composants primitifs de Nebel. Une conclusion pessimiste pourrait être : mais à quoi bon la RCO puisque les LT existent ? Le travail commencé ici n'est pas encore assez abouti pour fournir une réponse complètement optimiste à cette question. Il a cependant permis de mettre en lumière quelques point intéressants :

- la distinction parallèle, d'une part entre les deux relations de subsomption extensionnelle explicite ($\preceq$ et $\lhd_a$), et d'autre part, entre les ensembles d'attributs constituant une CNS ($\mathcal{P}_\alpha$) et les attributs seulement nécessaires ($\mathcal{P}'_\alpha$) : cette distinction explique aussi bien les règles de CLASSIC que les passerelles de TROPES.
- la problématique de la classification incertaine à la SHIRKA, dont la formalisation a montré la relative difficulté de la définition, et que les LT pourraient intégrer sans grande difficulté ;
- la non trivialité de la déduction dans un langage aussi « simple » que SHIRKA qui impose de bien distinguer les propriétés spécifiques proprement sémantiques des propriétés des descriptions du monde, ces dernières reflétant plus le point de vue de l'implémentation qui devrait être subalterne ;
- le tropisme classificatoire dans les systèmes d'objets par le biais de propriétés spécifiques comme l'univocité ou l'exhaustivité ; ce tropisme classificatoire devrait se traduire par des aides à la construction de la hiérarchie analogues à la classification des classes des LT ;
- le statut de second ordre des classes définies explicitement introduites et le fait que la sémantique des LT ne fait pas l'hypothèse de l'explication des attributs qui est en revanche implicite dans tous les systèmes d'objets.

1. Cité par [Gochet et Gribomont, 1991], page 18.

À la lumière de ces différents points, deux évolutions non convergentes se dessinent, la première entraînant les LT vers la problématique de la RCO, la seconde rendant les systèmes de RCO aussi sémantiquement sûrs que les LT. Au terme de ces évolutions, dont il reste à montrer la faisabilité, la différence entre les deux approches resterait, mais dans un cadre sémantique suffisamment unifié pour qu'une comparaison « à armes égales » soit possible.

À court terme, il faut donc voir si ce cadre de travail permet, d'une part de formaliser complètement des systèmes comme SHIRKA, TROPES ou FROME— tels qu'ils sont ou tels qu'ils devraient être —, d'autre part d'améliorer leur sémantique en montrant les limitations de leur implémentation et par une objectivation maximum de leurs propriétés spécifiques ou générales. Un premier pas a été accompli ici en posant clairement le cadre sémantique de la classification incertaine. Il reste à conclure sur chacun des SC concernés, spécifiquement. Plus généralement, la question se pose de la viabilité d'un SC dont la sémantique serait à la fois bien définie et plus stricte que celle des LT, par exemple *univoque*, *exhaustive* ou autre.

Une fois la sémantique de la RCO bien établie, la question liminaire de son intérêt et de son utilité relatives trouverait sans doute plusieurs catégories de réponses ou d'arguments positifs :

- une réponse pratique mais personnelle serait sa capacité à être intégrée à un langage de programmation par objets, au sens discuté dans [Carré *et al.*, 1995] dont la lecture montre bien que cette conclusion provisoire ne fait certainement pas l'unanimité ;

- un argument plus méthodologique que sémantique pourrait privilégier la fermeture de la RCO, opposée à l'ouverture des LT : la première permet un choix explicite des objets à représenter, alors que la seconde semble autoriser toutes les chimères par sa liberté de croisement. Le fait que l'explicitation de la relation d'incompatibilité permette d'annuler cette différence ne change rien à l'affaire. Il s'agit là d'un argument méthodologique : l'incompatibilité fait un peu penser au fameux *frame problem* de l'IA, qui oblige à énumérer explicitement tout ce qui n'a pas changé, ou n'a rien à voir ou enfin, ici, est incompatible.

- des arguments calculatoires (décidabilité, complétude, complexité) sont possibles mais peu probables.

En tout état de cause, les propriétés du SC qui peuvent être objectives doivent l'être (classification des classes définies, par exemple, et vérification de domaine pour les autres), et une aide interactive à l'utilisateur (comme celle proposée par [Leclère, 1996]) pourrait aider au maintien des autres propriétés.

Dans tous les cas, une première véritable justification de ce travail sera fournie par une formalisation complète d'un SC (autre que les LT), avec preuves de décidabilité des problèmes et de complétude et complexité des algorithmes.

Par ailleurs, il paraît intéressant de se pencher sur les rapports entre les SC et la théorie des ensembles (sur laquelle nous renvoyons à [Gochet et Gribomont, 1994], chapitres 1 et 2), en particulier en ce qui concerne deux points que nous n'avons pas du tout envisagé d'intégrer dans notre cadre bien que nous en ayons longuement parlé :

- l'interprétation des systèmes à prototypes (qui ne font pas la distinction entre classes et instances) dans la théorie axiomatique des ensembles, dont les seuls « éléments » sont des ... ensembles ;
- l'interprétation des systèmes d'objets réflexifs (avec classes et méta-classes) dans une théorie des ensembles *non fondés* — c'est-à-dire qui ne contient pas l'*axiome de fondation* qui assure plus ou moins que la relation d'appartenance est un *bon ordre* — et l'inclusion de ces systèmes réflexifs dans la problématique de la classification.

L'application et l'extension de ce cadre aux SC qui gèrent des connaissances incertaines ou approximatives est aussi souhaitable.

Annexe A

Notions fondamentales sur les ensembles ordonnés

Les ensembles ordonnés sont à la base des systèmes classificatoires et nous en rappelons les bases.

A.1 Ensembles, relations, fonctions

Le lecteur est supposé familier des bases de la théorie des ensembles.

Une relation binaire de *domaine* E et de *co-domaine* F est une partie du *produit cartésien* $E \times F$. Les relations étant des ensembles, toutes les opérations ensemblistes leur sont applicables. Étant donné une relation R de $E \times F$, la relation *duale* de R est la relation R^{d} de $F \times E$ définie par $xR^{\mathrm{d}}y \iff yRx$. On peut parler aussi de relation *inverse*.

Une *fonction totale* (ou *application*) est une relation particulière telle que chaque élément du domaine est en relation avec exactement un élément du co-domaine. Une *fonction partielle* est une relation telle que chaque élément du domaine est en relation avec au plus un élément du co-domaine : toute fonction partielle est totale sur son domaine. On utilisera les notations $\mathrm{dom}(f)$ et $\mathrm{codom}(f)$ pour désigner les domaines et co-domaines des fonctions (considérées comme totales) et des relations.

Une relation est dite *interne* lorsque ses domaine et co-domaine coïncident. Elle est alors *totale* si tout couple est en relation, soit par la relation, soit par sa duale.

Soit une fonction $f : E \longrightarrow F$, E' une partie de E et x un élément de F : on notera classiquement $f(E')$ l'ensemble des images de f pour tous les éléments de E' et $f^{-1}(x)$ l'ensemble des images inverses de x par f.

Étant donné un ensemble E, 2^E dénote l'ensemble des parties de E. L'ensemble des fonctions de E dans F, usuellement noté F^E, sera ici noté $E \longrightarrow F$, comme dans la théorie des types.

A.2 Ensembles ordonnés

Les rappels qui suivent sur les ensembles ordonnés sont tirés des introductions de [Nourine, 1993; Medina, 1995].

A.2.1 Ordres et préordres

Un *préordre* est une relation *transitive* et *réflexive* : c'est une relation interne. Une relation d'*ordre partiel* est une relation *transitive*, *réflexive* et *antisymétrique* : c'est un préordre. Une relation d'*ordre strict* est une relation *transitive*, *irréflexive* et *antisymétrique*. Une relation d'*ordre total* est une relation d'ordre pour laquelle tout couple d'éléments est comparable.

NOTE *A.1* *Notation*

Si $<$ est la notation d'une relation d'ordre stricte, on notera implicitement $\leq$ l'ordre partiel correspondant mais on ne fera aucun hypothèse générale sur l'utilisation de $>$ (ou $\geq$) pour les relation duales.

On obtient un ordre à partir d'un préordre en quotientant le domaine par la relation d'équivalence associée aux circuits du préordre. Soit $\leq$ le préordre, E son domaine : la relation R est définie par $xRy \iff x \leq y \ \& \ y \leq x$. C'est une relation d'équivalence (transitive, réflexive et symétrique). La relation quotient $\leq_R$ est définie sur l'ensemble quotient E/R par : $\hat{x} \leq_R \hat{y} \iff \exists x \in \hat{x}, y \in \hat{y} : x \leq y$.

A.2.2 Notions de base

Étant donné une relation R (sans circuit), un arc de transitivité est une paire ab, avec $a \neq b$ et aRb, telle qu'il existe c différent de a et b, tel que aRc et cRb. La *fermeture transitive* de R est la plus petite relation contenant R qui contienne tous ses arcs de transitivité : on pourra la noter R^*. Inversement, la *réduction transitive* de R — on parle aussi de graphe de couverture — est la plus grande relation contenue dans R qui ne contienne aucun arc de transitivité. Cette dernière notion n'est pas définie si la relation a des circuits, mais on peut toujours l'appliquer au graphe quotient.

Étant donné deux ensembles ordonnés $(A, \leq_A)$ et $(B, \leq_B)$, une application f : $(A, \leq_A) \longrightarrow (B, \leq_B)$ est dite *monotone* lorsque c'est un *morphisme d'ordre*, c'est-à-dire que

$$\forall a, a' \in A : a \leq_A a' \Longrightarrow f(a) \leq_B f(a')$$

que l'on exprime aussi bien par le « diagramme commutatif » suivant :

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\leq_A"'] & B \arrow[d, "\leq_B"] \\ A \arrow[r, "f"'] & B \end{tikzcd}$$

La monotonie suppose toujours deux relations d'ordre, que l'on omettra souvent lorsqu'il n'y a pas d'ambiguïté. Par abus de langage, on pourra aussi qualifier de monotone, sans préciser les relations, une fonction qui n'est monotone que relativement à la relation duale, donc de $(A, \leq_A) \longrightarrow (B, \geq_B)$, où $\geq_B$ est la relation duale de $\leq_B$.

Soit $(A, \leq)$ un ensemble ordonné : pour toute partie B de A, on définit l'ensemble des *maximaux* (resp. *minimaux*) notés $\max(B)$ (resp. $\min(B)$) par l'ensemble des éléments $x \in B$ tels que $\forall y \in B, x \leq y$ (resp. $y \leq x$) $\Longrightarrow x = y$. Lorsque ces maximaux (resp. minimaux) sont uniques, on peut parler de *maximum* (resp. *minimum*).

On définit de même un *majorant* (resp. *minorant*) de B comme un élément y de A tel que $\forall x \in B, x \leq y$ (resp. $y \leq x$). On peut bien sûr parler des *majorants minimaux* ou des *minorants maximaux* qui présentent un intérêt particulier.

Enfin, un *puits* est un sommet de A sans successeur (donc minimal), et une *source* un sommet sans prédecesseur (donc maximal). On utilise souvent les termes de *feuille* et *racine* bien qu'ils ne s'appliquent à strictement parler qu'aux arvborescences.

A.2.3 Idéaux et antichaînes

Soit $(A, \leq)$ un ensemble ordonné (ou préordonné?). Un *idéal* est une partie B de A telle que $\forall x, y \in A : x \in B \;\;\&\;\; x \leq y \Longrightarrow y \in B$. Comme pour la monotonie, on utilisera le même terme d'idéal en ce qui concerne la relation duale $\leq^{\mathrm{d}}$, alors que l'on trouve plutôt dans la littérature le nom de « filtre » : pour les distinguer, on pourra parler d'idéal supérieur ou inférieur.

Une *antichaîne* d'une relation (d'ordre) est un ensemble d'incomparables par cette relation. Soit $(A, \leq)$ une relation d'ordre, A^* ($A^*_{\leq}$ en cas d'ambiguïté) dénote l'ensemble des antichaînes de A, inclus dans 2^A, l'ensemble des parties de A.

D'abord, les idéaux et les antichaînes sont en bijection : tout idéal est caractérisé par une unique antichaîne, l'ensemble de ses minimaux, et, réciproquement, toute antichaîne engendre un idéal, l'ensemble des majorants de chacun de ses éléments. Lorsque l'idéal est engendré par un unique élément, on le dit *principal*.

En particulier, les maximaux (resp. minimaux) d'un sous-ensemble de A forment une antichaîne et toute antichaîne coïncide avec l'ensemble des maximaux (resp. minimaux) de l'idéal qu'elle engendre.

Ensuite, l'ordre $\leq$ peut être étendu aux idéaux : l'inclusion des idéaux fait de l'ensemble des idéaux un treillis distributif (voir plus loin).

En revanche, l'ordre $\leq$ peut être étendu aux antichaînes (A^*) de plusieurs façons différentes :

$$X \leq^* Y \overset{\text{def}}{\Longleftrightarrow} \begin{cases} (a) & \forall x \in X, \exists y \in Y, x \leq y \\ (b) & \forall y \in Y, \exists x \in X, x \leq y \\ (c) & (a) \;\&\; (b) \end{cases} \tag{A.1}$$

Ces trois définitions ne sont bien sûr pas équivalentes et, si la relation définie par (c) est l'intersection des deux autres, inversement (a) et (b) sont le résultat de la composition (resp. à droite et à gauche) de (c) avec l'inclusion ensembliste. Dans les trois cas, il s'agit d'ordres, mais (a) et (b), qui sont exactement duales, sont en plus des treillis. (b) est le treillis des antichaînes habituellement considéré pour lequel la bijection avec les idéaux est un isomorphisme de treillis. Ces trois variantes seront notées respectivement $\leq_a^*$, $\leq_b^*$ et $\leq_c^*$ [1].

D'une façon générale, toute fonction f à valeur dans 2^A et pour laquelle l'image de tout élément est un *idéal* supérieur (resp. inférieur) est caractérisée par l'antichaîne minimale (resp. maximale) de ces images, au sens de la relation $\leq_a^*$ (resp. $\leq_b^*$). Il suffit alors de considérer la fonction $\underline{f}$ (resp. $\overline{f}$) à valeur dans A^*, telle que $\underline{f}(x) = \min_{\leq}(f(x))$ (resp. $\overline{f}(x) = \max_{\leq}(f(x))$). Ainsi, les systèmes à objets ne s'intéressent en général qu'à $\underline{Isa}(o)$, les plus classiques en faisant un singleton, ce qui permet alors de parler de *la* classe (ou *le type*) de o. Théoriquement, il n'y a bien sûr aucune différence, mais pratiquement cela peut avoir une incidence algorithmique non négligeable.

Notons que les antichaînes maximales (resp. minimales) se comparent suivant l'ordre $\leq_a^*$ (resp. $\leq_b^*$) [2].

A.3 Treillis

La notion de treillis est la première généralisation de l'ordre total : dans un ordre total, tout couple d'éléments a et b possède un maximum et un minimum, respectivement a ou b. Dans un treillis, tout couple d'éléments a et b possède une borne inférieure $a \wedge b$, qui est un unique minorant maximal, ainsi qu'une borne supérieure [3] $a \vee b$, qui est un unique majorant minimal.

$$\forall \beta, \gamma \in \mathcal{X} : \begin{cases} \beta \wedge \gamma \leq \beta \\ \beta \wedge \gamma \leq \gamma \\ \forall \delta : \delta \leq \beta \ \& \ \delta \leq \gamma \Longrightarrow \delta \leq \beta \wedge \gamma \end{cases} \tag{A.2}$$

L'existence de bornes inférieure et supérieure implique, par définition, leur unicité [4]. Lorsque l'on applique ces opérateurs à α et β, on sous-entend implicitement l'existence de cette (ces) borne(s) mais cela n'implique en rien une structure de treillis : certains couples peuvent avoir une borne inférieure et/ou supérieure, d'autres pas.

Un *demi-treillis inférieur* (resp. *supérieur*) ne possède que la borne correspondante. Notons seulement qu'un demi-treillis inférieur avec un élément maximum (et réciproquement) est un treillis.

A.3.1 Treillis des idéaux ou des antichaînes

L'ensemble des idéaux d'un ordre, eux-mêmes ordonnés par la relation d'inclusion, constitue un treillis distributif :

$$\alpha \wedge (\beta \vee \gamma) = (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \tag{A.3}$$

1. * est une notation ambigüe, utilisée pour les fermetures transitives.
2. *Vérifier*.
3. On peut aussi parler d'*infimum* et de *supremum*.
4. Attention aux faux-amis ! en anglais, une *lower bound* est un minorant : il faut une *greatest lower bound* pour avoir la borne inférieure.

$$\alpha \vee (\beta \wedge \gamma) = (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

Cette relation d'inclusion des idéaux correspond à la version (a) de l'ordre sur les antichaînes.

A.3.2 Treillis infinis

Deux propriétés deviennent intéressantes lorsqu'un treillis est infini (ces propriétés sont vraies pour tout treillis fini) : le treillis peut être *complet* ou *continu*.

Dans un *treillis complet*, toute partie (et non pas tout couple d'éléments) possède des bornes inférieure et supérieure. Tout treillis fini est complet : dans un treillis non complet, seule une partie infinie peut ne pas avoir de borne.

Les *treillis continus* sont des treillis qui possèdent de bonnes propriétés d'approximation finie de l'infini. Ces treillis continus ont été introduits par [Scott, 1972] pour fonder la sémantique des langages de programmation.

Jusqu'à maintenant, nous n'avons pas discuté de la finitude, nous contentant d'imposer un caractère fini à $\mathcal{X}$, $\mathcal{P}$ et $\mathcal{W}$, car il semble peu probable qu'il soit nécessaire de s'intéresser à des hiérarchies infinies, même virtuellement. [Hautamäki, 1986] s'intéresse à des structures algébriques assez comparables aux systèmes classificatoires en supposant explicitement le caractère dénombrable de $\mathcal{P}$. Seuls les $\mathcal{P}_\alpha$ sont finis. Il y définit un *point de vue* comme une partie finie de $\mathcal{P}$.

On retrouve ce besoin d'infini dans [Wand, 1994] où il est justifié par des considérations de génie logiciel, chaque programmeur n'ayant accès qu'à une partie des attributs du système au développement duquel il participe. Dans les deux cas, l'ensemble des attributs est infini, mais il reste fini dans chaque classe et dans chaque objet.

A.3.3 Treillis de Galois

La notion de treillis de Galois d'une relation est à peine moins connue que celle de treillis [Nourine, 1993; Godin *et al.*, 1995]. Soit R une relation de $E \times F$ — c'est-à-dire la *correspondance* (E, R, F) : les *pavés maximaux* de R sont les couples (X, Y) tels que

$$\text{(A.4)} \qquad \begin{aligned} Y &= \{y \in F \mid \forall x \in X : xRy\} \\ X &= \{x \in E \mid \forall y \in Y : xRy\} \end{aligned}$$

L'ensemble des couples (X, Y) ordonné par la relation $(X, Y) \preceq (X', Y')$ ssi $X \subseteq X'$ (ou, de façon équivalente à cause de la maximalité, $Y \supseteq Y'$) forme un treillis appelé treillis de Galois de la relation R, noté $Gal(E, R, F)$.

Annexe B

Les logiques terminologiques

L'OBJECTIF DE CETTE ANNEXE est de faire quelques rappels sur différentes logiques terminologiques. On décrira principalement les formalismes $\mathcal{TF}$ et $\mathcal{AF}$ de [Nebel, 1990], le langage (de programmation) CLASSIC [Resnick *et al.*, 1993] et les langages (formels) de la famille $\mathcal{AL}$ [Donini *et al.*, 1994].

Ces rappels montreront deux façons différentes de parler des logiques terminologiques, la première plus proche des langages de programmation, la seconde plus mathématique.

L'annexe se termine par la définition des principaux constructeurs.

B.1 CLASSIC

CLASSIC est un membre de la communauté LT que nous avons beaucoup cité [Brachman *et al.*, 1991; Resnick *et al.*, 1993; Borgida et Patel-Schneider, 1994]. La syntaxe des descriptions est décrite dans la figure B.1. À cette syntaxe s'ajoutent tout une bibliothèque de fonctions Common Lisp, comprenant toutes les fonctions nécessaires pour l'introduction des entités du langage, en particulier :

- trois fonctions Common Lisp pour introduire des concepts définis, primitifs ou primitifs et disjoints (au sens de ∇), la dernière fonction permettant de décrire un ensemble de concepts primitifs deux à deux disjoints (donc strictement équivalent au `disjoint` de $\mathcal{TF}$, mais plus commode) ;
- une fonction pour créer des rôles primitifs, avec des possibilités de hiérarchie de rôles (sous-rôles) et de rôle inverse ;
- une fonction pour créer un individu ;
- deux fonctions pour modifier un individu, de façon monotone ou non monotone ;
- une fonction pour fermer un rôle d'un individu[1].

1. Une note sybilline du manuel annonce qu'un opérateur `close` n'est pas fourni parce qu'il rendrait la description de concepts auto-épistémique, donc non monotone.

<descr>	::=	`THING` | <HOST-descr> | <CLASSIC-descr>
<individual-name>	::=	<HOST-individual> | <CLASSIC-indiv-name>
<fn>	::=	*; une fonction Common Lisp retournant* ***T**, **?** ou **NIL***
<arg>	::=	<Common-Lisp-expr>
<HOST-descr>	::=	`HOST-THING` | <HOST-concept-name> |
		(**and** <HOST-descr>$^+$) |
		(**one-of** <HOST-indiv>*) |
		(**test-h** <fn> <arg>*) |
		(**min** <number>) |
		(**max** <number>)
<HOST-concept-name>	::=	<symbol> *; défini comme un concept « host »*
<HOST-individual>	::=	<string> | <number> | '<Common-Lisp-expr>
<CLASSIC-descr>	::=	`CLASSIC-THING` | <CLASSIC-concept-name> |
		(**and** <CLASSIC-descr>$^+$) |
		(**one-of** <CLASSIC-indiv-name>*) |
		(**test-c** <fn> <arg>*) |
		(**all** <role-name> <descr>) |
		(**at-least** <positive-int> <role-name>) |
		(**at-most** <non-negative-int> <role-name>) |
		(**fills** <role-name> <individual-name>$^+$) |
		(**same-as** (<attr-name>$^+$) (<attr-name>$^+$))
<CLASSIC-concept-name>	::=	<symbol> *; défini comme un concept « classic »*
<CLASSIC-indiv-name>	::=	<symbol> *; défini comme un individu « classic »*
<role-name>	::=	<mrole-name> | <attr-name>
<mrole-name>	::=	<symbol> *; défini comme un rôle multi-valué*
<attr-name>	::=	<symbol> *; défini comme un attribut mono-valué*

FIG. B.1 – *Syntaxe de* CLASSIC, *d'après [Resnick* et al., *1993]*

Sur la base d'un langage assez classique et proche de $\mathcal{TF}$, CLASSIC présente plusieurs traits originaux :

- la prise en compte des types de base (HOST-*concept*) ;
- la définition de concepts par des tests procéduraux (constructeurs `test-h` et `test-c`) : le second constructeur désigne une fonction de test à 3 valeurs, la valeur intermédiaire `?` n'étant pas très claire (cf. note 6.14, page 157) ;
- la distinction statique (non liée aux classes) des rôles multi-valués et des attributs mono-valués ;
- la description de classes relativement à des instances, par les constructeurs `fills` et `one-of` ;
- la description d'individus en utilisant la totalité du langage de description des classes ;
- l'identification de chemins d'attributs par le constructeur `same-as` ;

<terminology>	::=	<term-intro> | <restriction>*
<term-intro>	::=	<concept-intro> | <role-intro>
<concept-intro>	::=	<atom-concept> $\doteq$ <concept> |
		<atom-concept> $\dot{\leq}$ <concept>
<role-intro>	::=	<atom-role> $\doteq$ <role> |
		<atom-role> $\dot{\leq}$ <role>
<restriction>	::=	(**disjoint** <concept> <concept>)

<concept>	::=	<atom-concept> | **Anything** |
		(**and** <concept>$^+$) |
		(**all** <role> <concept>) |
		(**atleast** <number> <role>) |
		(**atmost** <number> <role>)
<role>	::=	<atom-role> | **anyrelation**
<atom-concept>	::=	<symbol>
<atom-role>	::=	<symbol>

<world-descr>	::=	{<obj-descr> | <rel-descr>}*
<obj-descr>	::=	(<atom-concept> <object>)
<rel-descr>	::=	(<atom-role> <object> <object>) |
		(<atom-role> <object> (**atleast** <number>)) |
		(<atom-role> <object> (**atmost** <number>))

FIG. B.2 – *Syntaxe des formalismes $\mathcal{TF}$ (en séparant description et introduction) et $\mathcal{AF}$, d'après [Nebel, 1990]*

- un rudimentaire niveau méta, puisqu'il est associé à chaque concept générique un concept individuel, instance d'une espèce de méta-classe nommée `concept` ;

- des règles dont les prémisses et conclusions s'expriment par des descriptions de concepts et qui permettent des inférences exécutées au niveau des individus seulement ; ces règles peuvent faire intervenir des fonctions Common Lisp, ce qui complique notablement par rapport aux passerelles.

L'ensemble des ces caractéristiques montre bien que CLASSIC a été conçu pour être utilisable et utilisé, avec un bon compromis entre l'élégance formelle, l'expressivité et la possibilité d'ouvertures procédurales.

B.2 $\mathcal{T}F$ et $\mathcal{A}F$

La syntaxe des formalismes $\mathcal{TF}$ et $\mathcal{AF}$ (syntaxe des descriptions et des introductions de concept) est décrite par la figure B.2 [Nebel, 1990].

Il faut rajouter à cette syntaxe formelle deux contraintes qui conduisent à « ordonner » la terminologie : un terme atomique ne peut être introduit deux fois et tous les termes atomiques à droite des connecteurs $\doteq$ ou $\dot{\leq}$ doivent avoir déjà été introduits. Notons que c'est une contrainte assez commune qui se traduit en pratique par une contrainte chronologique d'introduction, sauf pour certains systèmes comme CLOS

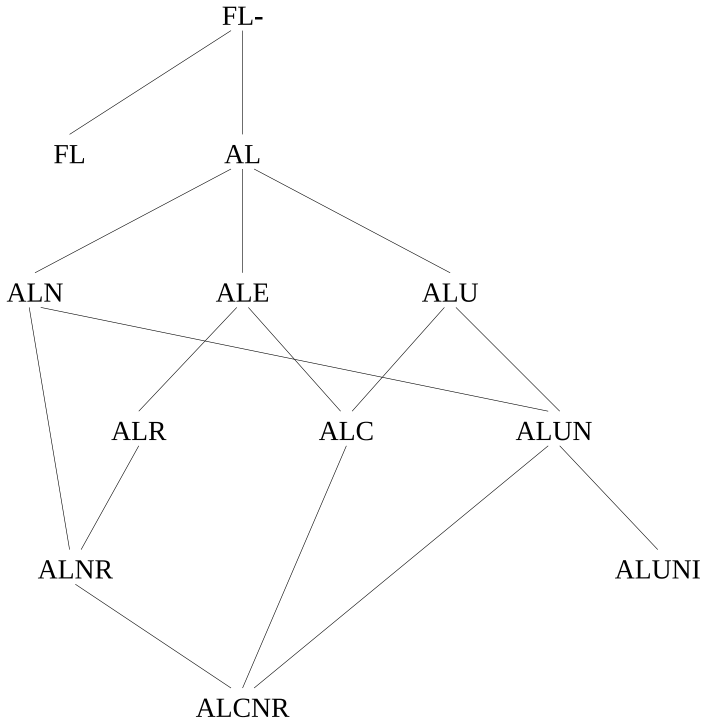

FIG. B.3 – *La famille de langages $\mathcal{AL}$*

qui permettent d'introduire une classe avant d'avoir introduit certaines de ses superclasses.

B.3 La famille $\mathcal{AL}$

La famille de langages $\mathcal{AL}$ se décline comme suit (figure B.3) :

$\mathcal{FL}$- comprend les constructeurs `and`, `all` et `some` (ce dernier désignant la quantification existentielle non qualifiée) ;

$\mathcal{FL}$ y ajoute le constructeur `range` ;

$\mathcal{AL}$ (qui signifie *attributive language*) ajoute à $\mathcal{FL}$- la négation de concept atomique primitif (constructeur `a-not`) ;

$\mathcal{ALU}$ y ajoute l'union (constructeur `or`) ;

$\mathcal{ALE}$ ajoute la quantification existentielle qualifiée (constructeur `c-some`) ;

$\mathcal{ALC}$ étend $\mathcal{ALU}$ avec la négation généralisée (*complement*) : le constructeur `not` implique alors la disjonction ;

$\mathcal{ALN}$ ajoute à $\mathcal{AL}$ la restriction de cardinalité (*number restriction*), avec les constructeurs `atleast` et `atmost` (le premier impliquant `some`) ;

$\mathcal{ALR}$ étend $\mathcal{ALE}$ avec l'intersection de rôles, ou sous-rôles, avec le constructeur `and-role`.

Toutes les combinaisons sont possibles, même si ces diverses extensions ne sont pas indépendantes [Schmidt-Schauß et Smolka, 1991; Donini *et al.*, 1994] : le langage $\mathcal{TF}$ de [Nebel, 1990] est en fait $\mathcal{ALNR}$.

B.3.1 Syntaxe de $\mathcal{AL}$

Il s'agit de la syntaxe du langage formel et non pas de la syntaxe d'un langage de programmation effectif. Elle est tirée de [Donini *et al.*, 1994; Buchheit *et al.*, 1993] : C et D désignent des concepts, Q et R des rôles et A un concept atomique (primitif).

$C, D \longrightarrow A \mid$	concept atomique (primitif)
$\top \mid$	
$\bot \mid$	
$\neg A \mid$	(négation atomique)
$\neg C \mid$	(négation généralisée)
$C \sqcap D \mid$	(conjonction)
$C \sqcup D \mid$	(disjonction)
$\forall R.C \mid$	(quantification universelle)
$\exists R \mid$	(quantification existentielle non qualifiée)
$\geq nR \mid$	(cardinalité min)
$\leq nR \mid$	(cardinalité max)
$Q \sqcap R$	(conjonction de rôles)

Le tableau suivant compare les différents langages (ce sont des analogies, pas des équivalences, au moins en ce qui concerne les SC) :

SC	$\mathcal{TF}$	$\mathcal{AL}$
$\top$	`Anything`	$\top$
$\bot$	`Nothing`	$\bot$
$\sqcap$	`and`	$C \sqcap D$
$\sqcup$	`or`	$\sqcup$
type	`all`	$\forall$
∇	`a-not`	$\neg A$
∇	`not`	$\neg C$
—	`some`	$\exists$
card	`atleast`	$\geq$
card	`atmost`	$\leq$
—	`and-role`	$Q \sqcap R$

Enfin, dans $\mathcal{AL}$ la description du monde est constituée d'assertions de la forme $C(a)$ ou $R(a, b)$ ou R et C sont des rôles et concepts quelconques, l'interprétation des instances vérifiant l'hypothèse du nom unique.

B.4 Résultats de complexité

B.5 Tous les constructeurs

NB. Les constructeurs qui suivent sont tirés, pour la plupart, de [Nebel, 1990][2], mais ils sont interprétés dans le cadre de notre formalisme, c'est-à-dire que chaque attribut est interprété par une fonction et non par une relation. Les symboles c, r et n désignent respectivement des concepts (classes), des rôles (attributs) et des nombres entiers.

Les principaux constructeurs primitifs cités plus ou moins directement dans cet ouvrage sont les suivants :

$$\begin{aligned}
\texttt{and}(c_1, ..., c_n)^{\mathcal{I}} &= \bigcap_{i=1,n} c_i^{\mathcal{I}} \\
\texttt{or}(c_1, ..., c_n)^{\mathcal{I}} &= \bigcup_{i=1,n} c_i^{\mathcal{I}} \\
\texttt{not}(c)^{\mathcal{I}} &= \Delta \backslash c^{\mathcal{I}} \\
\texttt{all}(r, c)^{\mathcal{I}} &= \{x \in \Delta \mid r^{\mathcal{I}}(x) \subseteq c^{\mathcal{I}}\} \\
\texttt{atleast}(n, r)^{\mathcal{I}} &= \{x \in \Delta \mid |r^{\mathcal{I}}(x)| \geq n\} \\
\texttt{atmost}(n, r)^{\mathcal{I}} &= \{x \in \Delta \mid |r^{\mathcal{I}}(x)| \leq n\} \\
\texttt{rvm}(r_1, r_2)^{\mathcal{I}} &= \{x \in \Delta \mid r_1^{\mathcal{I}}(x) \subseteq r_2^{\mathcal{I}}(x)\} \\
\texttt{and-role}(r_1, ..., r_n)^{\mathcal{I}} &= \bigcap_{i=1,n} r_i^{\mathcal{I}} \\
\texttt{or-role}(r_1, ..., r_n)^{\mathcal{I}} &= \bigcup_{i=1,n} r_i^{\mathcal{I}} \\
\texttt{not-role}(r)^{\mathcal{I}} &= (\Delta \rightarrow 2^{\Delta}) \backslash r^{\mathcal{I}} \\
\texttt{fills}(r, o) &= \{x \in \Delta \mid o^{\mathcal{I}} \in r^{\mathcal{I}}(x^{\mathcal{I}})\} \\
\texttt{one-of}(o_1, ..., o_n) &= \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}
\end{aligned}$$

Les autres constructeurs de rôles, s'expriment plus simplement sous forme de fonction de leur argument :

$$\begin{aligned}
\texttt{compose}(r_1, r_2)^{\mathcal{I}}(x) &= \bigcup_{y \in r_1^{\mathcal{I}}(x)} r_2^{\mathcal{I}}(y) \\
\texttt{range}(r, c)^{\mathcal{I}}(x) &= r^{\mathcal{I}}(x) \cap c^{\mathcal{I}} \\
\texttt{inverse}(r)^{\mathcal{I}}(x) &= \{y \in \Delta \mid x \in r^{\mathcal{I}}(y)\}
\end{aligned}$$

Le constructeur `a-not` a la même sémantique que `not`, mais il ne s'applique qu'à des concepts atomiques primitifs.

2. S'il y a une certaine variabilité dans la littérature, elle repose plus sur le vocabulaire que sur l'interprétation.

À ces constructeurs primitifs, s'ajoutent diverses variantes syntaxiques :

$$\begin{aligned}
\texttt{single}(r) &= \texttt{atmost}(1, r) \\
\texttt{some}(r) &= \texttt{atleast}(1, r) \\
\texttt{c-atleast}(n, r, c) &= \texttt{atleast}(n, \texttt{range}(r, c)) \\
\texttt{c-atmost}(n, r, c) &= \texttt{atmost}(n, \texttt{range}(r, c)) \\
\texttt{c-some}(r, c) &= \texttt{c-atleast}(1, r, c)
\end{aligned}$$

Enfin, s'ajoutent à ces constructeurs quelques opérateurs qui se traduisent par des contraintes supplémentaires non définitionnelles :

$$\begin{aligned}
\texttt{cover}(c, c_1, ..., c_n) &\Longrightarrow c^{\mathcal{I}} = \bigcap_i c_i^{\mathcal{I}} \\
\texttt{disjoint}(c_1, ..., c_n) &\Longrightarrow \bigcap_i c_i^{\mathcal{I}} = \emptyset
\end{aligned}$$

Les analogues (mais pas équivalents) définitionnels respectifs sont `or` et `not`.

Table des figures

Bibliographie

[Abadi et Cardelli, 1996] M. Abadi et L. Cardelli. *A theory of objects.* Monographs in Computer Science. Springer Verlag, 1996.

[Aït-Kaci et Nasr, 1986] H. Aït-Kaci et R. Nasr. LOGIN: a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[Aït-Kaci et Podelski, 1993] H. Aït-Kaci et A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.

[Aït-Kaci, 1991a] H. Aït-Kaci. A glimpse of Paradise. In *Next Generation Information System Technology*, éditeurs J.W. Schmidt et A.A. Stogny, pages 15–25. Springer-Verlag, 1991.

[Aït-Kaci, 1991b] H. Aït-Kaci. An overview of LIFE. In *Next Generation Information System Technology*, éditeurs J.W. Schmidt et A.A. Stogny, pages 42–58. Springer-Verlag, 1991.

[Atkinson *et al.*, 1989] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, et S. Zdonik. The object-oriented database system manifesto. 1989.

[Attardi, 1991] G. Attardi. An Analysis of Taxonomic Reasoning. In Lenzerini et al. [1991], pages 29–49.

[Baader et Hanschke, 1991] F. Baader et P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc.* 12^{th} *International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[Baader et Hollunder, 1991] F. Baader et B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2:22–27, 1991.

[Bardou et Dony, 1995] D. Bardou et C. Dony. Propositions pour un nouveau modèle d'objets dans les langages à prototypes. In Napoli [1995], pages 93–112.

[Benoit *et al.*, 1986] Ch. Benoit, Y. Caseau, et Ch. Pherivong. LORE, un langage objet relationnel et ensembliste. In *Actes des Journées Langages Orientés Objet*, éditeurs P. Cointe et J. Bézivin, numéro 48, pages 3–13. Afcet Informatique, Bigre+Globule, 1986.

[Berthier, 1994] Berthier. L'agent rationnel, objet de l'IA? *Revue d'Intelligence Artificielle*, 8(4):327–349, 1994.

[Borgida et Patel-Schneider, 1994] A. Borgida et P.F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *JAIR*, 1:277–308, 1994.

[Borgida, 1992a] A. Borgida. Description logics are not just for the flightless-birds: a new look at the utility and foundations of description logics. Rapport technique, Rutgers University, 1992.

[Borgida, 1992b] A. Borgida. From type systems to knowledge representation: natural semantics specifications for description logics. *Int. J. on Intelligent and Co-operative Information Systems*, 1(1):93–126, 1992.

[Borgida, 1995a] A. Borgida. Description Logics in Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[Borgida, 1995b] A. Borgida. Types in programming languages and description logics in knowledge representation: a comparison. 1995. version courte de [Borgida, 1992b].

[Boyland et Castagna, 1996] J. Boyland et G. Castagna. Type-safe compilation of covariant specialization: a practical case. In *Proceedings of the Tenth European Conference on Object-Oriented Programming, ECOOP'96*, éditeur P. Cointe, pages 3–25, Linz, Austria, 1996. Springer-Verlag.

[Brachman *et al.*, 1991] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Resnick, et A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE Language. In *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, éditeur J. Sowa, pages 401–456. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[Brachman et Schmolze, 1985] R.J. Brachman et J.G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman, 1977] R.J. Brachman. What's in a Concept: Structural Foundations for Semantic Networks. *International Journal of Man-Machine Studies*, 9:127–152, 1977.

[Brachman, 1979] R.J. Brachman. On the Epistemological Status of Semantic Networks. In *Associative Networks: Representation and Use of Knowledge by Computers*, éditeur N.V. Findler, pages 3–50. Associated Press, New York, 1979.

[Brachman, 1983] R.J. Brachman. What IS-A is and isn't: An Analysis of Taxonomic Links in Semantic Networks. *Computer*, 16(10):30–37, 1983.

[Brachman, 1985] R. Brachman. "I lied about the trees", or defaults and definitions in knowledge representation. *AI Magazine*, 6(3):80–93, 1985.

[Bruce et Longo, 1990] K. B. Bruce et G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87:196–240, 1990. (réédité dans [Gunter et Mitchell, 1994]).

[Buchheit *et al.*, 1993] M. Buchheit, F.M. Donini, et A. Schaerf. Decidable reasoning in terminologic knowledge representation systems. *JAIR*, 1:109–138, 1993.

[Bunge, 1977] M. Bunge. *Ontology I: The Furniture of the World*, volume 3 de *Treatise on Basic Philosophy*. Reidel, Boston, 1977.

[Bunge, 1979] M. Bunge. *Ontology II: A World of Systems*, volume 4 de *Treatise on Basic Philosophy*. Reidel, Boston, 1979.

[Calvanese *et al.*, 1994] D. Calvanese, M. Lenzerini, et D. Nardi. A unified framework for class-based representation formalisms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94), Bonn, Germany*, pages 109–120. 1994.

[Capponi, 1993] C. Capponi. Classification des classes par les types. In Habib et Oussalah [1993], pages 215–224.

[Capponi, 1995] C. Capponi. *Identification et exploitation des types dans un modèle de connaissances par objets*. Thèse d'informatique, Université Grenoble I, 1995.

[Caraux *et al.*, 1995] G. Caraux, O. Gascuel, G. Andrieu, et D. Levy. Approches informatiques de la reconstruction d'arbres phylogénétiques à partir de données moléculaires. *Technique et Science Informatiques*, 14(2):113–139, 1995.

[Carpenter, 1992] B. Carpenter. *The logic of typed feature structures*. Numéro 32 dans Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[Carré *et al.*, 1990] B. Carré, L. Dekker, et J-M. Geib. Multiple and evolutive representation in the ROME language. In *Second International Conference on Technology of Object-Oriented Languages and Systems*, Paris, june 1990.

[Carré *et al.*, 1995] B. Carré, R. Ducournau, J. Euzenat, A. Napoli, et F. Rechenmann. Classification et objets : programmation ou représentation. In *Actes des 5e journées nationales du PRC-GDR Intelligence Artificielle*, pages 213–237. Teknea, 1995.

[Carré et Dekker, 1991] B. Carré et L. Dekker. Inheriting object-oriented features through meta-programming. In *Proceedings of the First East European Conference on Object-Oriented Programming, EurOOP'91*, éditeur Augustin Mrazik, Bratislava, sep. 1991.

[Carré, 1989] B. Carré. *Méthodologie orientée objet pour la représentation des connaissances, concepts de points de vue, de représentation multiple et évolutive d'objet*. Thèse d'informatique, Université de Lille, 1989.

[Caseau, 1985] Y. Caseau. Les objets et les ensembles dans LORE. In *Actes RFIA'85*, 1985.

[Caseau, 1987] Y. Caseau. *Etude et réalisation d'un langage objet :* LORE. Thèse d'Université, Orsay, 1987.

[Caseau, 1996] Y. Caseau. La définition de règles logiques sur des objets dans le langage CLAIRE. In Dennebouy [1996], pages 170–182.

[Castagna, 1996a] G. Castagna. Instance variables specialization in object-oriented programming. 1996.

[Castagna, 1996b] G. Castagna. Le modèle fondé sur la surcharge : une visite guidée. *Technique et Science Informatiques*, 15(6):673–708, 1996.

[Chabre, 1988] B. Chabre. Le filtrage en YAFOOL. Rapport de stage ENST, Sema Group, 1988.

[Chambers et Ungar, 1992] C. Chambers et D. Ungar. *Lisp and Symbolic Computation*, 4(3), 1992. Numéro spécial consacré au langage SELF.

[Cointe, 1987] P. Cointe. Metaclass are first class: the OBJVLISP model. In *Proceedings of the Second ACM Conference on Object-Oriented Programming, Languages and Applications, OOPSLA'87*, éditeur N. Meyrowitz, Orlando, 1987. ACM Press.

[Coupey et Fouqueré, 1994] P. Coupey et Ch. Fouqueré. Classer des concepts définis avec des défauts et des exceptions. In Rechenmann [1994], pages 69–80.

[Coupey et Fouqueré, 1995] P. Coupey et Ch. Fouqueré. Defaults in concept definitions: an application. In *Proc. KRUSE'95*, pages 112–121, 1995.

[Crampé et Euzenat, 1996] I. Crampé et J. Euzenat. La révision dans un modèle d'objets simple. In Dennebouy [1996], pages 134–149.

[Dac, 1972] P. Dac. *Pensées*. Le Cherche Midi, Paris, 1972.

[David et Krivine, 1987] J.-M. David et J.-P. Krivine. Utilisation de prototypes dans un système expert de diagnostic : le projet DIVA. In *Actes 7e Journées Internationales "Les systèmes experts et leurs applications"*, Avignon, mai 1987.

[David et Krivine, 1988] J.-M. David et J.-P. Krivine. Acquisition de connaissances expertes à partir de situations types. In *Actes 8e Journées Internationales "Les systèmes experts et leurs applications"*, Avignon, mai 1988.

[Dekker et Carré, 1992] L. Dekker et B. Carré. Multiple and dynamic representation of frame with point of view in FROME. In Habib et Oussalah [1992], pages 97–111.

[Dekker, 1993] L. Dekker. La réification des filtres en langage FROME. In Habib et Oussalah [1993], pages 23–35.

[Dekker, 1994] L. Dekker. FROME *: représentation multiple et classification d'objets avec points de vue*. Thèse d'informatique, Université de Lille, 1994.

[Dennebouy, 1993] Y. Dennebouy. *Un langage visuel pour la manipulation de données*. Thèse d'informatique, EPFL, Lausanne, CH, 1993.

[Dennebouy, 1996] éditeur Y. Dennebouy. *Actes des journées Langages et Modèles à Objets, LMO'96*, Leysin, octobre 1996.

[Dicky *et al.*, 1994] H. Dicky, C. Dony, M. Huchard, et T. Libourel. Un algorithme d'insertion avec restructuration dans les hiérarchies de classes. In Rechenmann [1994], pages 125–136.

[Dicky *et al.*, 1996] H. Dicky, C. Dony, M. Huchard, et T. Libourel. On automatic class insertion with overloading. In *Proceedings of the Eleventh ACM Conference on Object-Oriented Programming, Languages and Applications, OOPSLA'96*, pages 251–267. ACM Press, 1996.

[Dionne *et al.*, 1992] R. Dionne, E. Mays, et F.J. Oles. A non-well-founded approach to terminological cycles. In *AAAI'92*, pages 761–766, 1992.

[Dionne *et al.*, 1993] R. Dionne, E. Mays, et F.J. Oles. The equivalence of model-theoretic and structural subsumption in description logics. In *Proceedings of the 13th* IJCAI*, Chambéry, France*, pages 710–716, 1993.

[Dobbie et Topor, 1995] G. Dobbie et R. Topor. On the declarative and procedural semantics of deductive object-oriented systems. *Journal of Intelligent Information Systems*, 4:193–219, 1995.

[Donini *et al.*, 1994] F.M. Donini, M. Lenzerini, D. Nardi, et A. Schaerf. Deduction in Concept Languages: From Subsumption to Instance Checking. *Journal of Logic and Computation*, 4:1–30, 1994.

[Ducournau *et al.*, 1994] R. Ducournau, M. Habib, et G. Simonet. Méta-héritage et héritage non-transitif. Rapport de Recherche 94-021, L.I.R.M.M., Montpellier, 1994.

[Ducournau *et al.*, 1995] R. Ducournau, M. Habib, M. Huchard, M.-L. Mugnier, et A. Napoli. Le point sur l'héritage multiple. *Technique et Science Informatiques*, 14(3):309–345, 1995.

[Ducournau et Habib, 1989] R. Ducournau et M. Habib. La multiplicité de l'héritage multiple. *Technique et Science Informatiques*, 8(1):41–62, 1989.

[Ducournau et Habib, 1991] R. Ducournau et M. Habib. Masking and conflicts, or to inherit is not to own. In Lenzerini et al. [1991], chapitre 14.

[Ducournau, 1991] R. Ducournau. *Y3 :* YAFOOL*, le langage à objets.* Sema Group, 1991.

[Ducournau, 1996] R. Ducournau. Les incertitudes de la classification incertaine. In Dennebouy [1996], pages 183–200.

[Dugerdil, 1988] P. Dugerdil. *Contribution à l'étude de la représentation des connaissances fondée sur les objets. Le langage* OBJLOG. Thèse d'informatique, Université d'Aix-Marseille II, 1988.

[Eco, 1994] U. Eco. *La recherche de la langue parfaite.* Seuil, Paris, 1994.

[Ellis, 1993] G. Ellis. Efficient retrieval from hierarchies of objects using lattice operations. In *Proc. of the 1st Int. Conf. on Conceptual Structures*, numéro 699 dans LNCS. Springer-Verlag, 1993.

[Euzenat et Rechenmann, 1995] J. Euzenat et F. Rechenmann. Shirka, 10 ans, c'est Tropes? In Napoli [1995], pages 13–34.

[Euzenat, 1993a] J. Euzenat. Définition abstraite de la classification et son application aux taxonomies d'objets. In Habib et Oussalah [1993], pages 235–246.

[Euzenat, 1993b] J. Euzenat. On a purely taxonomic and descriptive meaning for classes. In *Proceedings of the IJCAI'93 Workshop on Object-based Representation Systems*, éditeur A. Napoli, Nancy, France, 1993. Rapport de Recherche CRIN 93-R-156.

[Euzenat, 1994] J. Euzenat. Classification dans les représentations par objets : produits de systèmes classificatoires. In *Actes de la conférence Reconnaissance des Formes et Intelligence Artificielle*, 1994.

[Fargier *et al.*, 1996a] H. Fargier, J. Lang, R. Martin-Clouaire, et T. Schiex. Traitement de problèmes de décision sous incertitude par des problèmes de satisfaction de contraintes. *Revue d'Intelligence Artificielle*, 10(4):??, 1996.

[Fargier *et al.*, 1996b] H. Fargier, J. Lang, et T. Schiex. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proc. of AAAI-96*, page??, 1996.

[Faucher, 1991] C. Faucher. *Elaboration d'un langage extensible fondé sur les schémas : le langage* OBJLOG+. Thèse d'informatique, Université d'Aix-Marseille III, 1991.

[Faucher, 1992] C. Faucher. Extensibilité d'un langage de schémas relativement à ses aspects. In Habib et Oussalah [1992], pages 30–42.

[Ferber et Volle, 1988] J. Ferber et Ph. Volle. Using coreference in object-oriented representation. In *Proceedings of ECAI'88*. 1988.

[Ferber, 1989] J. Ferber. *Objets et agents : une étude des structures de représentation et de communication en Intelligence Artificielle*. Thèse d'Etat, Université Paris VI, 1989.

[Fikes et Kehler, 1985] R. Fikes et T. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, September 1985.

[Finin, 1986] T.W. Finin. Interactive classification: a technique for acquiring and maintaining knowledge bases. *Proceedings of the IEEE*, 74(10):1414–1421, 1986.

[Gensel, 1995] J. Gensel. *Contraintes et représentation des connaissances par objets. Application au modèle Tropes*. Thèse d'informatique, Université Grenoble I, 1995.

[Ghelli et Orsini, 1991] G. Ghelli et R. Orsini. Types and subtypes as partial equivalence relations. In Lenzerini et al. [1991], chapitre 12, pages 191–209.

[Girard, 1995] P. Girard. *Construction hypothétique d'objets complexes*. Thèse d'informatique, Université Grenoble I, 1995.

[Gochet et Gribomont, 1991] P. Gochet et P. Gribomont. *Logique, méthodes pour l'informatique fondamentale*, volume 1. Hermès, Paris, 1991.

[Gochet et Gribomont, 1994] P. Gochet et P. Gribomont. *Logique, méthodes pour l'informatique fondamentale*, volume 2. Hermès, Paris, 1994.

[Godin *et al.*, 1995] R. Godin, G. Mineau, R. Missaoui, et H. Mili. Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. *Revue d'Intelligence Artificielle*, 9(2):105–137, 1995.

[Gonzalez-Gomez, 1996] M. Gonzalez-Gomez. *Représentation des connaissances approximatives et classification approchée d'instances dans un langage de schémas : le système CAIN*. Thèse d'informatique, Université d'Aix-Marseille III, 1996.

[Granger et Thonnat, 1985] C. Granger et M. Thonnat. Un système de vision fondé sur une méthode structurelle de classification. In *Cognitiva 85*, 1985.

[Granger, 1985] C. Granger. *Reconnaissance d'objets par mise en correspondance en vision par ordinateur*. Thèse de 3e cycle, Université de Nice, 1985.

[Granger, 1986] C. Granger. Fuzzy reasoning in a knowledge-based system for object recognition. In *Proc. of ECAI'86*, pages 163–170, 1986.

[Granger, 1988] C. Granger. An Application of Possibility Theory to Object Recognition. *Fuzzy Sets and Systems*, 28(4):351–362, 1988.

[Guinaldo, 1995] O. Guinaldo. Techniques d'indexation pour aider à la classification dans le modèle des graphes conceptuels. In Napoli [1995], pages 53–68.

[Guinaldo, 1996] O. Guinaldo. *Étude d'un gestionnaire d'ensembles de graphes conceptuels*. Thèse d'informatique, Université Montpellier II, 1996.

[Gunter et Mitchell, 1994] éditeurs C. A. Gunter et J. C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. The MIT Press, 1994.

[Habib et Oussalah, 1992] éditeurs M. Habib et M. Oussalah. *Actes des journées Représentation Par Objets, RPO'92*, La Grande Motte, juin 1992. EC2.

[Habib et Oussalah, 1993] éditeurs M. Habib et M. Oussalah. *Actes des journées Représentation Par Objets, RPO'93*, La Grande Motte, juin 1993. EC2.

[Hautamäki, 1986] A. Hautamäki. Points of view and their logical analysis. *Acta Philosophica Fennica*, 41:1–156, 1986.

[Horty, 1991] J.F. Horty. A credulous theory of mixed inheritance. In Lenzerini et al. [1991], chapitre 2.

[Horty, 1994] J.F. Horty. Some direct theories of nonmonotonic inheritance. In *Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 2: Nonmonotonic Reasoning*, éditeurs D. Gabbay et C. Hogger. Oxford University Press, 1994.

[ILO, 1991] ILOG, Gentilly. SMECI*, reference manual*, 1991.

[ILO, 1995] ILOG, Gentilly. ILOG TALK *reference manual, Version 3.2*, 1995.

[ILO, 1996] ILOG, Gentilly. POWER CLASSES *reference manual, Version 1.4*, 1996.

[Johnson, 1964] W.E. Johnson. *Logic*. Dover, New York, 1964.

[Kahn, 1987] G. Kahn. Natural semantics. Rapport Technique RR 601, I.N.R.I.A., 1987.

[Kaindl, 1994] H. Kaindl. Object-oriented approaches in software engineering and artificial intelligence. *Journal of Object-Oriented Programming*, 6(8), 1994.

[Karp, 1993] P.D. Karp. The design space of knowledge representation systems. Technical Report 520, SRI AI Center, 1993.

[Kay, 1993] A.C. Kay. The early history of SMALLTALK. In *Proc. of the 2nd History of Programming Languages Conference, HOPL-II*, éditeurs J.A.N. Lee et J.E. Sammet. ACM Press, 1993.

[Keene, 1989] S.E. Keene. *Object-Oriented Programming in* COMMON LISP*. A programmer's guide to* CLOS. Addison-Wesley, Reading, MA, 1989.

[Kifer *et al.*, 1993] M. Kifer, G. Lauser, et J. Wu. Logical foundations of object-oriented and frame-based languages. Rapport Technique 93/06, SUNY at Stony Brook, 1993. to appear in Journal of ACM.

[Lalande, 1926] A. Lalande. *Vocabulaire technique et critique de la philosophie*. Presses Universitaires de France, Paris, 1926.

[Leclère, 1994] M. Leclère. Diverses problématiques de classification dans le modèle des graphes conceptuels. In Rechenmann [1994], pages 81–94.

[Leclère, 1995] M. Leclère. *Les connaissances du niveau terminologique des graphes conceptuels*. Thèse d'informatique, Université Montpellier II, 1995.

[Leclère, 1996] M. Leclère. C-CHiC : Construction coopérative de hiérarchies de catégories. *Revue d'Intelligence Artificielle*, 10(1):57–100, 1996.

[Lenzerini *et al.*, 1991] éditeurs M. Lenzerini, D. Nardi, et M. Simi. *Inheritance Hierarchies in Knowledge Representation and Programming Langages*. John Wiley & Sons, 1991.

[Levesque et Brachman, 1987] H.J. Levesque et R.J. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence*, 3(2):78–93, 1987.

[Levy et Rousset, 1995] A.Y. Levy et M.-C. Rousset. CARIN: a representation language combining horn rules and description logic. In *IJCAI'95*, 1995.

[Lipkis, 1982] T. Lipkis. A KL-ONE Classifier. In *Proceedings of the 1981 KL-ONE Workshop, Jackson, New Hampshire*, éditeurs J.G. Schmolze et R.J. Brachman, pages 126–143, 1982.

[MacGregor, 1991] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In Sowa [1991], pages 385–400.

[Magnan, 1994] M. Magnan. *Réutilisation des composants : les exceptions dans les objets composites.* Thèse d'informatique, Université Montpellier II, sept. 1994.

[Malenfant, 1996] J. Malenfant. Programmation par prototypes. *Technique et Science Informatiques*, 15(6):709–733, 1996.

[Mariño *et al.*, 1990] O. Mariño, F. Rechenmann, et P. Uvietta. Multiple perspectives and classification mechanism in object oriented classification. In *Proc. ECAI'90*, pages 425–430, 1990.

[Mariño, 1991] O. Mariño. Classification d'objets composites dans un système de représentation des connaissances multi-points de vue. In *Actes RFIA'91*, 1991.

[Mariño, 1993] O. Mariño. *Raisonnement classificatoire dans une représentation à objets multi-points de vue.* Thèse d'informatique, Université Grenoble I, 1993.

[Masini *et al.*, 1989] G. Masini, A. Napoli, D. Colnet, D. Leonard, et K. Tombre. *Les langages à objets.* InterEditions, 1989.

[Medina, 1995] R. Medina. *Arborescence des états globaux et analyse d'exécutions distribuées.* Thèse d'informatique, Université Montpellier II, 1995.

[Minsky, 1975] M. Minsky. A framework for representing knowledge. In *The psychology of computer vision*, éditeur P.H. Winston, pages 211–281. McGraw-Hill, 1975.

[Mosses, 1990] P.D. Mosses. Denotational semantics. In *Formal Models and Semantics*, éditeur J. Van Leeuwen, volume 1 de *Handbook of Theoretical Computer Science.* Elsevier, Amsterdam, 1990.

[Motschnig-Pitrik et Mylopoulos, 1992] R. Motschnig-Pitrik et J. Mylopoulos. Classes and instances. *Int. Journal of Intelligent and Cooperative Systems*, 1(1):61–92, 1992.

[Mugnier et Chein, 1992] M.-L. Mugnier et M. Chein. Conceptual graphs: fundamental notions. *Revue d'Intelligence Artificielle*, 6:365–406, 1992.

[Mulet et Cointe, 1993] Ph. Mulet et P. Cointe. Définition d'un noyau réflexif pour un langage à prototypes. In Habib et Oussalah [1993], pages 101–116.

[Mulet et Marco, 1994] Ph. Mulet et J. Marco. De la parenté entre les environnements de MIT SCHEME et les prototypes de SELF. In Rechenmann [1994], pages 167–178.

[Mulet, 1995] Ph. Mulet. *Réflexion et langages à prototypes.* Thèse d'informatique, Université de Nantes, 1995.

[Myers *et al.*, 1992] B.A. Myers, D.A. Giuse, et B. VanderZanden. Declarative programming in a prototype-instance system: object-oriented programming without writing method. In *Proceedings of the Seventh ACM Conference on Object-Oriented Programming, Languages and Applications, OOPSLA'92*, éditeur A. Paepcke, Vancouver, 1992. ACM Press.

[Napoli *et al.*, 1994] A. Napoli, C. Laurenço, et R. Ducournau. An object-based approach to organic synthesis planning. *Int. J. Human-Computer Studies*, 41:5–32, 1994.

[Napoli, 1991] A. Napoli. Le raisonnement par classification. In *Le Raisonnement en Intelligence Artificielle*, chapitre 8. InterEditions, 1991.

[Napoli, 1992] A. Napoli. *Représentation par objets et raisonnement par classification en Intelligence Artificielle.* Thèse d'Etat, Université de Nancy, 1992.

[Napoli, 1995] éditeur A. Napoli. *Actes des journées Langages et Modèles à Objets, LMO'95*, Nancy, octobre 1995. INRIA.

[Nebel, 1985] B. Nebel. How well does a vanilla loop fit into a frame. *Data and knowledge engineering*, 1:181–194, 1985.

[Nebel, 1990] B. Nebel. *Reasoning and revision in hybrid representation systems.* Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1990.

[Nguyen et Rieu, 1989] G.T. Nguyen et D. Rieu. Schema Evolution in Object-Oriented Database Systems. *Data & Knowledge Engineering*, 4:43–67, 1989.

[Nourine, 1993] L. Nourine. *Quelques propriétés algorithmiques des treillis.* Thèse d'informatique, Université Montpellier II, 1993.

[Padgham et Nebel, 1993] L. Padgham et B. Nebel. Combining classification and non-monotonic inheritance reasoning: a first step. In *Proc. 7^{th} International Symposium on Methodologies for Intelligent Systems*, pages 15–18, Trondheim, Norway, 1993.

[Padgham et Zhang, 1993] L. Padgham et T. Zhang. A terminological logic with defaults: a definition and an application. In *Proceedings of the 13th* IJCAI*, Chambéry, France*, pages 663–668, 1993.

[Paramavisam et Plaisted, 1996] M. Paramavisam et D. A. Plaisted. Automated deduction techniques for classification in concept languages. Technical Report TR96-015, CS Department, University of North Carolina, Chapel Hill, 1996.

[Parent et Spaccapietra, 1996] Ch. Parent et S. Spaccapietra. Bases de données orientées objets. Support de cours, École CIMPA-INRIA-UNESCO *Langages et Modèles à Objets*, 1996.

[Patel-Schneider, 1989a] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272, 1989.

[Patel-Schneider, 1989b] P.F. Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38(3):319–351, 1989.

[Pelenc et Ducournau, 1996] S. Pelenc et R. Ducournau. Un modèle pour l'intégration des contraintes et des objets. In *Journées du GDR de Programmation, thème objet*, Limoges, 1996.

[Perrot, 1996] J.-F. Perrot. Principes implémentatoires des langages à objets. Support de cours, École CIMPA-INRIA-UNESCO *Langages et Modèles à Objets*, 1996.

[Popper, 1934] K. R. Popper. *La logique de la découverte scientifique*. Payot (1973 pour la trad. française), 1934.

[Procop et Pivot, 1987] M. Procop et B. Pivot. *Définition et réalisation d'une fonctionnalité de classification dans* SHIRKA. Mémoire d'année spéciale « intelligence artificielle », ENSIMAG, Grenoble, 1987.

[Projet Sherpa, 1995] Projet Sherpa. TROPES 1.0 reference manual. Rapport interne, INRIA Rhônes-Alpes, Grenoble, 1995.

[Quillian, 1968] M.R. Quillian. Semantic memory. In *Semantic information processing*, éditeur M. Minsky, pages 227–270. MIT Press, 1968.

[Quine, 1977] W.V. Quine. *Relativité de l'ontologie et autres essais*. Aubier Montaigne, Paris, 1977.

[Quine, 1993] W.V. Quine. *Quiddités. Dictionnaire philosophique par intermittence*. L'ordre philosophique. Seuil, Paris, 1993.

[Quinqueton *et al.*, 1992] J. Quinqueton, P. Vismara, J.C. Régin, M. Py, et C. Laurenço. RESYN : un système d'aide à la synthèse en chimie organique. In *Systèmes Experts et Applications, Avignon'92*, pages 305–318, 1992.

[Rechenmann et Vignard, 1985] F. Rechenmann et P. Vignard. CRIKA : quand les règles rencontrent les objets. Rapport de Recherche RR 468, INRIA, Sophia-Antipolis, 1985.

[Rechenmann, 1988] F. Rechenmann. SHIRKA : système de gestion de bases de connaissances centrées-objet. Manuel de référence. Rapport technique, IMAG – INRIA Rhône-Alpes, Grenoble, 1988.

[Rechenmann, 1992] F. Rechenmann. Method inheritance and selection in class based knowledge models. In *Proceedings of the ERCIM workshop on Theoretical and Experimental Aspects of Knowledge Reprsentation*, éditeurs C. Meghini et F. Sebastiani, pages 205–208, 1992.

[Rechenmann, 1994] éditeur F. Rechenmann. *Actes des journées Langages et Modèles à Objets, LMO'94*, Grenoble, octobre 1994. IMAG.

[Resnick *et al.*, 1993] L.A. Resnick, A. Borgida, R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, et K.C. Zalondek. CLASSIC description and reference manual for the COMMON LISP implementation (version 2.2). Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1993.

[Rieu *et al.*, 1992] D. Rieu, G.T. Nguyen, et J. Escamilla. Méthodes et représentation des connaissances. In Habib et Oussalah [1992], pages 17–30.

[Rossazza, 1990] J.-P. Rossazza. *Utilisation de hiérarchies de classes floues pour la représentation des connaissances imprécises et sujettes à exceptions : le système* SORCIER. Thèse d'université, Toulouse, 1990.

[Russel, 1921] B. Russel. *Introduction à la philosophie mathématique.* Payot, Paris, 1921. (trad. française, 1991).

[Schaff, 1960] A. Schaff. *Introduction à la sémantique.* Editions Anthropos, Paris, 1960.

[Schmidt-Schauß et Smolka, 1991] M. Schmidt-Schauß et G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[Scott, 1972] D.S. Scott. Continuous lattices. *Lecture Notes in Mathematics*, 274:97–136, 1972.

[Scott, 1976] D. Scott. Data types as lattices. *SIAM J. Computer*, 5:452–487, 1976.

[Simonet, 1994] G. Simonet. *Héritage non-monotone à base de chemins et de graphes partiels.* Thèse d'informatique, Université Montpellier II, sept. 1994.

[Sowa, 1984] J. Sowa. *Conceptual structures – information processing in mind and machine.* Addison Wesley, Reading, Massachussets, 1984.

[Sowa, 1991] éditeur J. Sowa. *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann Publishers, San Mateo, California, 1991.

[Steele, 1990] G.L. Steele. *Common Lisp, the Language.* Digital Press, second édition, 1990.

[Tarski, 1952] A. Tarski. The semantic conception of truth and the foundation of semantics. In *Semantics and the philosophy of language*, éditeur L. Linsky. Urbana, 1952.

[Thomason, 1992] R.H. Thomason. NETL and subsequent path-based inheritance theories. *Computers and Mathematics with Applications*, 23(2-5):179–204, 1992.

[Tort, 1989] P. Tort. *La raison classificatoire.* Aubier, 1989.

[Touretzky, 1986] D.S. Touretzky. *The Mathematics of Inheritance Systems.* Pitman, London, 1986.

[Vismara, 1995] Ph. Vismara. *Reconnaissance et représentation d'éléments structuraux pour la description d'objets complexes. Application à l'élaboration de stratégies de synthèse en chimie organique.* Thèse d'informatique, Université Montpellier II, 1995.

[Vismara, 1996] Ph. Vismara. Appariements dirigés pour le raisonnement par classification sur des hiérarchies de graphes. In Dennebouy [1996], pages 201–214.

[Volle, 1989] Ph. Volle. Coréférence et mécanismes déductifs dans un langage de représentation par objets. In *PRC-IA Représentation par objets*. Caen, 1989.

[Wand, 1989] Y. Wand. A proposal for a formal model of objects. In *Object-Oriented Concepts, Databases and Applications*, éditeurs W. Kim et F.H. Lochovsky, pages 537–559. ACM Press, 1989.

[Wand, 1994] M. Wand. Type inference for objects with instance variables and inheritance. In Gunter et Mitchell [1994], pages 97–120.

[Wille, 1982] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, éditeur I. Rival, pages 445–470. Reidel, 1982.

[Wille, 1992] R. Wille. Concept lattices and conceptual knowledge systems. *Computers Mth. Applic.*, 23(6-9):493–515, 1992.

[Winograd, 1975] T. Winograd. Frame representation and the declarative / procedural controversy. In *Representation and Understanding: studies in cognitive science*, éditeurs D.G. Bobrow et A.M. Collins, pages 185–210. Academic Press, 1975.

[Wittgenstein, 1945] L. Wittgenstein. *Investigations philosophiques*. Gallimard, Paris, 1945.

[Woods et Schmolze, 1992] W. Woods et J. Schmolze. The KL-ONE family. *Computers and Mathematics with Applications*, 23(2-5):133–177, 1992.

[Woods, 1975] W.A. Woods. What's in a Link: Foundations for Semantic Networks. In *Representation and Understanding: Studies in Cognitive Science*, éditeurs D.G. Bobrow et A. Collins, pages 35–82. Associated Press, New York, 1975.

[Woods, 1991] W. Woods. Understanding subsumption and taxonomy: a framework for progress. In Sowa [1991], pages 45–94.

[Wright *et al.*, 1984] J.M. Wright, M.S. Fox, et D. Adam. *SRL2 User Manual*, 1984.

Index des références

Index des noms cités

Index des langages cités

Index des règles

Index des notations

Index

– D –