Mémoire pour l'Habilitation à Diriger les Recherches

Expressivité / Efficacité : le Dilemme de la Programmation Logique Inductive

Céline Rouveirol

Table des matières

Chapitre 1

Introduction

La Programmation Logique Inductive (PLI) est une jeune discipline[1] que l'on a souvent définie comme *l'intersection de la Programmation Logique et de l'Apprentissage Symbolique* [MDR94]. À l'origine, l'objectif commun des chercheurs de cette communauté était d'appliquer des outils de *formalisation issus de la Programmation Logique* à des techniques d'apprentissage dans des langages de concepts complexes, restrictions de la logique des prédicats du premier ordre (LPO dans la suite du document).

L'Apprentissage Symbolique couvre de nombreux domaines (voir par exemple [Mit97] pour une introduction à l'apprentissage), définis par le type de problème considéré. La PLI, à l'instar de la communauté d'apprentissage dans son ensemble, s'est concentrée sur le problème de l'apprentissage de définitions de concepts discriminantes (apprentissage *discriminant* dans la suite du document), même si certains travaux de PLI ont abordé d'autres problématiques, comme l'apprentissage descriptif, le regroupement conceptuel ou même l'apprentissage par renforcement.

Illustrons d'abord intuitivement la nécessité de langages de représentation plus puissants que la logique propositionnelle ou la logique attribut-valeur[2] en prenant un exemple dans le domaine de la biologie moléculaire [SKMS97, SKB99]. Supposons que le problème soit de discriminer des molécules mutagènes et non-mutagènes [SMSK96] (il est admis que les molécules mutagènes ont un rôle dans le déclenchement du cancer). Dans un langage propositionnel, les molécules mutagènes et non-mutagènes sont représentées en fonction d'un *nombre fixe d'attributs* (nominaux, numériques ou booléens), chaque attribut étant une fonction de la molécule décrite. Des corrélations entre les valeurs de ces attributs et l'activité mutagène de la molécule peuvent être apprises, avec des algorithmes d'apprentissage tels que la régression linéaire, les réseaux neuronaux, etc.

Par contre, ces logiques ne sont pas adaptées pour représenter l'aspect *structurel* du problème, que ce soit au niveau des entrées (des exemples) ou des sorties (les définitions de concept obtenues). D'une part, les molécules en entrée du système d'apprentissage ont des *structures* très différentes les unes des autres, que ce soit en termes de leur nombre d'atomes (chaque molécule possède entre 20 et 40 atomes) et de leurs caractéristiques, ou en termes de *liaisons* entre ces atomes. D'autre part, on peut chercher à apprendre un concept faisant intervenir une *sous-structure commune* à ces molécules, une sous-structure étant vue comme un ensemble non spécifié d'atomes et de liens entre ces atomes. Il est clair que le formalisme propositionnel ou attribut-valeur n'a pas une puissance de représentation permettant de représenter des formules

1. Le premier workshop de PLI a eu lieu en 1991 [Mug91].
2. connue également sous le nom de logique zéro +, voir annexe.

aussi complexes que celles-ci ; il est alors nécessaire d'adopter une technique d'apprentissage capable de traiter des langages de représentation d'exemples et de concepts plus puissant, offrant à la fois souplesse et puissance de représentation pour les entrées et pour les résultats de l'apprentissage, comme par exemple, des langages de la logique des prédicats du premier ordre.

Nous nous sommes spécialement attaché, au cours de nos travaux, à la résolution de problèmes d'apprentissage dans des langages de type Datalog sans récursion et sans négation. L'apprentissage discriminant dans de telles restrictions de la logique des prédicats, que nous nommerons dans la suite *apprentissage relationnel*, préexiste à la PLI (voir entre autres [Plo70, Rey70, Sha83, KG86, SB86, BG90, Bis92]). La PLI prend sa source dans ces approches, mais elle met tout particulièrement l'accent sur la modélisation formelle des mécanismes déductifs et inductifs à la base de ce type d'apprentissage. Ceci comprend, entre autres, l'étude des espaces de recherche propres à l'apprentissage en LPO et de leurs structures, la définition d'opérateurs d'apprentissage permettant de les parcourir et la caractérisation de leurs propriétés, en particulier de leur complexité.

Intuitivement, la complexité d'un processus d'apprentissage est influencée par la taille de l'espace de recherche de l'algorithme d'apprentissage (en première approximation, le langage cible tout entier) ainsi que par la complexité des opérations qui permettent de parcourir cet espace de recherche et d'évaluer ce parcours. Le constat est sans surprise : l'apprentissage PLI est en général considérablement plus complexe que l'apprentissage dans des langages cibles propositionnels (espaces de recherche potentiellement infinis, opérations inductives et déductives indécidables ou NP-complètes). Le besoin au niveau des applications impose néanmoins la recherche de restrictions qui à la fois ont un sens en terme de l'application traitée et qui permettent l'implantation de méthodes d'inférences inductives et déductives relativement efficaces afin de former la base d'outils d'apprentissage opérationnels. Le dilemme central de la PLI, à l'instar de beaucoup d'approches d'Intelligence Artificielle symboliques, est donc la recherche d'un bon compromis entre la puissance du langage de concept choisi et la complexité de l'apprentissage dans ce langage [LB85].

La recherche en PLI, très active durant les dix dernières années, à la fois théorique et expérimentale, a permis de faire progresser largement la compréhension des difficultés propres à l'apprentissage relationnel, ainsi que de faire émerger quelques propositions de langages du premier ordre et d'opérations élémentaires inductives et déductives offrant une réponse satisfaisante au dilemme cité ci-dessus. Par ailleurs, certains travaux ont également mis en avant les limites de la PLI, qui sont dans une large partie héritées du modèle sous-jacent inspiré de la programmation logique . Celles-ci comprennent entre autres les difficultés de la PLI, sous sa forme actuelle, à modéliser un raisonnement inductif probabiliste ou encore à manipuler des informations numériques.

Nous décrivons dans ce document nos contributions au domaine de la PLI au cours de ces dernières années en les organisant autour de trois axes. Le chapitre 2 est entièrement consacré à l'usage des biais d'apprentissage en PLI. Les biais en apprentissage discriminant regroupent toute information autre que les exemples et contre-exemples du concept à définir, qui vont permettre de sélectionner une définition parmi toutes les définitions candidates. Si l'utilisation des biais d'apprentissage est loin d'être restreinte aux systèmes de PLI, nous montrerons dans ce chapitre comment ils deviennent des acteurs majeurs de l'apprentissage PLI, dans le sens où ils vont permettre de contenir la complexité intrinsèque de l'apprentissage relationnel. Nous présenterons un modèle opérationnel d'une famille de systèmes de PLI dont ils sont les paramètres principaux. En nous appuyant sur ce modèle, nous présenterons deux travaux visant à une meilleure utilisation des biais.

Dans le chapitre 3, nous explorerons les liens entre l'apprentissage PLI et l'apprentissage propositionnel. Nous aborderons la question de l'utilisation de techniques d'apprentissage propositionnelles guidées par les données pour la résolution de problèmes d'apprentissage relationnels : est-elle possible et si oui, sous quelles conditions ? Nous donnerons une première ébauche de réponse en présentant deux systèmes implantant de telles techniques. Dans le chapitre 4, nous remettrons en cause le choix de Datalog comme langage de concept cible pour la PLI. En particulier, nous considérerons deux extensions de l'apprentissage PLI discriminant à des langages hybrides : langage de clauses contraintes d'une part, afin de mieux intégrer le traitement des informations numériques en PLI, et langage hybride combinant logique définie et logique de description, afin d'étendre l'expressivité des langages de Horn grâce aux connecteurs offerts par les logiques de descriptions. Enfin, nous conclurons ce document par le chapitre 5 en décrivant quelques enjeux et perspectives à moyen terme de ce domaine.

Chapitre 2

Biais et configuration de systèmes de Programmation Logique Inductive

Le travail présenté dans ce chapitre a été réalisé en collaboration avec H. Adé (Université Catholique de Leuven, Belgique), P. Albert (Ilog, France), C. Nédellec (LRI, France), et F. Torre (LRI, France). Ce chapitre présente une synthèse des résultats publiés dans les articles [RADR93, NR93, NR94, RA94, NRA$^+$96, TR97a].

2.1 Motivations

La PLI, depuis ses débuts, s'est attachée, entre autres [1], à proposer des modèles logiques des opérations élémentaires d'apprentissage discriminant.

Définition 2.1.1 (Problème d'apprentissage discriminant [Mit82])
Un problème d'apprentissage discriminant se pose par la donnée :

- *d'un langage des instances* $\mathcal{L}_e$*,*
- *d'un langage des hypothèses* $\mathcal{L}_h$*,*
- *d'une relation de couverture entre formules de* $\mathcal{L}_h$ *et formules de* $\mathcal{L}_e$*,*
- *d'un ensemble d'apprentissage,* $E = E^+ \cup E^-$*, ensemble de formules de* $\mathcal{L}_e$*, exemples et contre-exemples d'un concept cible.*

L'algorithme d'apprentissage va parcourir $\mathcal{L}_h$ ou un de ses sous-ensembles afin de construire une définition du concept cible, satisfaisant le critère d'arrêt de l'apprentissage – dans une version idéale que nous adopterons dans la suite, la complétude et la correction par rapport à l'ensemble d'apprentissage. Sans rentrer dès à présent dans les détails techniques, cette définition est suffisante pour faire apparaître ces opérations élémentaires :

- le test de couverture entre une hypothèse et un exemple associé à la relation de couverture choisie (opération déductive) et
- implicitement, les opérateurs d'apprentissage qui permettent de parcourir $\mathcal{L}_h$ en transformant une hypothèse en un ensemble de nouvelles hypothèses (opération inductive)

1. D'autres branches de l'apprentissage, comme le regroupement conceptuel, ont également et plus récemment été abordées dans la cadre de la PLI. Nous n'aborderons pas ce domaine dans ce mémoire.

La section 2.2 présente un résumé synthétique de l'état de l'art concernant ces résultats. Ces travaux ont donné le jour à une caractérisation logique de ces opérations et ont permis d'exploiter des résultats issus de la Programmation Logique, en particulier en terme de décidabilité et de complexité de ces opérations. Beaucoup de ces résultats sont peu encourageants quant à l'efficacité de l'implémentation «brute» de ces opérations dans un langage de la logique des prédicats, et même dans un langage Datalog sans récursion et sans négation. Très rapidement, un aller-retour s'est installé dans la communauté PLI entre l'implémentation de systèmes opérationnels et l'étude théorique des inférences inductives et déductives plus contraintes mises en place dans ces systèmes. Ces contraintes ou restrictions s'expriment par l'intermédiaire de méta-connaissances portant le nom de *biais* qui permettent d'exprimer des propriétés ou des préférences sur les solutions attendues, autres que celles données à travers l'ensemble d'apprentissage. Ces contraintes peuvent être exploitées afin d'optimiser les opérations élémentaires citées précédemment.

Très vite, on a assisté à un foisonnement de biais de toute sorte en PLI, dont la sémantique et l'influence sur le modèle logique de la PLI étaient mal étudiées. L'une des tâches du contrat ESPRIT ILP BRA 6020 était de recenser, classifier, comparer les biais et leurs rôles dans les systèmes de PLI de l'époque. Un des résultats de cette tâche est une typologie des biais utilisés en PLI [NRA+96]. Conjointement, nous avons proposé [NR93, NR94] un modèle alternatif au modèle logique pour les systèmes de PLI : un modèle opérationnel abstrait des systèmes de PLI mettant en lumière l'influence et l'interaction des biais d'un système de PLI dans le fonctionnement effectif de ces systèmes, permettant de comparer leur comportement sur un problème d'apprentissage spécifique. Nous nous sommes appuyés pour la construction de ce modèle, outre sur les résultats de la communauté de PLI, sur les résultats et formalismes développés dans la communauté d'Acquisition des Connaissances (maintenant Ingénierie des Connaissances) [RA94, RA96]. La section 2.4 présente une partie de ce modèle mettant en lumière l'influence des biais sur le processus d'apprentissage et certaines de leurs interactions pour une famille de systèmes de PLI, les systèmes à stratégie *«générer et tester»*.

Ce modèle ayant permis de poser les bases d'une réflexion sur l'interaction entre biais, relation de généralité et opérateurs d'apprentissage dans les systèmes de PLI, nous présentons enfin deux de nos travaux qui sont des développement orthogonaux autour de ce modèle. Le premier, en collaboration avec Fabien Torre [TR97a] traite d'une meilleure exploitation de certains biais afin d'améliorer la recherche d'un système *«générer et tester»* (section 2.5). Ce travail a abouti à la définition de nouvelles relations de généralité et de nouveaux opérateurs d'apprentissage intégrant directement la notion de biais de langage, et permettant, entre autres, de dépasser des résultats négatifs connus sur certaines relations de généralité classiques en PLI.

Ce modèle générique peut également être envisagé comme la spécification d'une boîte à outils à l'usage des développeurs d'applications PLI. Le but à terme est double : concevoir et créer une librairie de composants à *faible granularité* permettant à un ingénieur de la connaissance de configurer aisément et finement des méthodes d'apprentissage de concepts de type PLI afin de les adapter aux besoins d'une application à base de connaissances spécifique. Dans ce cadre, et sur la base de notre modèle générique, le système $\mathcal{HAIKU}$ a été implémenté (section 2.6). Dans le chapitre 5, nous présenterons brièvement une application préliminaire de ce système configurable à un problème d'extraction de connaissances dans les bases de données.

2.2 État de l'art en formalisation PLI

Nous présentons ici un bref résumé des recherches en PLI sur lesquelles nous nous appuierons au cours de ce chapitre. Nous avons regroupé les définitions dans l'annexe 5, qui introduit également les notations classiquement utilisées en PLI. Ce résumé n'a pas pour but d'être exhaustif. Nous renvoyons le lecteur intéressé à [MDR94, LD94, BG95a, NCdW97] pour autant d'introductions à la PLI.

Nous nous plaçons dans le cadre de *l'apprentissage à partir d'implication logique* [MDR94], classiquement utilisé en PLI pour l'apprentissage de définitions de concepts à partir d'exemples et de contre-exemples d'un concept cible[2].

Soient $\mathcal{L}_e$, $\mathcal{L}_b$ et $\mathcal{L}_h$ des langages de la logique du premier ordre. Étant donné un ensemble d'apprentissage $E = E^+ \cup E^-$, avec E^+ et E^- des ensembles d'exemples positifs et négatifs pour un concept cible, décrits dans un langage $\mathcal{L}_e$, une théorie du domaine B, décrite dans un langage $\mathcal{L}_b$, il s'agit de former une hypothèse H décrite dans un langage $\mathcal{L}_h$ telle que

$$\begin{array}{ll}
B \cup E^- \not\models \Box & \text{(correction a priori)} \\
B \not\models E^+ & \text{(nécessité a priori)} \\
B \cup H \models E^+ & (H \text{ est complète}) \\
B \cup H \cup E^- \not\models \Box & (H \text{ est correcte}).
\end{array}$$

Dans ce modèle logique de l'apprentissage discriminant, la *relation de couverture* entre hypothèses de $\mathcal{L}_h$ et exemples de $\mathcal{L}_e$ est l'implication logique ($\models$). $\mathcal{L}_h$ est structuré par une relation d'ordre partiel, appelée *relation de généralité* ou de *subsomption*. Intuitivement, une hypothèse H_1 est plus générale qu'une hypothèse H_2 si l'ensemble des exemples couverts par H_1 inclut l'ensemble des exemples couverts par H_2. Dans le cas général en PLI, la relation de généralité est modélisée, ainsi que la relation de couverture, par l'implication logique entre formules de $\mathcal{L}_h$. $\mathcal{L}_h$ forme l'espace de recherche potentiel (on verra par la suite que $\mathcal{L}_h$ n'est heureusement jamais exploré exhaustivement en cours d'apprentissage). Comme on a dans la plupart des cas en PLI $\mathcal{L}_e \subseteq \mathcal{L}_h$, on confondra, sauf mention explicite, relation de couverture et relation de généralité dans la suite de ce document.

L'implication logique étant indécidable [SS88] pour la logique des prédicats du premier ordre, ce cadre reste purement formel, et de nombreux travaux en PLI ont en donc proposé des restrictions, la plus usitée étant la restriction de $\mathcal{L}_h$ et $\mathcal{L}_b$ à des langages définis (voir annexe).

Relation de généralité / couverture

La relation de généralité la plus largement utilisée en PLI est la θ-subsomption [Plo70], équivalente à l'implication logique sans théorie du domaine pour des clauses non tautologiques et non récursives. Malheureusement, pour la θ-subsomption, le test de subsomption, tout en étant décidable, est NP-difficile dans le pire des cas, même dans un langage de clauses de Horn sans symbole de fonction [KL94]. Depuis, de nombreuses restrictions de la θ-subsomption ont été étudiées (voir par exemple la subsomption OI [ELMS96], la subsomption faible [BS99]) et/ou des langages restreints pour lesquels le test de θ-subsomption est polynomial ont été identifiés (ij-détermination [MF92], k-localité [KL94]).

2. Nous renvoyons à [DR97] pour une discussion sur les différents cadres pour l'apprentissage PLI, en particulier, celui de l'*apprentissage à partir d'interprétations* qui permet d'apprendre des formules *caractéristiques*, et non discriminantes.

Afin de pouvoir traiter la notion de généralité en présence de théorie du domaine, des relations de subsomption relativement à une théorie du domaine ont été introduites ou remises au goût du jour : la subsomption relative [Plo71] et sa restriction aux théories non récursives, la subsomption généralisée [Bun88]. Signalons également des relations de généralité plus puissantes, adaptées à l'apprentissage de programmes récursifs contenant des clauses autorésolvantes [IA95, Mug95], que nous ne décrirons pas ici.

Opérateurs d'apprentissage

En pratique, l'espace de recherche $\mathcal{L}_h$ est parcouru à l'aide d'*opérateurs* d'apprentissage ou *opérateurs de raffinement* [Sha83, NCdW97]. Ces opérateurs d'apprentissage satisfont la relation de généralité, notée $\prec$, imposée sur l'espace de recherche : un opérateur d'apprentissage $\mathcal{O}$ est soit *ascendant* (toute formule $D \in \mathcal{O}(C)$ est telle que D est plus générale que C, noté $C \preceq D$, soit *descendant* (pour tout $C \in \mathcal{L}_h$, pour tout $D \in \mathcal{O}(C)$, $D \preceq C$).

De nombreuses propriétés ont été étudiées pour ces opérateurs d'apprentissage. Après diverses versions de complétude [Rou94, Mug95, vdLNC94a], une propriété plus élaborée connue sous le nom d'*idéalité* (voir [vdLNC94b] pour une première présentation et [NCdW97] pour une présentation plus détaillée) a recueilli les suffrages de la communauté PLI. Un opérateur d'apprentissage $\mathcal{O}$ est idéal s'il est localement fini, propre [3] et complet [4]. Il n'existe pas d'opérateurs idéaux pour des espaces de recherche ordonnés par la θ-subsomption [vdLNC94b, vdL95], même pour des langages définis sans symbole de fonction. Il en existe par contre pour des relations de généralité plus restreintes que la θ-subsomption, comme la subsomption-OI, la subsomption faible, ou encore la subsomption empirique [CBS95].

La notion d'idéalité est très forte : un opérateur d'apprentissage ascendant idéal permet à partir de toute clause C de $\mathcal{L}_h$, d'atteindre toute clause de $\mathcal{L}_h$ plus générale que C. La recherche avec un opérateur idéal est donc très redondante, car *tous les chemins* qui mènent à une même hypothèse vont être développés. L'idéalité reste donc une propriété formelle qui ne saurait assurer l'efficacité d'un opérateur d'apprentissage. La communauté de PLI concentre en ce moment davantage son attention sur une propriété plus faible : l'*optimalité* [DRB93]. Un opérateur optimal est localement fini, faiblement complet (toutes les clauses de $\mathcal{L}_h$ sont atteignables à partir de la clause initiale de l'espace de recherche uniquement) et non-redondant (l'opérateur ρ ne génère plus qu'une fois une hypothèse donnée de l'espace de recherche à partir de la clause maximalement générale de l'espace). L'optimalité paraît plus adaptée pour caractériser les opérateurs avec de bonnes propriétés algorithmiques.

Si l'on désire implanter sur la base de ces modèles logiques des programmes d'apprentissage relativement efficaces, il est dans la plupart des cas nécessaire de contraindre davantage les spécifications de la tâche d'apprentissage, soit en restreignant *a priori* l'espace de recherche, soit en affaiblissant la relation de subsomption, soit enfin en élaguant dynamiquement l'espace de recherche.

2.2.1 Où l'on reparle de biais

T. Mitchell [Mit91] a donné le premier une définition du *biais* d'apprentissage : *toute information (autre que la stricte complétude et correction par rapport aux exemples d'apprentissage) permettant de choisir une généralisation parmi d'autres.* Il a par ailleurs prouvé que le biais,

3. Pour toute clause $C \in \mathcal{L}_h$, $\mathcal{O}(C)$ ne contient pas de clauses logiquement équivalentes à C.

4. Pour un opérateur $\mathcal{O}$ ascendant, et pour toute clause $C \in \mathcal{L}_h, \mathcal{O}$ permet d'atteindre toutes les clauses de $\mathcal{L}_h$ plus générales que C. Voir annexe 5 pour une définition détaillée de l'idéalité.

loin d'être un simple artifice destiné uniquement à améliorer les performances d'un système d'apprentissage, est *nécessaire* à l'étape d'induction qui est à la base de tout système de généralisation : hors biais, un système d'apprentissage discriminant n'apprendra que la disjonction des exemples positifs, sans pouvoir de prédiction (voir [Tor00] pour un rappel très pédagogique de l'argumentation de Mitchell).

Nous utiliserons dans la suite la classification suivante des biais (on retrouve des classifications similaires dans [Mit91, Utg86]). Nous distinguerons trois catégories de biais :

– *les biais de langage* : ils définissent le *langage du concept cible*, en restreignant *a priori* la syntaxe des hypothèses de l'espace de recherche et les combinaisons possibles entre ces hypothèses pour former la définition de concept finale. Il inclut toute restriction syntaxique sur l'expression du concept cible courant.

– *les biais de recherche* : biais dynamiques par définition, ils définissent implicitement l'espace des hypothèses que le programme va explorer effectivement et/ou dans quel ordre ces hypothèses vont être développées.

– *les biais de validation* : ils définissent les critères d'arrêt de l'apprentissage, pour une hypothèse ou pour une définition de concept donnés.

Les biais expriment, dans le meilleur des cas (d'aucuns les nomment alors *sémantiques*), des connaissances à propos du problème d'apprentissage à résoudre, en désignant des zones privilégiées de l'espace de recherche. Ils permettent d'exprimer à travers un ensemble de contraintes des informations supplémentaires sur le problème d'apprentissage ; on attend donc un résultat plus pertinent. Sinon, (ils se nomment alors *syntaxiques*), ils expriment une contrainte algorithmique nécessaire à la terminaison ou à l'efficacité du processus d'apprentissage, par exemple en définissant une restriction adéquate de l'espace de recherche rendant le test de subsomption polynômial, ou en exprimant une notion de «simplicité» des formules recherchées.

Les biais offrent donc, s'ils sont déclaratifs, la possibilité à un utilisateur, certes averti, d'adapter un système d'apprentissage (et de PLI en particulier) à la résolution d'un problème d'apprentissage spécifique. L'existence de biais, parfois mal intégrés au modèle logique précédemment décrit, résulte en une plus grande difficulté de caractérisation du comportement des systèmes d'apprentissage. En effet, l'adéquation des biais d'un système de PLI conditionne le succès de l'apprentissage. Si les contraintes imposées par les biais sont trop fortes ou mal ciblées, la (ou les) hypothèse(s) décrivant le concept cible peuvent être exclues de l'espace de recherche ou tout chemin vers cette hypothèse interdit. Si le biais est trop faible, le système de PLI peut «errer» dans l'espace de recherche sans converger sur la solution attendue, même si elle appartient à l'espace de recherche. L'expression des biais adaptés à un problème spécifique d'apprentissage est donc une étape cruciale (et non triviale) lors de l'application d'un système d'apprentissage, surtout en PLI, où la taille de l'espace de recherche potentiel $\mathcal{L}_h$ fait que le choix d'un biais trop faible a des conséquences désastreuses.

2.3 Un modèle de l'apprentissage *«générer et tester»*

Les «briques de base» (opérations élémentaires d'apprentissage, et biais qui agissent comme des paramètres de ces opérations) sont donc disponibles afin de tenter de construire un modèle générique des systèmes de PLI. Ce modèle a pour but, dans un premier temps, d'expliciter le rôle des biais dans les systèmes d'apprentissage de PLI, et d'étudier le comportement empirique de ces systèmes en fonction des différents biais qui leur sont imposés. Brièvement, nous souhaitons produire une description de haut niveau de ces systèmes en termes de *structure* et *nature de connaissances* et de *nature du raisonnement*, plutôt qu'en terme algorithmique [Cau94a].

Nous nous sommes à ce sujet inspiré, afin de clarifier le statut de ce modèle, de nombreux travaux issus de la communauté d'Ingénierie des Connaissances.

2.3.1 Ingénierie des Connaissances et modélisation

Le domaine de l'Ingéniérie des Connaissances s'intéresse à l'étude de techniques permettant de rationaliser la construction d'applications à bases de connaissances. Ce domaine est fortement connecté à l'Apprentissage en particulier parce que les objectifs de ces deux disciplines – éliciter des connaissances déclaratives pour la résolution de problèmes – sont similaires, même si leurs hypothèses de travail et leurs méthodes sont différentes. En Ingénierie des Connaissances (IC), le développeur de systèmes à bases de connaissances, ou *ingénieur de la connaissance* fait partie intégrante et active de la boucle d'acquisition des connaissances, alors que l'apprentissage considère souvent comme résolue cette étape d'interaction avec un utilisateur, et se concentre sur le développement de techniques automatiques avec une faible interaction avec l'utilisateur, dont la tâche se limite souvent à fournir un ensemble d'exemples d'apprentissage[5].

La recherche en IC a produit un ensemble de *méthodes d'acquisition des connaissances* qui fournissent à un ingénieur de la connaissance à la fois un ensemble de formalismes de représentation (graphiques ou non) ainsi qu'un ensemble de directives pour exprimer la connaissance qu'un humain utilise pour résoudre un problème sous forme de représentation informatique.

On peut distinguer deux grandes familles d'approches en IC : les *méthodes orientées domaine* et les *méthodes génériques*. Les méthodes orientées domaine, comme par exemple [MM89] et [Mar88], sont dédiées à la modélisation et la résolution d'une classe de problèmes (diagnostic, classification, etc.). Elles proposent une représentation des connaissances dédiée au problème ainsi qu'un ensemble d'outils pour opérationnaliser cette connaissance. Bien que très utiles pour ces problèmes spécifiques, ces méthodes manquent de généralité pour être adaptées à nos besoins.

Les méthodes génériques dont les fleurons sont KADS [SWB92, WVdVSA92], Commet [Ste92] et VITAL [O'H92] ont pour but de fournir des formalismes généraux afin de décrire des modèles conceptuels (au niveau connaissance) de méthodes de résolution de problèmes, détaillant comment un problème ou une classe de problèmes peuvent être représentés et résolus.

Sans vouloir rentrer dans le débat sur le niveau connaissance, qui est bien au delà des limites que nous nous sommes fixées pour ce mémoire, nous reprenons plutôt ici une citation de Clancey [Cla85] qui traduit parfaitement notre interprétation du niveau connaissances : « *Une description au niveau connaissances est une spécification d'un système en termes de connaissances qu'il doit posséder pour résoudre la tâche qui lui est assignée... Pour répondre à cette exigence, il est nécessaire de caractériser la méthode de raisonnement suivie par le système ainsi que la structure des connaissances qui sous-tend ce raisonnement.* » Le niveau connaissance permet donc de décrire et d'interpréter le comportement de systèmes informatiques complexes, sans que le modèle ait à détailler la structure interne de ceux-ci, mais simplement en observant leurs interactions avec l'extérieur. Dans cette interprétation, le niveau connaissance nous semble un bon médium à la fois pour fournir une base de description «unificatrice» de nos systèmes d'apprentissage et de leurs similarités indépendamment de leurs particularités d'implantation, ainsi

5. Notons d'ailleurs l'exception des systèmes *apprentis*, voir en particulier [Néd92].

qu'un bon niveau d'abstraction pour communiquer avec un ingénieur de la connaissances, ceci en accord avec [VdVA92].

Le modèle au niveau connaissances va à la fois fournir un formalisme de modélisation utile pour représenter une famille de systèmes d'apprentissage, ainsi qu'un ensemble de notations graphiques qui va faciliter l'interaction avec l'utilisateur.

2.3.2 Méthodes génériques en IC

Ces méthodologies s'appuient sur la notion de *librairies de composants réutilisables* [Ste93, BBVVdV93]. Nous analysons dans la suite l'adéquation à nos besoins des deux principales méthodes génériques d'IC, Commet et CommonKADS.

Commet

La méthode *Component of Expertise* (Commet) met l'accent sur la réutilisation des composants. Elle fournit des formalismes graphiques pour représenter des *décompositions de tâches* et des *flots de données*. Une tâche peut être décomposée en sous tâches et le flot de données est représenté graphiquement, permettant de spécifier la connexion entre les sous-tâches et les entités du domaine sur lesquelles les tâches s'appliquent.

Des méthodes de résolution de problèmes prédéfinies peuvent être utilisées pour spécifier le type de décomposition à appliquer pour une tâche donnée. Des éditeurs spécifiques sont utilisés pour définir les types de leurs paramètres et un moteur de propagation assure la cohérence globale du système en cours de conception.

Une caractéristique importante de Commet est la séparation des connaissances du domaine en deux modèles différents : le *modèle du domaine* qui stocke toute la connaissance du domaine qui est invariante d'une session à une autre, et le *modèle du cas* qui stocke toute la connaissance spécifique au problème en cours de résolution. La distinction est faite du point de vue réutilisabilité, le modèle du domaine étant *a priori* plus réutilisable que le modèle du cas.

KREST [Ste93], l'outil associé à la méthodologie fournit des services de gestion de bibliothèques pour stocker et accéder à des méthodes prédéfinies ou des composants définis par l'utilisateur. Notons enfin que KREST permet de produire un code exécutable à partir de la spécification graphique de l'application.

CommonKADS

CommonKADS partage certains des objectifs de Commet, il met par contre l'accent sur une autre ré-utilisabilité, celle des descriptions au niveau connaissances de classes de problèmes et de leur solutions.

CommonKADS distingue trois niveaux dans sa modélisation : le *niveau domaine*, qui représente la connaissance du domaine modélisé, le *niveau inférence*, qui représente les raisonnements abstraits s'appuyant sur le niveau domaine, en utilisant un formalisme étendu de flots de données, et le *niveau tâche*, qui représente l'algorithmique précise de la tâche à effectuer. Le niveau inférence et le formalisme utilisé, les *structures d'inférences*, qui permettent à la fois de spécifier une représentation abstraite (non algorithmique) des différentes étapes de résolution de problème, et la manière de les instancier au niveau domaine, nous ont paru particulièrement adaptées à nos objectifs de modélisation.

2.3.3 Choix d'un formalisme de représentation

Les objets que nous devons représenter sont les opérations d'apprentissage, leurs paramètres (leurs biais) et leurs connexions.

Nous devons expliciter l'ensemble des configurations possibles du système d'apprentissage, qui correspondent à toutes les décompositions possibles de la tâche de haut niveau, apprendre une définition de concept. Les deux approches évoquées, CommonKADS et KREST, ont des caractéristiques intéressantes pour notre problème, mais aucune n'est suffisante. Par exemple, nous avons besoin de deux fonctionnalités que nous ne trouvons pas réunies dans la même approche. Commet permet de représenter le système configuré final et de générer le programme opérationnel à partir de ses spécifications ; CommonKADS permet de représenter l'ensemble des composants de façon abstraite ainsi que leurs multiples décompositions possibles.

La possibilité de pouvoir représenter de multiples décompositions pour une tâche étant essentielle, nous avons choisi de représenter notre modèle grâce à une extension [AB92] des notations CommonKADS [SWB92]. Nous utilisons deux formalismes :

– une *hiérarchie de décomposition de tâches* qui représente graphiquement les décompositions possibles en sous tâches d'une tâche d'apprentissage, une notation spécifique explicitant les alternatives lors de la décomposition.

– une notation pour les *structure d'inférences* qui permet d'exprimer intuitivement les flots de données au sein d'une décomposition. Une notation spécifique permet de distinguer les paramètres de configuration qui doivent être instanciés au cours de la configuration.

Nous avons choisi – et étendu – les notations de CommonKADS, pour représenter notre modèle au niveau connaissances des systèmes *«générer et tester»*. Ce modèle a été conçu avec KADS-TOOL [AB92], un outil de support de la méthodologie CommonKADS, étendu en ce qui concerne la décomposition des étapes d'inférence.

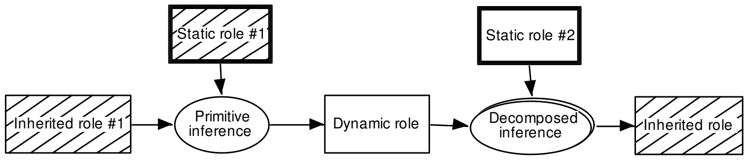

FIG. 2.1 – *Exemples de notations graphiques.*

La structure d'inférence de la figure 2.1 résume les notations que nous avons utilisées, fondées sur les notations CommonKADS, qui permettent de représenter à la fois l'ensemble des configurations possibles, et le système résultant. Une *structure d'inférence* organise un ensemble d'opérations, appelées *étapes d'inférence*, représentées par des ellipses. Une ellipse simple représente une étape d'inférence primitive, directement associée à un algorithme. Une ellipse grisée représente une étape d'inférence intermédiaire, qui peut être décomposée en de multiples structures d'inférence. Un rectangle simple représente un *rôle dynamique*. Il spécifie le flot des données entre deux ou plusieurs étapes d'inférence. Un rectangle en gras représente un *rôle statique*, c'est-à-dire, un rôle qui est utilisé mais non modifié au cours du raisonnement modélisé par l'étape d'inférence.

Pour la description du modèle au niveau connaissances de la famille d'algorithmes choisis, nous avons appliqué la correspondance suivante :

les opérations d'apprentissage sont représentées par des *étapes d'inférence*.

les décompositions possibles d'une opération d'apprentissage sont représentées par différentes *structures d'inférence* qui réalisent une étape d'inférence composite.

le flot de données entre opérations d'apprentissage est représenté par des *rôles dynamiques* qui lient entre elles les étapes d'inférence dans une même structure d'inférence.

les biais sont représentés par des *rôles statiques* associés aux étapes d'inférence. En effet, ces biais ne sont pas modifiés par ces étapes d'inférence mais uniquement par l'utilisateur en cours d'expérimentations.

Les contraintes contrôlant les décompositions, c'est-à-dire spécifiant les configurations interdites ou permises ne sont pas représentées en tant que telles dans le modèle. Pour l'instant, nous fournissons à l'ingénieur de la connaissances des «directives» de configurations textuelles.

L'enchaînement des différentes étapes d'inférence dans une structure d'inférence donnée est spécifié par un algorithme, qui joue le rôle du niveau tâche dans un modèle KADS.

2.4 Algorithmes génériques de $\mathcal{HAIKU}$

2.4.1 Stratégies de parcours de l'espace de recherche

D'un point de vue pratique, T. Mitchell [Mit82] regroupe les systèmes d'apprentissage en deux grandes familles : les systèmes à stratégie *guidée par les exemples* et les systèmes à stratégie *«générer et tester»*. Les systèmes d'apprentissage guidés par les exemples explorent leur espace de recherche en étant guidés par les exemples d'apprentissage, c'est-à-dire par les *similarités* observées entre les hypothèses courantes et les exemples positifs [6] et/ou les dissimilarités observées entre les hypothèses courantes et les exemples négatifs. L'apprentissage est donc implicitement contrôlé par l'ordre de présentation des exemples, ou tout au moins par l'ordre dans lequel ils sont traités. Les systèmes GOLEM [MF92] ou CILGG [Kie96] sont les rares systèmes d'apprentissage PLI à stratégie guidée par les données.

Par opposition, les systèmes d'apprentissage *«générer et tester»* génèrent dans un premier temps des hypothèses candidates suivant une stratégie *indépendante des exemples*, puis valident ensuite ces hypothèses sur l'ensemble d'apprentissage. L'avantage d'une approche de type *«générer et tester»* est qu'elle permet de définir très précisément à l'aide de biais d'apprentissage un espace de recherche fini et la manière de le parcourir. Pour cette raison, elle est utilisée par une majorité des systèmes de PLI (par exemple, CLINT [DR92], FOIL [Qui90], CLAUDIEN [DRB93, DRD97], PROGOL [Mug95], etc.) [7]. Par contre, la définition d'un biais adapté pour un système *«générer et tester»* requiert de la part de l'utilisateur une connaissance pointue à la fois du fonctionnement du système d'apprentissage de PLI et du domaine d'application.

En contrepartie, une approche guidée par les données, si elle s'appuie davantage sur les exemples de l'ensemble d'apprentissage et demande a priori moins d'expertise de l'utilisateur, peut s'avérer très coûteuse. Si l'on utilise par exemple une approche par calcul de moindres généralisés successifs, la progression dans l'espace de recherche peut être très lente. Par ailleurs, signalons que les opérateurs guidés par les données n'échappent pas à la complexité inhérente à l'apprentissage PLI. Par exemple, si la taille du moindre généralisé de deux clauses définies

6. Le plus utilisé de ces opérateurs d'apprentissage en PLI est la *least general generalisation* de Plotkin [Plo70] qui calcule le moindre généralisé de n exemples sous θ-subsomption.

7. Voir [NR93] pour plus de détails.

sous θ-subsomption est polynomiale, l'étape de réduction nécessaire pour élaguer le moindre généralisé est elle de complexité exponentielle.

Après cette brève discussion sur les stratégies d'apprentissage, nous présentons dans la suite en parallèle le modèle au niveau connaissances des algorithmes PLI *«générer et tester»* ainsi que les biais que nous avons répertoriés et leur influence dans le processus d'apprentissage. Nous renvoyons à [NRA+96] pour un panorama exhaustif des biais de langage et à [NR94, RA94] pour un exposé plus large du modèle.

Afin de décrire les systèmes d'apprentissage *«générer et tester»*, nous avons bien sûr repris le point de vue de Mitchell qui analyse l'apprentissage comme un processus de résolution de problème et nous l'avons précisé et étendu afin de rendre explicites les biais ainsi que les processus élémentaires sur lesquels ils agissent. Dans la suite, l'espace de recherche sera noté HS (pour *Hypothesis Space*). Plusieurs représentations sont possibles pour l'espace de recherche, nous choisissons ici, comme dans la majorité des systèmes de PLI, de représenter l'espace de recherche courant par une de ses bornes, notée CBD (pour $CurrentBestDefinition$). Ce type de représentation de l'espace suppose qu'une relation de subsomption est définie sur le langage de concept $\mathcal{L}_h$. Selon la direction de recherche choisie (ascendante ou descendante), l'espace de recherche effectif du système d'apprentissage sera inclus dans l'ensemble des hypothèses de $\mathcal{L}_h$ plus générales ou plus spécifiques que CBD. Nous appellerons dans la suite *hypothèse* une formule de $\mathcal{L}_h$, une définition de concept pouvant dans le cas général être formée d'un ensemble de telles hypothèses (on parle alors de définition de concept *disjonctive*).

2.4.2 Algorithme de haut niveau

La structure d'inférence la plus abstraite de l'apprentissage *«générer et tester»* peut se décomposer (voir figure 2.2) en deux sous-tâches : le *calcul d'une représentation initiale* pour HS et la *recherche* elle-même dans HS. Ces tâches ont en entrée, classiquement, les exemples positifs et négatifs du concept cible, et éventuellement une théorie du domaine.

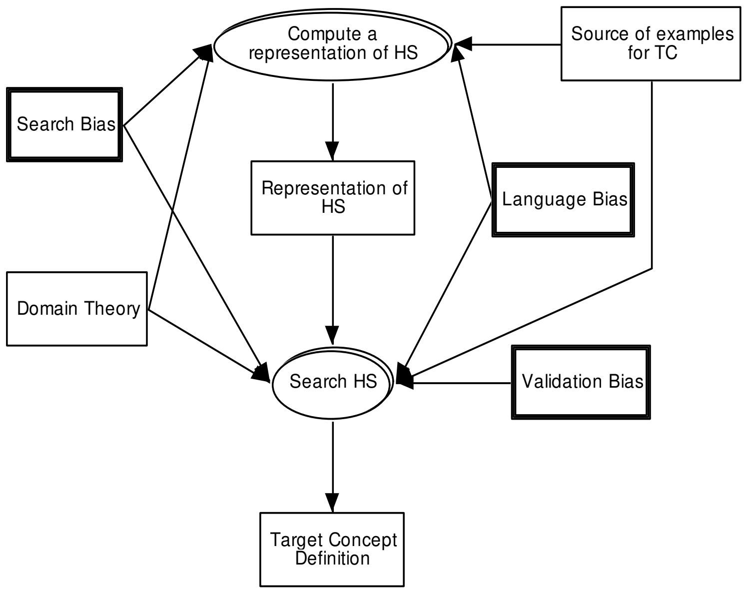

FIG. 2.2 – *Le point de vue résolution de problème sur l'apprentissage* «générer et tester»*, structure d'inférence de haut niveau*

La structure d'inférence révèle bien sûr que ces deux tâches et leurs biais associés sont fortement interdépendants. On distinguera dans la suite deux types d'approches. La première, dite *à biais de langage prédominant* est applicable si beaucoup d'informations sur la forme du concept cible sont disponibles (biais de langage fort). Un biais de langage *statique* définit un espace de recherche fini, préalablement à la recherche. Si le biais statique est suffisamment fort, il est intéressant d'adopter une approche précompilant l'espace de recherche (voir par exemple [BG95b, DRD97]), le biais de recherche se limitant alors à un ordre d'exploration des hypothèses de l'espace. Dans ce cas, définir un biais de langage adéquat, bien que tâche délicate pour un utilisateur, permet de limiter énormément l'effort de recherche. Signalons dans ce cadre l'utilisation de grammaires qui permettent de dénoter de façon très compacte l'ensemble des clauses de l'espace de recherche : les Clause sets [BG95b] et plus récemment, $\mathcal{DLAB}$ [DDR96], utilisé dans les systèmes $\mathcal{CLAUDIEN}$, TILDE [BDRJD99], etc.). Moins répandus, on trouve également la notion de schéma d'ordre deux pour contrôler l'expansion de l'espace de recherche (en particulier [MWKE93, Tau94]), les esquisses d'algorithmes [BJ93], et les modèles de règles [KW92], etc.

Si le biais de langage statique n'est pas suffisamment fort, ou s'il n'est pas *a priori* explicitable par l'utilisateur, la tâche d'apprentissage s'organise de manière différente. Dans une approche dite *générative*, le système d'apprentissage doit exploiter au maximum les informations contenues dans les exemples, ainsi qu'élaborer une stratégie de recherche sophistiquée afin de limiter la complexité de la recherche. Le calcul de la borne initiale de HS est dans ce cas contrôlé par le biais du langage et par les caractéristiques de la recherche qui s'effectuera ensuite (relation de généralité ordonnant HS, opérateurs d'apprentissage).

Si le système d'apprentissage est descendant, il maintient la borne supérieure de son espace de recherche, qui est souvent initialement réduite à une clause unitaire représentant le concept à apprendre. Certains systèmes descendants maintiennent également une borne inférieure de l'espace assurant la finitude de l'espace de recherche. Comme dans le cas des systèmes ascendants, cette borne inférieure est dans la plupart des cas calculée à partir d'un des exemples positifs du concept cible (noté TC, pour *Target Concept*), complété le cas échéant en utilisant une étape de déduction faisant intervenir la théorie du domaine (voir [RADR93, ADRB95] pour une description détaillée des biais de complétion pour les systèmes d'apprentissage ascendants). Le biais de langage est donc imposé par le langage de description des instances $\mathcal{L}_e$, et éventuellement par la théorie du domaine [DR92, Rou94, Mug95].

Le biais de langage appliqué à ce niveau a pour but de limiter la complexité du calcul de la borne inférieure de l'espace de recherche et/ou la complexité de la recherche ultérieure. On observe couramment dans les systèmes de PLI des contraintes sur la longueur et la complexité des clauses, ou sur le type/nombre de variables dans une clause hypothèse [DR92]. Signalons également des contraintes plus sophistiquées comme les schémas d'instanciations admissibles pour les variables d'une hypothèse, les modes d'entrée-sortie [Sha83, BG95b], ou l'ij-détermination [MF92], qui visent à limiter la complexité du test de subsomption, le calcul de la borne initiale de l'espace de recherche [Mug95] ou le calcul du moindre généralisé entre deux exemples [MF92].

La recherche dans HS va ensuite s'effectuer par modification itérative de CBD. La recherche dans HS est contrôlée par les biais de recherche, et par les biais de validation, et nous allons détailler dans la suite une de ses décompositions possibles (section 2.4.3).

Le niveau tâche le plus général de notre modèle (voir figure 2.3) contrôle, au sens algorithmique, l'enchaînement des sous-tâches identifiées ainsi que les flots de données entre celles-ci. Il gouverne la façon dont les différentes hypothèses d'une définition de concept disjonctive vont être apprises. L'exemple donné à la figure 2.3, implémente une stratégie de *couverture itérative* (voir [Bos95, Tor00] pour une discussion des stratégies d'apprentissage de définitions disjonctives) : une hypothèse est ajoutée à la définition courante du concept si elle satisfait un critère local d'apprentissage (par exemple, pour un apprentissage de type ascendant, la correction par rapport à l'ensemble d'apprentissage). Si la définition courante du concept ne satisfait pas le critère d'arrêt (toujours pour les systèmes ascendants, la complétude par rapport à l'ensemble d'apprentissage), une nouvelle hypothèse est apprise, éventuellement avec un nouvel ensemble d'apprentissage et/ou un nouveau biais.

gt(Source_Exemples, TD, B_Langage, B_Recherche, B_Validation)
% Stratégie de couverture itérative pour l'apprentissage disjonctif
Definition_Concept := $\emptyset$
Tant que $Critere\_Arret \in B\_validation$ n'est pas satisfait
CBD := calcul_borne_initiale(Source_Exemples,TD,B_Langage,B_Recherche)
PCD := modif_borne(Source_Exemples,TD,CBD,B_Langage, B_Recherche, B_Validation)
Fin Tant que
retourner($Concept\_Definition$).

FIG. 2.3 – *Niveau tâche de haut niveau de $\mathcal{HAIKU}$,* «générer et tester»

En dernier lieu, la définition de concept apprise est retournée ou l'apprentissage échoue, si le système ne trouve pas de solution au problème d'apprentissage dans la configuration donnée.

2.4.3 Algorithme central : modification de la borne

Une fois l'espace de recherche délimité par une ou deux bornes, le schéma de base de la recherche est détaillé à la figure 2.4.

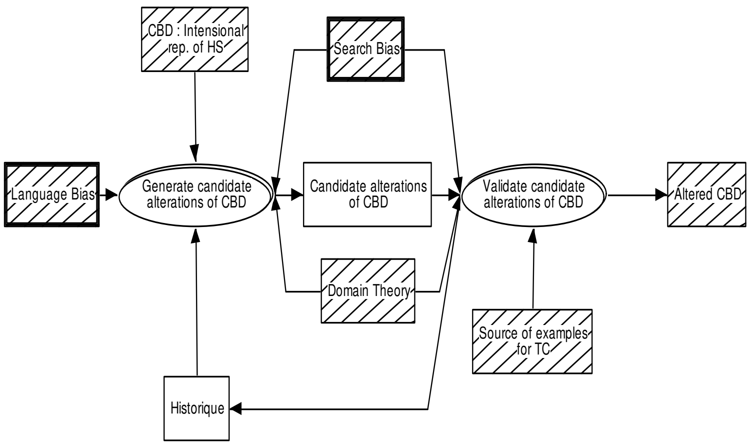

FIG. 2.4 – *Réduction de l'espace de recherche par modification itérative d'une de ses bornes (structure d'inférence)*

Étant donné une borne de l'espace de recherche CBD, le système de PLI engendre des transformations candidates de CBD, suivant une stratégie définie par le *biais de recherche*. En ce sens, le biais de recherche détermine quelle partie de l'espace de recherche préalablement définie par les biais de langage statiques va être effectivement explorée, et dans quel ordre.

Le biais de recherche peut être un biais *restrictif* ou *de préférence* [RG90] : un biais restrictif détermine quelles hypothèses de l'espace de recherche seront ignorées (par rapport à l'espace potentiel défini par les biais de langage statiques), alors qu'un biais de préférence n'exclut aucune hypothèse, mais fixe l'ordre dans lequel les hypothèses candidates doivent être générées ou explorées.

Chaque transformation de la borne CBD doit ensuite être évaluée contre les exemples, et chaque transformation de la borne validée va donc généraliser la borne inférieure (resp. spécialiser la borne supérieure) de l'espace de recherche et donc ainsi le réduire de façon monotone. Toute l'expertise du développeur de systèmes de PLI consiste à rendre l'ensemble des hypothèses candidates à évaluer contre les exemples le plus petit possible, car le test de couverture d'une hypothèse contre un ensemble d'exemples est particulièrement coûteux, même si les exemples sont représentés sous leur forme la plus triviale, c'est-à-dire un ensemble de faits (voir section 2.2).

Génération des hypothèses

Pour des systèmes à biais de langage prédominant, pour lesquels toutes les hypothèses de l'espace de recherche sont connues *a priori*, le biais de recherche se limite à spécifier dans quel ordre les hypothèses pré-compilées seront développées et testées contre les exemples. Nous nous intéresserons par la suite aux biais de recherche pour les approches *génératives* (figure 2.5).

Dans ce type de systèmes, les hypothèses candidates sont générées dynamiquement par l'application d'un opérateur d'apprentissage à l'une hypothèse de CBD. Le, ou les opérateurs applicables ainsi que la stratégie d'application de ces opérateurs font alors partie du biais de recherche. Si le système est ascendant (resp. descendant), la borne de l'espace de recherche va être généralisée (resp. spécialisée).

L'espace de recherche effectif est la clôture des opérateurs d'apprentissage choisis sur la borne initiale. Cet espace de recherche est partiellement ordonné par une relation de généralité qui dépend de l'opérateur choisi. Parmi les opérateurs PLI les plus étudiés et employés [8], citons :

1. l'opérateur d'*abandon de littéral*, qui comme son nom l'indique, généralise les hypothèses en abandonnant un littéral dans le corps de cette clause [DR92, Rou94, ELMS96], et son opérateur dual de spécialisation *par ajout de littéral* [BG88, Qui90, Mug95].

2. les opérateurs qui utilisent dynamiquement la théorie du domaine : l'opérateur d'*inversion de la résolution* nommé absorption [SB86, MB92] et son opérateur dual de spécialisation, le dépliage [BG88] qui consiste à appliquer une étape de résolution entre une clause courante à spécialiser et une clause de la théorie du domaine.

La relation de généralité sur l'espace des hypothèses est ensuite exploitée pour élaguer les hypothèses incohérentes ou incomplètes de l'espace de recherche (voir section suivante).

Étant donné une borne initiale, et un opérateur d'apprentissage, il reste à définir une stratégie de parcours de l'espace de recherche réel. Ceci se fait à travers

- le choix de l'hypothèse à modifier dans CBD, si elle est disjonctive. On retrouve là les stratégies classiques de recherche dans un espace étudiées en Intelligence Artificielle : largeur d'abord, [Néd92, DR92], profondeur itérative [DRD97] pour les stratégies non

8. Les opérateurs de PLI sont bien entendu plus nombreux et des opérateurs spécifiques, optimisés pour des langages cibles ou des relations d'ordre particuliers ont été développés, mais leur description et leur comparaison dépassent le cadre de ce document.

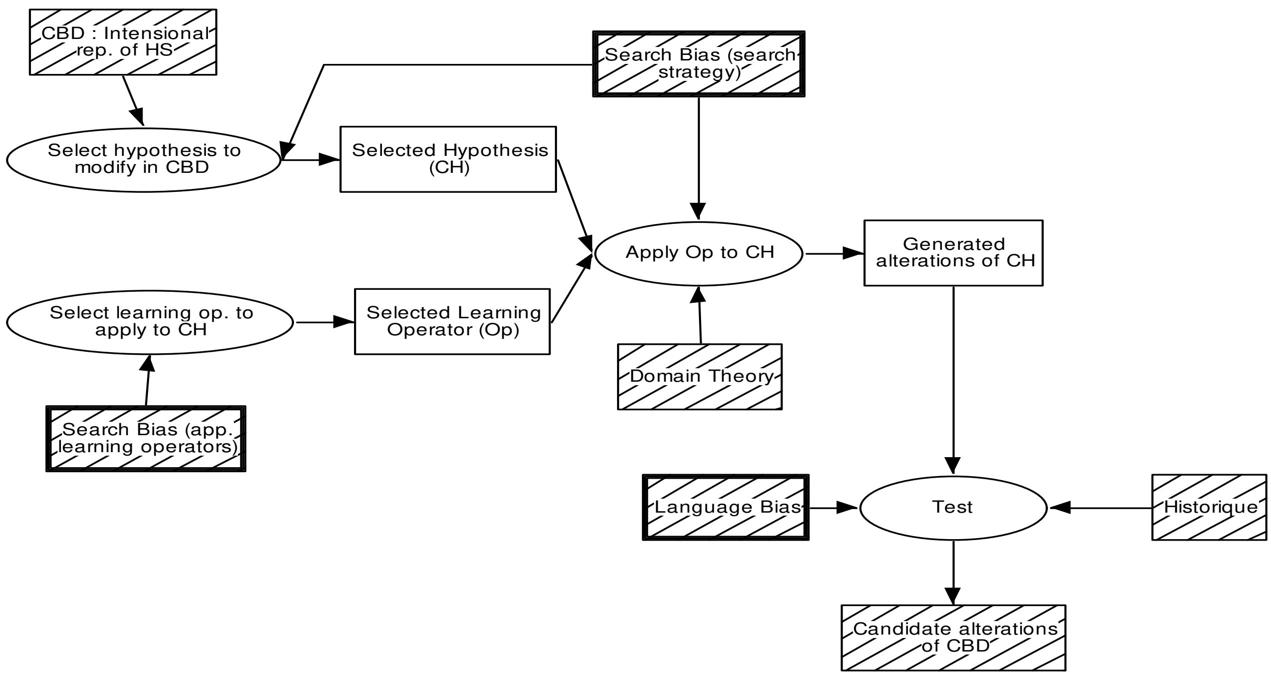

FIG. 2.5 – *Génération des transformations candidates de CBD*

informées, meilleur d'abord [Mug95], gradient [Qui90]. Comme le faisait remarquer D. Page dans une récente présentation invitée [Pag00], pour l'instant les stratégies de recherche restent relativement classiques et déterministes, avec des exceptions notables comme [GS93, AVK95, TNM00], qui implantent des stratégies de recherche de type algorithme génétique.

– le «pas» de l'opérateur. Certains systèmes définissent des «macro-opérateurs» d'apprentissage, qui correspondent à plusieurs applications successives d'un des opérateurs élémentaires signalés plus haut. Ceci permet de ne pas parcourir exhaustivement l'espace de recherche de l'opérateur *«générer et tester»* et offre un solution au problème du «plateau», qui est une des limitations des techniques *«générer et tester»*.

  On admettra qu'une recherche heuristique se fonde sur un score associé à chaque hypothèse de l'espace de recherche, en général une combinaison linéaire du nombre d'exemples positifs et négatifs de l'ensemble d'apprentissage couverts par l'hypothèse à évaluer. Même si l'on dispose d'un opérateur strict [9], il peut exister dans l'espace de recherche d'un opérateur de larges zones contenant des hypothèses ayant un même score d'évaluation identique sur l'ensemble d'apprentissage [10] : le système d'apprentissage peut donc ainsi errer dans des plateaux de la fonction d'évaluation. Une approche intéressante de ce point de vue est la notion de réductions empiriquement conservatives [CBS95], qui permet d'assurer qu'un système d'apprentissage n'explore que des hypothèses non équivalentes du point de vue de la couverture de l'ensemble d'apprentissage. Une autre technique, classique en Intelligence Artificielle, est celle du *lookahead*, qui préconise le développement de l'arbre

9. Un opérateur strict ne génère jamais deux hypothèses équivalentes du point de vue logique, voir annexe.

10. Ceci, soit parce que le langage des hypothèses est mal adapté et contient de nombreux prédicats non pertinents, soit parce que la fonction d'évaluation est inadéquate, soit enfin parce que l'ensemble d'apprentissage est trop dispersé.

de recherche sur plusieurs niveaux afin de de diminuer la sensibilité aux plateaux ou aux maxima locaux. Une version amoindrie du lookahead total est celle des *clichés relationnels* proposée par [SP91] : certains motifs, contenant plusieurs littéraux sont explorés à la demande, en particulier les motifs introduisant de nouvelles variables existentielles [Qui91, DRBDVL00].

Enfin, concluons cette section sur la génération des hypothèses candidates par un dernier point sur le contrôle des opérateurs ascendants ou descendants. L'opérateur d'abandon de littéral est relativement facile à contrôler, car, la clause sur laquelle il s'applique ayant un nombre fini de littéraux, il a un espace de recherche fini. L'opérateur d'ajout de littéraux est par contre beaucoup plus délicat à contrôler, le nombre des littéraux ajoutables étant potentiellement infini. Les opérateurs descendants ont néanmoins la faveur des développeurs de systèmes de PLI les plus récents, sans doute parce que les clauses les plus courtes et donc les plus simples sont plus vite atteintes par une stratégie descendante que par une stratégie ascendante.

La manière classique de contrôler une recherche du plus général au plus spécifique est l'utilisation d'une borne inférieure de l'espace de recherche [Mug95]. Cette borne, en général construite à partir d'un exemple positif «graine», n'est pas transformée mais elle sert de filtre ou de «générateur» aux littéraux ajoutables : un littéral ajouté afin de spécialiser une clause incorrecte appartenant à la borne supérieure courante de l'espace de recherche généralise nécessairement un littéral de cette borne inférieure. Ce biais s'accompagne souvent d'un biais syntaxique sur la longueur maximum des clauses du concept cible, ce qui achève de rendre l'espace de recherche fini.

Élagage avant évaluation contre les exemples

Détaillons l'élagage des clauses générées avant évaluation contre l'ensemble d'apprentissage. L'élagage classique [Mit82] en apprentissage exploite les échecs des tests de couverture d'hypothèses déjà explorées au cours de la recherche. Nous détaillerons dans la suite l'élagage pour un algorithme d'apprentissage ascendant [11]. La couverture d'un seul exemple négatif (si le biais de validation d'une hypothèse est la correction vis-à-vis de l'ensemble d'apprentissage) entraîne l'arrêt immédiat du développement d'une hypothèse par un opérateur ascendant. En effet, comme l'ensemble des exemples couverts par une hypothèse H est inclus dans l'ensemble des exemples couverts par une quelconque de ses généralisations, si H ne satisfait pas le critère de correction, aucune de ses généralisations ne pourra le satisfaire. Les hypothèses détectées comme incorrectes sont stockées dans un historique de l'apprentissage ; chaque nouvelle hypothèse générée est testée et éventuellement élaguée contre cet ensemble d'hypothèses incorrectes. Une hypothèse plus générale qu'une hypothèse incorrecte sera donc rejetée sans qu'il soit nécessaire d'avoir recours au coûteux test contre l'ensemble d'apprentissage.

Empiriquement, le test contre cet historique est moins coûteux que le test de couverture sur l'ensemble d'apprentissage. Les hypothèses candidates ayant passé avec succès ce filtre doivent enfin être évaluées contre l'ensemble d'apprentissage (l'historique, étant construit incrémentalement au cours de l'apprentissage, ne saurait se substituer complètement à l'ensemble d'apprentissage). Les hypothèses validées à l'issue du test contre les exemples permettront la mise à jour de la borne courante de l'espace de recherche, alors que les hypothèses échouées seront stockées dans l'historique.

11. Le traitement de l'élagage est dual pour un système descendant, le critère d'élagage étant la non couverture d'un exemple positif pour un langage de concept conjonctif, ou d'un seuil fixé d'exemples positifs, pour un langage de concept disjonctif.

Évaluation contre les exemples

Nous ne détaillerons pas cette étape, ni les biais qui la gouvernent. Elle s'appuie principalement sur une procédure de test de subsomption, qui, sauf dans des langages de concept extrêmement simples est NP-complète. Illustrons simplement sur un exemple simple la complexité du test de θ-subsomption. Soit une hypothèse $H : c \leftarrow p(X,Y), p(Z,T)$ et un exemple $E : c \leftarrow p(a,b), p(a,d), p(a,f)$. Il existe un nombre exponentiel (3^2) de substitutions telles que H θ-subsume E. Le choix et l'implantation du test de subsomption sont bien sûr directement liés au choix du langage des hypothèses et plus marginalement au langage des exemples. Cette étape est de loin la plus coûteuse de toutes dans un système *«générer et tester»* (voir section 2.2), ce qui explique tous les travaux afin, d'une part d'optimiser ce test de subsomption, et d'autre part, de développer des recherches heuristiques permettant d'atteindre au plus vite la ou les hypothèses cibles.

Le modèle des systèmes *«générer et tester»* que nous venons de décrire permet donc à un utilisateur d'avoir une idée relativement claire du fonctionnement de ces systèmes, au niveau global tout d'abord, puis de façon de plus en plus détaillée s'il le désire, jusqu'à atteindre les feuilles d'une décomposition, qui correspondent à des opérations élémentaires, comme la généralisation d'une hypothèse ou le test de subsomption. La compréhension de ce fonctionnement est de notre point de vue une pré-requis incontournable de la bonne utilisation d'un système de PLI, argumentation que nous développerons à la section 2.6.

Nous allons maintenant décrire dans la suite deux développements autour de ce modèle : le premier autour d'une meilleure intégration des biais de langage dans le paradigme *«générer et tester»*, le second concernant le développement d'un système d'apprentissage *«générer et tester»* configurable pour faciliter l'utilisation de système de PLI en situation réelle.

2.5 Biais de langage et recherche *«générer et tester»*

Nous présentons dans cette section le travail de Fabien Torre, que j'ai encadré et auquel j'ai collaboré. Cette recherche a été présentée dans [Tor95, TR97a], et nous renvoyons à [Tor00] pour un exposé détaillé des points techniques, dont nous présentons ici un résumé des résultats.

Nous nous intéressons ici aux biais de langage du point de vue du développeur du système de PLI, et nous nous posons en particulier la question suivante : sous quelles conditions un biais de langage permet-il d'améliorer les performances d'un système de PLI à approche générative (voir section 2.4.2)? Plus précisément, nous avons vu à la section précédente que la couverture d'un exemple négatif ou la non couverture d'un exemple positif permettait d'élaguer de manière admissible (c'est-à-dire sans perte de solution) la recherche dans l'espace des hypothèses. Peut-on étendre ce type d'élagage à certains biais de langage, et lesquels? On se restreint ici aux biais de langage qui s'expriment comme une propriété sur une hypothèse de la forme $f(H)\ \mathcal{R}_P\ k$, avec f une fonction de $\mathcal{L}_h$ dans un domaine quelconque $\mathcal{D}_f$ ($\mathcal{D}_f$ est le plus souvent $\mathbb{N}$ ou $\{vrai, faux\}$) et k est un paramètre de la propriété. Un biais de langage devient donc une propriété explicite d'une hypothèse, au même titre que sa complétude et sa correction vis-à-vis de l'ensemble d'apprentissage. Nous renvoyons à [Tor00] pour l'expression de la plupart des biais communément employés en PLI sous cette forme; donnons simplement ici l'exemple de la taille en nombre de littéraux d'une clause, qui s'exprime par $|H| \leq n$.

Soit $\mathcal{R}_O$ un opérateur d'apprentissage vu comme une relation de $\mathcal{L}_h$ dans $\mathcal{L}_h$. Préalablement à la caractérisation de l'élagage admissible de $\mathcal{R}_O$ selon une propriété P, introduisons la notion de propriété privée.

Définition 2.5.1 (Propriété privée)
Une propriété P est dite privée par rapport à une relation $\mathcal{R}_O$ si et seulement si, pour tout couple d'hypothèses (H, H') de $\mathcal{L}_h$ en relation par $\mathcal{R}_O$ et pour toute valeur du paramètre de la propriété P, le fait que H ne vérifie pas P implique que H' ne la vérifie pas non plus.

Le corollaire direct de cette définition est que l'élagage de $\mathcal{R}_O$ selon une propriété P (biais de langage, couverture ou rejet d'un exemple de l'ensemble d'apprentissage) n'est admissible que si P est privée par rapport à $\mathcal{R}_O$. Dans ce cas (voir figure 2.6), dès que l'opérateur $\mathcal{R}_O$ sort de la zone de l'espace de recherche où P est satisfaite, on est assuré de pouvoir élaguer la recherche par rapport au biais de langage P sans perte de solution.

Malheureusement, bien peu des biais de langage PLI sont privés pour les opérateurs d'apprentissage habituellement utilisés en PLI, qui satisfont déjà un ordre partiel de généralité. Notons cependant que la propriété qui fixe une borne supérieure sur la taille d'une clause est privée pour l'opérateur d'ajout de littéral, ce qui explique, par delà sa simplicité, sa large utilisation dans les systèmes d'apprentissage descendants [Mug95]. Force est donc de constater que les biais de langage dans un cadre *«générer et tester»* classique ne permettent pas un élagage fort. Seuls les biais privés doivent être testés au moment de la génération des hypothèses (voir section 2.4.3), ce qui ne permet pas d'améliorer fondamentalement les performances de l'apprentissage *«générer et tester»*.

Un changement de point de vue est cependant possible, afin de tenter d'utiliser les biais de langage pour un élagage dynamique de l'espace de recherche : étant donné un ensemble de propriétés fixées $P_1, \ldots, P_n$, existe-t-il des opérateurs qui rendent ces propriétés privées, permettant donc un élagage dynamique de l'espace de recherche par rapport à celles-ci ? Ces opérateurs sont à rechercher dans la *relation naturelle* de ces propriétés.

Définition 2.5.2 (Relation naturelle)
Étant donné un espace de recherche $\mathcal{L}_h$, la relation naturelle d'une propriété P dans $\mathcal{L}_h \times \mathcal{L}_h$ est le plus grand sous-ensemble de $\mathcal{L}_h \times \mathcal{L}_h$ tel que P soit privée.

F. Torre a prouvé qu'une propriété P de la forme $f(H)\ \mathcal{R}_P\ k$ a une *unique* relation naturelle, notée $\succeq_P$, et qu'elle est définie par

$$\forall H, H' \in \mathcal{L}_h : H \succeq_P H' \text{ si et seulement si } f(H)\ \mathcal{R}_P\ f(H')$$

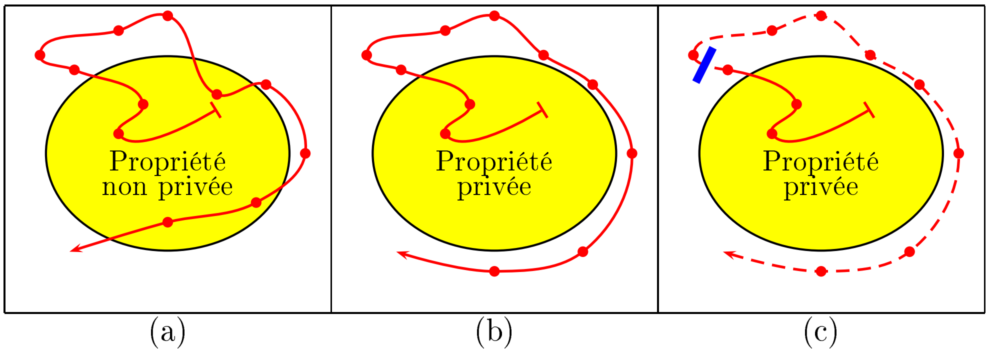

FIG. 2.6 – *Propriétés privée et non privée. La figure (a) montre que l'élagage sûr n'est pas possible par rapport à une propriété non privée. Dans le cas (b), l'élagage est sûr et (c) exhibe sa réalisation pratique.*

Exemple 2.5.1
Considérons comme propriété la couverture d'un exemple positif e*, notée* $H \models e$*. On en déduit immédiatement qu'un opérateur permettant d'élaguer sa recherche si un exemple positif n'est plus couvert doit être inclus dans la relation naturelle de la couverture, soit dans l'ensemble des couples* H, H' *tels que* $H \models H'$ *: l'opérateur doit donc être descendant (voir section 2.2). Dualement, si l'on désire élaguer les descendantes d'une hypothèse qui couvre un exemple négatif, l'opérateur d'apprentissage doit nécessairement satisfaire la relation naturelle duale : pour tout* H, H' *tels que* $H' = \mathcal{O}(H)$*,* $H' \models H$*, l'opérateur doit donc être ascendant.*

La formalisation choisie pour la relation naturelle permet de caractériser simplement la relation naturelle de la conjonction de plusieurs propriétés (l'intersection de leurs relations naturelles respectives) ainsi que celle de la négation d'une propriété (le complémentaire dans $\mathcal{L}_h \times \mathcal{L}_h$ de sa relation naturelle), ce qui est suffisant pour caractériser les relations naturelles des combinaisons de propriétés qui nous intéressent.

Un opérateur pouvant élaguer de façon admissible par rapport à une propriété P doit être inclus dans la relation naturelle de cette propriété. Cette relation a volontairement été choisie la plus large possible, afin de laisser la plus grande latitude *a posteriori*, c'est-à-dire au niveau de la définition de l'opérateur uniquement, afin de définir un type de parcours particulier de l'espace de recherche.

Nous nous sommes en particulier fixé le cadre suivant : nous avons recherché des opérateurs *idéaux* (voir section 2.2 et annexe) dans des relations naturelles de conjonctions de propriétés faisant intervenir la complétude ou la correction par rapport à l'ensemble d'apprentissage. Nous considérons comme ordre partiel de généralité la θ-subsomption, restriction de l'implication logique (voir annexe 5). Les relations naturelles des conjonctions de propriétés seront notées $\succ^{\theta}_{f}$, où $\succ^{\theta}$ représente la θ-subsomption et où f correspond à la fonction f de la propriété $P = f(H)\ \mathcal{R}_P\ k$. Par exemple, la relation d'ordre $\succ^{\theta}_{||}$ est définie par $H \succ^{\theta}_{||} H'$ ssi $H \succ^{\theta} H'$ et $|H| \leq |H'|$.

Il n'existe pas d'opérateur idéal pour les espaces de clauses ordonnés par la θ-subsomption. Van Der Laag et Nienhuys-Cheng, dans [vdLNC94a], proposent deux conditions de non-existence d'un opérateur idéal pour un ordre de généralité $\succ$. Ces conditions de non-existence s'appuient sur la constatation qu'un opérateur idéal calcule au minimum pour chaque hypothèse de $\mathcal{L}_h$ un *ensemble couvrant* de cette hypothèse.

Définition 2.5.3
On appelle couverture supérieure (resp. inférieure) d'une hypothèse $H \in \mathcal{L}_h$*, une hypothèse* $H' \in \mathcal{L}_h$ *telle que* $H' \succ H$ *(resp.* $H \succ H'$*) et il n'existe pas d'hypothèse* $H'' \in \mathcal{L}_h$ *telle que* $H' \succ H'' \succ H$ *(resp.* $H \succ H'' \succ H'$*).*
Un ensemble couvrant supérieur (resp. inférieur) de H *pour un ordre de généralité* $\succ$ *est un ensemble maximal de couvertures supérieures (resp. inférieures) de* H *incomparables deux à deux.*

Intuitivement, la première de ces conditions (suffisantes) de non-existence d'un opérateur idéal est qu'il est impossible de caractériser cet ensemble couvrant. Ceci vient du fait qu'il peut exister dans $\mathcal{L}_h$ des chaînes infinies (ascendantes ou descendantes) non couvertes (voir figure 2.7). Par exemple, et pour une chaîne infinie descendante, il existe un nombre infini d'hypothèses $D_1 \succ \ldots \succ D_n$ qui sont plus générales qu'une hypothèse C de $\mathcal{L}_h$, et aucune couverture supérieure de C qui soit couverte par toutes les D_i. La figure 2.8 exhibe une chaîne infinie descendante sous θ-subsomption dans un langage Datalog.

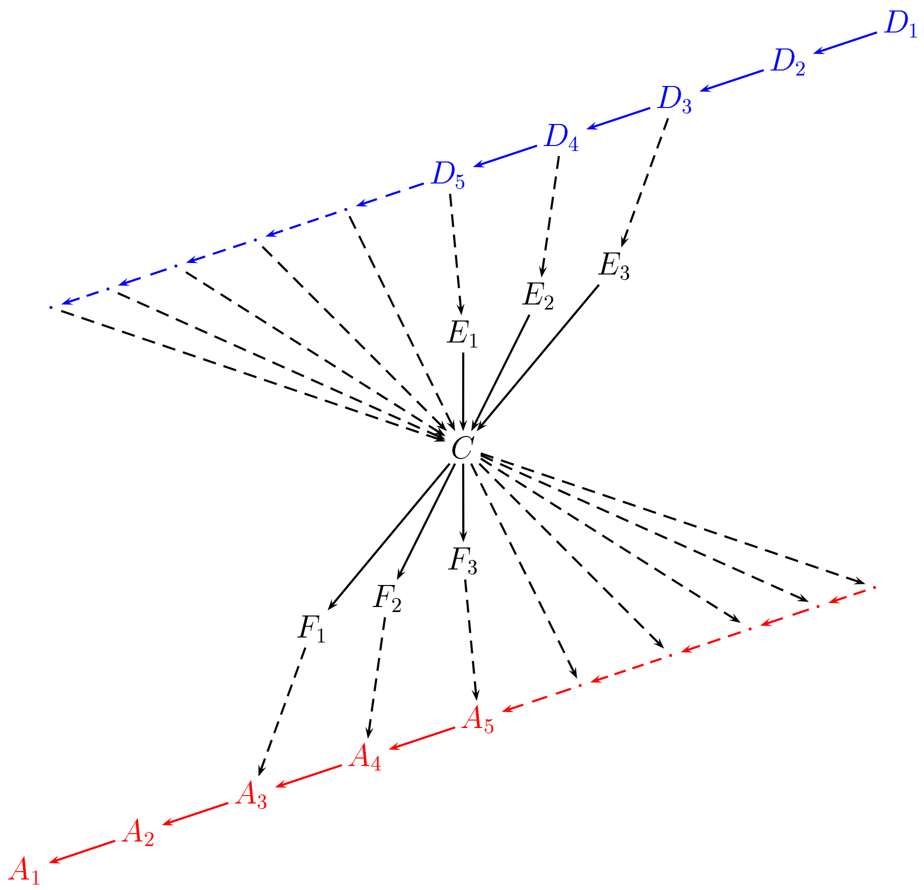

FIG. 2.7 – *Chaînes infinies non couvertes*

La deuxième condition de non existence d'un opérateur idéal est qu'il peut exister dans $\mathcal{L}_h$ des hypothèses ayant un ensemble couvrant inférieur ou supérieur infini (ce cas est beaucoup plus difficile à exhiber et nous renvoyons à [NCdW97] pour un exemple d'ensemble couvrant inférieur infini sous θ-subsomption).

En miroir à ces deux conditions de non-existence d'opérateur idéal pour la θ-subsomption, nous pouvons donc énoncer deux conditions nécessaires pour l'existence d'un opérateur idéal dans des espaces ordonnés par une relation $\succ_f^\theta$.

- un certain nombre de quantités associées à une clause augmentent à l'infini sur les chaînes infinies ascendantes et descendantes, comme le nombre de variables existentielles, le nombre d'occurrences des symboles de prédicats/fonctions, la taille. Afin de briser ces chaînes infinies, la propriété f doit nécessairement borner l'une au moins de ces quantités. Les autres biais de langage, comme la profondeur [12] sont inopérants pour la suppression de chaînes infinies pour la θ-subsomption.

- des ensembles couvrants inférieurs infinis pouvant exister dans les langages cible avec symboles de fonctions, même si l'on borne le nombre de variables existentielles, il est nécessaire de borner au moins une quantité parmi le nombre d'occurrences des symboles de prédicats/fonctions dans la clause ou la taille de la clause.

Ces conditions d'existence d'un opérateur idéal pour $\succ_f^\theta$ ne sont que nécessaires, une fois un opérateur satisfaisant ces conditions construit, il reste encore à prouver qu'il est effectivement idéal. Nous renvoyons à [TR97b] et à [Tor00] pour la définition d'un opérateur idéal pour $\succ_{||}^\theta$.

En résumé, cet axe de recherche a permis certes de définir de nouveaux opérateurs idéaux, de définir un nouveau biais Max_Occ qui permet de borner le nombre d'occurrences de symboles de fonctions ou de prédicats dans une clause [13] mais surtout, il a produit un modèle original de la recherche pour un système *«générer et tester»* où les biais de langage sont introduits dès la définition de l'opérateur d'apprentissage : le biais de langage est donc utilisé également pour *ordonner* l'espace de recherche et n'est plus uniquement une contrainte syntaxique à satisfaire par une hypothèse.

Dans sa thèse, Fabien Torre a poursuivi une réflexion plus prospective sur une meilleure utilisation des biais de langages. D'une part, il a observé que certaines associations de biais de langage conduisent à la définition d'une relation naturelle trop pauvre pour permettre un parcours de l'espace de recherche, et donc l'apprentissage. D'autre part, la notion d'idéalité est encore trop faible pour déboucher sur l'implantation d'opérateurs réellement efficaces. Il s'est donc orienté vers la notion, plus prometteuse, d'opérateur optimal [DRB93], qu'il a élargie aux cas où la recherche n'est pas nécessairement descendante à partir de la clause vide. Les opérateurs qu'il a définis comme *parfaits* (nous renvoyons à [Tor00] pour une description détaillée de ces opérateurs) utilisent au maximum les biais de langages, certains pour l'élagage dynamique de l'espace de recherche, certains pour la construction de bornes initiales de l'espace de recherche assurant *in fine* un parcours complet et optimal de l'espace de recherche.

12. On sait que pour toute clause C telle que $C \succ^\theta D$, $prof(C) \leq prof(D)$. La profondeur ne peut donc pas être utilisée pour «casser» une chaîne infinie descendante.

13. Bien que cet axe n'ait pas été développé dans sa thèse, tout laisse à penser que ce biais, qui peut également permettre de limiter le «non-déterminisme» d'une clause, pourrait servir de base à la définition d'un nouveau langage PAC-apprenable, de même que la notion de k-localité [KL94].

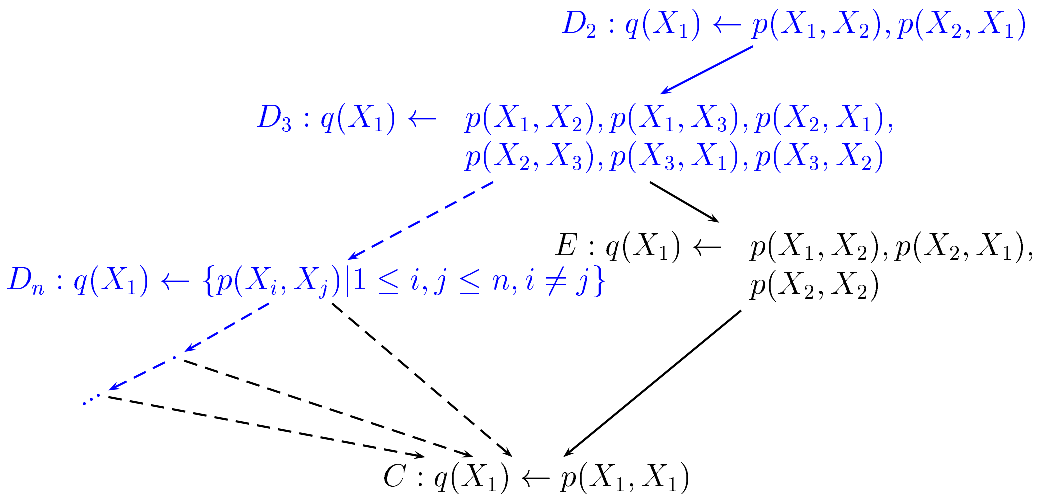

FIG. 2.8 – *Chaînes infinies non couvertes sous θ-subsomption. $\{D_i\}_{i \geq 2}$ est une chaîne infinie non couverte de C.*

2.6 Un outil de PLI configurable

Nous présentons dans cette section un autre développement du modèle présenté aux sections 2.3 et 2.4. Ce modèle est un premier pas vers un système d'apprentissage configurable : il décrit des décompositions de tâches complexes en tâches plus simples, des agencements de ces tâches, et les biais qui les contrôlent. Dans ce cadre, l'acquisition des biais adaptés devient un aspect crucial de la modélisation du problème d'apprentissage automatique, au même titre que la sélection de l'ensemble d'apprentissage.

Choisir une décomposition spécifique ou le biais adéquat est un problème difficile, et même si le modèle fourni peut servir de base à un module explicatif, la configuration réclame une expertise dans les mécanismes d'apprentissage sous-jacents. C'est la réinterprétation au niveau du processus d'apprentissage du fameux «goulet d'étranglement» de Feigenbaum [Fei77].

Devant la complexité des systèmes de PLI, le foisonnement des biais mis en œuvre afin de rendre leur fonctionnement efficace, ainsi que la complexité des applications à base de connaissances nécessitant une approche PLI comme la découverte dans les bases de données relationnelles (voir chapitre 5), on conçoit sans peine la difficulté d'un utilisateur. Étant donné un problème d'apprentissage de concepts, de multiples systèmes de PLI peuvent être utilisés, chacun avec ses caractéristiques, au niveau du langage du concept cible, des techniques d'apprentissage mises en œuvre, des caractéristiques attendues de l'ensemble d'apprentissage, etc. Il n'existe néanmoins pas de méthodologie concernant l'application des systèmes de PLI ou leur intégration dans une boucle d'acquisition des connaissances, dans laquelle le système de PLI interagit avec d'autres systèmes d'Acquisition des Connaissances utilisant des techniques plus manuelles ou avec des éléments de performance, comme des résolveurs de problèmes avancés.

Dans un autre cadre que celui de la PLI, le contrat ESPRIT Machine Learning Toolbox [MLT93] avait déjà abordé ce problème ; un des résultats de ce projet a été le développement d'une «boîte à outils» regroupant un ensemble de systèmes d'apprentissage très divers (allant d'outils d'analyse de données à des systèmes multi-stratégie de modélisation de théorie en ordre un comme MOBAL [MWKE93]) dans un environnement unifié, afin que ces systèmes puissent collaborer à la résolution d'un même problème d'apprentissage. Cet environnement, entre autres, définissait une interface unifiée pour tous les algorithmes, ainsi qu'un langage commun pour les entrées-sorties de tous les systèmes, qui pouvaient donc partager les informations fournies par l'utilisateur ainsi que leurs résultats. Enfin un «Consultant», implémenté comme un système expert, aidait l'utilisateur à sélectionner les algorithmes qui pouvaient résoudre ses besoins spécifiques en terme d'application.

Depuis, des systèmes faisant appel à des librairies de méthodes d'apprentissage ou d'analyse de données, avec un niveau encore moindre d'intégration (architectures *plug-in*) ont été développés (parmi eux, KEPLER [WWSE96], MLC++ [KS96], Clementine, par ISL). Tous ces systèmes partagent un même point de vue : le niveau d'intégration des différents algorithmes constituants est faible, les «blocs de base» étant des systèmes d'apprentissage complets eux-mêmes. Le corollaire est qu'ils intègrent des systèmes très divers, et offrent dans la plupart des cas un spectre relativement représentatif de l'apprentissage et parfois de l'analyse de données. Pour certains d'entre eux, il existe un module puissant de visualisation des résultats, aidant l'analyse de ceux-ci. Par contre, l'aide apportée au choix d'un algorithme spécifique parmi les algorithmes disponibles, ou le choix des paramètres de ces algorithmes reste faible, et demande une connaissance préalable des techniques sous-jacentes. Les données partagées par les systèmes sont au mieux les seules entrées-sorties de chacun des systèmes d'apprentissage.

Dans le domaine de l'acquisition de connaissances, un travail important de recherche a été réalisé sur les modèles génériques en Acquisition des Connaissances [BVdV93, VHTWS92]. Plus proche de notre problématique, [VdV94, Slo94] décrit un «kit d'apprentissage» configurable développé sous KREST (voir section 2.3.3), implémentant une famille d'algorithmes de type arbre de décision [Qui86] et fondé sur un modèle au niveau connaissances de ces différents algorithmes. Ce travail permet d'obtenir un système opérationnel à partir de l'assemblage de code LISP, en utilisant les fonctionnalités de génération de code de KREST. Par ailleurs, KREST n'offre pas la possibilité de représenter différentes décompositions pour une tâche. Dans un autre registre, citons [TGL93] qui utilise le niveau inférence de KADS pour contrôler l'apprentissage au niveau domaine. Enfin, mentionnons [Cau94b] qui élabore un statut et une représentation de l'information de contrôle au sein d'une méthodologie d'IC, qui est similaire à notre souci de représentation du biais et de ses influences en apprentissage.

2.6.1 Problématique

Nous nous concentrons ici sur le choix et le paramétrage d'outils de PLI, sans aborder les problèmes d'intégration d'un tel système d'apprentissage dans une boucle d'IC [14].

Dans ce contexte, l'utilisateur doit donc itérer jusqu'à satisfaction ou épuisement la boucle suivante :

1. sélectionner un système de PLI d'après sa description/documentation,
2. assigner les biais qui contrôlent les performances du système,
3. exécuter le système - analyser les résultats, s'arrêter avec succès ou retourner à étape 2 ou même 1.

Chacune des étapes de cette boucle est délicate, et leur itération essai-erreur peut être fastidieuse. Nous avons adopté une approche originale dans l'objectif de concevoir un outil d'apprentissage *configurable* couvrant la famille des systèmes d'apprentissage *«générer et tester»*. Notre but est par ce système de proposer à la fois une solution au choix du système d'apprentissage

14. La problématique de l'interaction entre Apprentissage et Acquisition ou Ingénierie des Connaissances a suscité un vif intêret et de nombreux débats au début des années 90. Nous renvoyons à [Néd95] pour une synthèse.

de PLI ainsi qu'une plate-forme d'expérimentations pour l'ajustement de ses biais. Un système d'apprentissage est construit par l'assemblage et le paramétrage d'opérations de base élémentaires. Cette solution offre des possibilités d'adaptation et de variations plus fines qu'avec des systèmes boîtes à outils, de par l'intégration et la composition de composants de base à faible granularité. Le spectre des systèmes d'apprentissage disponibles dans un tel outil configurable est par contre moins large que pour les systèmes boîte à outils cités précédemment, car nous présupposons, par opposition, que les systèmes d'apprentissage que notre architecture peut émuler partagent un même modèle de fonctionnement de haut niveau (voir figure 2.4). Ce modèle de fonctionnement commun permettra en outre d'offrir une aide plus précise à l'utilisateur lors de la configuration de cet outil.

Plus précisément un tel outil est :

- adaptable : face à un problème d'apprentissage spécifique décrit en termes de caractéristiques de l'ensemble d'apprentissage, de la théorie du domaine, de connaissances disponibles sur le ou les concepts à apprendre, l'utilisateur va pouvoir assembler un ensemble de procédures d'apprentissage élémentaires adaptées (par opposition à des méthodes ou systèmes complets dans des systèmes de type boîte à outils) et contrôler chacune de celles-ci finement s'il le désire. Une configuration définit un système d'apprentissage spécifique. La possibilité de re-configurer le système générique en cours d'apprentissage permet d'adapter la recherche de façon plus opportuniste qu'avec une librairie de méthodes. Sélectionner le système d'apprentissage approprié pour une situation spécifique revient donc à trouver une configuration adaptée du système, évitant les problèmes d'installation, test, traduction des données et des résultats intermédiaires.

- simple et homogène : la configuration d'un système d'apprentissage se fait par l'assemblage d'un ensemble prédéfini d'opérations d'apprentissage et en instanciant les biais relatifs aux composants choisis. Un ensemble de notations graphiques représentant les décompositions possibles de chaque étape de l'algorithme d'apprentissage rend possible de spécifier graphiquement une configuration de façon intuitive, ce qui simplifie grandement la phase d'expérimentations avec différents algorithmes d'apprentissage, ou différentes variations d'un même algorithme.

2.6.2 Stratégies de configuration

C'est un sujet relativement peu étudié en PLI et en apprentissage en général : l'adaptation du système d'apprentissage à une tâche d'apprentissage spécifique est à la charge de l'utilisateur, en collaboration avec le développeur du système.

La configuration dans notre cas s'effectue, sur la base du modèle au niveau connaissances décrit, en choisissant les décompositions de tâches adaptées et en instanciant de façon adéquate les paramètres de contrôle ou biais. Le système d'apprentissage instancié est l'ensemble des opérations d'apprentissage connectées entre elles, et les valeurs pour leur paramètres de contrôle associés.

Rappelons le dilemme de l'étape de configuration : si le biais est fort (c'est-à-dire si l'on dispose de beaucoup de contraintes sur la forme du concept cible et/ou la stratégie heuristique de parcours de l'espace de recherche), l'apprentissage peut s'effectuer rapidement, mais au risque d'écarter la ou les solutions du langage cible ou encore de ne pas atteindre la définition cible si elle appartient à l'espace de recherche. Par ailleurs, si le biais est trop faible, on augmente la probabilité que la définition du concept cible appartienne à l'espace de recherche, mais également la complexité de la recherche et le risque d'épuiser les ressources de calcul (ou la patience de l'expert) avant d'atteindre le résultat.

La stratégie la plus couramment employée [Utg86, DR92, ADRB95] est d'utiliser un biais fort, puis de relaxer le biais si l'apprentissage débouche sur un échec [15].

2.6.3 Une implémentation

Nous avons développé le système configurable $\mathcal{HAIKU}$ sur la base du modèle décrit dans les sections précédentes, en Prolog Eclipse. Le traitement des biais de langage a été implanté par F. Torre, au cours de son stage de DEA. Une interface graphique a été développée en Tcl / Tk par D. Faure pendant une partie de son stage de DESS pour assister la configuration par l'utilisateur. Une implantation modulaire permet une extension rapide et facile des capacités du système en intégrant de nouvelles opérations, comme nous le verrons au chapitre 5.

Au delà de l'environnement graphique, $\mathcal{HAIKU}$ fournit une batterie d'outils d'inspection de certains paramètres de l'apprentissage permettant d'observer les performances du système, tant en terme de complexité (profilage des diverses opérations, taille de la borne, nombre d'hypothèses générées, nombre d'hypothèses testées contre les exemples, etc.), qu'en terme de résolution du problème d'apprentissage (hypothèses ayant échoué au niveau du test contre l'ensemble d'apprentissage, couverture de l'ensemble d'apprentissage par une hypothèse déjà développée dans l'espace de recherche, etc.). Le type et le niveau de détail des informations fournies à l'utilisateur sont sélectionnés par lui, et modifiables à tout instant d'une session d'apprentissage.

Cet outil permet à un développeur d'applications intégrant un outil d'apprentissage de PLI de prototyper rapidement un système d'apprentissage de PLI adapté à une application spécifique, en testant par exemple les performances de diverses instanciations du système sur des échantillons de la base d'apprentissage.

2.7 Bilan

Ce chapitre a présenté un ensemble de travaux qui s'articulent autour d'une idée directrice : la maîtrise de la complexité de l'apprentissage PLI dans le cadre d'une stratégie *«générer et tester»*. Une première partie a présenté un modèle détaillé, générique et opérationnel de systèmes *«générer et tester»*, où les biais apparaissent comme des paramètres d'opérations élémentaires d'apprentissage discriminant.

Notre initiative de modélisation opérationnelle n'est bien sûr pas isolée en PLI. Citons en particulier l'algorithme générique GENCOL ([DRB92], repris dans [MDR94]). Ce modèle n'a pas, à notre connaissance, donné lieu à une implémentation et reste donc un outil théorique pour la description de systèmes de PLI. Le lien que nous avons établi avec les travaux et les outils de modélisation au niveau connaissances développés dans la communauté d'ingénierie des Connaissances est également notre originalité par rapport à GENCOL.

Dans une deuxième partie, nous avons présenté un cadre formel permettant d'élargir l'influence des biais de langage dans un système à stratégie *«générer et tester»*: il sont utilisés bien sûr pour tester les hypothèses candidates, mais également pour ordonner l'espace de recherche et peuvent donc servir à élaguer dynamiquement celui-ci. Un des résultats théoriques de ce travail est que la combinaison de certains biais de langage avec la θ-subsomption permet de casser

15. Signalons que, contrairement à [Utg86, BM92, Wro94a], nous ne relaxons pas le biais de langage en introduisant de nouveaux prédicats.

ces chaînes infinies en permettant de borner des quantités qui croissent infiniment le long de ces chaînes. Un autre résultat est la définition d'opérateurs idéaux pour ces ordres complexes. L'obtention d'opérateurs idéaux n'est pas en soi un résultat résolument novateur, l'originalité est plutôt l'explicitation du raisonnement qui a servi à les construire.

Enfin, dans une dernière partie, nous avons montré que le modèle que nous avons développé pouvait servir de base à l'implémentation d'un système d'apprentissage *«générer et tester»* configurable. Dans le cadre de la PLI, plusieurs systèmes sont paramétrables en terme de biais déclaratifs [NRA+96]. Citons ici le système NINA [ADRB95], et dans une version préliminaire [RADR93], qui formalise les biais de langage des systèmes ascendants de PLI. NINA implémente un système qui permet, à travers ce qui allait devenir $\mathcal{DLAB}$ [DDR96], de paramétrer les biais de complétion d'un système d'apprentissage discriminant ascendant. CLAUDIEN [DRD97], puis WARMR et TILDE utilisent intensivement $\mathcal{DLAB}$ afin de paramétrer le biais de langage dans le cadre d'un système d'apprentissage descendant. Citons MILES-CTL [Tau94] qui permet un paramétrage du biais de langage fondé sur l'idée de schémas d'ordre deux restreints, proches de ceux de Mobal [MWKE93]. Une comparaison détaillée peut être trouvée dans [NRA+96]. Soulignons que ces systèmes mettent tout particulièrement l'accent sur le paramétrage du biais de langage, et abordent plus superficiellement le paramétrage du biais de recherche, qui s'inscrit dans une perspective plus large d'apprentissage multi-stratégique.

Chapitre 3

Transformation Problème PLI - Problème Attribut Valeur

Nous avons étudié au chapitre précédent les systèmes d'apprentissage de type *«générer et tester»*, et nous avons en particulier observé que le contrôle de la complexité du processus d'apprentissage se réalise à travers la mise au point de biais de langage et/ou de recherche sophistiqués.

Dans ce chapitre, nous allons étudier plus en détail une famille d'approches complémentaires, représentée, entre autres, par les systèmes de PLI tels LINUS [LD94], STILL [SR97b, SR00], REPART [ZG98], SP [KPH98] et Propal [AR00]. Ces systèmes abordent la résolution d'un problème d'apprentissage relationnel en deux étapes. Dans un premier temps, ils reformulent le problème d'apprentissage PLI en un problème d'apprentissage propositionnel [1] ; une fois ce changement de représentation effectué, des algorithmes efficaces (espaces de recherche finis, heuristiques de recherche éprouvées, notamment en ce qui concerne le traitement du bruit) dédiés à ce formalisme plus simple peuvent être directement appliqués sur le nouveau problème reformulé. Si les caractéristiques discriminantes du problème relationnel initial ont été préservées par le changement de représentation, ce type d'approche constitue un excellent moyen de transférer l'expertise acquise en apprentissage propositionnel à la résolution d'un problème de PLI.

Ces changements de représentation, appelés *propositionnalisations* par la suite, procèdent de la manière suivante. Intuitivement, étant donné un *motif*, noté M, une formule de l'espace de recherche initial (relationnel), chaque exemple relationnel E_r est reformulé en l'ensemble des substitutions θ_i tel que M (ou un sous-ensemble de M) couvre E_r étant donné la substitution θ_i. L'ensemble d'apprentissage relationnel est donc transformé en une représentation tabulaire imposée, dans une première approximation, par les variables de M.

Mais, pour les mêmes raisons qui font que le test de couverture pour une relation de subsomption telle la θ-subsomption est NP-difficile, même pour des langages Datalog non récursifs et sans négation, l'ensemble d'apprentissage reformulé peut être de taille prohibitive, ainsi que hautement redondant [Alp00], pour des problèmes d'apprentissage dans des domaines faiblement structurés [SR00]. Il s'agit donc, afin de rendre ce type d'approches applicables à la résolution de problèmes relationnels, de contrôler d'une manière ou d'une autre la *taille* de l'ensemble d'apprentissage reformulé.

Nous allons suivre dans ce chapitre le plan suivant. Après une présentation intuitive du mé-

1. Par abus de langage, nous utiliserons le terme *propositionnel* dans la suite de ce document dans le sens *décrit dans ou manipulant un langage de la logique attribut-valeur ou booléenne* (voir annexe).

canisme de la propositionnalisation, nous présenterons deux systèmes à la conception desquels nous avons contribué, qui utilisent tous deux une approche fondée sur la propositionnalisation : le système *STILL*, en collaboration avec M. Sebag et le système *Propal*, en collaboration avec E. Alphonse. Nous comparerons leurs techniques de propositionnalisation respectives ainsi que la façon dont ils font face au problème de taille de l'ensemble d'apprentissage reformulé, chacun dans un contexte d'apprentissage différent. Nous conclurons sur des perspectives de recherche dans ce domaine.

3.1 Propositionnalisation

3.1.1 Lien apprentissage attribut-valeur – apprentissage PLI

Certains systèmes de PLI tels [Qui90, BDR98] étendent leurs techniques d'apprentissage, initialement développées dans un contexte propositionnel, afin de traiter des langages de concept plus complexes. En particulier, les systèmes développés à l'Université de Leuven [DRBDVL00] procèdent de la manière suivante. Les exemples ne sont pas, comme dans les méthodes propositionnelles classiques, les lignes d'un tableau attribut-valeur, mais des *interprétations de Herbrand*. La stratégie de recherche du système d'apprentissage propositionnel est conservée autant que possible, mais les opérateurs d'apprentissage et le test de couverture entre hypothèses et exemples sont modifiés afin de pouvoir traiter des langages relationnels. Les méthodes d'apprentissage propositionnelles sont donc sources d'inspiration de nouvelles stratégies d'apprentissage PLI, mais elles ne sont pas utilisées dans leur contexte initial pour résoudre un problème d'apprentissage relationnel.

Par opposition, la propositionnalisation est un changement de représentation qui construit effectivement des «projections» du problème d'apprentissage relationnel en un ensemble de problèmes propositionnels, ce qui va permettre d'utiliser les techniques propositionnelles directement dans leur contexte original pour la résolution d'un problème d'apprentissage relationnel.

3.1.2 Propositionnalisation d'un problème d'apprentissage

Soit T_r l'ensemble d'apprentissage relationnel, représenté comme un ensemble de clauses de type Datalog instanciées, soit $\succeq_r$ un ordre partiel sur un langage cible relationnel $\mathcal{L}_{hr}$, qui est un langage de type Datalog non récursif et sans négation. Nous considérons ici que $\succeq_r$ est la θ-subsomption ou l'une de ses restrictions qui autorisent l'existence d'opérateurs idéaux (voir annexe).

Le notion pivot de la propositionnalisation est celle de *motif de propositionnalisation*, noté M dans la suite de ce chapitre. Le motif de propositionnalisation est une clause de $\mathcal{L}_{hr}$, qui est soit fixée par un utilisateur, soit extraite d'un exemple positif de T_r, appelé «graine» de l'espace de recherche.

La propositionnalisation va permettre d'étudier les généralisations de M sous $\succeq_r$, et ceci dans un espace de recherche plus simple que $(\mathcal{L}_{hr}, \succeq_r)$. Pour ce faire, elle calcule, pour chaque exemple E_r de l'ensemble d'apprentissage relationnel, l'ensemble des substitutions θ_i telles que M (ou un sous-ensemble de M) couvre E_r selon la relation de couverture $\succeq_r$. Chaque exemple d'apprentissage E_r sera ensuite reformulé en autant de vecteurs propositionnels de taille fixe que de substitutions θ_i. Les attributs de la représentation tabulaire de E_r sont choisis en fonction du motif et de l'espace de recherche relationnel initial $(\mathcal{L}_{hr}, \succeq_r)$.

Exemple 3.1.1
Soit l'exemple $E_r : cc \leftarrow c(a), p(a,b), q(b), q(a), r(c)$ *et le motif* $M : cc \leftarrow p(V,W), q(X), q(Y)$, *et soit* $\succeq_r$ *la* θ*-subsomption. On obtient quatre substitutions* θ_i *telles que* $M\theta_i \subseteq E_r$*, qui correspondent aux quatre manières différentes d'apparier* $q(X), q(Y)$ *avec* $q(b), q(a)$ *sous* θ*-subsomption. On peut ensuite construire à partir de ces substitutions plusieurs représentations tabulaires pour* E_r*. Les attributs pour redécrire les* θ_i *peuvent être simplement les variables de* M *et les valeurs de ces attributs sont les constantes assignées à chacune de ces variables par* θ_i*. On peut également compléter ces attributs par un ensemble d'attributs booléens, par exemple, un ensemble d'égalités entre paires de variables de* $\mathcal{M}$*. L'ajout d'attributs booléens permet d'enrichir l'ensemble des généralisations de* M *qui sont atteignables dans le nouvel espace de recherche propositionnel (voir section 3.2).*

Si l'on veut prendre en compte lors de l'apprentissage les cas où M *s'apparie uniquement partiellement avec les exemples de* T_r*, il est adéquat d'adjoindre à la représentation tabulaire précédente un ensemble d'attributs booléens indiquant si un littéral particulier de* M *a pris part au calcul de la substitution* θ_i *(voir section 3.3).*

Si l'on se restreint au cas de l'appariement total de M avec un exemple d'apprentissage relationnel E_r, E_r est théoriquement reformulé, sous θ-subsomption et étant donné un motif M, en un ensemble de $\prod_{j=1}^{N} n_j^{m_j}$ vecteurs propositionnels, où N est le nombre de symboles de prédicats apparaissant dans M, n_j et m_j sont respectivement le nombre d'occurrences du symbole de prédicat p_j dans E_r et dans M. Cette explosion combinatoire vient du fait que, dans un langage cible $\mathcal{L}_{hr}$, les variables existentielles de M s'apparient de façon non déterministe avec les constantes dans les exemples de T_r. Dans la suite, nous appellerons *langage relationnel non déterministe* un tel langage de concept $\mathcal{L}_{hr}$.

Notons le problème d'apprentissage reformulé par $(E_{av}, \mathcal{L}_{av}, \succeq_{av})$. Le nouvel espace de recherche $(\mathcal{L}_{av}, \succeq_{av})$ est un treillis dont la borne inférieure est la reformulation (unique) du motif dans $\mathcal{L}_{av}$. L'ensemble d'apprentissage E_{av} reformulé peut donc être traité par des méthodes propositionnelles, selon la tâche d'apprentissage initiale. Évidemment, la complexité de méthodes fondées sur une telle reformulation repose sur la taille de l'ensemble d'apprentissage reformulé E_{av}.

3.1.3 Travaux apparentés

L'idée d'exemples comme un ensemble de substitutions d'une formule logique apparaît par ailleurs dans de nombreux travaux, exprimée de façon plus ou moins explicite [FZW95], Claudien [DRB93], puis PROGOL [SC97].

LINUS [LD94] est le premier système de PLI qui propose une approche à base de propositionnalisation pour résoudre un problème d'apprentissage relationnel en utilisant une technique propositionnelle. Par contre, et afin de contourner le problème de la taille potentiellement exponetielle de l'ensemble d'apprentissage reformulé, le langage du concept cible est restreint[2] de telle manière qu'un exemple relationnel est transformé en un unique exemple propositionnel.

Le système REPART [ZG98], et son ancêtre REMO [ZG94, ZG96] utilisent eux aussi la propositionnalisation, mais n'abordent pas explicitement le problème de la taille de l'ensemble

2. Clauses définies sans variables existentielles pour LINUS, ou restreint à des variables existentielles déterministes uniquement pour DINUS.

d'apprentissage reformulé. Par contre, l'utilisateur, à travers le choix d'un motif de propositionnalisation pertinent, peut indirectement contrôler la taille des exemples reformulés, en choisissant, par exemple, un motif faiblement non-déterministe.

3.1.4 Nouvelle tâche d'apprentissage

Comme mis en avant dans [ZG96, ZG98, DR98], la tâche d'apprentissage dans l'espace de recherche reformulé n'est plus l'induction d'une définition de concept couvrant toutes les instances positives reformulées et rejetant toutes les instances négatives reformulées du concept cible (voir chap. 2, section 2.2), mais est maintenant un problème *multi-instance*:

Définition 3.1.1 (Problème multi-instance [DLLP96])
Soit un problème d'apprentissage PLI $P_r : (T_r, \mathcal{L}_{hr}, \succeq_r)$. *Soit un problème d'apprentissage reformulé* $P_{av} : (E_{av}, \mathcal{L}_{av}, \succeq_{av})$, *extrait de* P_r *selon le motif* M. *Soit* $proj(E, M)$ *l'ensemble d'exemples de* E_{av} *extraits de* E *par projection sur le motif* M. *La tâche d'apprentissage pour le problème reformulé consiste à trouver une hypothèse* $H \in \mathcal{L}_{av}$ *telle que*

- *pour chaque exemple positif* $E \in T_r$, *il existe au moins un exemple* $e \in proj(E, M)$ *tel que* $H \succeq_{av} e$
- *pour chaque exemple négatif* $CE \in T_r$ *: il n'existe aucun* $ce \in proj(CE, M)$ *tel que* $H \succeq_{av} ce$.

L'équivalence entre le problème d'apprentissage initial et un problème reformulé selon un ensemble de motifs $\{M_1, \ldots, M_n\}$ dépend bien évidemment des caractéristiques conjointes de tous ces problèmes, en particulier de la relation de subsomption $\succeq_r$, du ou des motifs de propositionnalisation choisis, etc. Cette propriété d'équivalence est démontrée au cas par cas pour chaque changement de représentation spécifique, car il n'existe pas pour l'instant de conditions nécessaires ou suffisantes de préservation des solutions du problème d'apprentissage relationnel par propositionnalisation.

3.2 $DiVS\_LP$

Je présente ici l'extension de *DiVS*, conçu par M. Sebag [Seb94b, Seb94a], à l'induction de clauses définies [SR94] ; ce système, que nous nommerons par commodité $DiVS\_LP$ est le résultat d'une collaboration entre M. Sebag et moi-même[3]. Après avoir présenté brièvement l'algorithme de *DiVS* (section 3.2.1), j'introduirai la technique de propositionnalisation que nous avons développée pour adapter *DiVS* à l'apprentissage de clauses définies (section 3.2.2). J'aborderai ensuite rapidement à la section 3.2.4 l'heuristique d'appariement stochastique, développée dans le système STILL afin de faire face au problème de la taille exponentielle de l'ensemble d'apprentissage reformulé.

3.2.1 *DiVS*: brève description

Apprentissage dans *DiVS*

L'algorithme de base de *DiVS* est l'espace des versions disjonctif, conçu pour dépasser les limites de l'algorithme d'élimination des candidats [Mit82] dans le cas d'un problème d'ap-

3. L'extension de *DiVS* à l'induction de clauses contraintes [SR95] sera présentée au chapitre 4.

prentissage disjonctif, ou bruité (nous renvoyons à [Seb94a] pour une argumentation sur les limites de l'algorithme d'élimination des candidats). Afin de contourner ces limitations, et étant donné un ensemble d'apprentissage $E = E^+ \cup E^-$, *DiVS* construit, pour *chaque* exemple $E_i \in E$, une formule appelée *étoile* de E_i et notée $G(E_i)$. Cette formule caractérise l'ensemble des formules maximalement générales couvrant E_i et rejetant tous les exemples CE_j tels que $classe(CE_j) \neq classe(E_i)$. *DiVS* induit donc n telles hypothèses, où $n = card(E)$.

C'est un fait bien connu depuis [Hau88] que le nombre des hypothèses maximalement générales et correctes vis-à-vis d'un ensemble d'apprentissage peut être, dans le pire des cas, exponentiel en fonction du nombre d'exemples négatifs du concept cible, et ceci même pour un langage cible booléen. C'est pourquoi la plupart des systèmes d'apprentissage se contentent de calculer un sous-ensemble de taille fixe de G maximisant un critère heuristique.

Afin de caractériser tout de même $G(E_i)$, l'algorithme d'*induction paresseuse* de *DiVS* décompose encore le problème d'apprentissage. Au lieu de construire $G(E_i)$ comme un ensemble d'hypothèses correctes et complètes[4], $G(E_i)$ est caractérisée comme la conjonction logique, pour tout exemple CE_j appartenant à une autre classe que celle de E_i, d'une formule discriminant maximalement E_i et CE_j:

$$G(E_i) = \bigwedge_{j=1}^{|E^-(E_i)|} G(E_i, CE_j)$$

où $E^-(E_i)$ dénote l'ensemble des exemples de E dont la classe est différente de celle de E_i.

Enfin, la formule discriminant maximalement un exemple positif E d'un exemple négatif pour E, noté CE, est la *disjonction* des descripteurs maximalement discriminants pour E et CE, construits comme suit. Pour chaque attribut att_i ayant pour valeur v_i dans E et V_i dans CE et tel que $v_i \neq V_i$, le descripteur maximalement discriminant suivant est :

- $att_i \in Int_i$, où Int_i est le plus grand intervalle contenant v_i sans contenir V_i, si att_i est un attribut numérique

- $att_i = v_i$ si att_i est un attribut nominal

- $att_i = G(v_i)$, où $G(v_i, V_i)$ est le plus grand généralisé de v_i qui rejette V_i, si att_i est un attribut à valeur dans un type hiérarchique.

La formule $G(E_i)$ ainsi obtenue n'est pas simplifiée : en particulier, si un exemple négatif est subsumé par un autre, produisant donc une formule redondante dans le $G(E_i)$ final, elle n'est pas élaguée. Ceci introduit éventuellement un surcoût au niveau de la classification, mais qui est compensé par l'économie du coût de la détection d'exemples négatifs maximalement spécifiques.

Classification dans *DiVS*

Ces hypothèses induites sont ensuite utilisées pour classifier un nouvel exemple NE dans un algorithme de type *k-plus proches voisins* : chaque étoile $G(E_i)$ qui couvre NE va voter pour l'appartenance de NE à la classe de E_i, la classe finale de NE étant assignée à la majorité des hypothèses votantes.

Nous allons dans la suite de cette section décrire l'extension des algorithmes principaux de *DiVS*, l'induction paresseuse (section 3.2.2), et la classification (section 3.2.3), à la manipulation de clauses définies.

4. Notons que dans ce cadre, la complétude se ramène à la couverture du seul exemple positif E_i.

3.2.2 *Induction paresseuse* étendue à l'apprentissage de clauses définies

$DiVS_LP$conserve la stratégie d'apprentissage de $DiVS$, par décompositions successives du problème d'apprentissage initial. Par contre, le calcul de la généralisation maximalement générale discriminant un exemple positif d'un exemple négatif doit être étendue pour traiter des exemples relationnels. Le cadre d'apprentissage est celui de l'apprentissage par implication (voir chap. 2). Les exemples et contre-exemples sont représentés comme des clauses définies complètement instanciées. La relation de subsomption de $DiVS\_LP$ est la θ-subsomption, notée $\succeq_\theta$ dans la suite. Afin de faciliter la présentation, l'exemple et le contre-exemple sont décrits sous forme de clauses canoniques.

Définition 3.2.1
Soit C une clause définie quelconque. La forme canonique de C est de la forme $\mathcal{C}\theta$, telle que $\mathcal{C}$ est la variabilisation maximale de C (toutes les variables de $\mathcal{C}$ sont différentes) et θ est la substitution telle que $C = \mathcal{C}\theta$.

Nous adoptons dans la suite comme ordre partiel sur cette partie contrainte l'inclusion de substitutions décrite, par exemple, dans [LMM87]. Dans la suite, nous notons sous forme canonique $E = \mathcal{E}\theta$ et $CE = \mathcal{CE}\delta$.

Définition 3.2.2
Une clause $H = \mathcal{H}\gamma$ est une généralisation discriminante de l'exemple $E = \mathcal{E}\theta$ contre le contre-exemple $CE = \mathcal{CE}\delta$ ssi

1. *$H \succ_\theta E$ (généralisation de E)*
2. *(a) soit, H discrimine CE structurellement, i.e. $\mathcal{H} \not\succeq_\theta \mathcal{CE}$.*
   (b) soit, $\mathcal{H} \succeq_\theta \mathcal{CE}$, mais pour toute substitution μ_k telle que $\mathcal{H}\mu_k \subseteq \mathcal{CE}$, $\mu_k.\gamma \nsubseteq \delta$.

Rappelons que nous ne recherchons que des formules discriminantes *maximalement générales.* Le point 2.a) de la définition 3.2.2 permet de mettre en place la *discrimination par prédicat.* Ce cas se produit s'il existe au moins un littéral dans $\mathcal{E}$ qui ne peut «s'apparier» avec aucun littéral dans $\mathcal{CE}$. Comme, par définition de la forme canonique, toutes les variables de $\mathcal{E}$ sont différentes, un tel littéral discriminant est nécessairement construit sur un symbole de prédicat qui n'apparaît pas dans $\mathcal{CE}$. Ce littéral est donc une généralisation maximalement discriminante de E, et va former une des disjonctions de $G_{pred}(E, CE)$. Nous ne détaillerons pas plus ce cas relativement trivial de discrimination. Nous allons donc nous concentrer sur le point 2.b) de la discrimination, nommée *discrimination par contrainte.*

$$G(E, CE) = G_{pred}(E, CE) \vee G_{contr}(E, CE)$$

Dans ce cas, $DiVS\_LP$ utilise une approche à base de propositionnalisation.

Changement de représentation pour l'apprentissage discriminant

Le motif de propositionnalisation, M, est ici le sous-ensemble des littéraux de $\mathcal{E}$ construits sur des symboles de prédicats non discriminants, c'est-à-dire qui apparaissent à la fois dans $\mathcal{E}$ et $\mathcal{CE}$. Les attributs de la représentation tabulaire sont au minimum les variables de M, et ils peuvent être complétés par l'ensemble des égalités entre paires de variables de M qui sont satisfaites par E.

Un seul vecteur propositionnel est extrait de E, v_θ. Les exemples propositionnels extraits de CE sont les valeurs de ces attributs pour *toutes* les substitutions telles que $\mathcal{C} \succeq_\theta CE$. Rappelons qu'un symbole de prédicat donné peut avoir un nombre d'occurrences quelconque dans $\mathcal{C}$ et $\mathcal{CE}$, il y a donc dans le pire des cas un nombre exponentiel de substitutions telles que $\mathcal{C}$ θ-subsume CE et donc autant de vecteurs négatifs propositionnels extraits de CE (voir section 3.1).

Exemple 3.2.1
Soient E et CE un exemple positif et négatif de la classe cc[5].

$E : cc \leftarrow c(a), p(a,b), q(b), q(a), r(b)$

$CE : cc \leftarrow c(a), p(a,b), p(b,c), q(b), q(c)$

M est construit ici comme la variabilisation maximale de E moins le littéral $r(b)$, car le symbole de prédicat r n'apparaît pas dans CE, i.e., $M : cc \leftarrow c(U), p(V,W), q(X), q(Y)$. Comme attributs booléens supplémentaires, nous allons considérer les égalités entre paires de variables de $\mathcal{M}$ qui sont satisfaites dans E, i.e., $U = V, U = Y, V = Y, W = X$.

E est projeté sur l'élément le plus spécifique du treillis booléen, v_θ. Détaillons maintenant l'étape de reformulation de l'exemple CE. La propositionnalisation construit en tout huit substitutions σ_i telles que $M\sigma_i \subseteq CE$. L'ensemble d'apprentissage $\{E, CE\}$ reformulé est décrit à la figure 3.1.

M	U	V	W	X	Y	$U=V$	$U=Y$	$V=Y$	$W=X$
v_θ	a	a	b	b	a	1	1	1	1
v_{σ_1}	a	a	b	b	c	1	0	0	1
v_{σ_2}	a	a	b	c	b	1	0	0	0
v_{σ_3}	a	a	b	b	b	1	0	0	1
v_{σ_4}	a	a	b	c	c	1	0	0	0
v_{σ_5}	a	b	c	b	c	0	0	0	0
v_{σ_6}	a	b	c	c	b	0	0	1	1
v_{σ_7}	a	b	c	b	b	0	0	1	0
v_{σ_8}	a	b	c	c	c	0	0	0	1

FIG. 3.1 – *Ensemble d'apprentissage $\{E, CE\}$ reformulé selon le motif M*

D'après la définition 3.1.1, le problème de discrimination relationnel se transforme en problème de discrimination propositionnel : la contrainte discriminant l'exemple négatif PLI de l'exemple positif PLI est la conjonction, pour tous les vecteurs propositionnels extraits de CE, des contraintes maximalement générales permettant de discriminer le vecteur propositionnel extrait de E et un vecteur propositionnel extrait de CE.

$$G_{contr}(E, CE) = M[G(v_\theta, v_{\sigma_1}) \wedge \ldots \wedge G(v_\theta, v_{\sigma_n})]$$

Le problème de discrimination dans ce contexte se pose abstraitement comme l'exploration conjointe de deux treillis dont les bornes inférieures respectives sont fixées, comme dans PROGOL [Mug95] par l'exemple positif E. On dispose, d'une part, d'un treillis «logique» des parties de M et d'autre part, d'un treillis propositionnel des contraintes (ici, de simples substitutions) portant sur un sous-ensemble des variables de $\mathcal{M}$ et généralisant θ. La condition 1 de

5. Par souci de clarté, et afin de ne pas alourdir notre propos avec des considérations sur la génération de clauses connectées, le concept cible est d'arité zéro.

la définition 3.2.2 est toujours trivialement satisfaite. La discrimination va ici être guidée par la discrimination dans le treillis propositionnel, la généralisation dans le treillis logique se bornant à introduire les seuls littéraux nécessaires à l'introduction de la variable discriminante[6].

Considérons finalement le niveau élémentaire de discrimination, qui permet de discriminer le vecteur propositionnel positif v_θ d'un vecteur propositionnel négatif de v_{σ_i}. Nous pouvons maintenant appliquer directement les opérateurs algébriques de discrimination dirigés par les données décrits pour *DiVS* afin de former une substitution élémentaire maximalement générale γ telle que γ généralise θ (le vecteur propositionnel extrait de E) et $\mu_k\gamma$ soit incohérente avec δ (le vecteur propositionnel extrait de CE) (voir définition 3.2.2).

Étant donné la propositionnalisation que nous avons décrite ci-dessus, un attributs att_j de la représentation tabulaire est soit nominal (il représente la valeur de la variable X_j du motif M pour un appariement entre M et un exemple relationnel), soit booléen (il code alors l'égalité entre deux variables du motif, pour un appariement entre M et un exemple relationnel). Si att_j représente une variable du motif, et si la valeur de att_j pour v_θ, notée $att_j.v_\theta$ est différente de la valeur de att_j pour v_{σ_i}, $DiVS\_LP$ construit alors la substitution maximalement discriminante $att_j = att_j.v_\theta$. Si att_j est un attribut booléen, la discrimination sera uniquement possible si $att_j.v_{\sigma_i} = 0$.

Exemple 3.2.2
Si nous reprenons l'exemple 3.2.1, $G_{pred}(E, CE) = cc \leftarrow r(Z)$*. La disjonction des contraintes maximalement générales obtenues pour* v_θ *et* v_{σ_1} *est* $(Y = a \vee U = Y \vee V = Y)$*. La contrainte* $G_{contr}(E, CE)$ *caractérisant les généralisations maximalement discriminantes pour* E *et* CE *est*

$$(Y = a \ \vee \ U = Y \ \vee \ V = Y) \ \wedge \ (X = b \ \vee \ Y = a \ \vee \ U = Y \ \vee \ V = Y \ \vee W = X) \ \wedge$$

$$(Y = a \ \vee \ U = Y \ \vee \ V = Y) \ \wedge \ (X = b \ \vee \ Y = a \ \vee \ U = Y \ \vee \ V = Y \ \vee W = X) \ \wedge$$

$$(V = a \ \vee \ W = b \ \vee \ Y = a \ \vee \ U = V \ \vee \ U = Y \ \vee \ V = Y \ \vee \ W = X) \ \wedge$$

$$(V = a \ \vee \ W = b \ \vee \ X = b \ \vee \ Y = a \ \vee \ U = V \ \vee \ U = Y) \ \wedge$$

$$(V = a \ \vee \ W = b \ \vee \ Y = a \ \vee \ U = V \ \vee \ U = Y \ \vee \ W = X) \ \wedge$$

$$(V = a \ \vee \ W = b \ \vee \ X = b \ \vee \ Y = a \ \vee \ U = V \ \vee \ U = Y \ \vee \ V = Y)$$

La formule ainsi obtenue, de même que pour DiVS, *n'est pas simplifiée (on observe sur cet exemple que la contrainte extraite de* v_{σ_1} *est strictement identique à celle extraite de* v_{σ_3}*, de même que pour* v_{σ_2} *et* v_{σ_4}*). Une généralisation maximalement discriminante sous forme clausale que l'on pourrait extraire de cette contrainte et de* M *est* $g : cc \leftarrow q(a)$*.*

3.2.3 Classification de $DiVS\_LP$

La classification dans $DiVS\_LP$ procède comme nous l'avons dit selon un algorithme de *k plus proches voisins*, algorithme que nous ne détaillerons pas ici. Nous nous concentrerons plutôt sur le processus permettant de décider quand un nouvel exemple est un voisin d'un exemple de l'ensemble d'apprentissage.

Très naturellement, un exemple NE est voisin d'un exemple E si NE est couvert par l'étoile de E, $G(E)$. Comme $G(E)$ est la conjonction des $G(E, CE_i)$ pour tous les CE_i contre-exemples de E (voir section 3.2.2), il suffit donc de pouvoir décider si un exemple est couvert

6. Éventuellement avec une contrainte de connexion sur $\mathcal{H}$.

par $G(E, CE_i)$, c'est-à-dire la généralisation maximalement discriminante de E et CE_i. Plus précisément :

Définition 3.2.3
Soit $G(E, CE_i)$ sous la forme $(G_{pred}(E, CE_i), M, \wedge_k G(v_\theta, v_{\sigma_k}))$. Soit M_c le sous-ensemble de M restreint aux littéraux construits sur des symboles de prédicats communs à M et NE. NE est couvert par $G(E, CE_i)$ ssi

1. *soit $G_{pred}(E, CE_i) \succeq_\theta NE$*
2. *il existe au moins une substitution τ telle que $M_c\tau \subseteq NE$ et, pour tout vecteur propositionnel v_{σ_k} extrait de CE, $\tau \models G(v_\theta, v_{\sigma_k})$.*

Notons que la contrainte $G(v_\theta, v_{\sigma_k})$ n'étant pas une simple conjonction d'égalités, on ne peut simplement tester l'inclusion entre τ et $G(v_\theta, v_{\sigma_k})$.

Exemple 3.2.1
Toujours à la suite des exemples 3.2.1 et 3.2.2, soit la nouvelle observation à classifier O_1 : $c(a), p(b, c), q(a), q(c)$ (conjonction de littéraux, éventuellement existentiellement quantifiée). La représentation tabulaire de O selon le motif M est :

M	U	V	W	X	Y
$\tau_{O,1}$	a	b	c	a	c
$\tau_{O,2}$	a	b	c	c	a

On constate que $\tau_O \models G_{contr}(E, CE)$ donc O est «voisin» de E. Par contre, l'observation O' : $c(d), p(d, c), q(b), q(c)$ n'est pas couverte par $G_{contr}(E, CE)$.

3.2.4 Heuristique stochastique : le système STILL

Nous avons présenté dans les paragraphes précédents le mécanisme de propositionnalisation ainsi que son intégration dans *DiVS_LP* afin de résoudre, par une technique propositionnelle guidée par les données, un problème d'apprentissage relationnel.

Afin de rendre le passage à l'ordre un opérationnel, surtout dans le cas de problèmes faiblement structurés, comme le problème de la mutagénèse, [SR97a, SR97b, SR00] introduit avec l'*appariement stochastique* un biais lié aux ressources accordées à l'étape de propositionnalisation, ou plus précisément au test de couverture à la base de la propositionnalisation. Pour un exemple négatif relationnel, l'ensemble des substitutions telles que $M \succeq_r E_r$, noté Σ est de taille exponentielle (voir section 3.1.2). La subsomption stochastique va elle permettre de représenter chaque exemple négatif relationnel E_r par un échantillon stochastique de taille constante (fixé par l'utilisateur) de substitutions de Σ. La subsomption stochastique est correcte, incomplète certes (en effet, seule une fraction des exemples négatifs propositionnels sont rejetés), mais de complexité linéaire en le nombre de variables du motif. La conséquence en est que la complexité du processus de construction d'une étoile $G(E)$ est en $\mathcal{O}(N \times V^2)$ [7], où N est le nombre d'exemples dans l'ensemble d'apprentissage relationnel, et V le nombre de variables du motif.

De même pour la classification, seul un échantillon de taille fixe de substitutions entre le motif de l'étoile généralisé M_c et l'observation à classifier est exploré (construit). Le processus

7. Si l'on se restreint à une propositionnalisation sans les attributs booléens codant les égalités entre les variables du motif.

de classification est donc encore potentiellement incomplet, mais de complexité polynomiale en nombre de variables du motif, ainsi qu'en nombre d'exemples d'apprentissage relationnels.

D'autres biais de classification sont également mis en œuvre dans STILL, que nous ne détaillerons pas ici. Le système STILL, développé par M. Sebag, obtient des résultats empiriques dans le domaine de la mutagénèse qui sont excellents, tant au niveau des temps d'exécution que des taux de prédiction obtenus (voir [SRP96, SR97b, SR00]), et ceci pour des tailles d'échantillons de substitutions engendrés faibles, tant pour l'apprentissage que pour la classification.

Dans les derniers développements de ce travail [SR00], une amélioration a encore été apportée, qui exploite la partition suivante des variables de M : certaines variables sont des variables *objets* (qui peuvent éventuellement être typées) et d'autres sont des variables *attributs*, l'idée de base étant que les variables attributs sont des fonctions des variables objets. Dans ce cas, l'appariement des variables objets détermine complètement l'appariement des variables attributs, réduisant d'autant la partie strictement NP-difficile du test de subsomption.

3.3 Propositionnalisation Sélective

Dans cette partie du chapitre, le but est d'étudier l'applicabilité d'une approche à base de propositionnalisation pour construire une définition de concept relationnel *complète et correcte*, sous forme clausale, à partir d'un ensemble d'exemples positifs et négatifs[8]. Nous reprenons ici le critère de concision et donc de compréhensibilité du concept appris, même si le prix à payer est une perte d'efficacité du processus d'apprentissage, par rapport à l'approche stochastique de la propositionnalisation précédente.

Notre motivation, similaire à celle qui a conduit à l'extension de $DiVS$ à l'ordre un, est que l'utilisation d'une méthode algébrique d'apprentissage, guidée par les données, permet à un utilisateur de s'affranchir, au moins partiellement et dans un premier temps, de la mise au point fort délicate de biais de langage ou de recherche sophistiqués essentiels au fonctionnement correct de techniques *«générer et tester»*.

Nous proposons dans cette section une technique dite de *propositionnalisation sélective* ou *paresseuse*. Au lieu de générer le problème reformulé, avant apprentissage, comme l'ensemble des appariements entre un motif et les exemples d'apprentissage relationnel, l'idée de la propositionnalisation sélective est de ne générer que les exemples propositionnels **utiles** à la tâche de discrimination. Comme nous allons le voir dans la suite, la notion d'utilité repose à la fois sur la notion dynamique de tâche courante de discrimination du système d'apprentissage et sur l'exploitation de l'ordre partiel sur l'espace des instances propositionnelles. Cette méthode, en alternant propositionnalisation et résolution algébrique ainsi qu'en évitant autant que possible la génération d'instances propositionnelles redondantes limite la taille du problème reformulé, tout en assurant l'apprentissage de définitions de concepts correctes et complètes.

3.3.1 Propositionnalisation dans *Propal*

Soit un motif M, variabilisation maximale d'un exemple positif sous $\succeq_r$. La représentation tabulaire, de même que pour $DiVS\_LP$, est complétée par un ensemble de contraintes d'égalité entre variables de M satisfaites par l'exemple graine, noté $contr(M)$.

Chaque littéral l_i de $\mathcal{M}$ (resp. chaque contrainte c_j de $contr(M)$) définit un attribut booléen b_{l_i}, (resp. b_{c_j}) qui va être utilisé pour reformuler un exemple relationnel E_r (positif ou négatif)

8. Cette définition peut être disjonctive, mais on recherche, même si ce point n'est pas explicité dans les algorithmes classiques d'apprentissage de définitions disjonctives (comme [Mic83]), un nombre faible de disjonctions.

de T_r en un ensemble de vecteurs booléens, comme détaillé dans l'algorithme de la fig. 3.2 :

propositionnalise(e,M)
% retourne un vecteur booléen associé à e étant donné M.
$\sigma_k : = \emptyset$
$Deja\_Apparie : = \emptyset; A\_Apparier : = \mathcal{M}$
Répéter
Sélectionner un littéral l_i de $A\_Apparier$
$A\_Apparier : = A\_Apparier - l_i$
si il existe une subst. σ telle que $e \prec_r Deja\_Apparie.\sigma_k \cup l.\sigma_k.\sigma$
alors $\sigma_k : = \sigma_k.\sigma$; $b_{l_i} : = 1$;
$Deja\_Apparie : = Deja\_Apparie \cup l_i$
sinon $b_{l_i} : = 0$
jusqu'à ce que $A\_Apparie = \emptyset$
Pour tout $c_j \in \mathcal{C}$ **faire**
si $c_j.\sigma_k$ est satisfait pour e
alors $b_{c_j} = 1$
sinon $b_{c_j} = 0$
retourner$(b_{l_1}, \ldots, b_{l_n}, b_{c_1}, \ldots, b_{c_m})$.

FIG. 3.2 – *Algorithme de Propositionnalisation de* Propal

L'espace de recherche PLI initial (ainsi par ailleurs que l'espace des instances) est transformé ici en un treillis booléen de borne inférieure $M \cup contr(M)$, ordonné par l'inclusion booléenne. La différence principale de cette propositionnalisation avec celle de $DiVS\_LP$ est qu'elle permet de représenter *l'appariement partiel* entre le motif et un exemple. La généralisation du motif par abandon d'un littéral ou par abandon d'un lien d'égalité entre deux variables du motif est donc traitée en tant que telle. Par contre, il n'y a plus de treillis attribut-valeur, et donc plus de possibilités de discriminer les exemples négatifs à travers une substitution du motif. Une discrimination par valeur reste néanmoins possible si le motif n'est pas complètement variabilisé.

Exemple 3.3.1
Reprenons l'exemple 3.2.1 déjà développé pour $DiVS\_LP$, et modifions-le légèrement en ajoutant un exemple positif supplémentaire. Supposons que l'espace de recherche relationnel est toujours ordonné par la θ-subsomption. E est la «graine» qui va permettre de construire le motif de propositionnalisation.
$E : cc \leftarrow c(a), p(a, b), q(b), q(a), r(b)$
$E' : cc \leftarrow c(a), p(a, b), p(b, c), q(c), q(a), r(a)$
$CE : cc \leftarrow c(a), p(a, b), p(b, c), q(b), q(c)$
M est construit comme la variabilisation maximale de E sous θ-subsomption, i.e., $\mathcal{M} : cc \leftarrow c(U), p(V, W), q(X), q(Y), r(Z)$. $contr(M)$ est la conjonction des égalités entre paires de variables de M qui sont satisfaites dans E', i.e., $contr(M) : U = V, U = Y, V = Y, W = X, X = Z, W = Z$.

Par construction de l'algorithme à la fig. 3.2, E est projeté sur l'élément le plus spécifique du treillis, booléen cette fois-ci, v_θ. Les substitutions entre le motif et les exemples relationnels sont bien sûr les mêmes que dans l'exemple 3.2.1 (pour CE). Pour chaque substitution telle qu'un sous-ensemble de M (par opposition à la méthode précédente qui teste l'appariement complet entre M et l'exemple relationnel) couvre l'exemple relationnel à reformuler, un vecteur *booléen est engendré selon l'algorithme 3.2. Un extrait de l'ensemble d'apprentissage reformulé est décrit à la figure 3.3*[9].

M	$c(U)$	$p(V,W)$	$q(X)$	$q(Y)$	$r(Z)$	$U=V$	$U=Y$	$V=Y$	$W=X$	$X=Z$	$W=Z$
v_θ	1	1	1	1	1	1	1	1	1	1	1
$v_{\theta'_1}$	1	1	1	1	1	1	1	1	0	0	0
$v_{\theta'_2}$	1	1	1	1	1	1	0	0	0	1	0
$v_{\theta'_3}$	1	1	1	1	1	1	0	0	0	0	0
$v_{\theta'_4}$	1	1	1	1	1	1	1	1	0	1	0
...											
v_{σ_1}	1	1	1	1	0	1	0	0	1	-	-
v_{σ_2}	1	1	1	1	0	1	0	0	0	-	-
v_{σ_3}	1	1	1	1	0	0	0	0	0	-	-
v_{σ_4}	1	1	1	1	0	0	0	1	1	-	-
...											

FIG. 3.3 – *Extrait de l'ensemble d'apprentissage reformulé selon le motif M*

3.3.2 Propositionnalisation sélective

La propositionnalisation sélective exploite deux idées relativement simples afin d'optimiser la taille de l'ensemble d'apprentissage reformulé :

- en s'inspirant des méthodes d'incorporation de test [BD86] ainsi que des développements plus récents en résolution de problèmes de satisfaction de contraintes [Tsa93], l'information récoltée lors de la résolution du problème d'apprentissage (c'est-à-dire, la borne courante de l'espace de recherche) est utilisée afin d'éviter d'engendrer des vecteurs booléens non pertinents vis-à-vis de la tâche courante d'apprentissage. Par exemple, dans le cas d'une recherche descendante, il est inutile de générer des vecteurs booléens qui ne sont couverts par aucune des hypothèses maximalement générales maintenues par l'algorithme.

- l'ordre partiel sur l'espace des instances peut être utilisé afin de générer une approximation de l'ensemble des vecteurs booléens reformulés les plus spécifiques, car eux seuls sont suffisants pour apprendre une définition complète et correcte.

Ordre partiel sur l'ensemble des instances

Une très large fraction des vecteurs booléens générés par une approche de propositionnalisation exhaustive n'est jamais utilisée directement lors de la construction d'une définition de concept correcte et complète (nous les appellerons exemples *redondants* dans la suite du papier) et peut donc ne pas être générée sans nuire pour autant au résultat de l'apprentissage (par exemple, voir fig. 3.3 $v_{\theta'_3}$ et v_{σ_3} par rapport à $v_{\theta'_1}$ et v_{σ_1} respectivement).

En ce qui concerne les exemples négatifs, et d'après [SR90, SR96], il existe un ordre partiel (*nearest-miss* en anglais) sur l'espace des instances négatives, et il a été prouvé que seules

9. Afin de faciliter la lecture du tableau, nous n'y avons noté que les vecteurs booléens extraits des substitutions les plus spécifiques sur M.

les instances négatives les plus spécifiques étaient suffisantes pour assurer la correction de la définition de concepts apprise.

La taille de l'ensemble des formules non redondantes de l'espace des instances booléennes est bornée par le nombre maximal de formules incomparables d'un treillis booléen, soit $\binom{n}{\lfloor n/2 \rfloor}$ avec $n = |M|$. Cette borne supérieure est toujours exponentielle, mais bien moins élevée que la taille de l'espace des substitutions entre un motif de taille n et un exemple de même taille. Par exemple, si l'on considère le problème d'apprentissage de mutagénèse[10], avec comme relation de généralité la θ-subsomption, la taille de l'espace de substitutions entre un motif et un exemple relationnel est dans le pire des cas 40^{40}, contre 2^{40} pour le nombre maximal de vecteurs booléens non redondants. Cette borne théorique reflète la complexité intrinsèque d'une méthode à base de propositionnalisation appliquée à l'apprentissage relationnel dans le cadre de problème non déterministe.

Afin de pouvoir valider expérimentalement la notion de propositionnalisation sélective, nous l'avons «mise en situation» dans un algorithme très simple d'apprentissage, que nous détaillons dans la section suivante. Nous aborderons également l'évaluation de l'efficacité de la propositionnalisation sélective en terme d'approximation de l'ensemble des instances booléennes non redondantes, et nous verrons sur un exemple simple comment ce facteur influence la qualité de la résolution algébrique de l'algorithme d'apprentissage.

3.3.3 Algorithme d'apprentissage de *Propal*

Propal est un algorithme d'apprentissage relationnel qui utilise une stratégie de résolution «à la AQ» [Mic83], que nous décrivons à la figure 3.4. L'algorithme calcule un ensemble de formules maximalement générales et correctes par une recherche descendante dans un treillis booléen ordonné par l'inclusion booléenne. Comme la taille de G est exponentielle [Hau88] en le nombre d'exemples négatifs, l'algorithme utilise une heuristique très simple de recherche par faisceau: seules les n hypothèses ayant le meilleur score de couverture sur les exemples positifs de l'ensemble d'apprentissage sont conservées à chaque étape de spécialisation. À notre connaissance, *Propal* est le premier algorithme d'apprentissage de PLI qui apprend des concepts sous forme clausale en utilisant un opérateur de spécialisation guidé par les données.

On utilise la propositionnalisation sélective aux étapes 1 et 3 de l'algorithme 3.4. Nous nous concentrerons ici sur la propositionnalisation des exemples négatifs (étape 1). Si l'hypothèse courante g est incorrecte par rapport à l'exemple négatif relationnel ce, il existe une substitution σ telle que $g\sigma \subseteq ce$. Comme il existe un isomorphisme entre les sous-ensembles de M (vu comme une clause Datalog) et le treillis booléen ayant M (vu comme un vecteur booléen) comme élément maximalement spécifique, il existe un vecteur booléen négatif, plus spécifique ou égal à g, qui peut être extrait de ce.

Plus le vecteur b extrait de ce est spécifique, plus le facteur de branchement de l'opérateur algébrique (étape 2) sera réduit. Donc, après avoir calculé σ, *Propal* complète σ en appariant de façon gloutonne les littéraux de $M - g$ avec ce. Pour ce faire, *Propal* implémente une heuristique polynomiale de recherche dans l'espace des appariements entre $M - g$ et ce, fondé sur un algorithme d'appariement de graphe [SHW97][11]. Nous ne pouvons donc assurer que le vecteur extrait est le plus spécifique possible, mais seulement qu'il est «proche» d'un des plus

10. Nous rappelons que dans ce problème d'apprentissage, les exemples et contre-exemples (des molécules) ont des descriptions comptant entre 20 et 40 atomes.

11. En effet, rechercher le vecteur booléen le plus spécifique est équivalent à un problème de max-clique, qui est bien plus complexe qu'un problème de subsomption.

Mest construit à partir d'un exemple positif graine (voir algorithme figure 3.2)
$G \leftarrow \{\top\}$ (l'élément le plus général du treillis)

Pour chaque $ce \in CE$ **faire**
  Répéter
    Sélectionner $g \in G$
    (1) Calculer un vecteur booléen b à partir de ce (* $M \prec_b \ b \preceq_b \ g$ *)
    Si b est égal à M**alors**
      on ne peut pas discriminer structurellement ce
    Sinon
      (2) Spécialiser G pour discriminer b *(* résolution algébrique *)*
      (3) Évaluer chaque élément de G en fonction de la couverture des exemples positifs
      Mettre à jour G *(* stratégie par faisceau *)*
    Finsi
  Jusqu'à ce que tous les éléments de G soient corrects
Finpour
return(G)

FIG. 3.4 – *Calcul de n éléments de G*

spécifiques. En contrepartie, l'appariement de complexité exponentielle est restreint au calcul de la substitution σ.

Nous ne décrirons pas ici l'opérateur de spécialisation algébrique, mais nous renvoyons à [Gam95] pour une description détaillée. Une fois que b a été rejeté (étape 2), il n'est pas nécessaire de le stocker. Enfin, si la propositionnalisation génère un vecteur négatif égal à M, il n'existe pas de solution discriminante.

Exemple 3.3.2
Reprenons le problème d'apprentissage PLI de l'exemple 3.3.1. Supposons que l'algorithme d'apprentissage soit en train d'évaluer l'hypothèse incorrecte g : $cc \leftarrow$, c'est-à-dire la clause maximalement générale de l'espace de recherche. Cette clause est incorrecte avec la substitution $\sigma_0 = \emptyset$. Supposons que l'heuristique polynomiale de complétion de σ_0 construise la substitution négative $\sigma = \{U/a, V/a, W/b, X/c, Y/b\}$, conduisant à la construction du vecteur booléen $\{111101000--\}$, correspondant à v_{σ_2} dans le tableau 3.3. L'opérateur de spécialisation, afin de rejeter ce vecteur booléen négatif, va construire les raffinements (sains et minimaux) de g :$\{cc \leftarrow r(Z); cc \leftarrow c(U), q(Y), U = Y; cc \leftarrow p(V,W), q(Y), V = Y\}$. Les deux premières de ces hypothèses sont correctes et complètes, la troisième couvre encore l'exemple négatif et devra donc être encore affinée. Si l'heuristique polynomiale été plus efficace et avait produit le vecteur booléen négatif $\{111101001--\}$ correspondant à v_{σ_1} dans le tableau 3.3, seule les deux premières spécialisations de g auraient été produites.

3.3.4 Apport de la propositionnalisation sélective

Tout d'abord, grâce à la recherche descendante effectuée directement dans l'espace de recherche ILP, le nombre de vecteurs générés par la propositionnalisation sélective est égal dans le pire des cas, et ceci à chaque étape de spécialisation, à l'espace des substitutions entre g (au lieu de $\mathcal{M}$) et un exemple négatif relationnel. Le gain est très important, surtout si la description du motif contient de nombreux prédicats qui ne sont pas discriminants.

Nous pouvons ensuite assurer qu'à chaque étape de spécialisation, le vecteur booléen extrait est nécessairement incomparable ou plus spécifique que les vecteurs générés précédemment.

Enfin, la qualité de l'approximation de l'ensemble des vecteurs booléens non redondants (et donc de l'heuristique de calcul d'un vecteur booléen négatif le plus spécifique possible) a un impact direct, comme le montre l'exemple 3.3.2, sur le facteur de branchement de l'étape de spécialisation et donc sur la performance de la recherche : le cas extrême étant que si le vecteur booléen négatif extrait est un near-miss (identique à la borne inférieure du treillis booléen sauf pour un attribut), il n'y a qu'une seule spécialisation candidate de G.

Le système que nous venons de décrire a été implémenté par E. Alphonse et testé sur une version purement structurelle du problème de la mutagénèse. Les résultats expérimnetaux ont permis d'évaluer empiriquement sur un problème réel le pourcentage de l'espace des appariements explorés [AR99] ainsi que la prédiction des définitions obtenues [AR00].

3.4 Conclusion

Nous avons présenté dans ce chapitre deux systèmes d'apprentissage complètement différents du point de vue de la stratégie qu'ils emploient ainsi que de leurs objectifs, mais qui utilisent tous deux une étape de propositionnalisation afin d'appliquer des opérateurs algébriques de spécialisation guidés par les données.

Dans les deux cas, la propositionnalisation aura permis «l'importation» en PLI de techniques de résolution guidée par les données, et qui sortent du cadre du calcul de moindre généralisé. Ces techniques permettent d'être moins dépendant (sans a priori exclure leur utilisation) de biais forts des approches *«générer et tester»*. Les contextes d'apprentissage très différents de ces deux systèmes ont amené à adopter des solutions orthogonales afin de résoudre le problème de la taille de l'espace d'apprentissage reformulé, et tout laisse à penser que de nouvelles directions sont encore à explorer sur le sujet.

Les perspectives de ce champ de recherche sont nombreuses. Du point de vue théorique, la notion de complétude de la propositionnalisation, au sens de la préservation des solutions du problème d'apprentissage a été abordée pour l'instant de manière ad-hoc[Zuc96, AR98], mais demande à être formalisée de manière plus systématique. Du point de vue pratique, il faut maintenant confirmer les résultats très encourageants des premières évaluations de la propositionnalisation sélective sur des problèmes purement structurels en l'étendant à des problèmes de discrimination relationnels généraux, où la discrimination peut s'effectuer à la fois par la structure (occurrences de prédicats, liens d'(in)égalité entre variables) et par les valeurs d'arguments.

Chapitre 4

La lisière de la PLI

Une des limitations des systèmes de PLI est l'induction de programmes faisant intervenir des informations numériques. Les problèmes d'évaluation et d'apprentissage de littéraux contenant des variables numériques sont une conséquence du choix de la Programmation Logique comme cadre de formalisation de la PLI. La manipulation et l'induction efficace de programmes logiques contraints est pourtant une des conditions sine qua non de l'application de systèmes de PLI à des problèmes réels. Nous allons dans la première partie de ce chapitre détailler les différentes approches des systèmes de PLI pour traiter ce problème majeur, ainsi que notre contribution, sous la forme d'une extension de la PLI à l'apprentissage de clauses logiques contraintes, s'appuyant sur des résultats en Programmation Logique Contrainte. Ce travail a été réalisé en collaboration avec M. Sebag, dans la continuation de l'extension de *DiVS* à l'apprentissage de clauses définies présentée au chapitre 3.

Dans la deuxième partie de ce chapitre, nous tenterons d'explorer les liens de la PLI avec un autre domaine limitrophe, celui des Logiques de Descriptions. Les logiques de descriptions (DL dans la suite de ce chapitre) ont été spécifiquement conçues pour représenter et manipuler des connaissances de type hiérarchique. Les DLs et les logiques définies sont deux sous-ensembles de la logique des prédicats d'ordre un aux pouvoirs d'expressivité non comparables [Bor96]. En effet, les DLs proposent des connecteurs originaux (cardinalité, quantification sur les *fillers* de rôles), ainsi qu'une forme limitée de négation qui sont au delà de l'expressivité d'un langage logique défini. De plus, ces langages ont été largement étudiés du point de vue de leur complexité et certains disposent d'algorithmes de calcul de moindre généralisé de complexité polynomiale. Par contre, si les logiques définies offrent une plus grande flexibilité pour représenter des relations d'arité quelconque, ainsi que des jointures arbitraires entre ces relations, c'est au prix d'un coût exponentiel pour le test de subsomption pour des langages définis non restreints.

Il est donc naturel que la PLI s'intéresse à ces logiques, dans le cadre de la recherche d'un nouveau compromis expressivité-complexité pour le raisonnement inductif. Le travail que nous avons débuté récemment dans ce domaine, en collaboration avec V. Ventos [RV00], est d'explorer la faisabilité et l'utilité de l'apprentissage dans un langage hybride logique de descriptions - logique définie, le langage CARIN [LR98b]. En posant des restrictions adéquates à la fois sur la composante Clause de Horn ainsi que sur la composante logique de descriptions, nous espérons obtenir un langage à la fois expressif et ayant un bon comportement au niveau complexité de l'apprentissage.

4.1 Apprentissage de clauses contraintes

Les nombres et la logique des prédicats d'ordre un ne font pas bon ménage, une illustration en est qu'il n'existe pas d'axiomatisation finie de $\mathbb{N}$ en logique d'ordre un. Les symboles de fonction sont non interprétés en Programmation Logique, ils sont simplement traités comme foncteurs des termes de Herbrand. Par exemple, le symbole de fonction - n'étant pas interprété, les deux membres de l'équation suivante ne peuvent pas s'unifier et l'équation $X - Y = 0$ ne se résoudra jamais. Dans la pratique, les langages Prolog les plus répandus offrent une forme limitée de foncteurs interprétés, par le biais du prédicat prédéfini *is*[1]. Ce constructeur évalue un terme *clos* construit à partir de constantes numériques et de fonctions arithmétiques prédéfinies et retourne la valeur numérique correspondante. Cependant, cette solution reste limitée, car *is* ne s'applique qu'à des termes clos. Comme illustration supplémentaire, le littéral $Z\ is\ X - Y$ ne se résoudra jamais sauf si X et Y sont tous deux instanciés à des valeurs numériques.

Les problèmes d'apprentissage susceptibles de faire intervenir à la fois une forte composante structurelle PLI ainsi que des aspects numériques (comme par exemple, l'apprentissage de règles de classification dans les domaines de l'architecture, de la conception de pièces mécaniques ou de la détection de pannes dans des réseaux [May99]) semblent donc hors de portée de la PLI.

4.1.1 État de l'art

Évidemment, de nombreux chercheurs ont voulu contourner cette limitation de la PLI. Les techniques de propositionnalisation (voir chap. 3) offrent une solution, modulo le problème de taille de l'ensemble d'apprentissage reformulé que nous avons déjà largement évoqué. Ces systèmes bénéficient ainsi de tout l'éventail des techniques développées par les systèmes d'apprentissage propositionnels, notamment en ce qui concerne la discrétisation des attributs numériques (voir par exemple [DKS95] pour un état de l'art).

Des systèmes tels FOIL [QCJ93], FORS [KB97] et PROGOL [SC97] court-circuitent la résolution PROLOG, en faisant appel, de manière plus ou moins sophistiquée, à des traitement «câblés» pour les littéraux numériques. Par exemple, PROGOL introduit dans la borne inférieure de son espace de recherche un certain nombre de littéraux dits *à évaluation paresseuse*. L'utilisateur fournit pour ces prédicats les modes et types des arguments, ainsi qu'une procédure décrite sous forme d'un programme Prolog faisant appel - notamment - au prédicat prédéfini `is`. Ce programme Prolog permet de calculer, pour ces littéraux, les arguments de sortie à partir des arguments d'entrée. Les variables d'entrée sont donc instanciées, et les variables de sortie sont, en attendant mieux, des constantes de Skolem. Lorsqu'il s'agira de spécialiser une clause courante par l'ajout d'un littéral à évaluation paresseuse, un ensemble d'exemples positifs et négatifs pour ce littéral[2], sont collectés, et la procédure fournie par l'utilisateur pour ce prédicat est synthétisée. Par exemple, dans le cas d'une régression linéaire, les constantes de Skolem représentant les coefficients de l'équation linéaire sont calculés. Le littéral numérique ne sera ajouté afin de spécialiser la clause courante que si des coefficients peuvent être calculés tels que l'erreur quadratique de l'équation synthétisée soit inférieure à un seuil fixé par l'utilisateur. Par le biais de la théorie du domaine, l'utilisateur peut fournir des programmes définissant des schémas de relations numériques à induire, ainsi que le moyen de les évaluer sur les exemples. Il peut fournir, par exemple, la discrétisation d'une variable continue, (voir en particulier les travaux autour de PROGOL et des numériques [SM95, SC97])

1. Voir également les prédicats prédéfinis $\leq$, etc.
2. Les substitutions de la clause courante induites par les exemples positifs et négatifs.

L'évaluation et l'apprentissage de clauses contenant les variables numériques ont été traités dans la plupart des cas en PLI comme une verrue au modèle logique, ce qui est tout à fait injustifié, quand on connaît les développements récents de la Programmation Logique Contrainte, ainsi qu'en résolution de problèmes de satisfaction de contrainte. Dans la suite de cette section, nous présentons l'extension de DiVS [Seb94b, Seb94a] à l'induction de clauses contraintes [SR95, SR96]. Nous présentons d'abord dans la section 4.1.2 une notion de couverture/généralité entre clauses contraintes, qui nous permettra d'étendre la définition de généralisation discriminante à un langage de clauses contraintes. Nous donnerons ensuite rapidement les grandes lignes de l'algorithme d'apprentissage de clauses contraintes maximalement générales, fondé sur une technique de propositionnalisation. Nous nous appuierons pour cette partie sur la description de $DiVS\_LP$, déjà décrit au chapitre 3.

4.1.2 Relation de généralité entre clauses contraintes

Nous introduisons tout d'abord quelques notations et définitions de base de la Programmation Logique Contrainte (notée PLC dans la suite), empruntées pour la plupart à [VHD91].

Soit $\mathcal{L}$ un langage de clauses définies sans symbole de fonction autre que des constantes (DATALOG), $\mathcal{L} = \mathcal{L}_h \cup \mathcal{L}_c$, tels que $\mathcal{L}_h$ ne contient que des symboles de prédicats non interprétés, alors que $\mathcal{L}_c$ définit un ensemble de symboles de prédicats interprétés.

Définition 4.1.1
Une contrainte *est un atome construit sur un symbole de prédicat de $\mathcal{L}_c$. Nous appellerons dans la suite un* atome *(par abus de langage, et par opposition à une contrainte) un littéral construit sur un symbole de prédicat de $\mathcal{L}_h$.*

Définition 4.1.2
Une clause contrainte *est une clause définie de la forme :*

$$H \leftarrow B_1 \wedge \ldots \wedge B_m \wedge c_1 \wedge \ldots \wedge c_n$$

telle que H, $B_1, \ldots, B_m$ sont des atomes et $c_1, \ldots, c_n$ sont des contraintes. Dans la suite, $c_1 \wedge \ldots \wedge c_n$ est appelé la partie contrainte *de la clause contrainte, et $H \leftarrow B_1 \wedge \ldots \wedge B_m$ la* partie définie. *Par ailleurs, nous imposons que toutes les variables apparaissant dans $c_1 \wedge \ldots \wedge c_n$ soient liées à des variables apparaissant dans $B_1 \wedge \ldots \wedge B_m$.*

Un programme logique contraint *est un ensemble fini de clauses contraintes.*

La notion de conséquence logique se note $P, \mathcal{T} \models C$, ou $P \models_{\mathcal{S}} C$, où P est un programme logique contraint, $\mathcal{S}$ est une structure définissant $\mathcal{D}$ le domaine d'interprétation des variables de P et les prédicats interprétés sur $\mathcal{D}$, et $\mathcal{T}$ est la théorie axiomatisant $\mathcal{S}$. Si la structure $\mathcal{S}$ peut être axiomatisée en LPO, les deux formalisations sont équivalentes; mais des structures telles que $\mathbb{N}$ n'ont pas d'axiomatisation finie en LPO. Dans ce cas, on se ramènera à une sémantique algébrique.

Dans la suite, on utilisera la notation $\mathcal{D} \models$ pour l'implication entre des expressions contraintes.

L'implication entre clauses contraintes fait intervenir l'implication logique sur leurs parties définies, évaluée par unification dans le plus petit modèle de Herbrand et l'implication sur leurs parties contraintes, qui est évaluée sur $\mathcal{D}$ à travers les prédicats interprétés.

Définition 4.1.3
Soit c une contrainte, et $\mathcal{S}$ la structure définissant $\mathcal{D}$ le domaine d'interprétation des variables de c. c est cohérente *(ou* satisfiable*) si et seulement si il existe au moins une instanciation des variables de c dans $\mathcal{D}$ telle que c soit vraie par rapport à $\mathcal{S}$, notée $\mathcal{D} \models \exists c$. Plus généralement,*

étant donné un ensemble (i.e., une conjonction) de contraintes σ et une contrainte c, c est cohérente avec σ, *si et seulement si $\mathcal{D} \models \exists(\sigma \wedge c)$. Un ensemble de contraintes σ est* incohérente *ou insatisfiable par rapport à $\mathcal{S}$ si et seulement si $\mathcal{D} \models \forall(\neg\sigma)$.*

Définition 4.1.4 (Généralisation de contraintes)
Étant donnés deux (ensembles de) contraintes σ et ρ, σ généralise *ρ , noté $\rho \succeq_{\mathcal{S}} \sigma$ si et seulement si $\mathcal{D} \models \forall(\sigma \rightarrow \rho)$. Nous dirons également que σ* satisfait *ρ par rapport à $\mathcal{S}$.*

Exemple 4.1.1
Soient les contraintes $\sigma = (X > 10)$ et $\rho = (X > 5)$, pour X ayant $\mathbb{N}$ comme domaine d'interprétation. On a $\rho \succeq_{\mathcal{S}} \sigma$, car toutes les instanciations de X dans $\mathbb{N}$ qui rendent σ vraie rendent aussi ρ vraie. De manière équivalente, le domaine de X dans σ (e.g. $]10, \infty)$) est inclus dans le domaine de X dans ρ (e.g. $]5, \infty)$).

Nous pouvons maintenant présenter notre relation de généralité sur les clauses contraintes. Elle a été introduite pour la première fois sous une forme préliminaire dans [SR95]. Elle a été reprise depuis, par exemple dans [AF97].

Définition 4.1.5
Soient deux clauses contraintes sous forme canonique[3], $G_1\gamma_1$ et $G_2\gamma_2$, $G_1\gamma_1$ généralise *$G_2\gamma_2$, noté $G_1\gamma_1 \succeq_C G_2\gamma_2$, si et seulement si G_1 θ-subsume G_2 et $\gamma_1\sigma$ généralise γ_2, c'est-à-dire*

$G_1\gamma_1 \succeq_C G_2\gamma_2$ si et seulement si il existe une substitution σ telle que $G_1\sigma \subseteq G_2$ et $\gamma_1\sigma \succeq_{\mathcal{S}} \gamma_2$

Définition 4.1.6
Une clause $H = \mathcal{H}\gamma$ est une généralisation discriminante de l'exemple $E = \mathcal{E}\theta$ contre le contre-exemple $CE = \mathcal{CE}\delta$ si et seulement si

1. *$H \succeq_C E$*
2. (a) *soit H discrimine CE_j par sa partie définie, i.e. $\mathcal{H}$ ne θ-subsume pas $\mathcal{CE}$.*
   (b) *soit, pour toute substitution σ_i telle que $\mathcal{H}\sigma_i \subseteq \mathcal{CE}$, $\gamma\sigma_i \not\succeq_{\mathcal{S}} \delta$*

4.1.3 Discrimination par contrainte

De même que dans le chapitre 3, nous allons détailler ici le point 2.b de la définition 4.1.6, qui concerne la discrimination par contrainte, par opposition à la discrimination par prédicat, traitée au point 2.a. La version contrainte de *DiVS_LP* fonctionne également sur la base d'une technique de propositionnalisation. Le motif de propositionnalisation est également la restriction de $\mathcal{E}$ aux littéraux construits sur des symboles de prédicats apparaissant à la fois dans $\mathcal{E}$ et $\mathcal{CE}$. Il s'agit donc maintenant de construire la contrainte maximalement générale discriminant θ (la contrainte représentant l'exemple positif) de toutes les σ_i (l'ensemble des contraintes représentant l'exemple négatif). Cette contrainte est la disjonction des contraintes atomiques maximalement discriminantes, dans le même esprit que pour *DiVS*.

Considérons uniquement des variables contraintes nominales et numériques. Le domaine d'interprétation d'une variable nominale est un ensemble fini non ordonné; le domaine d'interprétation d'une variable numérique est un intervalle de $\mathbb{N}$ ou $\mathbb{R}$. Nous considérons le langage de contraintes suivant sur ces variables :

- contraintes unaires de domaine sur les variables nominales ou numériques ($X \in V$), V étant un intervalle si X est une variable ordonnée.

3. Voir la définition 3.2.1, section 3.2.2, chapitre 3.

– contraintes $(X = Y)$ ou $(X \neq Y)$ sur des variables nominales.

Synthèse de contraintes discriminantes unaires

La contrainte la plus simple pour discriminer E de CE par rapport à une variable contrainte X_i est la contrainte de domaine $X_i \in \mathcal{D}$ avec $\mathcal{D}$ un ensemble si X est une variable nominale (non ordonnée) et $\mathcal{D}$ un intervalle si X est de type ordonné.

Si une variable X de domaine $\mathcal{D}$ du motif est instanciée par θ dans E et par $\sigma_i.\delta$ dans CE, il s'agit de trouver le plus grand sous-ensemble / intervalle de $\mathcal{D}_i$ qui contienne $X\theta$ sans contenir $X\sigma_i\delta$. Un tel sous-ensemble $\mathcal{D}_i$ existe et est unique si la conjonction de contraintes sur X dans θ et la négation de la contrainte sur X dans $\sigma_i\delta$ est satisfiable, en particulier, si $X\theta$ et les $X\sigma_i\delta$ ont des valeurs différentes pour X nominale, ou ont une intersection vide si X est une variable contrainte de type ordonné. Dans ce cas, X est dite *discriminante.*

Exemple 4.1.2
Soit un exemple positif $c \leftarrow p(2,4)$, soit un exemple négatif $c \leftarrow p(2,1), p(1,5)$. Le motif de propositionnalisation est $c \leftarrow p(X,Y)$, où X et Y sont des variables contraintes de domaine $\mathbb{N}$. X n'est pas discriminant, car elle prend la valeur 2 à la fois pour l'exemple et un contre-exemple propositionnel. Par contre, la contrainte $1 < Y < 5$ est maximalement discriminante pour cet exemple et ce contre-exemple.

4.1.4 Bilan

Nous avons défini une relation de généralité contrainte qui permet d'exprimer naturellement dans un cadre PLI l'apprentissage discriminant de clauses contraintes, en utilisant des résultats de la communauté de programmation logique contrainte. L'algorithme d'apprentissage à la base du système DiVS a été étendu pour l'apprentissage de clauses contraintes, en utilisant une méthode à base de propositionnalisation.

STILL (voir chapitre 3) a été implanté par M. Sebag dans deux versions, contrainte et non contrainte. Les expériences sur l'application de la mutagénèse [KS95] donnent de meilleurs résultats avec l'apprentissage de généralisations maximalement discriminantes contraintes, mais finalement assez marginalement [SR97a]). Ces résultats peuvent s'expliquer de plusieurs façons. D'une part, le langage de contraintes peut ne pas être suffisant pour améliorer très sensiblement l'apprentissage, si le problème de la mutagénèse a une faible composante numérique, ou que le langage de contraintes manipulé est trop simple. Enfin, ces résultats sont fortement liés à la méthode d'apprentissage employée. Des expérimentations supplémentaires sur d'autres domaines, ainsi que dans le cadre d'autres méthodes d'apprentissage discriminant doivent être menées afin de confirmer l'utilité de la technique proposée.

4.2 Apprentissage dans CARIN-$\mathcal{ALN}$

Notre but est de proposer un cadre d'apprentissage et d'étudier l'intérêt du point de vue complexité de l'apprentissage dans un langage hybride logique de descriptions - logique définie en présence d'une théorie du domaine également hybride, contenant un ensemble de clauses de Horn et une terminologie. Dans un premier temps, nous présenterons la famille de langage CARIN et à cette occasion, les notations de logique de descriptions. Puis, nous donnerons brièvement, section 4.2.2, la définition formelle de l'inférence centrale dans CARIN, l'implication existentielle. Nous présenterons ensuite notre cadre d'apprentissage, puis nous définirons le test de couverture entre une hypothèse et un exemple (section 4.2.4) et la relation de généralité (section 4.2.5) associés à ce cadre, en soulignant par des exemples les particularités de ces deux opérations par rapport à leurs homologues définis pour un langage de Horn. Nous nous appuierons sur des résultats obtenus pour la famille de langages CARIN afin de prouver que ces deux opérations ainsi définies sont décidables, correctes et complètes. Nous conclurons cette section sur le bilan et les perspectives de ce travail théorique.

4.2.1 Les langages CARIN

CARIN est une famille de langages, chacun combinant une logique de descriptions $\mathcal{L}$ avec un ensemble de clauses de Horn. Plus précisément, une base de connaissances CARIN-$\mathcal{L}$ est formée de trois composantes. La première est une terminologie exprimée dans une logique de descriptions $\mathcal{L}$, la seconde est un ensemble de clauses de Horn et la troisième est un ensemble de faits clos. La terminologie est un ensemble d'énoncés de $\mathcal{L}$ à propos de *concepts* et de *rôles* du domaine représenté. Les concepts et les rôles de la terminologie peuvent aussi apparaître comme des prédicats dans le corps des clauses de Horn ainsi que dans l'ensemble de faits clos[4].

La composante terminologique : la logique de descriptions $\mathcal{ALN}$

Les logiques de descriptions sont une famille de formalismes de représentation des connaissances dont l'origine est le langage KL-ONE [Bra78]. Un nombre important de systèmes ont été développés depuis autour des DLs (e.g., CLASSIC [BPS94], FLEX [QDBK96]) et appliqués à des problèmes réels (voir CLASSIC dans [WWB+93]). Les DLs sont des restrictions de la LPO différentes de celles étudiées en PLI [CH94b] dans lesquelles le calcul de subsomption et sa complexité ont été également étudiés [DLNN91].

Dans les DLs, on manipule trois types d'objets formels : des *individus*, des ensembles d'individus appelés *concepts* qui sont représentés comme des prédicats unaires, et des relations entre deux individus appelés *rôles*, qui sont représentés comme des prédicats binaires. Un *concept* est défini comme un ensemble de propriétés satisfaites par les individus qui sont instances de ce concept. Ces propriétés sont exprimées par des *termes* qui sont construits à partir de concepts atomiques, de rôles et d'un ensemble de connecteurs. Les concepts sont partiellement ordonnés par une relation de subsomption, exprimant une relation d'inclusion entre concepts et qui est habituellement fondée sur une sémantique logique standard. Les inférences élémentaires étudiées dans les DLs sont la *classification de concepts*, qui calcule les relations de subsomption entre concepts (cette opération permet de construire automatiquement la terminologie) et la *reconnaissance d'instances*, qui permet de reconnaître quand un individu appartient à un concept.

Dans les DLs, la connaissance est séparée en deux composantes : la *composante terminologique* (appelée *T-box*), qui contient les définitions de concepts, ou en d'autres termes toute l'information qui va servir à classer les instances, et une *composante assertionnelle* (*A-box*), qui contient dans la plupart des cas des énoncés à propos d'individus. La T-box d'un langage CARIN-$\mathcal{L}$ est une T-box classique, la composante base de faits (section 4.2.1) sera une A-box étendue.

La complexité du raisonnement dans une base de connaissance en CARIN-$\mathcal{L}$ dépend de l'expressivité de la composante terminologique. La complexité du raisonnement dans les DLs varie en fonction de l'ensemble des connecteurs admis dans la règle syntaxique de la DL. CLASSIC

4. Nous renvoyons à [LR98b] pour une comparaison entre CARIN et d'autres langages hybrides.

est la DL la plus expressive pour laquelle le calcul de subsomption soit polynômial étant donné une sémantique non standard. En nous fondant sur les travaux sur la complexité du raisonnement dans les DLs, et contrairement aux travaux exposés dans [BNC00], nous avons choisi la DL $\mathcal{ALN}$ pour ses bonnes propriétés de complexité.

Le langage terminologique de $\mathcal{ALN}$ est défini en utilisant un ensemble $\mathbf{R}$ de rôles atomiques, un ensemble $\mathbf{P}$ de concepts atomiques, les constantes $\top$ and $\bot$, et la règle syntaxique suivante (C et D sont des concepts, P est un concept atomique, R est un rôle atomique, et n est un entier :

$C, D \rightarrow$	$\top$	le concept le plus general
	$\mid \bot$	le concept le plus specifique
	$\mid P$	concept atomique
	$\mid \neg P$	negation d'un concept atomique
	$\mid C \sqcap D$	conjonction de concept
	$\mid \forall R.C$	restriction de valeur
	$\mid \geq nR$	cardinalite pour R (minimum)
	$\mid \leq nR$	cardinalite pour R (maximum)

Cette règle syntaxique est utilisée pour définir des *termes* $\mathcal{ALN}$.

Exemple 4.2.1
$Train \sqcap \geq 2\ a\_voiture \sqcap \leq 4\ a\_voiture \sqcap \forall a\_voiture.carre \sqcap \forall a\_voiture. \leq 2\ a\_charge$ *décrit l'ensemble des trains qui ont entre deux et quatre voitures, qui sont toutes carrées et qui ont au plus deux charges.*

Une terminologie TC est formée d'énoncés ayant l'une des deux formes suivantes : **(1)** $A \equiv C$ (A est équivalent à C) ou **(2)** $A \sqsubseteq C$ (A est subsumé par C, ou C subsume A) où A est un nom de concept et C est un *terme* $\mathcal{ALN}$.

Afin d'avoir une bonne complexité pour les opérations concernant la T-Box, nous considèrerons uniquement des ensembles de définitions de concepts acycliques. Un nom de concept apparaît une seule fois à gauche d'un énoncé de la T-box. Un nom de concept A apparaissant en partie gauche d'un énoncé de type 1 est dit *défini* (i.e., A est doté de propriétés nécessaires et suffisantes). Un nom de concept A apparaissant en partie gauche d'un énoncé de type 2 est dit *primitif* (i.e., A est doté de propriétés nécessaires mais non suffisantes).

Exemple 4.2.2
La définition de concept
$train\_peu\_charge \equiv train \sqcap \geq 1\ a\_voiture \sqcap \forall\ a\_voiture.(\geq 1\ a\_charge \sqcap \leq 2\ a\_charge)$ *signifie qu'un train n'est pas trop chargé s'il a au moins une voiture et que chacune de ses voitures contient une ou deux charges.*

La sémantique (extensionnelle) de la composante terminologique est classique, et nous renvoyons à [LR98b] ainsi qu'à [RV00] pour plus de détails.

Exemple 4.2.3
$polygone \equiv forme\_fermee \sqcap \geq 1\ a\_cote.$
$non\_polygone \equiv forme\_fermee \sqcap \leq 0\ a\_cote.$
$triangle \equiv forme\_trois\_cotes.$
$forme\_trois\_cotes \equiv polygone \sqcap \geq 3\ a\_cote \sqcap \leq 3\ a\_cote.$
$forme\_quatre\_cotes \equiv polygone \sqcap \geq 4\ a\_cote \sqcap \leq 4\ a\_cote.$
$parallelogramme \equiv forme\_quatre\_cotes \sqcap cotes\_paralleles.$
$rectangle \equiv parallelogramme \sqcap angles\_droits.$
$carre \equiv rectangle \sqcap cotes\_egaux.$

$cercle \equiv non\_polygone \ \sqcap \ a\_rayon.$
$ellipse \equiv non\_polygone \ \sqcap \ a\_deux\_foyers.$
$voiture\_vide \equiv voiture \ \sqcap \ \leq 0 \ a\_charge.$
$train\_vide \equiv train \ \sqcap \ \forall a\_voiture.voiture\_vide.$

La classification permet d'engendrer automatiquement la taxonomie représentée partiellement à la figure 4.1 (nous n'avons représenté que les concepts et non les rôles). Notons que cette taxonomie est un graphe et non un arbre.

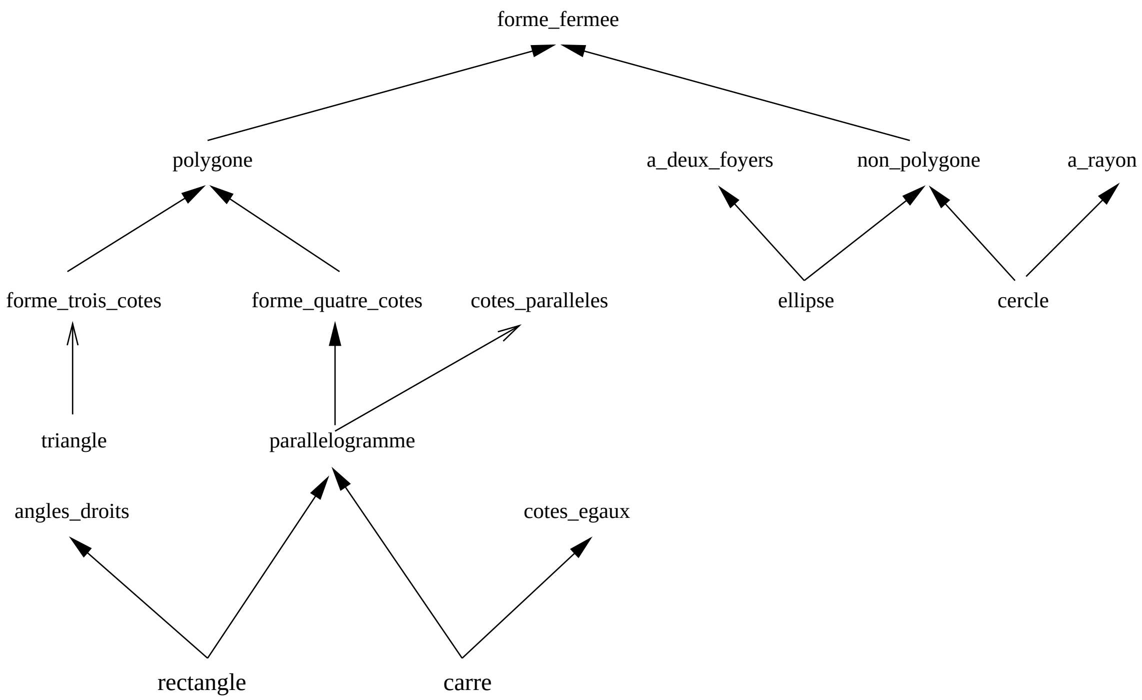

FIG. 4.1 – *Hiérarchie partielle pour TC.*

Base de règles

La composante clause de Horn, ou base de règles, R dans CARIN est un ensemble de clauses de Horn saines de la forme : $q(\overline{Y}) \leftarrow p_1(\overline{Y_1}) \wedge \ldots \wedge p_m(\overline{Y_m})$, où $\overline{Y}, \overline{Y_1}, \ldots, \overline{Y_m}$ sont des n-uplets de variables ou de constantes. Les symboles de prédicats p_i qui apparaissent dans le corps des clauses sont des *symboles de prédicats DL*, (i.e., des prédicats représentant des *concepts* ou *rôles* apparaissant dans la terminologie), ou des symboles de prédicats *ordinaires*. Un littéral pour un concept c défini dans la terminologie TC est de la forme $c(X)$, un littéral rôle pour un rôle r utilisé dans TC a la forme $r(X, Y)$. Un prédicat ordinaire est d'arité quelconque. Le prédicat de tête d'une clause doit être un prédicat ordinaire, car on suppose dans CARIN que TC décrit complètement la connaissance terminologique. Enfin, nous considérons uniquement des bases de règles non récursives.

Exemple 4.2.4
$a\_voiture\_carree(X) \leftarrow train(X), a\_voiture(X, Y), voiture(Y), carre(Y).$
$meme\_forme(X, Y) \leftarrow X \neq Y, carre(X), carre(Y).$
$meme\_forme(X, Y) \leftarrow X \neq Y, rectangle(X), rectangle(Y).$

$\vdots$
$motif(X,Y,Z) \leftarrow train(X), a\_voiture(X,Y), voiture(Y), a\_voiture(X,Z), voiture(Z),$
$\quad carre(Z), a\_charge(Z,U), meme\_forme(Y,U).$
$train\_est(X) \leftarrow train(X), a\_voiture(X,Y), voiture(Y), a\_voiture(X,Z), voiture(Z),$
$\quad a\_charge(Z,L), meme\_forme(Y,L).$
$train\_est(X) \leftarrow train(X), \forall a\_voiture.polygone(X).$

La composante base de faits

La base de faits est un ensemble de faits instanciés concernant des prédicats DL **et/ou** ordinaires. En cela, on peut parler d'une A-box étendue par rapport aux A-box classiquement utilisées en DL.

Exemple 4.2.5
$\mathcal{A} = \{train(a), a\_voiture(a,b), voiture(b), a\_charge(b,c), charge(c), meme\_forme(b,c),$
$a\_voiture(a,d), voiture(d), \leq 0\ a\_charge(d), voiture(e), a\_voiture(a,e), \leq 0\ a\_charge(e),$
$train\_est(a)\}.$

4.2.2 Raisonnement dans CARIN

L'inférence principale dans CARIN, l'*implication existentielle*, est décrite en détail dans [LR98b]. Formellement, le problème de l'implication existentielle pour un langage CARIN-$\mathcal{L}$ se pose de la façon suivante.

Définition 4.2.1 (Implication existentielle)
Soit TC une terminologie décrite dans une logique de descriptions $\mathcal{L}$, soit β et $Q_1, \ldots, Q_n$ des conjonctions quantifiées existentiellement de la forme : $(\exists \overline{Y} p_1(\overline{Y_1}) \wedge \ldots \wedge p_m(\overline{Y_m}))$, telles que les p_i soient des rôles ou des concepts apparaissant dans TC, les $\overline{Y_i}$ soient des n-uplets de variables et de constantes et $\overline{Y} \subseteq \overline{Y_1} \cup \ldots \cup \overline{Y_m}$. Soit Q une disjonction de la forme $Q_1 \vee \ldots \vee Q_n$. Les variables qui ne sont pas existentiellement quantifiées dans Q ou β sont considérées universellement quantifiées. Toute variable de Q doit aussi apparaître dans β. Le problème de l'implication existentielle est de décider si

$$\beta \cup TC \models Q$$

Dans [LR98b], un algorithme pour résoudre le problème d'implication existentielle est donné pour le langage CARIN-$\mathcal{ALCNR}$. Brièvement, cet algorithme construit un *système de contraintes* qui représente l'ensemble des modèles de $\beta \cup TC$ (nommés *modèles canoniques* dans la suite) puis vérifie la satisfaction de Q dans chacun de ces modèles. La preuve de la correction de cet algorithme pour $\mathcal{ALCNR}$[5] est fournie dans [LR98b], avec des résultats de complexité.

Nous allons ici nous appuyer sur ces résultats afin de décrire les inférences de base d'un algorithme d'apprentissage : le test de couverture entre une hypothèse et un exemple et la relation de subsomption (généralité) ordonnant l'espace de recherche. Comme cité plus haut, nous nous restreindrons à $\mathcal{ALN}$ pour le langage terminologique. L'algorithme d'implication existentielle dans ce cas est plus simple que celui décrit dans [LR98b], et en particulier, il n'existe qu'un seul modèle de $\beta \cup TC$. En conséquence, nous ne détaillerons pas ici l'algorithme d'implication existentielle donné dans [LR98b], mais nous illustrerons par des exemples ses

5. $\mathcal{ALCNR}$ est une logique de descriptions plus complexe que $\mathcal{ALN}$, comprenant, entre autres, la quantification existentielle sur les fillers de rôles, $\exists R.C$, la disjonction de concepts, et la conjonction sur les rôles.

particularités qui font qu'il ne se ramène pas à un simple opérateur de point fixe, comme la saturation [Rou94].

Exemple 4.2.6
Soit la terminologie TC*:*
$\{R_1 : carre \equiv rectangle \sqcap cotes\_egaux; R_2 : rectangle \equiv parallelogramme \sqcap angles\_droits\}$
(nous considérons que $parallelogramme$*, est un concept atomique, contrairement à la terminologie donnée à l'exemple 4.2.3).*
Soit l'exemple
$\mathcal{A} = \{train(a),\ \geq 1\ a\_voiture(a),\ \forall a\_voiture.rectangle(a),\ \forall a\_voiture.cotes\_egaux(a)\}$.
Le modèle modèle canonique de $\mathcal{A}$ *étant donné* TC *est*
$\{train(a), \geq\ 1\ a\_voiture(a), \forall\, a\_voiture.rectangle(a),$
$\forall a\_voiture.cotes\_egaux(a), \forall a\_voiture.parallelogramme(a), \forall a\_voiture.angles\_droits(a),$
$\forall a\_voiture.carre(a), a\_voiture(a, b), rectangle(b), cotes\_egaux(b), parallelogramme(b),$
$angles\_droits(b), carre(b)\}$

Plusieurs particularités sont à noter dans cet exemple. Tout d'abord, l'implication existentielle permet d'utiliser les équivalences de TC *«dans les deux sens» : en effet,* $\forall a\_voiture.paralle-logramme(a)$ *et* $\forall a\_voiture.angles\_droits(a)$ *ont été déduits par application de gauche à droite de* R_2 *de* TC*, alors que* $\forall a\_voiture.carre(a)$ *a été déduit par application de droite à gauche de* R_1*. Ensuite, la constante* b *a été générée automatiquement pour rendre compte du fait que* a *avait au moins une voiture. Enfin, les propriétés attachées à toute voiture de* a *ont été automatiquement attachées à* b*.*

Ensuite, CARIN permet d'énoncer, à la fois au niveau de TC, R ou d'une base de faits étendue, des contraintes sur le nombre d'individus en relation avec un individu donné à travers un rôle donné sans que ces individus soient nommés (e.g. $\geq\ 2a\_voiture(a)$: le train a a au moins deux voitures); de même, on peut caractériser tous les individus en relation avec un individu donné à travers un rôle sans pour autant les nommer (e.g. $\forall a\_voiture.carre(a)$: toutes les voitures du train a sont carrées). Ceci permet de dépasser le pouvoir d'expressivité des langages définis, car $\forall a\_voiture.carre(X)$ est équivalent à $\forall Y(a\_voiture(X,Y) \rightarrow carre(Y))$ [6].

D'autre part, les connecteurs de cardinalité permettent d'introduire une forme de négation implicite, en sus de la négation sur les concepts primitifs, que nous n'illustrerons pas ici. En effet, $\leq\ 2a\_voiture(X)$ est logiquement équivalent à
$a\_voiture(X,Y), a\_voiture(X,Z), a\_voiture(X,T) \rightarrow (T = Y \vee T = Z)$.
De même, $\geq\ 2a\_voiture(X)$ est équivalent à $a\_voiture(X,Y), a\_voiture(X,Z), X \neq Z$. En conséquence, une disjonction $Q_1 \vee Q_2$s peut être impliquée logiquement par $TC \cup H \cup \mathcal{A}_i$, alors ni Q_1ni Q_2 n'est vraie isolément (voir [LR98b], p172 pour une explication détaillée du calcul de modèle canonique).

Exemple 4.2.7
Soit une base de faits $\mathcal{A} = \{train(a), a\_voiture(a, b), carre(b)\}$ *et soient deux formules* Q_1 *:* $train(X), a\_voiture(X, Y), rectangle(Y), \geq 2\ a\_charge(Y)$ *et* Q_2 *:* $train(X), a\_voiture(X, Y),$ $cotes\_egaux(Y), \leq 4\ a\_charge(Y)$*.* $\mathcal{A}$ *implique logiquement* $Q_1 \vee Q_2$ *étant donné* TC*, même si aucun des* Q_i *n'est impliqué par* $\mathcal{A}$ *(on ne sait rien sur le nombre de charges de* b*).*

6. Ces langages du premier ordre avec variables quantifiées universellement dans le corps des clauses ont déjà été étudiés dans le cadre de l'apprentissage PLI, voir [Gon97, NCVLRDR99].

Enfin, l'absence d'hypothèse du monde clos sur la base de faits étendue permet de représenter des situations seulement partiellement connues au niveau des exemples. Il est possible également d'introduire explicitement une hypothèse partielle du monde clos, en *fermant* automatiquement les rôles, c'est-à-dire en ajoutant automatiquement les littéraux $\leq\ n\ r(a)$ pour l'observation de n atomes de la forme $r(a, Y)$ dans une base de faits étendue.

4.2.3 Le cadre d'apprentissage

Notre cadre d'apprentissage est celui de l'apprentissage par implication (voir chap. 2, section 2.2). Pour la clarté de la présentation, nous le présentons dans le cadre de l'apprentissage par interprétations, dans sa version orientée pour la classification [BDRJD99]. L'apprentissage par interprétations est équivalent à l'apprentissage par implication dans le langage dont nous nous préoccupons ici, non récursif [DR97]. Étant donné :

- un ensemble C de classes (chaque classe est étiquetée par un prédicat p_i, $1 \leq i \leq n$, et tous les symboles de prédicats représentant des classes ont la même arité),
- un ensemble E d'exemples classifiés : chaque exemple est de la forme $(e, p_i(\overline{a_j}))$, où p_i est la classe de l'exemple et e est un ensemble de faits instanciés où les constantes $\overline{a_j}$ apparaissent,
- une théorie du domaine BK

Le but de l'apprentissage discriminant est de trouver une hypothèse H telle que pour tout exemple $(e, p_i(\overline{a_j})) \ \in\ E$

- $H \wedge e \wedge BK \models p_i(\overline{a_j})$, et
- $H \wedge e \wedge BK \not\models p_j(\overline{a_j})$, pour tout $1 \leq j \leq n$ et $j \neq i$.

Représentation des exemples, de la théorie du domaine et du concept cible

La théorie du domaine est une théorie CARIN-$\mathcal{ALN}$: la partie hiérarchique est une terminologie $\mathcal{ALN}$ notée TC (section 4.2.1), et toute connaissance qui ne peut être représentée dans le langage terminologique est représentée dans R, la composante clause de Horn de la théorie CARIN-$\mathcal{ALN}$. Ceci inclut, en particulier, des clauses utilisant des variables existentielles, des clauses qui expriment des égalités entre restrictions de rôles, ou des clauses exprimant des relations entre plus de deux objets (voir exemple 4.2.4).

Un exemple e_i est une A-box étendue $\mathcal{A}_i$, à laquelle sont associés son étiquette de classe et un n-uplet de constantes $\overline{a_i}$ représentant l'instanciation du prédicat de classe pour e_i. Chaque base de faits représente exactement un exemple. L'ensemble d'apprentissage est donc un ensemble de bases de faits. Chaque exemple est un modèle de l'ensemble des clauses définissant son prédicat de classe.

Exemple 4.2.8
e_1 est représenté par $\mathcal{A}_1 = \{train(a), a\_voiture(a,b), voiture(b), carre(b), a\_charge(b,c),$
$charge(c), a\_voiture(a,d), voiture(d), ellipse(d), a\_charge(d,e), charge(e),$
$\leq 2\ a\_voiture(a), \leq 1\ a\_charge(b), \leq 1\ a\_charge(d), meme\_forme(c,e)\}$.
On associe à e_1 son prédicat de classe et son n-uplet de constante à travers le fait $train\_est(a)$.

Une hypothèse est un ensemble de clauses de Horn décrites dans le langage de la composante clause de Horn et concluant sur le prédicat cible[7].

Exemple 4.2.9
$H = \{train\_est(X) \leftarrow train(X), \geq 1\ a\_voiture(X), \forall\ a\_voiture. \leq 1\ a\_charge(X),$
$a\_voiture(X,Y), polygone(Y); train\_est(X) \leftarrow train(X), \leq 0\ a\_voiture(X)\}$

Le test de couverture ainsi que l'ordre partiel entre les hypothèses vont donc être sensiblement différents à leurs homologues dans langages définis classiques en PLI, comme nous allons le voir dans les sections suivantes.

4.2.4 Test de couverture des exemples

Nous allons maintenant formaliser le problème du test de couverture d'un exemple e_i, représenté comme une base de faits étendue CARIN-$\mathcal{ALN}$ par une hypothèse H définissant un concept cible p, étant donné une théorie CARIN-$\mathcal{ALN}$ T, formée d'une composante terminologique et d'une composante clause de Horn. Ce test réussit si $p(\overline{a})$, où $\overline{a}$ est le n-uplet de constantes associées à e_i, est logiquement impliqué par T et e_i. Plus formellement :

Définition 4.2.2 (Relation de couverture)
Étant donné une base de connaissances CARIN-$\mathcal{ALN}$ T (T est formé d'une composante terminologique TC et d'un ensemble de clauses de Horn R), une hypothèse H définissant un concept cible p, e un exemple représenté comme une base de faits $\mathcal{A}$ avec son n-uplet de constantes $\overline{a}$ associé, H couvre e si et seulement si :

$$T \cup H \cup \mathcal{A} \models p(\overline{a})$$

Tester la couverture de e par H revient à calculer le modèle canonique de $TC \cup R \cup H \cup \mathcal{A}$, puis à vérifier que $p(\overline{a})$ est satisfait dans ce modèle. Nous nous appuyons pour définir ce test sur le théorème suivant (p. 187, [LR98b]) :

Théorème 1
Soit Δ une base de connaissances CARIN-$\mathcal{ALCNR}$ telle que la composante clause de Horn ne soit pas récursive. Soit $p(\overline{a})$ une requête atomique close, où p est soit un atome DL soit un atome ordinaire. Le problème de déterminer si $\Delta \models p(\overline{a})$ est décidable.

Un algorithme correct et complet est fourni pour un tel test de satisfiabilité. Intuitivement, l'implication existentielle est utilisée pour calculer les modèles canoniques de $\mathcal{A}$ étant donné TC, puis un algorithme de chaînage avant est utilisé afin de calculer les extensions des prédicats ordinaires de $\mathcal{A}$ étant donné R. Les deux processus n'interfèrent pas, car R ne définit pas de prédicats DL. L'algorithme termine, car l'algorithme d'implication existentielle ainsi que le chaînage avant terminent tous deux (R est formé de règles non récursives).

Ces résultats nous permettent donc de conclure à l'existence d'un test de subsomption décidable, correct et complet pour CARIN-$\mathcal{ALN}$. Donnons quelques exemples de ce test.

Exemple 4.2.10
Soit H d'après l'exemple 4.2.9 et e_1 d'après l'exemple 4.2.8, soit T formée par la composante terminologique TC de l'exemple 4.2.3 et soit R vide. H_1 couvre e_1, puisque $TC \cup H \cup \mathcal{A}_1 \models$ $train\_est(a)$. Si l'on considère la base de faits étendue de l'exemple 4.2.6, $TC \cup H \cup \mathcal{A} \not\models$

7. Nous rappelons que la tâche d'apprentissage n'est pas d'étendre TC, mais R.

east_train(a) *puisque* $TC \cup H \cup \mathcal{A} \not\models \forall a\_voiture. \leq 1\ a\_charge(a)$ *: il n'y a pas d'hypothèse du monde clos sur* $\mathcal{A}$ *et rien ne permet de déduire de l'absence d'information sur les charges des voitures de* a *que* $\forall a\_voiture. \leq 1\ a\_charge(a)$.

Exemple 4.2.11
Soit e : $\mathcal{A} = \{train(a), \geq 2\ a\_voiture(a), \forall a\_voiture.voiture(a), \forall a\_voiture.carre(a)\}$, *avec l'étiquette de classe* $train\_est(a)$.
Si $H = \{train\_est(X) \leftarrow train(X), \geq 1\ a\_voiture(X), \forall a\_voiture.polygone(X)\}$, *H couvre e étant donné* TC *et* R *(vide).*
Si R *est telle que dans l'exemple 4.2.4 et* $H = \{east\_train(X) \leftarrow train(X), \geq 1\ a\_voiture(X),$ $a\_voiture(X,Y), voiture(Y), a\_voiture(X,Z), voiture(Z), meme\_forme(Y,Z)$, *H couvre e étant donné* TC *et* R.

4.2.5 Ordre partiel sur le langage d'hypothèses

Enfin, il nous reste à définir l'ordre partiel qui structure l'espace de recherche. C'est une extension de la subsomption généralisée [Bun88] (voir annexe) au cas où la théorie du domaine est formée d'un ensemble de clauses définies non récursives et d'une composante terminologique.

Définition 4.2.3 (Ordre de généralité sur le langage d'hypothèses)
Soient H_g *et* H_s *deux hypothèses de la forme :*

$$H_g = p(\overline{X}) \leftarrow p_1(\overline{Y_1}) \wedge \ldots \wedge p_m(\overline{Y_m})$$
$$H_s = p(\overline{X}) \leftarrow p'_1(\overline{Y'_1}) \wedge \ldots \wedge p_n(\overline{Y_n})$$

H_g *est plus général que* H_s *étant donné* TC *et* R *si et seulement si le corps de* H_s, TC *et* R, *impliquent logiquement le corps de* H_g *:*

$$TC, R, \exists \overline{Y'}(p'_1(\overline{Y'_1}) \wedge \ldots \wedge p_n(\overline{Y_n})) \models \exists \overline{Y}(p_1(\overline{Y_1}) \wedge \ldots \wedge p_m(\overline{Y_m}))$$

Si $tete(H_s) = tete(H_g)$ [8], le test de subsomption est en fait un test d'implication logique entre le corps des hypothèses (des conjonctions de littéraux quantifiées existentiellement). De même que pour le test de couverture de la section 4.2.4, nous allons utiliser un cas particulier d'utilisation de l'implication existentielle.

Rappelons que l'implication existentielle décide si $\beta \cup TC \models Q_1 \vee \ldots \vee Q_n$, où β et Q_i sont des conjonctions existentiellement quantifiées. Le cas que nous allons utiliser ici, où $n = 1$, est connu sous le terme de test d'*inclusion de requêtes* [LR98a].

La définition 4.2.3 indique que les corps de H_s et H_g, notés $corps(H_s)$ et $corps(H_g)$ doivent être comparés modulo la théorie du domaine exprimée comme $TC \cup R$. L'implication existentielle permet de les comparer étant donné TC, mais non étant donné R. Notons $compl_{TC}(H_s)$ la conjonction quantifiée existentiellement correspondant à la saturation de $corps(H_s)$ obtenue grâce à l'algorithme d'implication existentielle [9].

De même que pour le test de couverture, il suffit d'appliquer un algorithme de chaînage avant, étant donné R, sur $compl_{TC}(H_s)$, puis de vérifier si $corps(H_g)$ est satisfiable dans le modèle ainsi obtenu. Si c'est le cas, H_g est plus général que H_s, et non sinon. La complétude et la correction de l'implication existentielle [LR98b], ainsi que la complétude de la saturation avec

8. Nous faisons l'hypothèse, sans perte de généralité, que toutes les hypothèses pour le concept cible sont rectifiées, (i.e., elles ont les mêmes variables de tête).

9. Notons que l'algorithme d'implication existentielle reste complet et correct pour β non clos.

R, nous permettent de conclure à la complétude de notre test de subsomption entre hypothèses.

Exemple 4.2.12
Soit TC réduit à la formule :
$cercle \equiv non\_polygone \sqcap a\_rayon$
Soit R la clause :
$non\_polytrain(X) \leftarrow train(X), a\_voiture(X,Y), non\_polygone(Y)$.
Soient $H_s : train\_est(X) \leftarrow train(X), \geq 1a\_voiture(X), \forall a\_voiture.cercle(X)$. *et*
$H_g : train\_est(X) \leftarrow non\_polytrain(X)$.

H_g est plus général que H_s puisque :
$train(X), \geq 1a\_voiture(X), \forall a\_voiture.cercle(X) \cup TC \cup R \models non\_polytrain(X)$.

4.2.6 Bilan - perspectives

Nous avons présenté dans cette section un test de couverture et un ordre partiel de généralité pour l'apprentissage (discriminant) dans un langage hybride qui combine une logique de descriptions aux bonnes propriétés de complexité avec un langage de clauses de Horn. Notre travail présente la première tentative afin de combiner les deux langages pour l'apprentissage.

Le domaine des Logiques de Descriptions a soulevé un intérêt croissant dans la communauté d'apprentissage ces dernières années. Cohen et Hirsh [CH94b, CH94a] donnent des résultats théoriques et expérimentaux sur l'apprenabilité des logiques de descriptions. En particulier, ils prouvent que C-CLASSIC est PAC-apprenable. D'un point de vue pratique, ils proposent des algorithmes qui permettent d'apprendre des concepts C-CLASSIC à partir d'exemples positifs et négatifs de ces concepts. La principale limitation de leur approche est que les concepts et les exemples sont tous représentés dans un langage terminologique. Ceci implique une perte d'information pour les exemples, dans le sens où ils doivent être «abstraits» afin d'être représentés dans le langage terminologique. Par exemple, soit l'exemple défini par l'ensemble de faits : $train(a), a\_voiture(a,b), cercle(b), a\_voiture(a,c), rectangle(c)$, l'abstraction de cet exemple en C-CLASSIC est $(train \ \sqcap \ \geq \ 2 \ a\_voiture)$. Nous n'avons pas cette étape d'abstraction des exemples, puisqu'ils sont représentés dans une base de faits étendue, ce qui nous permet, en utilisant à la fois des atomes DL et ordinaires de représenter toute l'information disponible, qui sera donc exploitée le cas échéant au cours du calcul de la généralisation.

[KM94] est certainement le travail le plus proche du nôtre. Il présente le système KLUSTER, qui combine calcul de moindre généralisé et induction constructive afin d'apprendre des concepts exprimés en logique de descriptions. Plus précisément, KLUSTER met à jour une T-Box, étant donné des exemples représentés comme des A-Boxes. Un algorithme de calcul du moindre généralisé est donné pour la DL considérée. En cas de non correction du moindre généralisé obtenu, la T-Box est étendue avec de nouveaux concept intermédiaires. KLUSTER ne manipule pas, contrairement à notre approche de théorie du domaine ni de langage de concept hybride.

Les perspectives de ce travail sont fort nombreuses. Tout d'abord, il nous reste à préciser le calcul de complexité de ces opérations que nous venons de définir, en nous fondant sur des travaux proches comme [GLR00, Thi00], et à sélectionner les bonnes restrictions sur la composante clause de Horn afin d'obtenir un test de couverture polynômial, et éventuellement un nouveau langage PAC-apprenable. Nous nous inspirerons pour cela des restrictions utilisées en PLI, comme par exemple, les théories ij-déterminées [MF90, DRD94] ou k-locales [KL94].

Ensuite, un algorithme d'apprentissage doit être décrit, afin de valider expérimentalement l'apprentissage dans ce nouveau langage. Nous étudions en ce moment une approche d'apprentissage à base de calcul de moindre généralisé, sous hypothèse du monde clos, en collaboration avec J.-U. Kietz. Une autre direction prometteuse est l'exploitation de la flexibilité offerte par la partie DL pour représenter et apprendre à partir d'exemples incertains [BS98, BSV99].

Chapitre 5

Conclusion : enjeux et perspectives

Depuis ces quelques dix années, la PLI a obtenu des résultats théoriques importants, qui ont permis de mieux cerner les difficultés de l'apprentissage relationnel ainsi que d'apporter quelques éléments de solution pour la mise en pratique de l'apprentissage dans des langages d'ordre un restreints. Nous avons présenté dans les chapitres précédents notre contribution à ce domaine de recherche, organisé autour de trois thèmes principaux :

- les biais d'apprentissage en PLI, ou comment maîtriser la complexité de l'apprentissage relationnel dans des systèmes d'apprentissage à stratégie *«générer et tester»*;
- la propositionnalisation, ou comment adapter des techniques d'apprentissage guidées par les données à la résolution de problèmes d'apprentissage relationnels;
- l'extension du cadre défini de la PLI à l'apprentissage dans des langages de concepts plus complexes, comme l'apprentissage de clauses contraintes, ou l'apprentissage dans des langages hybrides logique définie - logique de descriptions.

La plupart de ces travaux ont été valorisés dans le cadre de contrats ESPRIT, tout d'abord le contrat *ILP* Basic Research Action 6020 (1991-1994), puis son successeur le contrat ILP^2 Long Term Research 20237 (1995-1999).

La recherche en PLI, d'abord principalement animée par des motivations théoriques, s'est investie de plus en plus dans le développement d'applications, où les systèmes de PLI se trouvent comparés à des systèmes attribut-valeurs (voir par exemple, [SMSK96]). Les domaines d'applications de l'apprentissage relationnel en général, et de la PLI en particulier sont en effet nombreux : Biologie, Apprentissage pour le Traitement Automatique du Langage Naturel (une illustration en est la création récente d'un workshop spécifique sur Apprentissage, Logique et Langage [CD99, CDNTKS00]), apprentissage et web [SC98], extraction dans les bases de données relationnelles. Cette confrontation aux applications réelles a fait évoluer le modèle logique initial, et nul doute qu'il continuera à évoluer sous la pression de nouvelles applications. Par exemple, une direction de recherche prometteuse dans le futur en PLI est un cadre d'apprentissage intégrant modèles d'apprentissage statistiques et PLI.

Nous allons plutôt nous attarder, en guise de conclusion, sur un domaine dans lequel nous avons déjà effectué quelques travaux préliminaires, et que nous pensons être un formidable défi en termes d'applications pour la PLI : l'extraction de connaissances à partir de bases de données relationnelles. Après avoir brièvement ébauché une architecture pour l'intégration d'un outil de PLI dans un environnement de découverte dans les bases de données, nous soulèverons un problème de recherche ouvert dans le domaine, celui de la sélection d'attributs et de relations pertinentes pour l'apprentissage discriminant.

La Découverte dans les Bases de Données est un domaine multi - disciplinaire qui rassemble des recherches dans des domaines aussi variés que les statistiques, l'apprentissage, les bases de données autour de la résolution d'un problème réel avec de gros enjeux, à la fois en terme de recherche et d'applications industrielles : l'extraction et l'accès à l'information stockée dans les bases de données relationnelles. Elle regroupe des problématiques très diverses comme l'explicitation d'expertise dans le but d'une prise de décision, l'optimisation de l'accès à l'information grâce à une restructuration de la base de données cible, la mise au point de techniques de résumé et de visualisation des données, le développement de langages de requêtes dédiés, etc. L'apprentissage lui-même n'est qu'une étape, assez peu coûteuse finalement, dans ce processus identifié comme *l'Extraction de Connaissances dans les bases de Données* (ECD dans la suite).

Les problèmes particuliers de l'apprentissage dans un cadre de l'ECD sont les suivants. Les données ont une taille que les systèmes d'apprentissage, même dans un formalisme attribut-valeur, traitent difficilement, tant en nombre de relations ou d'attributs de ces relations qu'en taille (en nombre de n-uplets) de ces relations. La tâche d'apprentissage est également dans la plupart des cas faiblement spécifiée : il est difficile d'exprimer le problème en termes de concept cible, d'exemples (positifs et négatifs), et surtout en termes d'attributs pertinents pour la résolution du problème. Dans un tel cadre, il est plus vraisemblable d'envisager l'extraction de connaissances comme un processus itératif, tel que la tâche d'apprentissage se précise au cours de la résolution, grâce à l'examen des résultats d'exécutions précédentes [Mor89].

Diverses tâches relevant de l'apprentissage ont été isolées en extraction des connaissances [Fay97]. La première regroupe le développements d'outils d'aide à la conception ou reconception d'une base de données. Plus précisément, il s'agit d'extraire d'une base de données des dépendances fonctionnelles, des règles d'associations, des contraintes d'intégrité, des motifs fréquents [Fla93, VLDDR94, DDR97], ou caractérisant des populations déviant par rapport à la norme [Wro97]. Les approches de PLI qui se sont illustrées dans ce domaine adoptent le cadre PLI dit «non monotone», le but étant de trouver des formules satisfaites par un sous-ensemble de la base de données cible, et non de discriminer exemples positifs et négatifs. Citons en particulier le travail de Dehaspe, Toivonen et King [DTK98], qui présente les résultats obtenus dans le cadre du développement d'une application de découverte de sous-structures fréquentes dans les composés chimiques.

Une deuxième tâche, somme toute assez proche de l'apprentissage discriminant classique *modélisation prédictive*, est de modéliser un attribut particulier d'une relation cible, en fonction des autres attributs des relations de la base de données.

Les systèmes de PLI peuvent s'attaquer à des problèmes d'extraction dans les bases de données *multi-relationnelles*, sans passer par une étape manuelle de construction d'une relation «universelle». En effet, un certain nombre de systèmes de PLI appliqués à la découverte dans les bases de données [LM95, BM96, BDR96, NR98] conservent les données sous leur forme initiale, i.e., elles restent gérées par leur SGBD natif. La fonction de couverture par rapport à l'ensemble d'apprentissage est donc complètement déléguée au SGBD. Ceci évite, dans le cas où les données sont de taille importante, d'avoir à gérer les problèmes d'optimisation du test de couverture, le chargement partiel des exemples en mémoire (voir [BDRJD99] pour une discussion). Le système d'apprentissage PLI, s'il adopte une stratégie *«générer et tester»* engendre des hypothèses sous forme logique, étant donné une stratégie indépendante des exemples. Lorsqu'une hypothèse doit être validée contre les exemples, elle est traduite automatiquement sous forme de requête au SGBD. Cette étape est relativement simple, du fait de la proximité du langage des hypothèses manipulé par le système de PLI et d'un langage de requêtes de type SQL. Le SGBD retourne au système de PLI le nombre d'exemples positifs et négatifs couverts par l'hypothèse, qui sur la base de cette évaluation va poursuivre sa recherche dans l'espace des hypothèses logiques.

La PLI permet, dans le cas de l'apprentissage à partir de bases de données multi-relationnelles normalisées, non seulement de traiter des exemples faiblement structurés, mais également d'apprendre à partir de ces exemples des règles complexes, faisant, le cas échéant, de multiples références à plusieurs relations de la base de données, reliées par des liens complexes de dépendances entre clés principales et étrangères. On retrouve, évidemment, les «nœuds» de complexité déjà largement évoqués au cours de ce manuscrit. Nous nous attacherons surtout ici à celui de la taille de l'espace des hypothèses, qui est aggravé par le fait que les données étant de taille imposante, il est encore plus crucial de limiter le nombre d'accès aux données. Au-delà des biais que nous avons évoqués dans le chapitre 2 afin de contraindre l'espace de recherche, le problème majeur ici est celui de la sélection des littéraux pertinents vis-à-vis de la tâche d'apprentissage. Si l'on se place en apprentissage multi-relationnel, au problème de la sélection d'attributs pertinents dans une relation, s'ajoute celui de la sélection des relations pertinentes vis-à-vis de la tâche d'apprentissage.

Pour illustration, développons une variation du fameux exemple d'apprentissage dit des *trains de Michalski* (voir la compétition [MMPS94]), revu et corrigé à la lumière de l'ECD. La base de données relationnelles contient trois relations TRAIN, VOITURE et CHARGE. Les clés de ces trois relations sont respectivement les attributs $Train\#$, $Car\#$ et $Load\#$. Il existe une association 1-n entre la relation $Train$ et la relation Car, ainsi qu'entre la relation Car et $Load$. On a par ailleurs les contraintes $Car.Train\# \subseteq Train.Train\#$ et $Load.Car\# \subseteq Car.Car\#$.

$Train\#$	$Direction$
T_1	$Ouest$
T_2	Est

$Load\#$	$Car\#$	$Forme\_C$
L_{11}	C_{11}	$Cercle$
L_{12}	C_{12}	$Rectangle$
L_{13}	C_{13}	$Cercle$
L_{21}	C_{21}	$Cercle$
L_{22}	C_{22}	$Rectangle$
L_{23}	C_{23}	$Triangle$
L_{24}	C_{24}	$Rectangle$

$Car\#$	$Train\#$	$Rang$	$Forme\_V$	$Longeur\_V$	$Forme\_toit$	$Cloison$	Nb_roues
C_{11}	T_1	1	$Rectangle$	$Court$	$Ouvert$	$Simple$	2
C_{12}	T_1	2	$Baquet$	$Court$	$Ouvert$	$Simple$	2
C_{13}	T_1	3	$Rectangle$	$Long$	$Plat$	$Simple$	2
C_{21}	T_2	1	$Rectangle$	$Court$	$Ouvert$	$Simple$	2
C_{22}	T_2	2	$Baquet$	$Court$	$Ouvert$	$Simple$	2
C_{23}	T_2	3	$Rectangle$	$Long$	$Plat$	$Simple$	2
C_{24}	T_2	4	$Rectangle$	$Long$	$Ouvert$	$Simple$	2

FIG. 5.1 – *Les tables* TRAIN, VOITURE *et* CHARGE

Dans cet extrait de la base de données, les trains T_1 et T_2 de la figure 5.1, ainsi que toutes les voitures et charges qui leur sont associées constituent respectivement un exemple positif et négatif du concept cible. La tâche d'apprentissage, par exemple, consiste à apprendre toutes les règles conjonctives, complètes et correctes qui discriminent les trains allant à l'est (`Train.Direction = 'Est'`) des trains allant à l'ouest (les autres).

Afin de donner une idée de la complexité intrinsèque du problème (qui reste malgré tout un problème jouet), évaluons grossièrement la taille de l'espace de recherche. Supposons que :

– la borne inférieure de l'espace de recherche est l'exemple T_2

– on choisit comme langage d'hypothèse un langage à granularité fine dans lequel chaque littéral correspond à un attribut des relations TRAIN, VOITURE et CHARGE : la représentation logique de T_2 contient 36 littéraux (T_2 contient 4 voitures, décrites chacune par 7 attributs et 4 charges, décrites chacune par 2 attributs)

– on sélectionne comme opérateur d'apprentissage l'abandon ou l'ajout de littéral, qui est un opérateur idéal pour la subsomption OI. [ELMS96]

La taille de l'espace de recherche est de 2^{36}, une approche exhaustive est donc inenvisageable, même si l'on considère que l'on dispose d'un test de couverture optimisé.

Dans un tel problème d'apprentissage, on se trouve donc confronté à un problème de sélection d'attributs, compliqué par le fait qu'un même attribut est référencé plusieurs fois, pour des objets (n-uplets) différents. Si la sélection d'attributs pertinents a suscité beaucoup de travaux en apprentissage attribut-valeur, c'est un thème pour l'instant sous-représenté dans la communauté PLI [Für97] [1].

Dans [NR98], nous avons présenté des opérateurs de changement de représentation, qui, couplés à des opérateurs classiques d'apprentissage PLI, comme l'abandon et l'ajout de littéral, permettent d'effectuer une recherche bi-directionnelle à divers niveaux d'approximation. Brièvement, il s'agissait dans ce travail de modifier le langage d'hypothèses manipulé par le système de PLI, afin de modifier le niveau de précision de la recherche. Dans un premier niveau de recherche «objet», un littéral d'une hypothèse représente un n-uplet d'une relation référencée dans l'exemple initial, graine de l'espace de recherche. La recherche à ce niveau consiste à ajouter ou à abandonner une référence complète à une relation (dans notre exemple, une voiture ou une charge et tous les attributs qui lui sont associés). Une fois isolées les meilleures hypothèses dans ce langage (par exemple, les hypothèses maximalement générales correctes et les hypothèses maximalement spécifiques complètes), un niveau plus fin dit «attribut» permet de poursuivre la recherche dans le nouvel espace ainsi défini. La recherche au niveau attribut consiste à ajouter ou à abandonner une référence à un attribut des relations encore référencées dans les bornes du nouvel espace. Abstraitement, on peut décrire ce type d'apprentissage comme un *lookahead* dirigé par la structure de la base de données (relation, attribut, puis valeur).

La stratégie ainsi décrite, si elle peut être remarquablement efficace, est malheureusement incomplète [Rou99]. En effet, il existe des ensembles d'apprentissage qui feront échouer cette stratégie - dans le sens où elle n'identifiera pas de solution satisfaisant le critère d'apprentissage, même si elle existe dans l'espace de recherche - et qui ne feraient pas échouer une stratégie exhaustive.

Ces recherches, ainsi que celles que nous avons décrites dans les chapitres précédents nous ont donc amené au constat suivant. La PLI permet l'apprentissage de langages cibles très puissants, et offrant une remarquable flexibilité à l'utilisateur pour l'expression de son problème d'apprentissage, le corollaire étant la complexité (parfois inacceptable) de l'apprentissage dans de tels langages. Plus encore qu'en apprentissage attribut-valeur, le développement d'applications ne peut être qu'un processus itératif, qui va permettre à l'utilisateur de préciser de plus en plus la tâche d'apprentissage. L'utilisateur peut être assisté dans cette tâche si on lui donne des moyens simples d'expérimenter et de restreindre a priori la recherche dans des zones prometteuses. Une autre voie, moins explorée, mais plus ambitieuse, est de lui suggérer en cours d'apprentissage des changements de représentation de son problème initial afin de le rendre plus contraint et donc plus efficace, ultimement attribut-valeur ou booléen. Ces changements de représentation

1. Signalons cependant [LGD95], qui se place d'ailleurs dans un cadre d'apprentissage à base de propositionnalisation (voir chap. 3).

peuvent s'appuyer, par exemple, sur des opérateurs de sélection d'attributs discriminants, sur la création de motifs faiblement déterministes (voir les travaux de Zucker [ZG96, Zuc96] sur le sujet), ou sur une reformulation du problème d'apprentissage dans un nouvel espace métrique [Seb97]. Cet axe de recherche, au lieu d'opposer l'apprentissage attribut-valeur et l'apprentissage relationnel, les rendrait complémentaires dans le développement d'applications relationnelles.

Annexe A

Notations et définitions PLI

Nous avons regroupé dans cette annexe un résumé succinct des notions de base en PLI. Elles sont extraites, entre autres, de [NCdW97]. Les notations et les concepts de Programmation Logique sont empruntés à [Llo87].

A.1 Apprentissage attribut-valeur

Dans l'apprentissage dit «attribut-valeur», le langage de représentation des exemples est le suivant: tous les exemples sont décrits selon un ensemble *fixe* d'attributs qui sont soit booléens, soit valués. Un attribut valué prend la forme $att_i = val_i$, où val_i appartient au domaine de définition (type) de l'attribut att_i. Ces attributs peuvent être à valeur dans un type *nominal* ou *discret non ordonné*, ou ordonné discret ($\mathbb{N}$ ou un de ses sous-ensembles) ou ordonné continu ($\mathbb{R}$ ou un de ses sous-ensembles). Les langages de concept pour un système d'apprentissage attribut-valeur peuvent être de complexités diverses. Dans les cas les plus simples, le langage de concept est construit strictement sur le même ensemble de descripteurs que le langage des instances, mais seul un sous-ensemble de descripteurs pertinents vis-à-vis de la tâche d'apprentissage discriminante apparaîtra dans les définitions de concepts finales. Parfois, on admet des descripteurs plus complexes dans le langage d'hypothèses : $att_i > val_i$, $att_i < val_i$ ou $att_i = val_i$, où val_i apparaît dans une hiérarchie de valeurs définie dans une théorie du domaine (voir [Mic83], pour une description détaillée des langages d'hypothèses attribut-valeur).

Le test de couverture entre hypothèses et exemples attribut-valeur est linéaire en fonction du nombre de descripteurs dans l'exemple.

A.2 Notations clauses

Nous travaillons dans ce document, sauf mention spéciale, dans des langages de concept Datalog sans négation et sans récursion. Une clause définie a la forme : $q(\overline{Y}) \leftarrow p_1(\overline{Y_1}), \ldots, p_m(\overline{Y_m})$ où $\overline{Y}, \overline{Y_1}, \ldots, \overline{Y_m}$ sont des tuples de variables ou de constantes. $q(\overline{Y})$ est la *tête* de la clause, le seul littéral positif et $p_1(\overline{Y_1}), \ldots, p_m(\overline{Y_m})$ le *corps* de la clause, une conjonction de littéraux. Les clauses sont saines, i.e., les variables apparaissant dans la tête de la clause apparaissent également dans au moins un des $\overline{Y_i}$. Les variables de $\overline{Y}$ sont appelées variables *universelles* de la clause, et par abus de langage, les variables apparaissant uniquement dans le corps de la clause sont appelées variables *existentielles*. Dans la plupart des cas en PLI, on impose que la clause soit *connectée*, c'est-à-dire qu'il existe un ensemble de littéraux permettant de connecter toute variable existentielle de la clause à une variable universelle.

Si dans un problème d'apprentissage donné, la connexion de la clause n'est pas un critère d'élagage (dans ce cas, tous les littéraux sont reliés à une variable de la tête, ou le littéral de tête de la clause n'a pas de variables), on peut sans perte de généralité se restreindre à travailler uniquement sur le corps des exemples et des hypothèses, considérés alors comme une conjonction de littéraux quantifiés existentiellement.

A.3 Cadre défini pour l'apprentissage discriminant

Muggleton et De Raedt dans [MDR94] introduisent la restriction suivante du cadre normal de l'apprentissage discriminant (voir chapitre 2), lorsque $\mathcal{L}_h$ est une langage défini. C'est ce cadre que nous utilison dans la majorité de nos travaux (sauf pour les travaux décrits au chapitre 4, qui consistent essentiellement à définir un nouveau cadre d'apprentissage pour des langages non définis). Étant donné un ensemble d'apprentissage $E = E^+ \cup E^-$, avec E^+ et E^- des ensembles d'exemples positifs et négatifs pour un concept cible, décrits dans un langage $\mathcal{L}_e$, une théorie du domaine B, décrite dans un langage défini $\mathcal{L}_b$, il s'agit de former une hypothèse H décrite dans un langage défini $\mathcal{L}_h$ telle que

pour tout $e \in E^-$,	e est faux dans $\mathcal{M}(B)$	(correction a priori)
il existe au moins un $e \in E^+$ tq	e est faux dans $\mathcal{M}(B)$	(nécessité a priori)
pour tout $e \in E^+$,	e est vrai dans $\mathcal{M}(B \cup H)$	(H est complète)
pour tout $e \in E^-$,	e est faux dans $\mathcal{M}(B \cup H)$	(H est correcte).

où $\mathcal{M}(B)$ et $\mathcal{M}(B \cup H)$ dénotent les plus petits modèles de Herbrand de B et $B \cup H$.

A.4 Relations de couverture

A.4.1 Préliminaires

Introduisons tout d'abord un ensemble de notations et définitions valables quelle que soit la relation de généralité utilisée. Nous utiliserons la notation suivante : soit une relation de généralité $\preceq$, un ordre partiel sur $\mathcal{L}_h$. Soient deux clauses C et $D \in \mathcal{L}_h$. Nous noterons $C \preceq D$, ou $D \succeq C$ si D subsume au sens large C selon la relation de subsomption $\succeq$. Deux clauses C et D sont logiquement équivalentes ou redondantes, noté $C \equiv D$, si et seulement si $C \succeq D$ et $D \succeq C$. D subsume strictement C, noté $C \prec D$, si $C \preceq D$ et $C \not\equiv D$).

Une clause D est un *moindre généralisé* (ou appartient à la couverture supérieure) de C selon $\preceq$[1] si et seulement si $C \prec D$ et il n'existe pas de E telle que $C \prec E$ et $E \prec D$. Dualement, C est un moindre spécialisé de D ou C appartient à la couverture inférieure de D.

A.4.2 Relations de couverture sans théorie du domaine

La relation de couverture et de généralité la plus employée en PLI est la θ-subsomption.

Définition A.4.1 (θ-subsomption[Plo70])
Une clause D θ-subsume une clause C, noté $C \prec_\theta D$ si et seulement si il existe une substitution θ telle que $D\theta \subseteq C$.

1. Voir chapitre 2.

Sous θ-subsomption et pour deux clauses quelconques de $\mathcal{L}_h$, il existe un moindre généralisé unique, et [Rey70, Plo70] donnent un algorithme polynomial en la taille des clauses en entrée pour le calculer. Par contre, ce moindre généralisé, à l'issue de cet algorithme doit être réduit, car il contient des littéraux logiquement redondants. Plus formellement :

Définition A.4.2 (Réduction)
Une clause H est réduite s'il n'existe pas de littéral l_i dans le corps de H tel que $\exists\theta, l_i\theta \subseteq corps(H)$.

L'algorithme de réduction donné dans [Plo70] fait appel [le nombre d'appels est linéaire en fonction du nombre de littéraux de la clause à réduire) à un test de subsomption qui est, lui, NP-complet. Enfin, il n'existe pas d'opérateurs idéaux dans les espaces de clauses (même pour des langages de type Datalog) ordonnés par la θ-subsomption. La communauté de PLI s'est donc orientée vers des relations de généralité plus contraintes que la θ-subsomption. Citons en deux qui sont depuis largement utilisées.

Définition A.4.3 (Subsomption OI [ELMS96])
Une clause D OI-subsume une clause C, noté $C \preceq_{OI} D$, si et seulement si il existe une substitution injective θ telle que $D\theta \subseteq C$.

Le fait que θ soit injective interdit que deux variables différentes de D soient associées au même terme dans C. Par exemple, soient les clauses C et D suivantes :

$D : c(X) \leftarrow p(X, Y)$ $\qquad$ $C : c(X) \leftarrow p(X, X)$

$C \not\preceq_{OI} D$.

Afin de modéliser une recherche dans un espace ayant pour borne supérieure une clause $\top$ et une borne inférieure notée $\perp$, Badea et Stanciu [BS99] ont introduit la *subsomption faible*. Afin de présenter la définition formelle de la subsomption faible, associons à chaque clause C de l'espace de recherche $\theta_\perp(C)$ qui est la substitution telle que $C\theta_\perp(C) \subseteq \perp$[2].

Définition A.4.4 (Subsomption faible [BS99])
Une clause D subsume faiblement une clause C par rapport à une clause $\perp$, noté $C \preceq_w D$ si et seulement si il existe une substitution θ telle que $D\theta \subseteq C$ et telle qu'il n'existe pas de littéraux L_1 et L_2 de D tels que $L_1\theta = L_2\theta$ et $\theta_\perp(C).\theta = \theta_\perp(D)$.

Soit une clause $\perp : c(a) \leftarrow p(a, a)$. $c(X) \leftarrow p(X, Y) \succeq_w c(X) \leftarrow p(X, X)$, par contre $c(X) \leftarrow p(X, Y), p(Y, X) \not\succeq_w c(X) \leftarrow p(X, X)$.

On a la propriété suivante : $C \succeq_w D \Rightarrow C \succeq_{OI} D \Rightarrow C \succeq_\theta D$, alors que la réciproque est fausse.

Les relations de généralité ci-dessus ne prennent pas en compte une théorie du domaine, ce qui est un énorme désavantage. Supposons dans un premier temps que la théorie du domaine est un programme défini. Plusieurs relations de généralité ont été définies afin de structurer un espace d'hypothèses en présence de théorie du domaine.

A.4.3 Relations de couverture avec théorie du domaine

Parmi elles, la subsomption relative [Plo71] est une extension de la θ-subsomption. Comme cette relation de généralité est indécidable dans le cas d'une théorie du domaine à plus petit modèle de herbrand infini [MF92], nous présenterons plutôt ici la subsomption généralisée,

2. Notons qu'il peut y avoir pour une même clause C plusieurs substitutions $\theta_{\perp i}(C)$ telles que $C\theta_{\perp i}(C) \subseteq \perp$.

une restriction de la subsomption relative introduite par Buntine [Bun88] pour les théories du domaine non récursives.

Définition A.4.5 (Subsomption généralisée [Bun88])
Soit T une théorie du domaine (programme défini). Soit σ une substitution de Skolem pour une clause de Horn C. Une clause D g-subsume, ou subsume au sens de la subsomption généralisée, une clause C étant donné une théorie du domaine T, noté $C, T \prec_g D$ si et seulement si il existe une substitution θ telle que $tete(D)\theta = tete(C)$ et $T, corps(D)\sigma \models corps(C)\theta\sigma$.

Le rôle de σ est d'instancier les variables non instanciées par θ dans le corps de C. La θ-subsomption est une relation de généralité strictement plus faible que la subsomption généralisée, c'est-à-dire

$$C \preceq_w D \Rightarrow C, T \preceq_g D$$

mais la réciproque est fausse. Par exemple, soient les clauses :

$C : c(X,Y) \leftarrow pere(X,Z), mere(Z,Y).$
$D : c(X,Y) \leftarrow pere(X,Z), parent(Z,Y).$
$T_1 : parent(X,Y) \leftarrow pere(X,Y).$ $T_2 : parent(X,Y) \leftarrow mere(X,Y).$

D g-subsume C étant donné T_1 et T_2, mais D ne θ-subsume pas C.

A.5 Opérateurs d'apprentissage

Dans les définitions suivantes, $\mathcal{L}_h$ est un langage de concept, et ρ un opérateur d'apprentissage $\rho : \mathcal{L}_h \longrightarrow 2^{\mathcal{L}_h}$.

Définition A.5.1 (Opérateur idéal)
ρ est idéal si et seulement si il est

- *localement fini : $\rho(C)$ est fini et calculable pour tout $C \in \mathcal{L}_h$*
- *propre : pour tout $C \in \mathcal{L}_h$, $\nexists D \in \rho(C)$ telle que $C \sim D$*
- *complet : pour tout $C, D \in \mathcal{L}_h$, $C \succ D \Rightarrow \exists E \in \rho^*(C)$ telle que $E \sim D$*

Définition A.5.2 (Non redondance)
Un opérateur ρ est non redondant ssi, pour tout C_1, C_2 et $D \in \mathcal{L}_h$, $D \in \rho^*(C_1) \cap \rho^*(C_2) \Rightarrow C_1 \in \rho^*(C_2)$ ou $C_2 \in \rho^*(C_1)$

Définition A.5.3 (Optimalité)
Un opérateur ρ est optimal si et seulement si

- *localement fini*
- *faiblement complet : $\rho^*(\top) = \mathcal{L}_h$*
- *non-redondant*

Définition A.5.4 (Minimalité)
Un opérateur ρ est minimal ssi $\rho(C)$ ne contient que des couvertures inférieures minimales de C et que pour tout $D_1, D_2 \in \rho(C), D_1 \not\succeq D_2$ et $D_2 \not\succeq D_1$.

Bibliographie

[AB92] Albert (P.) et Brunet (P.). – Kads-tool, a case tool for the *common*kads methodology. *In: Conference of International Association of Knowledge Engineers.*

[ADRB95] Adé (H.), De Raedt (L.) et Bruynooghe (M.). – Declarative Bias for Specific-to-General ILP Systems. *Machine Learning*, vol. 20, n1/2, 1995, pp. 119–154.

[AF97] Anthony (S.) et Frisch (A.M.). – Generating numerical literals during refinement. *In: Proc. of the seventh international Workshop on Inductive logic Programming (ILP'97)*, éd. par Lavrač (N.) et Džeroski (S.). pp. 61–76. – Springer Verlag.

[Alp00] Alphonse (E.). – Apprentissage de concepts relationnels par une propositionnalisation sélective. *In: Actes de la seconde Conférence d'Apprentissage (CAP'2000)*, éd. par De la Higuera (C.). pp. 11–20. – Hermes.

[AR98] Alphonse (E.) et Rouveirol (C.). – Object identity for relational learning. – Deliverable LRI1c(1), ESPRIT LTR 20238 (ILP2), 1998.

[AR99] Alphonse (E.) et Rouveirol (C.). – Selective propositionalization. *In: Proc. of the third European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'99)*, éd. par Zytkow (J. M.) et Rauch (J.). pp. 271–277. – Springer Verlag.

[AR00] Alphonse (E.) et Rouveirol (C.). – Lazy propositionalization for relational learning. *In: Proc. of the 14th European Conference on Artificial Intelligence (ECAI'2000)*, éd. par Horn (W.). pp. 256–260. – IOS Press.

[AVK95] Augier (S.), Venturini (G.) et Kodratoff (Y.). – Learning first order logic rules with a genetic algorithm. *In: Proceedings of The First International Conference on Knowledge Discovery & Data Mining (KDD'95)*, pp. 21–26.

[BBVVdV93] Breuker (J.), Bredeweg, Valente et Van de Velde (W.). – Reusable problem solving components: the *common*kads library. *In: 3rd KADS User Meeting, Munich.*

[BD86] Bennett (J. S.) et Dietterich (T. G.). – *The Test Incorporation Hypothesis and the Weak Methods.* – Rapport technique nCSTR-86-30-4, Oregon State University, Department of Computer Science, 1986.

[BdR94] Bergadano (F.) et de Raedt (L.) (édité par). – *Proc. of the 7th European Conference on Machine Learning.* – Springer Verlag, avril 1994, *Lecture Notes in Artificial Intelligence*, volume 784.

[BDR96] Blockeel (H.) et De Raedt (L.). – Relational knowledge discovery in databases. *In: Proc. MLnet Familiarization Workshop on ILP for KDD*, pp. 111–124.

[BDR98] Blockeel (H.) et De Raedt (L.). – Top-down induction of first order decision trees. *Artificial Intelligence*, vol. 101, 1998, pp. 285–297.

[BDRJD99] Blockeel (H.), De Raedt (L.), Jacobs (N.) et Demoen (B.). – Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, vol. 3, n1, 1999, pp. 59–93.

[BG88] Bergadano (F.) et Giordana (A.). – A knowledge intensive approach to concept induction. *In : Proc. of the 5th International conference on Machine Learning*. – Morgan Kaufmann.

[BG90] Bergadano (F.) et Giordana (A.). – Guiding induction with domain theories. *In : Machine Learning : an artificial intelligence approach, vol. 3*, éd. par Kodratoff (Y.) et Michalski (R.S.), pp. 474–492. – Morgan Kaufmann, 1990.

[BG95a] Bergadano (F.) et Gunetti (D.). – *ILP: from ML to Software Engineering*. – MIT Press, 1995.

[BG95b] Bergadano (F.) et Gunetti (D.). – Learning clauses by tracing derivations. *In : ILP: from ML to Software Engineering*. – MIT Press, 1995.

[Bis92] Bisson (G.). – Learning in FOL with a similarity measure. *In : Proc. of the 10th National Conference on Artificial Intelligence (AAAI'92)*, éd. par Swartout (W.). pp. 82–87. – MIT Press.

[BJ93] Brazdil (P.) et Jorge (A.). – Exploiting algorithm sketches in ILP. *In : Proc. of the 3rd International Workshop on Inductive Logic Programming*. – Technical Report, IJS-DP-6707. J. Stefan Institute.

[BM92] Bain (M.) et Muggleton (S.). – Non-monotonic learning. In Muggleton [Mug92], pp. 145–161.

[BM96] Brockhausen (P.) et Morik (K.). – Direct access of an ILP algorithm to a database management system. *In : Proc. MLnet Familiarization Workshop on ILP for KDD*, éd. par De Raedt (L. et al), pp. 95–110.

[BNC00] Badea (L.) et Nienhuys-Cheng (S. H.). – A refinement operator for description logics. In Cussens et Frisch [CF00], pp. 40–59.

[Bor96] Borgida (A.). – On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, vol. 82, 1996, pp. 353–367.

[Bos95] Boström (H.). – Covering vs. divide-and-conquer for top-down induction of logic programs. *In : Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, éd. par Mellish (C.S.). pp. 1194–1200. – Morgan Kaufmann.

[BPS94] Borgida (A.) et Patel-Schneider (P. F.). – Complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, vol. 1, 1994, pp. 278–308.

[Bra78] Brachman (R.J.). – *A Structural Paradigm for Representing Knowledge*. – Rapport technique n3605, BBN Report, 1978.

[BS98] Bouthinon (D.) et Soldano (H.). – An inductive logic programming framework to learn a concept from ambiguous examples. *In : Proc. of the 10th European Conference on Machine Learning (ECML'98)*, éd. par Nédellec (C.) et Rouveirol (C.). pp. 238–249. – Springer Verlag.

[BS99] Badea (L.) et Stanciu (M.). – Refinement operators can be (weakly) perfect. In Džeroski et Flach [DF99], pp. 21–32.

[BSV99] Bouthinon (D.), Soldano (H.) et Ventos (V.). – Aspects théoriques et pratiques de l'apprentissage en cas d'incomplétude des données. *In : Actes de CAP'99: Conférence d'apprentissage*, pp. 145–152.

[Bun88] Buntine (W.). – Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, vol. 36, 1988, pp. 375–399.

[BVdV93] Breuker (J.) et Van de Velde (W.). – *The CommonKADS Library.* – Rapport technique n KADS-II/TM.2/VUB/TR/054/1.1, 1993.

[Cau94a] Causse (K.). – *MCC, vers l'acquisition de connaissances de contrôle.* – Thèse de PhD, LRI, Université Paris Sud, Centre d'Orsay, 1994.

[Cau94b] Causse (K.). – A model for control knowledge. *In : Proc. of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop (Banff'94).* – Banff, Alberta, Canada, 1994.

[CBS95] Champesme (M.), Brézellec (P.) et Soldano (H.). – Réduction de l'espace de recherche en apprentissage : résultats théoriques et expérimentaux. *In : Dixièmes Journées Francophones sur l'Apprentissage*, pp. 65–84.

[CD99] Cussens (J.) et Džeroski (S.) (édité par). – *Proc. of the 1st Learning Language in Logic (LLL) Workshop.* – 1999.

[CDNTKS00] Cardie (C.), Daelemans (W.), Nédellec (C.) et Tjong Kim Sang (E.) (édité par). – *Proc. of the 4th Conference on Computational Natural Language Learning (CoNLL)and of the 2nd Learning Language in Logic (LLL) Workshop.* – 2000.

[CF00] Cussens (J.) et Frisch (A.M.) (édité par). – *Proc. of the 10th International Conference on Inductive Logic Programming (ILP'00).* – Springer Verlag, 2000, *Lecture Notes in Artificial Intelligence*, volume 1866.

[CH94a] Cohen (W. W.) et Hirsh (H.). – The learnability of the description logics with equality constraints. *Machine Learning*, vol. 2, n4, 1994, pp. 169–199.

[CH94b] Cohen (W. W.) et Hirsh (H.). – Learning the CLASSIC description logic: Theoretical and experimental results. *In : International Conference on Knowledge Representation and Reasoning*, pp. 121–133. – Morgan Kaufmann, 1994.

[CH94c] Cohen (W. W.) et Hirsh (H.) (édité par). – *Proc. 11th International Conference on Machine Learning.* – Morgan Kaufmann, 1994.

[Cla85] Clancey (W. J.). – heuristic classification. *Artificial Intelligence*, vol. 27, 1985, pp. 289–350.

[DDR96] Dehaspe (L.) et De Raedt (L.). – DLAB: A declarative language bias formalism. *In : Proc. of the 10th International Symposium on Methodologies for Intelligent Systems (ISMIS'96).* pp. 613–622. – Springer Verlag.

[DDR97] Dehaspe (L.) et De Raedt (L.). – Mining association rules in multiple relations. *In : Proc. of the 7th International Workshop on Inductive Logic Programming (ILP'97)*, éd. par Lavrač (N.) et Džeroski (S.). pp. 78–87. – Springer Verlag.

[DF99] Džeroski (S.) et Flach (P.) (édité par). – *Proc. of the 9th International Conference on Inductive Logic Programming (ILP'99).* – Springer Verlag, 1999, *Lecture Notes in Artificial Intelligence*, volume 1634.

[DKS95] Dougherty (J.), Kohavi (R.) et Sahami (M.). – Supervised and unsupervised discretization of continuous features. *In : Proc. of the twelfth Conference on Machine learning (ICML-95)*, éd. par Prieditis (A.) et Russel (S.). – Morgan Kaufmann.

[DLLP96] Dietterich (T.), Lathrop (R.) et Lozano-Perez (T.). – Solving the multi-instance problem with axis-parallel rectangles. *Artificial Intelligence*, vol. 89, 1996, pp. 31–71.

[DLNN91] Donini (F. M.), Lenzerini (M.), Nardi (D.) et Nutt (W.). – The complexity of concept languages. *In : Principles of Knowledge Representation and Reasoning: 2^{nd} International Conference*, éd. par Allen (J.A.), Fikes (R.) et Sandewall (E.). pp. 151–162. – Cambridge, Mass., 1991.

[DR92] De Raedt (L.). – *Interactive Theory Revision: An Inductive Logic Programming Approach.* – Academic Press, 1992.

[DR97] De Raedt (L.). – Logical settings for concept learning. *Artificial Intelligence*, vol. 95, 1997, pp. 187–201.

[DR98] De Raedt (L.). – Attribute-value learning versus inductive logic programming : The missing link. In Page [Pag98], pp. 1–8. Extended abstract.

[DRB92] De Raedt (L.) et Bruynooghe (M.). – A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, vol. 7(3), 1992, pp. 251–269.

[DRB93] De Raedt (L.) et Bruynooghe (M.). – A theory of clausal discovery. *In : Proc. of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, éd. par Bajcsy (R.). pp. 1058–1063. – Morgan Kaufmann.

[DRBDVL00] De Raedt (L.), Blockeel (H.), Dehaspe (L.) et Van Laer (W.). – Three companions for first order data mining. *In : Inductive Logic Programming for Knowledge Discovery in Relational Data Mining*, éd. par Lavrač (N.) et Džeroski (S.). – Springer Verlag, 2000. à paraître.

[DRD94] De Raedt (L.) et Džeroski (S.). – First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, vol. 70, 1994, pp. 375–392.

[DRD97] De Raedt (L.) et Dehaspe (L.). – Clausal discovery. *Machine Learning*, vol. 26, n2/3, 1997, pp. 99–146.

[DTK98] Dehaspe (L.), Toivonen (H.) et King (R. D.). – Finding frequent substructures in chemical compounds. *In : Proc. of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98).* pp. 30–36. – AAAI Press.

[ELMS96] Esposito (F.), Laterza (A.), Malerba (D.) et Semeraro (G.). – Refinement of datalog programs. *In : Proc. of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, pp. 73–94.

[Fay97] Fayyad (U.). – Knowledge discovery in databases : An overview. *In : Proc. of the 7th International Workshop on Inductive Logic Programming (ILP'97)*, éd. par Lavrač (N.) et Džeroski (S.). pp. 1–16. – Springer Verlag.

[Fei77] Feigenbaum (E.). – The art of artificial intelligence: Themes and case studies in knowledge engineering. *In : Proceedings of IJCAI'77*, pp. 1014–1029.

[Fla93] Flach (P. A.). – Predicate invention in inductive data engineering. *In : Proc. of the 6th European Conference on Machine Learning*, éd. par Brazdil (P.). pp. 83–94. – Springer Verlag.

[Für97] Fürnkranz (J.). – Dimensionality reduction in ILP : A call to arms. *In : Proc. of the IJCAI Workshop Frontiers of ILP*, éd. par De Raedt (L.).

[FZW95] Fensel (D.), Zickwolff (M.) et Wiese (M.). – Are substitutions the better examples? Learning complete sets of clauses with Frog. *In : Proc. of the 5th International Workshop on Inductive Logic Programming (ILP'95)*, éd. par De Raedt (L.). pp. 453–474. – Department of Computer Science, Katholieke Universiteit Leuven.

[Gam95] Gamberger (D.). – A minimization approach to propositional inductive learning. *In : Proc. of the 8th European Conference on Machine Learning (ECML'95)*, éd. par Lavrač (N.) et Wrobel (S.). pp. 151–160. – Springer Verlag.

[GLR00] Goasdoué (F.), Lattès (V.) et Rousset (M.-Ch.). – The use of CARIN language and algorithms for information integration: The PICSEL system. *International Journal of Cooperative Systems*, 2000. – à paraître.

[Gon97] Goncalves (M-E.). – Handling quantifiers in ILP. In Muggleton [Mug97], pp. 337–357.

[GS93] Giordana (A.) et Saitta (L.). – REGAL: An integrated system for learning relations using genetic algorithms. *In : Proc. of* 2^{nd} *International Workshop on Multistrategy Learning*, pp. 234–249.

[Hau88] Haussler (D.). – Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, vol. 36, n 2, 1988, pp. 177–221.

[IA95] Idestam-Almquist (P.). – Generalization of clauses under implication. *Journal of Artificial Intelligence Research*, vol. 3, 1995, pp. 467–489.

[KB97] Karalic (A.) et Bratko (I.). – First order regression. *Machine Learning Journal*, vol. 26:2/3, 1997, pp. 147–176.

[KG86] Kodratoff (Y.) et Ganascia (J.-G.). – Improving the generalization step in learning. *In : Machine Learning: an artificial intelligence approach, vol. 2*, éd. par Michalski (R. S.), Carbonell (J. G.) et Mitchell (T. M.), pp. 215–244. – Morgan Kaufmann, 1986.

[Kie96] Kietz (J-U.). – *Induktive Analyse Relationer Data*. – Thèse de PhD, Thèse Technischen Universität Berlin, 1996.

[KL94] Kietz (J-U.) et Lübbe (M.). – An efficient subsumption algorithm for inductive logic programming. In Cohen et Hirsh [CH94c], pp. 130–138.

[KM94] Kietz (J-U.) et Morik (K.). – A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, vol. 14, n 2, 1994, pp. 193–218.

[KPH98] Kramer (S.), Pfahringer (B.) et Helma (C.). – Stochastic propositionalization of non-determinate background knowledge. In Page [Pag98], pp. 80–94.

[KS95] King (R.D.) et Srinivasan (A.). – Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming*, vol. 13, n 3-4, 1995, pp. 411–434.

[KS96] Kohavi (R.) et Sommerfield (D.). – Data mining using MLC++, a ml library in C++. *In : Tools with AI.* – IEEE Computer Society Press.

[KW92] Kietz (J-U.) et Wrobel (S.). – Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton [Mug92], pp. 335–359.

[LB85] Levesque (H. J.) et Brachman (R. J.). – A fundamental tradeoff in knowledge representation and reasoning. *In : Readings in Knowledge Representation*, éd. par Brachman (R. J.) et Levesque (H. J.). – Morgan Kaufmann, 1985.

[LD94] Lavrač (N.) et Džeroski (S.). – *Inductive Logic Programming : Techniques and Applications.* – Ellis Horwood, 1994.

[LGD95] Lavrač (N.), Gamberger (D.) et Džeroski (S.). – An approach to dimensionality reduction in learning from deductive databases. *In : Proc. of the 5th International Workshop on Inductive Logic Programming (ILP'95)*, éd. par De Raedt (L.). pp. 337–354. – Department of Computer Science, Katholieke Universiteit Leuven.

[Llo87] Lloyd (J.W.). – *Foundations of Logic Programming, second extended edition.* – Springer Verlag, 1987.

[LM95] Lindner (G.) et Morik (K.). – Coupling a relational learning algorithm with a database system. *In : Proc. MLnet Familiarization Workshop on Statistics, ML and KDD*, éd. par et al (Y. Kodratoff).

[LMM87] Lassez (J. L.), Maher (M. J.) et Marriott (K.). – Unification revisited. *In : Foundations of Deductive Databases*, éd. par Gallaire (H.) et Minker (J.), pp. 293–322. – Morgan Kaufmann, 1987.

[LR98a] Levy (A.) et Rousset (M.-Ch.). – Verification of knowledge bases based on containment checking. *Artificial Intelligence*, vol. 101, n1-2, 1998.

[LR98b] Levy (A. Y.) et Rousset (M.-Ch.). – Combining horn rules and description logics in CARIN. *Artificial Intelligence*, vol. 104, 1998, pp. 165–209.

[Mar88] Marcus (S.). – Salt: a knowledge acquisition tool for propose and revise systems. *In : Automating Knowledge Acquisition for Expert Systems*, éd. par Marcus (S.). – Kluwer Academic, 1988.

[May99] Mayer (E.). – *Apprentissage Inductif de scénarios pour la supervision de réseaux de télécommunications.* – Thèse de PhD, Thèse de l'Université Rennes 1, 1999.

[MB92] Muggleton (S.) et Buntine (W.). – Machine invention of first-order predicates by inverting resolution. In Muggleton [Mug92].

[MDR94] Muggleton (S.) et De Raedt (L.). – Inductive logic programming: Theory and methods. *Journal of Logic Programming*, vol. 19, 1994, pp. 629–679.

[MF90] Muggleton (S.) et Feng (C.). – Efficient induction of logic programs. *In : Proceedings of the 1st Conference on Algorithmic Learning Theory.* pp. 368–381. – Ohmsha, Tokyo, Japan.

[MF92] Muggleton (S.) et Feng (C.). – Efficient induction in logic programs. In Muggleton [Mug92], pp. 281–298.

[Mic83] Michalski (R.S.). – A theory and methodology of inductive learning. *In : Machine Learning : an artificial intelligence approach, vol. 1*, éd. par Michalski (R. S.), Carbonell (J. G.) et Mitchell (T. M.), pp. 474–492. – Morgan Kaufmann, 1983.

[Mit82] Mitchell (T. M.). – Generalization as search. *Artificial Intelligence*, vol. 18, 1982, pp. 203–226.

[Mit91] Mitchell (T. M.). – The need for biases in learning generalizations. *In : Readings in Machine Learning*, pp. 184–191. – Morgan Kaufmann, 1991. First published as Technical Report TR-117, Department of Computer Science, Rutgers University, 1980.

[Mit97] Mitchell (T. M.). – *Machine Learning.* – Mac Graw Hill, 1997.

[MLT93] MLT Consortium. – Final public report, 1993.

[MM89] Marcus (S.) et McDermott (J.). – Salt: A knowledge acquisition language for propose and revise systems. *Artificial Intelligence*, vol. 39(1), 1989, pp. 1–38.

[MMPS94] Michie (D.), Muggleton (S.), Page (D.) et Srinivasan (A.). – *To the international computing community: A new East-West challenge.* – Rapport technique, Oxford University Computing laboratory, Oxford,UK, 1994. URL: ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z.

[Mor89] Morik (K.). – Sloppy modeling. *In : Knowledge Representation and Organisation in Machine Learning*, éd. par Morik (K.). pp. 107–134. – Springer Verlag.

[Mug91] Muggleton (S.) (édité par). – *Proc. of the 1st International Workshop on Inductive Logic Programming.* – University of Porto, 1991.

[Mug92] Muggleton (S.) (édité par). – *Inductive Logic Programming.* – Academic Press, 1992.

[Mug95] Muggleton (S.). – Inverse entailment and PROGOL. *New Generation Computing*, vol. 13, 1995, pp. 245–286.

[Mug97] Muggleton (S.) (édité par). – *Proc. of the 6th International Workshop on Inductive Logic Programming (ILP'96): Selected papers.* – Springer Verlag, 1997, *Lecture Notes in Artificial Intelligence*, volume 1314.

[MWKE93] Morik (K.), Wrobel (S.), Kietz (J-U.) et Emde (W.). – *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* – Academic Press, 1993.

[NCdW97] Nienhuys-Cheng (S-H.) et de Wolf (R.). – *Foundations of Inductive Logic Programming.* – Springer Verlag, 1997, *Lecture Notes in Artificial Intelligence*, volume 1228.

[NCVLRDR99] Nienhuys-Cheng (S.-H.), Van Laer (W.), Ramon (J.) et De Raedt (L.). – Generalizing refinement operators to learn prenex conjunctive normal forms. In Džeroski et Flach [DF99], pp. 245–256.

[Néd92] Nédellec (C.). – How to specialize by theory refinement. *In : Proc. of the 10th European Conference on Artificial Intelligence (ECAI'92)*, éd. par Neumann (B.). pp. 474–478. – John Wiley.

[Néd95] Nédellec (C.). – Integration of machine learning and knowledge acquisition. *Knowledge Engineering review*, vol. 10, n1, March 1995, pp. 77–81.

[NR93] Nédellec (C.) et Rouveirol (C.). – Hypothesis selection biases for incremental learning. *In: Proc. of the AAAI Spring Symposium Series workshop on Issues for Incremental Learning.*

[NR94] Nédellec (C.) et Rouveirol (C.). – *Specifications of the HAIKU system.* – Rapport technique n928, Université Paris Sud, Laboratoire de Recherche en Informatique, août 1994.

[NR98] Nédellec (C.) et Rouveirol (C.). – Focusing learning in relational databases. – Deliverable LRI1, ESPRIT LTR 20238 (ILP2), 1998.

[NRA+96] Nédellec (C.), Rouveirol (C.), Adé (H.), Bergadano (F.) et Tausend (B.). – Declarative bias in ILP. *In: Advances in Inductive Logic Programming*, éd. par De Raedt (L.), pp. 82–103. – IOS Press, 1996.

[O'H92] O'Hara (L.). – *Knowledge Acquisition in Vital.* – Rapport technique n D2.1.2, Esprit Project 5365, 1992.

[Pag98] Page (D.) (édité par). – *Proc. of the 8th International Workshop on Inductive Logic Programming (ILP'98).* – Springer Verlag, 1998, *Lecture Notes in Artificial Intelligence*, volume 1446.

[Pag00] Page (D.). – ILP: Just do it. In Cussens et Frisch [CF00], pp. 3–18.

[Plo70] Plotkin (G.). – A note on inductive generalization. *In: Machine Intelligence.* – Edinburgh University Press, 1970.

[Plo71] Plotkin (G.). – A further note on inductive generalization. *In: Machine Intelligence.* – Edinburgh University Press, 1971.

[QCJ93] Quinlan (J.R.) et Cameron-Jones (R.M.). – FOIL: A midterm report. *In: Proc. of the 6th European Conference on Machine Learning*, éd. par Brazdil (P.). pp. 3–20. – Springer Verlag.

[QDBK96] Quantz (J.), Dunker (G.), Bergmann (F.) et Kellner (I.). – *The FLEX system.* – Rapport technique, Technische Universität, Berlin, Germany, KIT-Report, 1996.

[Qui86] Quinlan (J.R.). – Induction of decision trees. *Machine Learning*, vol. 1, 1986, pp. 81–106.

[Qui90] Quinlan (J. R.). – Learning logical definitions from relations. *Machine Learning*, vol. 5, n 3, 1990, pp. 239–266.

[Qui91] Quinlan (J.R.). – Determinate literals in inductive logic programming. *In: Proc. of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, éd. par Mylopoulos (J.) et Reiter (R.). – Morgan Kaufmann.

[RA94] Rouveirol (C.) et Albert (P.). – Knowledge level model of a configurable learning system. *In: Proc. of the eigth European Knowledge Acquisition Workshop (EKAW'94)*, éd. par Steels (L.), Schreiber (G.) et Van de Velde (W.). pp. 374–393. – Springer Verlag.

[RA96] Rouveirol (C.) et Albert (P.). – Knowledge level model of a configurable learning system. *IEEE Transactions on Intelligent Systems and their Applications*, 1996, pp. 50–58.

[RADR93] Rouveirol (C.), Adé (H.) et De Raedt (L.). – Bottom-up generalization in ILP. *In : Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, éd. par Bergadano (F.), De Raedt (L.), Matwin (S.) et Muggleton (S.). pp. 59–70. – Morgan Kaufmann.

[Rey70] Reynolds (J. C.). – Transformational systems and the algebraic structure. *In : Machine Intelligence*. – Edinburgh University Press, 1970.

[RG90] Russell (S.) et Grosof (B.). – A sketch of autonomous learning using declarative bias. *In : Machine Learning, Meta-Reasoning and Logics*, éd. par Brazdil (P.) et K. (Konolige), pp. 19–54. – Kluwer Academic, 1990.

[Rou94] Rouveirol (C.). – Flattening and saturation: Two representation changes for generalization. *Machine Learning*, vol. 14, n2, 1994, pp. 219–232.

[Rou99] Rouveirol (C.). – Focusing strategy for learning complex concept definitions. – Deliverable LRI1b, ESPRIT LTR 20238 (ILP2), 1999.

[RV00] Rouveirol (C.) et Ventos (V.). – Towards learning in CARIN-$\mathcal{ALN}$. In Cussens et Frisch [CF00], pp. 191–208.

[SB86] Sammut (C.) et Banerji (R.). – Learning concepts by asking questions. *In : Machine Learning : an artificial intelligence approach*, éd. par Michalski (R. S.), Carbonell (J. G.) et Mitchell (T. M.), pp. 167–192. – Morgan Kaufmann, 1986.

[SC97] Srinivasan (A.) et Camacho (R.). – Experiments in numerical reasoning with inductive logic programming. *In : Machine Intelligence*. – Edinburgh University Press, 1997.

[SC98] Slattery (S.) et Craven (M.). – Combining statistical and relational methods for learning in hypertex domains. In Page [Pag98], pp. 38–52.

[Seb94a] Sebag (M.). – A constraint-based induction algorithm in FOL. In Cohen et Hirsh [CH94c], pp. 275–283.

[Seb94b] Sebag (M.). – Using constraints to building version spaces. In Bergadano et de Raedt [BdR94], pp. 257–271.

[Seb97] Sebag (M.). – Distance induction in first order logic. *In : Proc. of the 15th International Joint Conference on Artificial Intelligence, Workshop on Inductive Logic Programming*, éd. par Raedt de (L.).

[Sha83] Shapiro (E.Y.). – *Algorithmic Program Debugging*. – MIT Press, 1983.

[SHW97] Scheffer (T.), Herbrich (R.) et Wysotzki (F.). – Efficient θ-subsumption based on graph algorithms. In Muggleton [Mug97], pp. 212–228.

[SKB99] Srinivasan (A.), King (R.D.) et Bristol (D.W.). – An assessment of submissions made to the predictive toxicology evaluation challenge. *In : Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*. pp. 270–276. – Morgan Kaufmann.

[SKMS97] Srinivasan (A.), King (R. D.), Muggleton (S. H.) et Sternberg (M. J. E.). – The predictive toxicology evaluation challenge. *In : Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*. pp. 4–9. – Morgan Kaufmann.

[Slo94] Slodzian (A.). – *Configuring decision tree learning algorithms with KresT*. – Rapport technique, VUB, 1994.

[SM95] Srinivasan (A.) et Muggleton (S.). – Comparing the use of background knowledge by two ILP systems. *In : Proc. of the 5th International Workshop on Inductive Logic Programming (ILP'95)*, éd. par De Raedt (L.). – Department of Computer Science, Katholieke Universiteit Leuven.

[SMSK96] Srinivasan (A.), Muggleton (S.), Sternberg (M.J.E.) et King (R.D.). – Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, vol. 85, n1,2, 1996.

[SP91] Silverstein (G.) et Pazzani (M.J.). – Relational cliches: Constraining constructive induction during relational learning. *In : Proc. of the 8th International Workshop on Machine Learning (IWML'91)*, éd. par Birnbaum (L.) et Collins (G.). pp. 203–207. – Morgan Kaufmann.

[SR90] Smith (B. D.) et Rosenbloom (P. S.). – Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. *In : Proceedings of the 8th National Conference on Artificial Intelligence*, éd. par Dietterich (T. S. W.). pp. 848–853. – Hynes Convention Centre, juillet–août 1990.

[SR94] Sebag (M.) et Rouveirol (C.). – Induction of maximally general clauses consistent with integrity constraints. In Wrobel [Wro94b], pp. 195–216. GMD-Studien, 237.

[SR95] Sebag (M.) et Rouveirol (C.). – Constraint inductive logic programming. *In : Proc. of the 5th International Workshop on Inductive Logic Programming (ILP'95)*, éd. par De Raedt (L.). pp. 181–198. – Department of Computer Science, Katholieke Universiteit Leuven.

[SR96] Sebag (M.) et Rouveirol (C.). – Constraint inductive logic programming. *In : Advances in Inductive Logic Programming*, éd. par De Raedt (L.), pp. 277–294. – IOS Press, 1996.

[SR97a] Sebag (M.) et Rouveirol (C.). – Polynomial approximate learning in (constraint) logic programming. In Muggleton [Mug97], pp. 105–126.

[SR97b] Sebag (M.) et Rouveirol (C.). – Tractable induction and classification in first order logic via stochastic matching. *In : 15th Int. Join Conf. on Artificial Intelligence (IJCAI'97), Nagoya, Japon.* pp. 888–893. – Morgan Kaufmann.

[SR00] Sebag (M.) et Rouveirol (C.). – Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, vol. 38, 2000, pp. 41–62.

[SRP96] Sebag (M.), Rouveirol (C.) et Puget (J.F.). – Ilp+stochastic bias = polynomial approximate learning. *In : Proc. of* 3^{nd} *International Workshop on Multistrategy Learning.* – AAAI Press.

[SS88] Schmidt-Schauss (M.). – Implication of clauses is undecidable. *TCS: Theoretical Computer Science*, vol. 59, n3, 1988, pp. 287–296.

[Ste92] Steels (L.). – *Reusability and Configuration of applications by non-programmers.* – Rapport technique nAI-memo 92-4, AI Lab. Free University of Bruxelles, 1992.

[Ste93] Steels (L.). – The componential framework and its role in reusability. *In : Second Generation Expert Systems*, éd. par J-M. (David), J-P. (Krivine) et R. (Simmons). – Springer-Verlag, 1993.

[SWB92] Shreiber (G.), Wielinga (B.) et Breuker (J.). – *KADS: a Principled Approach to Knowledge-Based System Development.* – Academic Press, 1992.

[Tau94] Tausend (B.). – Representing biases for inductive logic programming. In Bergadano et de Raedt [BdR94], pp. 427–430.

[TGL93] Thomas (J.), Ganascia (J.G.) et Laublet (P.). – Model-driven knowledge acquisition and knowledge-biased machine learning. *In : Proceedings of the IJCAI-93 Workshop on Machine Learning and Knowledge Acquisition*, éd. par Tecuci (G.), Kedar (S.) et Kodratoff (Y.), pp. 220–235.

[Thi00] Thiam (S. M.). – Test de satisfiabilité dans CARIN-ALN : Application à la vérification de plans de requêtes dans PICSEL. – Rapport de DEA TIBD, 2000.

[TNM00] Tamaddoni-Nehzad (A.) et Muggleton (S. H.). – Searching the subsumption lattice by a genetic algorithm. *In : Proc. of the 10th International Conference on Inductive Logic Programming*, éd. par Cussens (J.) et A. (Frisch). pp. 243–252. – Springer Verlag.

[Tor95] Torre (F.). – Exploitation de biais de langage en apprentissage symbolique automatique dans un cadre logique. – Rapport de DEA, juin 1995.

[Tor00] Torre (F.). – *Intégration des biais de langage à l'algorithme générer et Tester - Contribution à l'apprentissage disjonctif.* – Thèse de PhD, Université Paris Sud, 2000.

[TR97a] Torre (F.) et Rouveirol (C.). – Natural ideal operators in inductive logic programming. *In : Proc. of the ninth European Conference on Machine Learning (ECML'97)*, éd. par van Someren (M.) et Widmer (G.). pp. 274–289. – Springer-Verlag.

[TR97b] Torre (F.) et Rouveirol (C.). – Opérateurs naturels en programmation logique inductive. *In : Actes 12èmes Journées Françaises d'Apprentissage (JFA'97)*, pp. 53–64.

[Tsa93] Tsang (E.). – *Foundations of Constraint Satisfaction.* – Academic Press, 1993.

[Utg86] Utgoff (P. E.). – Shift of bias for inductive concept learning. *In : Machine Learning: An Artificial Intelligence Approach*, éd. par Michalski (R. S.), Carbonell (J. G.) et Mitchell (T. M.). pp. 107–148. – Morgan Kaufmann.

[vdL95] van der Laag (P.R.J.). – *An Analysis of Refinement Operators in Inductive Logic Programming.* – Thèse de PhD, Erasmus Universiteit, Rotterdam, the Netherlands, 1995.

[vdLNC94a] van der Laag (P.R.J.) et Nienhuys-Cheng (S-H.). – Existence and nonexistence of complete refinement operators. In Bergadano et de Raedt [BdR94], pp. 307–322.

[vdLNC94b] van der Laag (P.R.J.) et Nienhuys-Cheng (S.H.). – A note on ideal refinement operators in ILP. In Wrobel [Wro94b], pp. 247–262. GMD-Studien, 237.

[VdV94] Van de Velde (W.). – Invited talk: Toward a toolkit for knowledge engineering. *Journées Acquisition des Connaissances*, 1994.

[VdVA92] Van de Velde (W.) et Aamodt (A.). – *Machine Learning Issues in CommonKads.* – Rapport technique n KADS-II/TII.4.3/TR/VUB/002/3.0, 1992.

[VHD91] Van Hentenryck (P.) et Deville (Y.). – Operational semantics of constraint logic programming over finite domains. *In : Proc. of PLILP'91.*

[VHTWS92] Van Heijst (G.), Terpstra (P.), Wielinga (B.) et Shadbolt (N.). – Using generalized directive models in knowledge acquisition. *In : Current developments in Knowledge Acquisition: Ekaw 92.* – Springer-Verlag.

[VLDDR94] Van Laer (W.), Dehaspe (L.) et De Raedt (L.). – Applications of a logical discovery engine. *In : Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pp. 263–274.

[Wro94a] Wrobel (S.). – Concept formation during interactive theory revision. *Machine Learning*, vol. 14, n2, 1994, pp. 169–192.

[Wro94b] Wrobel (S.) (édité par). – *Proc. of the 4th International Workshop on Inductive Logic Programming (ILP'94).* – Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien, 237.

[Wro97] Wrobel (S.). – An algorithm for multi-relational discovery of subgroups. *In : Proc. of the first Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'97)*, éd. par Komorowski (J.) et Zytkow (J.). pp. 78–87. – Springer Verlag.

[WVdVSA92] Wielinga (B.), Van de Velde (W.), Shreiber (G.) et Akkermans (D.). – The *commonkads* framework for knowledge modelling. *In : AAAI Knowledge Acquisition Workshop 1992, Banff, Canada.*

[WWB+93] Wright (J. R.), Weixelbaum (E. S.), Brown (K.), Vesonder (G. T.), Palmer (S. R.), Berman (J. I.) et Moore (H. H.). – A knowledge-based configurator that supports sales, engineering and manufacturing at ATT bell network systems. *In : Proc. of the Innovative Applications of Artificial Intelligence Conference*, pp. 183–193. – Menlo Park, California, 1993.

[WWSE96] Wrobel (S.), Wettschereck (D.), Sommer (E.) et Emde (W.). – Extensibility in data mining systems. *In : Proc. of the Second International Conference on Knowledge Discovery and Data Mining*, éd. par Simoudis (E. et al). – AAAI Press.

[ZG94] Zucker (J.-D.) et Ganascia (J.-G.). – Selective reformulation of examples in concept learning. In Cohen et Hirsh [CH94c], pp. 352–360.

[ZG96] Zucker (J.-D.) et Ganascia (J.-G.). – Changes of representation for efficient learning in structural domains. *In : Proc. of 13th International Conference on Machine Learning.* – Morgan Kaufmann.

[ZG98] Zucker (J.-D.) et Ganascia (J.-G.). – Learning structural indeterminate clauses. In Page [Pag98], pp. 235–244.

[Zuc96] Zucker (J.-D.). – *Appariements et changements de représentation pour l'apprentissage symbolique.* – Thèse de PhD, LAFORIA, Université Paris VI, 1996.