Fabien Torre
sous la direction de Céline Rouveirol

Exploitation de biais de langage en apprentissage symbolique automatique dans un cadre logique

Rapport de stage de DEA
Juin 1995

Table des matières

Remerciements

Je tiens en premier lieu à exprimer ma reconnaissance à Céline Rouveirol ; tout d'abord, pour avoir proposé ce sujet et pour m'avoir accepté comme stagiaire ; puis, pour sa présence constante, la qualité de ses conseils et sa bonne humeur.

Merci aux membres de l'équipe Inférence et Apprentissage que j'ai cotoyés pendant mon stage pour leur accueil et leur sympathie.

Enfin, je veux assurer de mon amitié mes collègues stagiaires, pour quelques discussions intéressantes qui m'ont permis d'améliorer mon travail... Mais surtout pour les innombrables bons moments que nous avons pu passer ensemble pendant cette année de DEA.

Introduction

Un des problèmes majeurs en programmation logique inductive concerne la taille de l'espace de recherche à explorer pour apprendre un concept. Une technique reconnue consiste à réduire la taille de cet espace en restreignant le langage de définition de concept (biais de langage). Une autre méthode consiste à focaliser dynamiquement l'exploration sur un sous-ensemble de l'espace de recherche en fonction d'exemples du concept à apprendre (biais de recherche). Le système HAIKU développé dans l'équipe Inférence et Apprentissage implémente un algorithme générique qui permet de modéliser différents types d'exploration de l'espace de recherche.

Le propos de ce stage est d'intégrer les biais de langage dans HAIKU de manière à produire une plate-forme qui permettra d'étudier les interactions entre les deux types de biais présentés ci-dessus.

Dans un premier chapitre, on présente rapidement la programmation logique inductive, en mettant en avant le problème de la taille des espaces utilisés. On décrit alors la solution que constituent les biais de langage, ainsi que leur utilisation dans les systèmes connus.

On se livre ensuite à une étude de ces biais de langage. Tout d'abord, en mesurant empiriquement leur effet sur la taille de l'espace des hypothèses. Puis, en remettant en cause cette approche, on s'intéressera à des propriétés plus formelles. C'est cette étude qui nous permet de définir l'utilisation que HAIKU fera de ces biais.

Après avoir constaté que les biais classiques ne présentent que rarement les propriétés désirées, nous sommes amenés au chapitre suivant, à chercher de nouveaux opérateurs et de nouveaux biais qui favoriseront l'apparition de propriétés sympathiques.

On termine avec le bilan de cette étude et l'utilisation que l'on peut espérer en faire par la suite.

Chapitre 1

ILP et biais de langage

1.1 Présentation

La programmation logique inductive (ILP) est une discipline récente qui emprunte à la fois à l'apprentissage et à la programmation logique. En tant que branche de l'apprentissage automatique, son but est de mettre au point, de formaliser des techniques permettant de construire des hypothèses à partir des exemples classifiés. On va construire la définition d'un concept en se basant sur des exemples et des contre-exemples de ce concept, mais aussi en s'aidant d'une théorie du domaine.

L'originalité d'ILP par rapport aux autres techniques de ce domaine, est de remplacer la représentation attribut-valeur habituellement utilisée comme langage de représentation du concept appris, par un formalisme du premier ordre. Ce formalisme sert à représenter aussi bien les exemples et les hypothèses que la théorie du domaine.

Enfin, une autre caractéristique intéressante est que les résultats obtenus sont naturellement compréhensibles par un humain.

Evidemment, le gain d'expressivité amène des espaces d'hypothèses de taille très importante, voire infinie. Ceci est un problème majeur en ILP ; il devient crucial de diriger la recherche vers les clauses les plus prometteuses. Tout moyen qui permet de donner ainsi une direction est appelé un *biais*.

Dans [NR94], on distingue trois familles de biais.

1. Les *biais de langage* indiquent quoi chercher en restreignant l'expressivité du langage de concept. Ils interdisent purement et simplement certaines hypothèses. Le biais de langage le plus simple consiste à limiter la taille des clauses.
2. Les *biais de recherche* disent comment chercher, c'est-à-dire, comment se déplacer parmi les hypothèses conservées par les biais de langage. On peut,

par exemple, stipuler que la recherche doit se faire en largeur d'abord, ou en profondeur d'abord.

3. Les *biais de validation* décident si une hypothèse peut faire partie ou non de la définition du concept. Ces critères se ramènent souvent à la correction (aucun exemple négatif n'est couvert) et à la complétude (tous les exemples positifs sont couverts), ou à des affaiblissements de ces notions.

Habituellement, les systèmes implémentent des biais qui semblent pertinents à leurs concepteurs ; ces choix sont alors irrémédiablement fixés dans le code du programme. L'esprit du système HAIKU ([NR94]) est de rendre *déclaratifs* certains des paramètres du processus d'apprentissage. Des systèmes de ce type ont déjà été développés (par exemple NINA [AB92] ou MILES [ST94]) amenant à chaque fois plus de déclarativité. Dans le système HAIKU, en ajustant les différents paramètres, on peut donc simuler une famille de systèmes de type Generate-and-Test existants et, tâche plus intéressante, il devient possible de mesurer l'effet de chaque choix sur la complexité et la qualité de l'apprentissage.

Pour notre part, nous nous intéresserons aux biais de langage et à leur intégration dans un modèle de type Generate-and-Test, le but final étant de permettre leur utilisation judicieuse dans HAIKU. Pour cela, nous allons décrire ces biais tels qu'ils apparaissent dans la littérature ILP. Avant cette description des biais, il convient d'ébaucher les algorithmes qui les utilisent. Nous allons donc maintenenant considérer deux approches de la programmation logique inductive.

1.2 Différentes méthodes

1.2.1 Approche transformationnelle

C'est une partie de ces méthodes qu'HAIKU se propose d'englober. On y trouve, parmi les sytèmes les plus connus : FOIL ([Qui90]), MARVIN ([SB86]) ou encore CLINT ([DRB92]).

On commence par définir quelques notions habituellement utilisées dans cette approche.

Définition 1.2.1 (opérateur d'apprentissage)
Un opérateur est une transformation qui s'applique à une hypothèse. Cette transformation peut souvent s'appliquer de manière non-déterministe, si bien que l'application d'un opérateur $\mathcal{O}$ à une hypothèse H fournit un ensemble de nouvelles hypothèses.

Parmi les opérateurs les plus courants, on trouve l'ajout ou l'abandon d'un littéral et l'absorption.

Définition 1.2.2 (absorption)
L'application de l'absorption à la clause initiale $T_i \leftarrow C_i$, étant donnée la clause de la théorie $T_t \leftarrow C_t$ telle qu'il existe une substitution θ et que $C_t\theta \subseteq C_i$, fournit la clause :

$$T_h \leftarrow Rel \wedge T_t\theta \text{ avec } C_i = C_t\theta \wedge Rel$$

Par exemple, l'absorption sur l'hypothèse

$$mere(X, Y) \leftarrow femme(X), parent(X, Y)$$

à l'aide de la clause de la théorie du domaine

$$\begin{array}{rcl} grand\_pere(A, C) & \leftarrow & pere(A, B), \\ & & parent(B, C), femme(B), \\ & & tom(A), helen(B), liz(C) \end{array}$$

fournit la clause :

$$\begin{array}{rcl} grand\_pere(A, C) & \leftarrow & pere(A, B), \\ & & mere(B, C), \\ & & tom(A), helen(B), liz(C) \end{array}$$

Chaque opérateur respecte un ordre partiel ($\preceq$, "... est plus général que...") sur les clauses, c'est-à-dire que l'on a l'une des deux propriétés suivantes

$$\begin{array}{rcl} \forall H\ \forall H',\ H' \in \mathcal{O}(H) & \Rightarrow & H \preceq H' \\ \forall H\ \forall H',\ H' \in \mathcal{O}(H) & \Rightarrow & H' \preceq H \end{array}$$

La propriété présentée par l'opérateur détermine le type du système. Si c'est la première, on passe du plus général au plus spécifique et le système est dit *ascendant* ou *bottom-up*. Dans l'autre cas, on dira que le système est *descendant* ou *top-down*.

La relation de généralité la plus simple entre les hypothèses est la θ-subsomption.

Définition 1.2.3 (θ-subsomption [Plo70])
On dit que C θ-subsume D ($C \preceq D$), si et seulement s'il existe une substitution θ telle que

$$\begin{array}{rcl} tete(C) & = & tete(D) \\ corps(C)\theta & \subseteq & corps(D) \end{array}$$

Par rapport à cette relation, l'ajout de littéraux est un opérateur de spécialisation alors que l'abandon constitue une généralisation. Par contre, une hypothèse et ses descendantes par absorption ne sont, en général, pas comparables pour cet ordre. L'opérateur d'absorption respecte la subsomption généralisée.

Définition 1.2.4 (subsomption généralisée [Bun88])
C subsume D au sens de la subsomption généralisée (par rapport à un programme logique P) si et seulement s'ils existent des substitutions σ et θ telles que

$$\begin{aligned} tete(C) &= tete(D)\sigma \\ P \wedge corps(C)\theta &\models corps(D)\sigma\theta \end{aligned}$$

Ainsi, ces systèmes, grâce à leur opérateur, vont pouvoir parcourir l'espace de recherche de manière monotone. Reste à savoir de quelle(s) hypothèse(s) on va partir. Pour les *top-down*, trouver l'hypothèse la plus générale est facile, il s'agit de la clause unitaire (la tête de la clause est le concept à définir et son corps est vide). Par contre, pour les *bottom-up* la découverte d'une clause maximalement spécifique est plus délicate. On utilise en général un exemple positif plus ou moins modifié. Certaines modifications fournissent une clause équivalente à l'exemple initial ; c'est le cas de la saturation utilisée dans ITOU ([Rou92]). D'autres fournissent une clause plus générale (variabilisation de l'exemple).

Plutôt que passer en revue les systèmes existants, nous allons travailler sur une version simplifiée de l'algorithme générique d'HAIKU ([NR94]) qui, après instanciation, peut s'identifier à chacun de ces systèmes.

On donne la boucle externe (table 1.1) et la boucle interne (table 1.2).

TABLE 1.1 – Boucle GT

```
1. Initialiser la définition du concept à ∅.
2. Tant que le critère d'arrêt n'est pas validé :
   (a) Calculer les hypothèses de départ et initialiser la
       définition courante avec ces hypothèses.
   (b) Explorer les clauses accessibles depuis les clauses
       de départ et récupérer une définition partielle
       (éventuellement plusieurs) (algorithme 1.2).
   (c) Ajouter la définition partielle à la définition du concept
       et tester le critère d'arrêt.
3. Renvoyer la définition du concept.
```

1.2.2 Approche exhaustive

Il s'agit tout simplement de générer *toutes* les clauses de l'espace de recherche. Puis de les parcourir, sans utiliser d'ordre de généralité sur les hypothèses, en tes-

TABLE 1.2 – Exploration GT

1. Tant que le critère d'arrêt de l'exploration n'est pas satisfait :
   (a) Choisir une hypothèse parmi celles de la définition courante.
   (b) Appliquer l'opérateur d'apprentissage sur cette hypothèse pour obtenir un ensemble de nouvelles hypothèses.
   (c) Parmi ces nouvelles clauses
      i. Enlever les clauses qui ne sont pas cohérentes par rapport aux hypothèses déjà échouées et celles qui sont redondantes avec des hypothèses de la définition courante.
      ii. Enlever celles qui ne sont pas valides vis-à-vis des exemples.
      iii. Dans la définition courante, remplacer l'hypothèse choisie au point 1a par les clauses issues de l'étape 1(c)ii.
   (d) Tester le critère d'apprentissage sur les hypothèses de la définition courante ; celles qui réussissent sont placées dans la définition partielle.
   (e) Tester le critère d'arrêt de l'exploration sur la définition partielle.
2. Renvoyer la définition partielle.

tant simplement pour chacune d'elle leur couverture des exemples. Cette méthode apparaît dans [BG95] pour le système TRACY.

Un moyen de générer toutes les hypothèses est d'utiliser un opérateur classique (définition 1.2.1). La table 1.3 donne un algorithme trivial qui utilise un opérateur de spécialisation pour construire cet espace.

TABLE 1.3 – Pré-processing avec opérateur de spécialisation

1. Initialiser l'ensemble résultat à $\emptyset$ et l'ensemble courant à la clause unitaire.
2. Tant que l'ensemble courant n'est pas vide :
   (a) Enlever une clause de l'ensemble courant et la placer dans l'ensemble résultat.
   (b) Appliquer l'opérateur à cette clause pour obtenir de nouvelles hypothèses.
   (c) Placer ces nouvelles clauses dans l'ensemble courant.
3. Rendre l'ensemble résultat.

Lorsque l'on a généré l'ensemble S de toutes les clauses candidates, le parcours se fait alors selon l'algorithme de la table 1.4 (qui utilise en plus un ensemble d'exemples $E+$ et un ensemble de contre-exemples $E-$).

TABLE 1.4 – Algorithme de TRACY

1. $T \leftarrow \emptyset$
2. pour chaque exemple positif $e+ \in E+$
   (a) trouver C telle que $C \vdash_{sld} e+$ *point de backtrack*
   (b) $T \leftarrow T \cup C$
   (c) si T dérive des exemples de $E-$ alors *backtracker*
3. rendre les clauses de T comme résultat

Si l'on suppose que les exemples sont des buts PROLOG, alors le problème de savoir si un exemple est couvert ou non par la théorie se ramène à résoudre ce but à l'aide des clauses de la théorie.

Les algorithmes 1.1 et 1.4 font apparaître une similitude entre les deux approches : à chaque fois, on dispose d'un opérateur capable de nous fournir des hypothèses.

Dans ce cadre commun, les biais de langage peuvent opérer à deux niveaux :

1. La définition des clauses autorisées a priori, est un biais de langage.
2. Après que l'opérateur ait généré une nouvelle hypothèse, un biais de langage peut décider s'il faut ou non poursuivre avec cette clause ; c'est-à-dire la confronter aux hypothèses déjà échouées et aux exemples dans les méthodes transformationnelles, réappliquer l'opérateur et conserver cette clause pour les méthodes exhaustives.

Ces considérations nous amènent à distinguer deux ensembles d'hypothèses qui sont présents (explicitement ou non) dans chacune des méthodes.

Définition 1.2.5 (espace potentiel)
C'est l'ensemble des clauses autorisées avant toute restriction. Il peut s'agir par exemple de l'ensemble des clauses de Horn que l'on est capable de construire en utilisant les prédicats de la théorie du domaine.

Définition 1.2.6 (espace de recherche)
Il est constitué des hypothèses de l'ensemble précédent qui vérifient les biais de langage. Ce sont ces clauses qui seront finalement candidates pour faire partie de la définition recherchée. Dans le cas des méthodes exhaustives, il s'agit des clauses qui sont fournies par le pré-processing (section 1.2.2). Pour les méthodes transformationnelles, cet ensemble n'apparaît pas explicitement, mais les clauses de la définition trouvée font nécessairement partie de cet ensemble.

Notre étude des biais de langage débute avec les différentes manières de représenter le premier de ces espaces.

1.3 Représentations de l'espace potentiel

Nous passons en revue les différents formalismes permettant de représenter l'espace potentiel. Pour chacun, nous donnerons à titre d'exemple le moyen de dénoter le langage suivant :

$$\begin{aligned}\mathcal{L} = \{ \quad & grandfather(X,Y) \leftarrow male(Y), parent(X,Z) \\ & grandfather(X,Y) \leftarrow female(Y), parent(X,Z) \\ & grandfather(X,Y) \leftarrow male(X), parent(X,Z) \\ & grandfather(X,Y) \leftarrow female(X), parent(X,Z)\end{aligned}$$

$$
\begin{aligned}
grandfather(X,Y) \leftarrow\ & male(Y), parent(X,Z), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & female(Y), parent(X,Z), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & male(X), parent(X,Z), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & female(X), parent(X,Z), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & male(Y), parent(X,Z), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & female(Y), parent(X,Z), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & male(X), parent(X,Z), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & female(X), parent(X,Z), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & male(Y), parent(X,Z),\\
& parent(Z,Y), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & female(Y), parent(X,Z),\\
& parent(Z,Y), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & male(X), parent(X,Z),\\
& parent(Z,Y), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & female(X), parent(X,Z),\\
& parent(Z,Y), parent(X,Y)
\end{aligned}
$$

}

1.3.1 Les ensembles de clauses

Cette représentation est introduite pour le système TRACY ([BG95]) que nous citions à la section 1.2.2 comme exemple des méthodes exhaustives.

Pour une clause, on donne les prédicats pouvant y apparaître, et pour chacun de ces prédicats les arguments possibles. Finalement l'espace est représenté par un *ensemble de clauses*.

On commence par définir les ensembles plus simples :

1. Les *ensembles de termes* fournissent les combinaisons d'arguments autorisés pour un prédicat. $est\_stable(\{A,B\},\{C,D\})$ autorise les littéraux suivants :
$$
\begin{aligned}
& est\_stable(A,C)\\
& est\_stable(A,D)\\
& est\_stable(B,C)\\
& est\_stable(B,D)
\end{aligned}
$$

2. Les *ensembles de prédicats* donnent les combinaisons de prédicats pouvant apparaître dans une clause. $\{cube(X), cube(Y), cube(Z)\}$ autorise les sé-

quences suivantes (en plus de la séquence vide) :

$$
\begin{array}{l}
cube(X) \\
cube(Y) \\
cube(Z) \\
cube(X), cube(Y) \\
cube(X), cube(Z) \\
cube(Y), cube(Z) \\
cube(X), cube(Y), cube(Z)
\end{array}
$$

Ces prédicats peuvent naturellement faire apparaître des *ensembles de termes* dans leurs arguments.

3. Finalement, on utilise les deux ensembles précédents pour construire des clauses qui, à leur tour, forment un *ensemble de clauses*.

L'espace défini à la section précédente est dénoté par l'*ensemble de clauses* suivant :

$$
\begin{array}{l}
\mathcal{L} = \{ \\
\quad \begin{array}{lll}
grandfather(X,Y) \leftarrow & male(\{X,Y\}), parent(X,Z), \\
 & \{parent(\{X,Z\},Y)\} \\
grandfather(X,Y) \leftarrow & female(\{X,Y\}), parent(X,Z), \\
 & \{parent(\{X,Z\},Y)\}
\end{array} \\
\}
\end{array}
$$

1.3.2 Les schémas

Les schémas ([Tau94]) utilisent eux des prédicats variables. On fait apparaître un symbole générique à la place du nom de prédicat et l'on indique simplement les variables que devra utiliser le littéral. Ainsi $P(X)$ dénote l'ensemble des littéraux $P(X)\theta$ où θ est une substitution du second ordre (qui remplace donc P par un nom de prédicat de la théorie du domaine à un argument).

L'exemple qui nous sert de fil conducteur est représenté par les schémas suivants :

$$
\begin{array}{l}
\mathcal{L} = \{ \\
\quad \begin{array}{ll}
grandfather(X,Y) \leftarrow & P(Y), Q(X,Z) \\
grandfather(X,Y) \leftarrow & P(Y), Q(X,Z), parent(X,Y) \\
grandfather(X,Y) \leftarrow & P(Y), Q(X,Z), parent(Z,Y)
\end{array}
\end{array}
$$

$$
\begin{aligned}
grandfather(X,Y) \leftarrow\ & P(Y), Q(X,Z), parent(X,Y), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & P(X), Q(X,Z)\\
grandfather(X,Y) \leftarrow\ & P(X), Q(X,Z), parent(X,Y)\\
grandfather(X,Y) \leftarrow\ & P(X), Q(X,Z), parent(Z,Y)\\
grandfather(X,Y) \leftarrow\ & P(X), Q(X,Z), parent(X,Y), parent(Z,Y)
\end{aligned}
$$

}

Des schémas plus sophistiqués ont été définis : on peut spécifier des conditions sur les prédicats candidats à la substitution (conditions qui peuvent par exemple porter sur l'arité) et sur les variables qui apparaîtront dans le littéral final.

1.3.3 Les modèles de clause

Les deux formalismes précédents sont complémentaires dans le sens où ce qui est facile à représenter dans l'un est difficile à représenter dans l'autre :

- les ensembles de clauses peuvent représenter de manière concise des clauses de longueurs différentes ; par contre, il faudra autant de schémas qu'il y a de longueurs possibles.
- si l'on considère des clauses de même longueur mais utilisant des prédicats différents, ce sont les schémas qui les représentent de la manière la plus naturelle.

L'idée développée dans [ADRB94], est alors d'unifier les deux représentations dans des *modèles de clause* pour cumuler les avantages.

Définition 1.3.1 (modèle de clause)
Un modèle de clause est de la forme

$$Head \leftarrow Body, BodySet$$

avec

- *Head, la tête, est un atome (variabilisé ou non)*
- *Body,* $A_1, \ldots, A_n (n \geq 0)$ *où les* A_i *sont des atomes (variabilisés ou non)*
- *BodySet, un ensemble de prédicats,* $\{A_1, \ldots, A_n\} (n \geq 1)$ *où les* A_i *sont des atomes*

Cette définition utilise les conventions suivantes.

- Un atome est de la forme $p(t_1, \ldots, t_n) (n \geq 0)$ où p est un prédicat, les t_i sont des termes ou des ensembles de termes (comme dans les ensembles de clauses).
- Un atome variabilisé est de la forme $P(t_1, \ldots, t_n)$ où P est un prédicat générique (au sens des schémas).

Voyons maintenant les clauses décrites par le modèle de clause $Head \leftarrow Body, BodySet$.

1. Si les littéraux du modèle ne contiennent pas d'ensembles de termes, alors ce modèle représente les clauses suivantes.

$$\{Head\theta \leftarrow Body\theta \cup B \mid \begin{array}{ll} \theta & \text{substitution du } 2^{\text{nd}} \text{ ordre} \\ B & \subseteq BodySet\} \end{array}$$

2. Si $BodySet = \{b_1, \ldots, b_n\}$ contient $b_i = p(T_1, \ldots, T_k)$ avec T_j un ensemble de termes $\{t_1, \ldots, t_l\}$, alors on dénote le même langage que la clause $Head \leftarrow Body, BodySet'$ où

$$\begin{array}{rcl} BodySet' & = & (BodySet - \{p(T_1, \ldots, T_k)\}) \\ & \cup & \{p(T_1, \ldots, T_{j-1}, t, T_{j+1}, \ldots, T_k) \mid t \in \{t_1, \ldots, t_l\}\} \end{array}$$

3. Si $Body = b_1, \ldots, b_n$ contient un $b_i = p(T_1, \ldots, T_k)$ où p peut être variable et $T_j = \{t_1, \ldots, t_l\}$, alors les clauses ainsi représentées sont les suivantes

$$\begin{array}{l} \{ \\ \quad Head \leftarrow \; b_1, \ldots, b_{i-1}, p(T_1, \ldots, T_{j-1}, t, T_{j+1}, \ldots, T_k), \\ \qquad\qquad\; b_{i+1}, \ldots, b_n, BodySet \mid t \in \{t_1, \ldots, t_l\} \\ \} \end{array}$$

Conformément à l'idée de départ, les schémas comme les ensembles de clauses sont des cas particuliers de ces modèles :

— les schémas sont des modèles qui n'utilisent ni les ensembles de prédicats, ni les ensembles de termes.

— les ensembles de clause sont des modèles sans prédicat variable.

Ce pouvoir de synthèse est visible sur notre exemple puisqu'il est représenté par une expression extrêmement concise :

$$\begin{array}{l} \mathcal{L} = \{ \\ \qquad grandfather(X, Y) \leftarrow \; P(\{X, Y\}), Q(X, Z), \{parent(\{X, Z\}, Y)\} \\ \} \end{array}$$

1.3.4 Les grammaires

L'idée développée dans [Coh94] est d'utiliser une grammaire hors-contexte (type 2 de la hiérarchie de Chomsky). Dans ce cadre, les symboles utilisés sont des littéraux ; on va donc considérer des l-symboles, des l-terminaux et des l-non-terminaux. *body*(T) est le l-symbole de départ où T est le concept que l'on cherche à définir.

Notation 1.3.1
$A \rightarrow B$ where P (où P est un but PROLOG) dénote l'ensemble de règles

$$\begin{cases} A\theta_1 & \rightarrow & B\theta_1 \\ A\theta_2 & \rightarrow & B\theta_2 \\ & \vdots & \\ A\theta_n & \rightarrow & B\theta_n \end{cases}$$

où θ_i est une subsitution associée à une preuve de P.

Plus qu'une représentation, les grammaires offrent une procédure pour générer les clauses dénotées.

Définition 1.3.2 (dérivation)
La notion de dérivation est évidemment très semblable à la dérivation classique à ceci près qu'elle fait intervenir l'unification.
— *Si $\alpha A'\beta$ est une chaîne de l-symboles,*
— *s'il existe une règle $A \rightarrow \gamma$ dans la grammaire G,*
— *si A et A' ont un plus général unificateur θ,*
alors on dit que $\alpha A'\beta$ dérive $(\alpha\gamma\beta)\theta$ en une étape dans G :

$$\alpha A'\beta \overset{G}{\Rightarrow} (\alpha\gamma\beta)\theta$$

Notation 1.3.2
$\alpha_1 \overset{G}{\Rightarrow} \alpha_2 \overset{G}{\Rightarrow} \cdots \overset{G}{\Rightarrow} \alpha_{n-1} \overset{G}{\Rightarrow} \alpha_n$ se réécrit $\alpha_1 \overset{G}{\Rightarrow}{}^* \alpha_n$.

Avec cette dérivation, les clauses représentées se définissent comme le langage dans les grammaires classiques.

Définition 1.3.3 (langage décrit)
Le langage décrit est l'ensemble des chaînes de l-terminaux α tels que $S \Rightarrow^* \alpha$ (où S est le l-symbole de départ). On dit qu'un tel α est un mot du langage.

Ce formalisme est sans aucun doute le plus expressif et le plus souple de ceux présentés ici. Cependant, il est délicat de demander à un utilisateur quelconque l'écriture d'une telle grammaire. Cela revient à exiger une procédure de génération

des clauses autorisées. De plus, cette procédure pour être efficace doit utiliser astucieusement les l-non-terminaux qui a priori n'ont pas de sens en dehors des grammaires. Bref, cette démarche ne cadre pas avec l'esprit déclaratif que nous avons voulu adopter.

La grammaire suivante permet de générer l'espace que nous avons considéré tout au long de ces descriptions.

$$\begin{aligned}
body(grandfather(X,Y)) &\rightarrow sexe(A), parent(X,Z), autre(X,Y,Z) \\
&\quad\ \ where\ member(A,[X,Y]). \\
sexe(A) &\rightarrow male(A). \\
sexe(A) &\rightarrow female(A). \\
autre(X,Y,Z) &\rightarrow []. \\
autre(X,Y,Z) &\rightarrow parent(X,Z). \\
autre(X,Y,Z) &\rightarrow parent(X,Y), autre(X,Y,Z).
\end{aligned}$$

Comme les modèles, les grammaires de Cohen réunissent les clauses de longueur variable (avec les règles récursives) et les prédicats variables (avec les règles disjonctives).

Nous allons maintenant passer en revue les biais de langage qui sont plus délicats à intégrer dans les langages de biais que nous venons de décrire.

1.4 Les différents biais de langage

Les biais de langage se scindent classiquement en deux groupes :

1. Les *biais syntaxiques* sont des restrictions sur la forme des hypothèses. Ils définissent les clauses bien formées. La limitation de la taille des clauses, que nous citions précédemment pour illustrer les biais de langage, est un biais syntaxique.
2. Les *biais sémantiques* nécessitent des informations qui ne sont pas contenues dans l'hypothèse elle-même. Ces informations proviennent soit de la théorie, soit des exemples. Ainsi, la théorie peut contenir la signature de ses prédicats ; on refuse alors les clauses qui ne vérifient pas ces signatures.

Pour notre part, nous diviserons les syntaxiques en deux sous-classes selon leur manière d'opérer : les limitations numériques et les conditions sur des ensembles de variables.

1.4.1 Limitations numériques

Etant donné une fonction sur l'ensemble des clauses, on borne la valeur de cette fonction pour nos hypothèses.

Nombre de littéraux dans le corps des clauses

Si l'on limite ce nombre à 1,

$$\begin{aligned} \text{on peut apprendre } member(X,L) &\leftarrow head(L,X) \\ \text{mais pas } member(X,L) &\leftarrow tail(L,T), member(X,T) \end{aligned}$$

Nombre de variables existentielles

Dans une clause de Horn, les variables de la tête sont implicitement quantifiées universellement et les autres existentiellement. Par exemple, la clause

$$\begin{aligned} intersection(X,Y,Z) \leftarrow\ & head(X,HX), \\ & member(HX,Y), \\ & tail(X,TX), \\ & intersection(TX,Y,W), \\ & cons(HX,W,Z) \end{aligned}$$

comporte trois variables existentielles (HX, TX et W).

Profondeur

On limite l'imbrication des foncteurs donnée par la fonction $Depth$.

$$\begin{aligned} Depth(V) &= 0 \\ Depth(c) &= 1 \\ Depth(f(t_1,\ldots,t_n)) &= 1 + max(\{Depth(t_1),\ldots,Depth(t_n)\}) \\ Depth(p(t_1,\ldots,t_n)) &= max(\{Depth(t_1),\ldots,Depth(t_n)\}) \\ Depth(H \leftarrow L_1 \wedge \ldots \wedge L_k) &= max(\{Depth(H),\ldots,Depth(L_k)\}) \end{aligned}$$

Ainsi, la clause $odd(s(s(X))) \leftarrow odd(X)$ a la profondeur du terme $s(s(X))$, c'est-à-dire 2.

Il s'agit du paramètre i de GOLEM ([MF90]).

Degré

L'idée intuitive est de limiter la longueur d'une chaîne d'instanciations aboutissant à l'instanciation d'une variable.

$$\begin{aligned} Level(c) &= 0 \\ Level(V) &= 0 \text{ si V est une variable de la tête} \\ Level(V) &= 1 + min(\{Level(T) \mid T \text{ et } V \text{ dans un même littéral}\}) \\ Level(C) &= max(\{Level(V) | V \in C\}) \end{aligned}$$

Par exemple, dans la clause suivante

$$\begin{aligned} grandfather(GF) \leftarrow\ & male(GF), \\ & parent(GF, C), \\ & parent(C, LC) \end{aligned}$$

GF, C et LC ont respectivement les degrés 0, 1 et 2 (la clause a donc le degré 2). Il s'agit du paramètre j de GOLEM ([MF90]).

1.4.2 Conditions sur les variables

Range-restricted

Toutes les variables de la tête doivent apparaître dans le corps. La clause

$$\begin{aligned} arche(X, Y, Z) \leftarrow\ & cube(X), \\ & cube(Y) \\ & cube(Z) \end{aligned}$$

est satisfaisante,

$$\begin{aligned} arche(X, Y, Z) \leftarrow\ & cube(X), \\ & cube(Y) \end{aligned}$$

ne l'est pas puisque la variable Z n'apparaît pas dans le corps.

Connexion

On demande que les hypothèses soient liées. Les constantes et les variables de la tête sont liées ; de même pour une variable apparaîssant dans un littéral avec un terme lié. On dit alors qu'une clause est liée si toutes ses variables sont liées. Typiquement, ce biais permet d'éviter les clauses du type

$$father(F) \leftarrow parent(F, C), \\ male(X)$$

Cette clause est *range-restricted* mais la variable X n'est pas liée (et par suite la clause non plus). A noter le lien très fort entre ce dernier biais et le degré : le degré d'une clause est défini si et seulement si cette clause est connectée. L'utilisation simultanée de ces deux biais n'a donc pas d'intérêt.
On trouve parfois une version plus forte de la connexion. Ainsi, le système FOIL ([Qui90]) parcourt la clause de gauche à droite et exige que chaque littéral rencontré soit lié au début de la clause. La clause donnée ici est connectée pour la première définition mais pas pour la seconde :

$$length(L, 1) \leftarrow null(TL), \\ tail(L, TL)$$

Cela tient au fait que la variable TL est liée après une première apparition seule.

1.4.3 Biais sémantiques

Comme à la section précédente, les conditions vont porter sur des ensembles de variables ; la différence est que ces biais vont utiliser en plus une information extérieure à la clause (c'est cela qui leur vaut le nom de sémantique).

Cette information prend la forme d'une étiquette pour chacun des arguments de chacun des prédicats pouvant apparaître dans les hypothèses. Comme précédemment, on peut définir des conditions sur les variables d'une hypothèse. Ces biais sont naturellement plus fins du fait de l'étiquetage.

On rencontre principalement deux étiquetages.

1. l'un utilise l'aspect fonctionnel des préditats ;
2. l'autre exploite le typage des prédicats.

La fonctionnalité (ou modes *input*/*output*) est utilisée, entre autre, dans les systèmes FILP ([BG93]) et TRACY ([BG95]).

A chaque prédicat P d'arité n, on associe un mode fonctionnel :
— m arguments de P sont déclarés comme *inputs* ;
— les $(n - m)$ autres arguments sont considérés comme *outputs* ;
Pour le prédicat *intersection* par exemple, la déclaration naturelle de mode est

$$intersection(in, in, out)$$

où l'on signifie que les deux premiers arguments sont des entrées et que le dernier contiendra le résultat. Par la suite, on supposera les modes suivants :

null(out)
head(in,out)
tail(in,out)
member(in,in)
notmember(in,in)

Avec ces informations, on peut contraindre la forme des hypothèses de différentes manières.

Utilisation des entrées de la tête

On demande que toutes les variables *inputs* de la tête soient présentes dans le corps.
Après confrontation avec ce biais, la clause

$$\begin{aligned} intersection(X,Y,Z) \quad &\leftarrow \quad null(X), \\ &\qquad\quad null(Z) \end{aligned}$$

sera rejetée puisque l'*input* Y n'apparaît pas dans le corps.

Apparition des sorties de la tête dans le corps

On aimerait que les variables *outputs* prennent une valeur dans le corps, ce qui revient à exiger l'apparition de ces variables dans le corps. La clause

$$\begin{aligned} intersection(X,Y,Z) \quad &\leftarrow \quad tail(X,TX), \\ &\qquad\quad intersection(TX,Y,W) \end{aligned}$$

n'est pas satisfaisante puisque l'*output* Z ne prend jamais de valeur.

Utilisation des résultats intermédiaires du corps

On veut que chaque résultat produit par un prédicat du corps soit utilisé (dans la suite du corps avec le statut *input* ou comme *output* de la tête). Par exemple, dans la clause ci-dessous, la variable HL prend une valeur mais n'est pas utilisée.

$$\begin{aligned} member(X,L) \quad &\leftarrow \quad head(L,HL), \\ &\qquad\quad tail(L,TL), \\ &\qquad\quad member(X,TL) \end{aligned}$$

Unicité de l'instanciation des sorties

On refuse la surcharge des *outputs* : une variable ne doit prendre qu'une seule fois le statut *output*. Grâce à ce biais, on évitera par exemple la clause suivante

$$\begin{array}{rcl} intersection(X,Y,Z) & \leftarrow & tail(X,TX), \\ & & head(X,HX), \\ & & intersection(TX,Y,Z), \\ & & intersection(TX,Y,W), \\ & & cons(HX,W,Z) \end{array}$$

puisque Z prend par deux fois une valeur.

Instanciation des entrées

Il serait souhaitable que chacune des variables utilisées comme *inputs* dans le corps ait été instanciée auparavant (soit parce que cette variable est un *input* de la tête, soit parce qu'elle est apparue en *output* dans un littéral précédent).

$$\begin{array}{rcl} intersection(X,Y,Z) & \leftarrow & head(X,HX), \\ & & intersection(TX,Y,W), \\ & & cons(HX,W,Z) \end{array}$$

car la variable TX est utilisée comme *input* alors qu'elle n'a jamais pris de valeur.

MILES ([ST94]) utilise un biais sémantique qui fait intervenir le typage. Ce biais nous intéressera moins car il est, en général, implicitement exploité dans les langages de biais (section 1.3).

Par exemple, la première apparition d'une variable dans un *ensemble de prédicats* (section 1.3.1 ou 1.3.3) fixe son type : si l'on autorise le littéral $head(L,X)$, on sous-entend que L est une liste et X un élément et il est bien clair que l'on ne fera pas apparaître des littéraux contrariant ce typage dans la suite de la clause.

De même dans les grammaires de Cohen (section 1.3.4), on peut facilement forcer les hypothèses à être cohérentes vis-à-vis des types. Avec la règle suivante, on oblige le prédicat *head* à utiliser une liste en premier argument et un élément en second.

$$\begin{array}{rcl} R(X,Y,Z,L1,L2,L3) & \rightarrow & head(A,B) \\ & where & member(A,[L1,L2,L3]), \\ & & member(B,[X,Y,Z]) \end{array}$$

Chapitre 2

Etude des biais classiques

Dans un premier temps, on mesure de manière empirique la force brute des biais, c'est-à-dire le nombre d'hypothèses satisfaisant le biais par rapport à la taille de l'espace de recherche. On discute ensuite de l'applicabilité de ces biais.

2.1 Force empirique des biais

On montre la procédure qui permet de quantifier l'influence des biais sur deux problèmes classiques dans la littérature ILP. Il s'agit d'apprendre le concept d'arche puis l'opération d'intersection.

2.1.1 Les arches

La théorie du domaine est donnée par les figures 2.1 et 2.2.

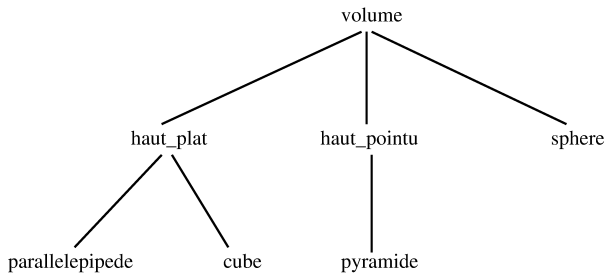

FIGURE 2.1 – Hiérarchie pour le problème d'arche

On veut apprendre la clause suivante pour le concept d'arche,

$$\begin{aligned} arche(X,Y,Z) \leftarrow \quad & haut\_plat(X), \\ & haut\_plat(Y), \end{aligned}$$

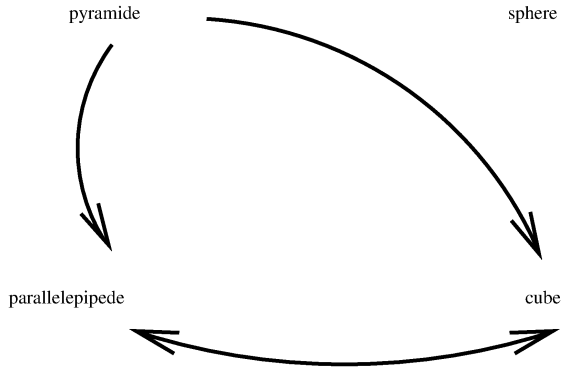

FIGURE 2.2 – Graphe de la relation est_stable

$$\begin{aligned} & volume(Z), \\ & est\_stable(Z, X), \\ & est\_stable(Z, Y). \end{aligned}$$

On utilise un *ensemble de clauses* (section 1.3.1) pour représenter l'espace potentiel. On construit cet ensemble en y mettant tous les prédicats de la théorie du domaine ; de plus, pour chacun de ces prédicat, toute variable peut être argument.

$$\begin{aligned} arche(X, Y, Z) \leftarrow \quad \{ & \\ & haut\_plat(\{X, Y, Z\}), \\ & volume(\{X, Y, Z\}), \\ & haut\_pointu(\{X, Y, Z\}), \\ & sphere(\{X, Y, Z\}), \\ & pyramide(\{X, Y, Z\}), \\ & parallelepipede(\{X, Y, Z\}), \\ & cube(\{X, Y, Z\}), \\ & est\_stable(\{X, Y, Z\}, \{X, Y, Z\}) \\ \}. & \end{aligned}$$

Cet *ensemble de clauses* autorise 2^{30}, soit 1 073 741 824, hypothèses différentes... Les arches, que l'on considère comme un problème jouet, illustrent parfaitement le problème de la taille des espaces d'hypothèses en ILP que nous évoquions en introduction.

Voyons maintenant les biais de la section 1.4 que nous pouvons utiliser sur ce problème. Ce problème est par nature non fonctionnel, ce qui nous empêche

d'utiliser les biais de modes *input/output*. De plus, l'*ensemble de clauses* utilisé n'introduit pas de nouvelles variables et, par suite, tous les biais qui portent sur les variables existentielles sont obsolètes : le nombre de variables existentielles, le degré et la connexion. Enfin, la profondeur n'a pas de sens non plus dans le cas des arches.

Finalement, les biais utilisables sont la limitation du nombre de littéraux (on fixe la limite à la longueur de la clause que l'on veut apprendre, soit 5) et *range-restricted*.

On peut aussi interdire certains littéraux ou certaines conjonctions.

1. Il n'y a pas de sens à tester $est\_stable(A, A)$.
2. Les *contraintes d'intégrité* : elles sont occasionnées par des noeuds de la hiérarchie qui se trouve sur des branches différentes. On exprimera par exemple
$$false \leftarrow sphere(A) \wedge cube(A)$$
3. Les *redondances* : il s'agit des noeuds qui se trouvent sur un même chemin dans la hiérarchie.

$sphere(A) \wedge volume(A)$ est équivalent à $sphere(A)$

Les résultats sont donnés à la table 2.1.

Ce tableau nous permet de faire plusieurs observations.

Les biais sont soit extrêmement puissants (réduisant l'espace à $10\,000^{ème}$ de sa taille initiale), soit extrêmement faibles (conservant plus de 99,9 % de l'espace de départ). Seule l'interdiction de certains littéraux ne présente pas cet extrémisme mais, on peut remarquer que ce biais aurait pu être satisfait si nous y avions pensé au moment de la définition de l'espace potentiel. Considérer ce biais n'aurait pas été très naturel avec les *ensembles de clauses* (mais néanmoins faisable). Cela devient trivial si l'on se place dans le cadre des grammaires de Cohen. Ainsi, la règle de production suivante n'autorise pas les littéraux $est\_stable(A, A)$:

$$\begin{aligned} stabilite(X, Y, Z) \;\; \rightarrow \;\; & est\_stable(A, B) \\ & where\ inclusion([A, B], [X, Y, Z]) \end{aligned}$$

Parmi les plus élagants, on trouve les contraintes d'intégrité si bien qu'il semble très souhaitable de posséder ce genre d'informations ; le système CLAUDIEN ([RB93]) utilise d'ores et déjà ce type de connaissance.

La limitation du nombre de littéraux est elle aussi très puissante et on peut envisager de l'utiliser comme un biais paramétré à la manière de CLINT ([DRB92]) ou NINA ([AB92]). CLINT essaye d'apprendre avec des langages très pauvres et les enrichit en cas d'échec. Avec cette idée, on peut ne considèrer que des clauses

NL : nombre de littéraux limités à 5 **RR** : *range-restricted*
LI : littéraux interdits **RE** : redondances
CI : contraintes d'intégrité

NL	**RR**	**LI**	**RE**	**CI**	**Taille ER**	%
					1 073 741 824	100
X					174 437	0.016
	X				1 072 956 159	99.9
		X			134 217 728	12.5
			X	X	110 592	0.01
X	X				137 245	0.013
X		X			101 584	0.009
X			X	X	34 151	0.003
	X	X			134 021 503	12.48
	X		X	X	108 335	0.01
		X	X	X	13 824	0.001
X	X	X			81 288	0.008
X	X		X	X	31 894	0.003
X		X	X	X	11 080	0.001
	X	X	X	X	13 271	0.001
X	X	X	X	X	10 527	0.001

TABLE 2.1 – Mesures sur le problème d'arche

de longueur 1 ; si les critères de correction et de complétude ne sont pas satisfaits, on recommence en autorisant les clauses de longueur 2, etc.

Le seul biais réellement faible est *range-restricted* qui n'élague quasiment pas. Il s'agit de savoir si cette caractéristique est lié au problème particulier des arches ou si elle est intrinsèque au biais. La réponse est claire. Ce biais peut être satisfait avec seulement deux littéraux de notre ensemble et, on peut ensuite ajouter à loisir les 27 autres littéraux... Plus formellement, on fait les hypothèses suivantes :

1. le concept à apprendre possède m variables ;
2. les prédicats utilisables ont un seul argument qui peut être l'une des m variables.
3. on a n prédicats utilisables.

On a alors un espace potentiel de $2^{n\times m}$ clauses et parmi elles, $(2^n-1)^m$ qui sont *range-restricted*. Le rapport entre la taille de l'espace élagué et la taille de l'espace de départ est donc :

$$\frac{(2^n-1)^m}{2^{n\times m}} = \left(1-\frac{1}{2^n}\right)^m$$

Si l'on veut réduire l'espace à une fraction B, on obtient la condition suivante :

$$n < \log_2\left(\frac{1}{1-B^{\frac{1}{m}}}\right)$$

Ainsi, si notre concept utilise 10 variables (ce qui est énorme, mais notre propos est de favoriser l'élagage par *range-restricted*) et que l'on veut réduire notre espace d'au moins la moitié (ce qui est peu demander par rapport à l'efficacité des autres biais), notre théorie ne pourra pas contenir plus de 3 prédicats... Les hypothèses faites ci-dessus ne sont pas très contraignantes puisque si l'on admet qu'un prédicat peut amener plusieurs variables (c'est le cas de *est_stable* dans le cas de l'arche), le phénomène ne fait que s'accentuer.

Comme le montre la figure 2.3, l'association de deux biais préserve les clauses qui satisfont ces deux biais.

Si l'on a la garantie que la définition cherchée vérifie chaque biais, notre intérêt est de minimiser le nombre des clauses communes à chaque biais. On peut mesurer l'efficacité d'une association par la mesure suivante :

$$Gain(B,B') = \frac{min(|E_B|,|E'_B|) - |E_B \cap E'_B|}{|E_{initial}|}$$

E_B (respectivement E'_B) est l'ensemble des hypothèses de $E_{initial}$ qui satisfont le biais B (respectivement B'). On mesure ainsi le nombre d'hypothèses supprimées par l'association elle-même, par rapport à la taille de l'espace initial. Le tableau

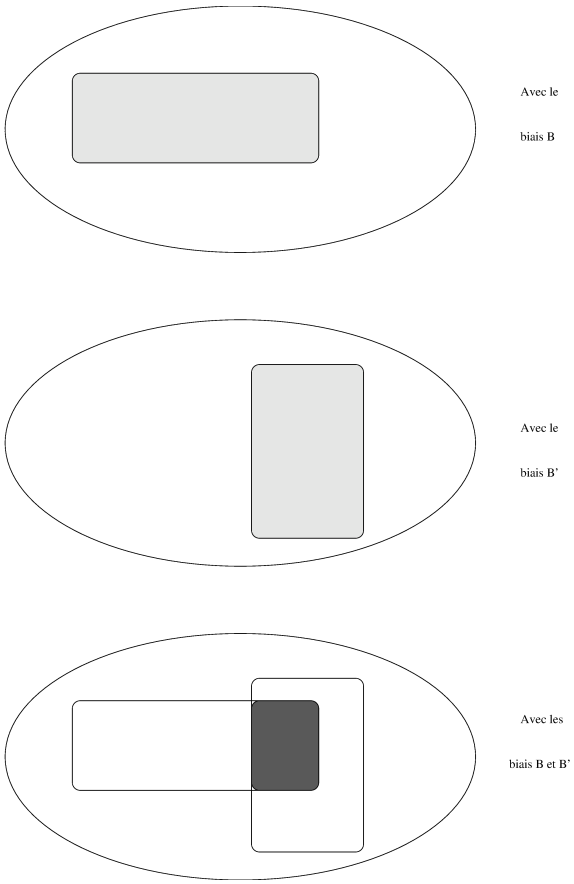

FIGURE 2.3 – Résultat de l'association de deux biais

NL : nombre de littéraux limités à 5 **RR** : *range-restricted*
LI : littéraux interdits **CI** : conjonctions interdites

NL	RR	3.10^{-5}
NL	LI	6.10^{-5}
NL	CI	7.10^{-5}
RR	LI	2.10^{-4}
RR	CI	2.10^{-6}
LI	CI	9.10^{-5}

TABLE 2.2 – Efficacité des associations

2.2 fournit l'efficacité des associations doubles. On y fait apparaître que l'intérêt des associations est très faible. Cela tient principalement au fait que, comme nous l'avons vu, quasiment tous les biais sont déjà très élagants seuls.

Le problème des arches ne nous a pas permis d'observer tous les biais de la section 1.4 et, en particulier les biais utilisant les modes *input/output*. On considère donc maintenant un problème de nature fonctionnelle.

2.1.2 L'intersection

Il s'agit de trouver la définition de l'opération $intersection(X, Y, Z)$ où les trois arguments sont des listes et Z est l'intersection de X et Y.

On utilise une théorie classique pour un problème de liste ; on suppose définis les prédicats les plus simples : *null*, *head*, *tail*, *member* et *notmember*. Pour ces prédicats, on reprend les modes que nous avions définis pour illustrer l'aspect fonctionnel des biais (section 1.4.3).

intersection(in,in,out)
null(out)
head(in,out)
tail(in,out)
member(in,in)
notmember(in,in)

Cette fois-ci, on veut apprendre plusieurs clauses.

$$
\begin{aligned}
intersection(X, Y, Z) \leftarrow \quad & null(X), \\
& null(Z). \\
intersection(X, Y, Z) \leftarrow \quad & head(X, HX), \\
& tail(X, TX), \\
& member(HX, Y), \\
& intersection(TX, Y, W), \\
& cons(HX, W, Z). \\
intersection(X, Y, Z) \leftarrow \quad & head(X, HX), \\
& tail(X, TX), \\
& notmember(HX, Y), \\
& intersection(TX, Y, Z).
\end{aligned}
$$

On utilise un *ensemble de clauses* (section 1.3.1) pour représenter l'espace potentiel : comme dans le cas de l'arche, on utilise tous les littéraux que l'on peut

construire à partir de la théorie. On se restreint cependant à des clauses bien typées (section 1.4.3) ; tacitement, on a X, Y, Z, TX, TY, TZ qui sont des listes et HX, HY, HZ qui sont des éléments. Pour le corps de $intersection(X, Y, Z)$, on a donc l'ensemble suivant,

$$
\begin{aligned}
\{ \\
&null(\{X, Y, Z\}), \\
&head(X, HX), \\
&head(Y, HY), \\
&head(Z, HZ), \\
&tail(X, TX), \\
&tail(Y, TY), \\
&tail(Z, TZ), \\
&member(\{HX, HY, HZ\}, \{X, Y, Z, TX, TY, TZ\}), \\
&notmember(\{HX, HY, HZ\}, \{X, Y, Z, TX, TY, TZ\}), \\
&intersection(\{X, Y, TX, TY\}, \{X, Y, TX, TY\}, \{W, Z\}), \\
&cons(\{HX, HY, HZ\}, \{W, TX, TY, TZ\}, \{W, Z\}) \\
\}
\end{aligned}
$$

qui autorise 2^{101} hypothèses différentes (!).

Il faut maintenant classifier les biais de langage. Comme pour les arches, on va repérer les biais qui n'ont pas d'intérêt parce qu'ils sont vérifiés par toutes les clauses de l'espace potentiel. En plus, apparaîssent des biais qui élimineraient certaines clauses de la définition cherchée. On considèrera finalement ceux qui ont échappé aux deux classes précédentes.

Si l'on demande que certaines variables de la tête soient présentes dans les corps (*range-restricted* et utilisation des *inputs* de la tête), on ne pourra pas apprendre la première clause de la procédure puisqu'elle n'utilise pas la variable *input* Y. De même, si l'on contraint le degré des clauses à être nul, on perd les deux autres clauses qui utilisent des variables auxiliaires.

D'autre part, nos termes se réduisent tous à des variables, si bien que limiter la profondeur n'a pas de sens. On a vu que ne garder que des clauses de degré 0 excluait des solutions ; si l'on admet les clauses de degré 1, on autorise toutes les clauses de l'*ensemble de clauses* considéré sauf celles qui ne sont pas connectées. Par conséquent, ce biais n'a pas de grand intérêt pour notre problème.

Finalement, on utilisera les biais : le nombre de littéraux limités à 5, le nombre de variables existentielles limitées à 3, la connexion, l'apparition des *outputs*, l'utilisation des *outputs* intermédiaires, le refus de surcharge des *outputs* et l'instanciation des *inputs*.

On refusera aussi les littéraux ayant l'une des formes suivantes :

intersection(A,A,_)
cons(_,A,A)

On pose des contraintes d'intégrité en disant que lorsque l'on a établi qu'une liste était vide, on ne teste pas ses composants ; par exemple :

$$false \leftarrow null(A) \wedge head(A, \_)$$

Dans la même idée, si l'on sait qu'un élément appartient à une liste, il est stupide de demander qu'il n'y appartienne pas :

$$false \leftarrow member(A, B), notmember(A, B)$$

Les résultats sont donnés à la table 2.3.

NL	: littéraux limités à 5	**NV**	: variables limitées à 3
CO	: connexion	**AO**	: apparition des *outputs*
UO	: utilisation des *outputs*	**PS**	: pas de surcharge
II	: instanciation des *inputs*	**LI**	: littéraux interdits
CI	: contraintes d'intégrité		

NL	**NV**	**CO**	**AO**	**UO**	**PS**	**II**	**LI**	**CI**	**Taille ER**	%
									2^{101}	100
X									83 463 472	10^{-21}
	X								3×10^{11}	10^{-17}
		X							*	*
			X						$2^{101} - 2^{73}$	99.9
				X					*	*
					X				435×2^{45}	10^{-13}
						X			6×10^{26}	0.024
							X		2^{96}	3.125
								X	2197×2^{56}	10^{-9}

* : en cours de calcul

TABLE 2.3 – Mesures sur le problème d'intersection

On retrouve sensiblement la même chose que pour les arches : les biais sont radicaux.

Les valeurs données ont été obtenues de deux manières différentes : d'une part par le calcul, d'autre part par l'expérimentation grâce à une méthode exhaustive (algorithme 1.3). Les résultats sont bien sûr identiques mais les tests font apparaître que les temps de traitement sont sensiblement les mêmes quels que soient les biais utilisés. C'est cette constatation sur l'expérimentation qui nous amène à la section suivante où nous allons définir des propriétés plus théoriques.

2.2 Propriétés des biais

Nous sommes amenés à relativiser le rôle de l'espace de recherche. Cela, parce qu'un opérateur doit travailler sur des clauses qui ne satisfont pas les biais pour, peut-être, en construire une satisfaisante. Par exemple, si l'opérateur est l'ajout d'un littéral et que l'on considère la condition *range-restricted*, il faudra plusieurs étapes pour trouver une clause satisfaisant ce biais. Pour le problème d'arche, on peut avoir la séquence suivante :

$$
\begin{array}{lcl}
arche(X,Y,Z) & \leftarrow & \\
arche(X,Y,Z) & \leftarrow & cube(X) \\
arche(X,Y,Z) & \leftarrow & cube(X), \\
 & & volume(Z) \\
arche(X,Y,Z) & \leftarrow & cube(X), \\
 & & volume(Z), \\
 & & est\_stable(Z,X) \\
arche(X,Y,Z) & \leftarrow & cube(X), \\
 & & volume(Z), \\
 & & est\_stable(Z,X), \\
 & & cube(Y)
\end{array}
$$

Les quatre premières clauses ne satisfont pas le biais *range-restricted* (la variable Y n'est pas utilisée dans le corps de ces clauses). Elles n'appartiennent donc pas à l'espace de recherche, mais elles permettent d'amener la dernière qui elle en fait partie.

Définition 2.2.1 (espace engendré)
Il s'agit des clauses qui sont effectivement construites par l'opérateur d'apprentissage, et testées. Les clauses de l'espace engendré qui ne sont pas dans l'espace de recherche, et leur nombre, dépendent de l'opérateur utilisé.

Ce comportement va à l'encontre de ce que l'on désirait en utilisant des biais

de langage, à savoir, se permettre d'ignorer certaines clauses (c'est-à-dire ne même pas les générer). Cela nous amène à demander des biais *privés*.

Notation 2.2.1
Pour un opérateur $\mathcal{O}$, on note $\mathcal{O}^*$ sa fermeture transitive.

Notation 2.2.2
Etant donné un biais B, on note $\mathcal{L}_B$ l'ensemble des clauses satisfaisant le biais B.

Définition 2.2.2 (biais privé)
Un biais B est privé pour un opérateur $\mathcal{O}$ si pour toutes hypothèses H et H' telles que $H' \in \mathcal{O}^*(H)$, on a $H \notin \mathcal{L}_B \Rightarrow H' \notin \mathcal{L}_B$.

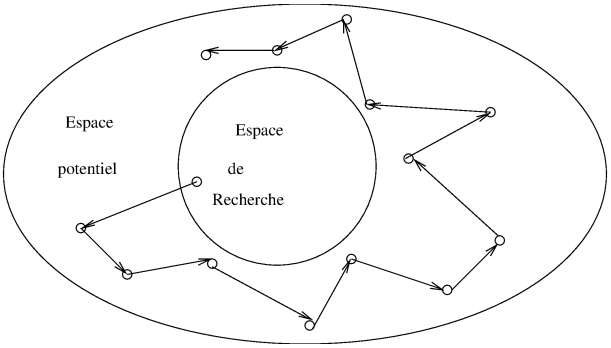

FIGURE 2.4 – Comportement d'un opérateur en présence d'un biais privé

Par exemple, si l'on limite la taille des hypothèses et que l'on possède une clause ne satisfaisant pas ce biais, l'ajout de littéraux ne peut amener que des hypothèses trop grandes. On dira donc que la limitation du nombre de littéraux est un biais privé pour l'ajout.

On définit pareillement la propriété duale.

Définition 2.2.3 (biais clos)
Un biais B est clos pour un opérateur $\mathcal{O}$ si pour toutes hypothèses H et H' telles que $H' \in \mathcal{O}^*(H)$, on a $H \in \mathcal{L}_B \Rightarrow H' \in \mathcal{L}_B$.

Supposons que nous disposions d'une hypothèse H et que nous la confrontions à un biais $\mathcal{B}$; voyons ce qu'il advient selon les propriétés de B par rapport à l'opérateur utilisé.

— B est à la fois clos et privé. Soit H satisfait ce biais et on a la garantie que toute application de l'opérateur sur H amènera de nouvelles hypothèses satisfaisantes. Soit H échoue et on est sûr qu'il n'y a pas d'intérêt à poursuivre l'application de l'opérateur sur cette hypothèse.

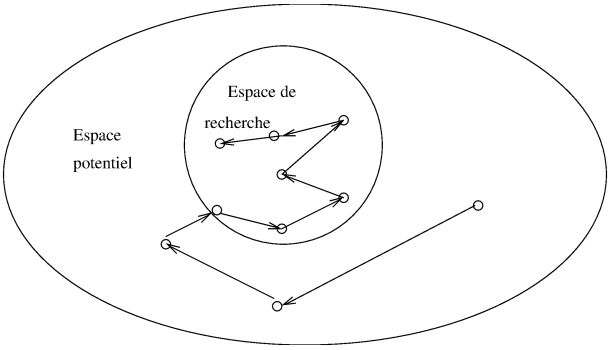

FIGURE 2.5 – Comportement d'un opérateur en présence d'un biais clos

— B est seulement clos. Si H vérifie le biais alors on est sûr, sans avoir à faire d'autres tests que ses descendantes le vérifieront aussi. Par contre, si H échoue, H ne peut pas devenir une définition partielle du concept cible, et pourtant il faut la conserver car l'application de l'opérateur sur H peut ramener de bonnes hypothèses.
— B est seulement privé. Si H échoue, on peut l'abandonner définitivement ; si elle passe le test, il faudra tout de même tester chacune de ses descendantes.
— B n'est ni clos ni privé. Que H soit retenue ou non, il faut poursuivre l'application de l'opérateur et confronter le biais à toutes les nouvelles hypothèses.

A priori, la vision du biais comme solution au problème de la taille de l'espace des hypothèses (section 1.1), voudrait nous faire privilégier les biais privés. Un biais non-privé est incapable d'élaguer quelle que soit la configuration et on prend alors le risque, non seulement de parcourir l'ensemble des hypothèses, mais aussi de tester pour rien ce biais sur chacune de ces hypothèses.

Par contre, il est plus difficile de décider si l'on veut ou non des biais clos. Ils peuvent être avantageux dans le sens où ils nous libérent de tests pesants dès qu'ils ont été validés. La contre-partie de l'absence de tests est l'acceptation de toutes les clauses que l'opérateur nous fournira.

Ces propriétés nous permettent d'ores et déjà de donner une nouvelle version de l'algorithme d'HAIKU (tables 2.4 et 2.5), qui prend en compte les biais de langage et ces propriétés. Globalement, nous allons intervenir à trois niveaux :

1. Sur les clauses de départ, on testera les biais à la fois clos et privés.
2. On vérifiera les clauses fournies par l'opérateur avant leur entrée dans la définition courante à l'aide des biais seulement privés.
3. Lorsque l'on essaye de faire passer une hypothèse de la définition courante vers la définition partielle, on utilisera le reste à savoir les biais seulement clos et ceux qui n'ont aucune propriété (soit, ceux qui ne sont pas privés).

Dans chaque cas, l'hypothèse qui échoue est définitivement abandonnée.

TABLE 2.4 – Nouvelle Boucle GT

1. Initialiser la définition du concept à $\emptyset$.
2. Tant que le critère d'arrêt n'est pas validé :
   (a) Calculer les hypothèses de départ et initialiser la définition courante ***avec celles qui valident les biais à la fois clos et privés***.
   (b) Explorer les clauses accessibles depuis les clauses de départ et récupérer une définition partielle (éventuellement plusieurs) (algorithme 2.5).
   (c) Ajouter la définition partielle à la définition du concept et tester le critère d'arrêt.
3. Renvoyer la définition du concept.

Voyons maintenant les propriétés des biais vus à la section 1.4 par rapport à deux opérateurs classiques : l'ajout et l'abandon de littéraux (tableaux 2.6 et 2.7).

On observe finalement plusieurs choses.

1. Il y a très peu de biais clos.
2. Il y a très peu de biais privés.
3. Aucun biais n'est à la fois clos et privé.
4. La superposition des deux tableaux précédents montre que pour un opérateur qui s'exprimerait en termes d'ajouts et d'abandons de littéraux, aucun des biais étudiés n'est clos ou privé. Or, tout opérateur peut être vu comme un remplacement de littéraux dans une clause.

Le dernier point est évidemment très général. Ainsi, dans le cas de l'absorption où les littéraux ajoutés sont très fortement liés à ceux enlevés par la théorie du domaine (section 1.2.1), il est possible de conserver quelques propriétés favorables (tableau 2.8).

Les algorithmes 2.4 et 2.5 montraient que seuls les biais privés avaient un pouvoir élagant ; il apparaît maintenant que pour un opérateur fixé, la plupart des biais ne sont pas privés et, par suite, ne sont pas utilisables pour l'élagage. Ce constat doit être relativisé par rapport aux résultats expérimentaux de la section précédente qui tendent à montrer que l'on n'a pas besoin d'utiliser plusieurs biais et qu'un seul peut être amplement suffisant.

TABLE 2.5 – Nouvelle Exploration GT

1. Tant que le critère d'arrêt de l'exploration n'est pas satisfait :
   (a) Choisir une hypothèse parmi celles de la définition courante.
   (b) Appliquer l'opérateur d'apprentissage sur cette hypothèse pour obtenir un ensemble de nouvelles hypothèses.
   (c) Parmi ces nouvelles clauses
      i. ***Enlever les clauses qui ne satisfont pas les biais seulement privés.***
      ii. Enlever les clauses qui ne sont pas cohérentes par rapport aux hypothèses déjà échouées et celles qui sont redondantes avec des hypothèses de la définition courante.
      iii. Enlever celles qui ne sont pas valides vis-à-vis des exemples.
      iv. Dans la définition courante, remplacer l'hypothèse choisie au point 1a par les clauses issues de l'étape 1(c)iii.
   (d) Tester le critère d'apprentissage sur les hypothèses de la définition courante ; celles qui réussissent sont placées dans la définition partielle ***seulement si elles vérifient les biais non-privés.***
   (e) Tester le critère d'arrêt de l'exploration sur la définition partielle.
2. Renvoyer la définition partielle.

Biais de langage	**Biais Clos**	**Biais Privé**
Nombre de littéraux	oui	non
Nombre de variables	oui	non
Profondeur	oui	non
Degré	non	non
Range-restricted	non	oui
Connexion	non	non
Fonctionnalité (1)	non	oui
Fonctionnalité (2)	non	oui
Fonctionnalité (3)	non	non
Fonctionnalité (4)	oui	non
Fonctionnalité (5)	non	non
Total	4	3

TABLE 2.6 – Propriétes des biais pour l'abandon

Biais de langage	**Biais Clos**	**Biais Privé**
Nombre de littéraux	non	oui
Nombre de variables	non	oui
Profondeur	non	oui
Degré	non	non
Range-restricted	oui	non
Connexion	non	non
Fonctionnalité (1)	oui	non
Fonctionnalité (2)	oui	non
Fonctionnalité (3)	non	non
Fonctionnalité (4)	non	oui
Fonctionnalité (5)	non	non
Total	3	4

TABLE 2.7 – Propriétes des biais pour l'ajout

Biais de langage	**Biais Clos**	**Biais Privé**
Nombre de littéraux	oui	non
Nombre de variables	*non*	non
Profondeur	non	non
Degré	*non*	non
Range-restricted	non	oui
Connexion	non	non
Fonctionnalité (1)	non	oui
Fonctionnalité (2)	non	oui
Fonctionnalité (3)	non	non
Fonctionnalité (4)	non	non
Fonctionnalité (5)	non	non
Total	1	3

TABLE 2.8 – Propriétes des biais pour l'absorption

Chapitre 3

Vers une meilleure exploitation des biais de langage

Au chapitre précédent, nous avons donné des propriétés sur les biais de langage ; compte-tenu de ces propriétés, nous avons pu intégrer ces biais assez astucieusement dans le système HAIKU. Astucieusement, parce que nous ne faisons jamais de tests inutilement : les biais non privés ne peuvent pas élaguer et sont donc testés seulement en fin d'apprentissage. Malheureusement, cela signifie aussi que des hypothèses qui ne satisfont pas ces biais sont générées, testées vis-à-vis des hypothèses déjà échouées et des exemples, etc.

On aimerait que le processus d'apprentissage ne travaille que sur des clauses satisfaisant les biais de langage. En termes d'espace, cela revient à demander que l'espace engendré soit inclus dans l'espace de recherche.

Pratiquement, on voudrait disposer de biais de langage et d'opérateurs tels que les biais soient à la fois clos et privés pour ces opérateurs. Pour réaliser cela, nous pouvons considérer deux approches :

1. soit on essaye de construire des opérateurs qui rendent tous les biais à la fois clos et privés ;
2. soit on cherche des biais qui sont clos et privés à la fois pour tous les opérateurs.

3.1 Vers de nouveaux opérateurs

3.1.1 Idée de départ

Le problème de trouver un opérateur qui rende tous les biais de langage à la fois clos et privés se ramène à découvrir un formalisme dans lequel nous pourrons exprimer tous les biais de langage : à la fois les littéraux autorisés et les autres.

De plus, on aimerait que ce formalisme fournisse naturellement une procédure permettant de générer les hypothèses qu'il dénote en respectant si possible un ordre de généralité. C'est cette procédure qui nous servirait alors d'opérateur d'apprentissage et, sans avoir aucun test à faire, nous aurions la garantie que les clauses produites sont des hypothèses vérifiant tous les biais. On retrouve bien que, pour cet opérateur, tout biais est à la fois clos et privé. Cela suppose tout de même de posséder une ou plusieurs clauses de départ pour pouvoir commencer à appliquer cet opérateur.

Enfin, ce formalisme doit être facile à utiliser : on doit pouvoir aisément spécifier les différents biais de langage.

Les grammaires de Cohen (1.3.4) amènent bien avec elles une procédure de génération mais elles ne sont pas capables de représenter tous les biais.

Si l'on se restreint, pour l'instant aux approches descendantes, on peut construire un tel opérateur assez simplement à partir de l'opérateur d'ajout.

On pourra effectuer cet ajout si et seulement s'il préserve la satisfaction des biais. Il ne s'agit bien sûr pas de tester la clause résultat auquel cas on aurait simplement reformulé le problème : cela ne serait plus visible de l'extérieur mais on engendrerait les mêmes clauses et, en conséquence, on ferait les mêmes tests coûteux.

L'idée est de représenter chaque biais par des contraintes qui portent exclusivement sur les littéraux ajoutés. Naturellement, certains biais peuvent avoir besoin d'informations sur la clause sur laquelle on applique l'opérateur. Aussi, notre opérateur gèrera une information pour chacun des biais.

Ces contraintes auront finalement la forme suivante :

$$\frac{[\{\mathrm{Plus_i}\} \cup \Sigma, \mathrm{I}_{\mathcal{B}\mathrm{iais}}, \Gamma] \ , \ \mathrm{Condition} \vdash \mathrm{Tete} \leftarrow \mathrm{Corps}}{[\Sigma', \mathrm{I}'_{\mathcal{B}\mathrm{iais}}, \Gamma'] \vdash \mathrm{Tete} \leftarrow \mathrm{Corps} \cup \{\mathrm{Plus_i}\}} \ \mathcal{B}\mathrm{iais}$$

On interprètera cette règle de la manière suivante :

- $\{Plus_i\}$ est l'ensemble des littéraux que l'on cherche à ajouter avec cette règle.
- Σ, littéraux qui sont candidats à l'ajout en plus des $\{Plus_i\}$.
- $I_{\mathcal{B}iais}$ est l'information du contexte qui est propre au biais $\mathcal{B}iais$: il peut y accéder et la modifier.
- Γ, c'est le reste du contexte. On y trouve : les informations de chaque biais et, éventuellement les conjonctions de littéraux que l'on ne désire pas voir apparaître.
- $Condition$, elle porte exclusivement sur $\{Plus_i\}$ et $I_{\mathcal{B}iais}$.
- $Tete \leftarrow Corps$, la clause sur laquelle nous sommes en train d'appliquer l'ajout ; nous savons déjà que cette clause satisfait le biais $\mathcal{B}iais$.

— Σ', il s'agit de Σ auquel on peut avoir enlevé certains littéraux (voir Γ').
— $I'_{\mathcal{B}iais}$ est la nouvelle valeur de $I_{\mathcal{B}iais}$.
— Γ', le reste du nouveau contexte. On ne modifie bien sûr pas les informations appartenant en propre aux autres biais. Par contre, nous devons mettre à jour les conjonctions interdites : on enlève de ces conjonctions les $Plus_i$ qui s'y trouvent ; si une conjonction ne contient plus qu'un seul littéral (!), celui-ci est supprimé de Σ pour donner Σ'.
— $Tete \leftarrow Corps \cup \{Plus_i\}$, c'est la clause dont on prouve par cette règle, qu'elle vérifie le biais $\mathcal{B}iais$.

Avec ce formalisme, on a la garantie de pouvoir représenter tous les biais car, au pire, on prend la clause elle-même comme information du contexte et l'on teste le biais sur la clause obtenue par ajout des nouveaux littéraux (et l'on a bien une condition qui ne porte que sur le contexte et les littéraux candidats). Evidemment, on cherchera à faire des conditions beaucoup plus astucieuses et cette méthode brutale sera réservée aux cas désespérés.

Si l'on considère que la *Condition* est un but PROLOG et que les différentes mises à jour peuvent être vues comme un autre but, alors l'application d'une règle revient à résoudre la requête

$$\leftarrow Condition, Action$$

Si l'on considère plusieurs biais de langage alors il faut pouvoir appliquer une règle pour chacun de ces biais, c'est-à-dire résoudre une requête par biais.

Pour ce qui est de la clause de départ, on a besoin d'une clause plus générale que les autres qui vérifie tous les biais : on utilise assez naturellement la clause vide en admettant qu'elle est satisfaisante pour tous les biais.

A noter que nos règles d'inférence ne sont pas orientées et que l'on peut donc les utiliser de haut en bas pour générer de nouvelles clauses, ou de bas en haut pour prouver qu'un biais valide une clause. Si l'on se place dans une implémentation en PROLOG, on retrouve cette double possibilité (génération ou vérification), simplement en jouant sur les parties de la règle que l'on instancie. Notre propos est ici de construire un opérateur et, par conséquent, nous ne nous intéressons qu'à l'aspect génération.

Voyons maintenant comment représenter les biais de langage dans ce formalisme.

3.1.2 Représentation des biais

Le format des règles d'inférence donné précédemment peut se décomposer en deux éléments :

1. L'information qui est stockée dans le contexte et son initialisation, c'est-à-dire sa valeur pour la clause vide.

2. Le test effectué pour déterminer si la nouvelle clause est satisfaisante pour le biais ou non. Cette condition ne doit porter que sur l'information du contexte et les littéraux candidats à l'ajout.

Ce sont ces éléments que nous allons spécifier pour chacun des biais de la section 1.4.

- Nombre de littéraux
  - Le nombre de littéraux déjà contenus dans le corps de la clause, initialisé à 0.
  - La valeur stockée ajoutée au nombre de littéraux ajoutés ne doit pas dépasser la taille limite des clauses.
- Nombre de variables existentielles
  - Les variables déjà apparues et le nombre de variables existentielles déjà dans le corps. On débute avec le couple (variables de la tête,0).
  - Le nombre de variables apparaîssant dans les nouveaux littéraux sans être déjà notées dans le contexte, plus le nombre de variables déjà recensées ne doit pas excéder le nombre maximum de variables existentielles.
- Profondeur
  - Rien.
  - Les nouveaux littéraux ne doivent pas contenir des termes de profondeur supérieure à la limite
- Degré : il s'agit du seul cas où nous sommes obligés de stocker quasiment toute la clause dans le contexte.
  - On représente chaque littéral par la liste des variables qu'il amène. Initialisé avec les variables de la tête et leur degré (0).
  - On vérifie que le degré n'est pas trop grand (à chaque fois, il faut recalculer la valeur du degré de la nouvelle clause en considérant tous ses littéraux).
- Range-restricted : On a deux règles. L'une qui fait passer de la clause vide à une clause *range-restricted* :
  - Les variables de la tête.
  - Ces variables doivent être incluses dans l'ensemble des variables des nouveaux littéraux.

  La seconde règle fait passer d'une clause *range-restricted* (autre que la clause vide) à une nouvelle clause *range-restricted* mais, elle n'utilise pas de condition et ne fait donc pas d'accès au contexte.
- Connexion
  - Les variables déjà connectées. Au départ, les variables de la tête sont connectées.
  - On vérifie pour chaque nouveau littéral que son ensemble de variables a au moins un élément en commun avec l'ensemble des variables connec-

tées.
- Utilisation des *inputs* : comme *range-restricted*, sauf que l'on ne considère que les variables ayant le statut *input*.
- Apparition des outputs de la tête : comme *range-restricted*, sauf que l'on ne considère que les variables ayant la statut *output*.
- Utilisation des *outputs* du corps
  - Les *outputs* de la tête.
  - Toutes les variables *outputs* des littéraux à ajouter doivent, soit apparaître en *inputs* dans ces mêmes littéraux, soit faire partie des *outputs* de la tête.
- Surcharge des *outputs*
  - Les variables qui ont déjà pris une valeur (c'est-à-dire qui sont apparues en *output*). Initialisé avec la liste vide.
  - Les *outputs* des nouveaux littéraux, ne doivent pas se trouver dans le contexte.
- Instanciation des *inputs*
  - L'ensemble des variables instanciées initialisé avec les *inputs* de la tête.
  - Les variables *inputs* des littéraux à ajouter doivent appartenir au contexte ; pour chaque littéral ajouté, on ajoute ses *outputs* dans le contexte.

Nous pouvons remarquer que les biais de langage dont on avait établi qu'ils étaient clos pour l'opérateur d'ajout habituel (*range-restricted*, utilisation des *inputs* et apparition des *outputs* de la tête), adopte dans notre formalisme une forme bien particulière : deux règles dont l'une a une condition toujours vérifiée. Par contre, parmi les autres, nous sommes incapables de distinguer les biais qui étaient privés. Cette observation nous servira pour utiliser au mieux la représentation.

3.1.3 Utilisation de la représentation

Nous avons donc vu que les biais de langage pouvaient s'exprimer de manière très concise dans notre représentation. Il faut maintenant préciser les modalités d'utilisation de ces règles.

Comme nous l'avons déjà évoqué, on part donc de la clause vide et on ajoute de nouveaux littéraux. Il reste à définir la stratégie d'ajout et la provenance des nouveaux littéraux.

L'ensemble des littéraux candidats peut être obtenu par développement de l'une des représentations décrites à la section 1.3. Une solution consiste, par exemple, à éclater un *ensemble de prédicats* (section 1.3.1) pour obtenir tous les littéraux possibles.

Il est clair que toute clause peut être construite en ajoutant à la clause vide les littéraux qui la composent. Nous préférerons une stratégie qui amène les clauses les plus simples d'abord. Cela pour minimiser les tests (puisqu'ils ne portent que

sur les littéraux ajoutés, autant en ajouter le moins possible).

Intuitivement, lorsque l'on a obtenu l'ensemble des littéraux candidats, on essaye d'ajouter un de ces littéraux. Si l'ajout est accepté, on a une nouvelle clause et le littéral ajouté n'est plus candidat. Sinon, on essaye à nouveau d'ajouter ce littéral mais accompagné d'un autre littéral, etc. On obtient donc, par étape, des clauses de plus en plus spécifiques (pour la θ-subsomption).

De plus, on a pu remarquer que les biais clos pour l'ajout classique avait une représentation particulière. Par contre, les biais privés ne sont pas différenciés des autres. Cette propriété doit être prise en compte au niveau de la stratégie d'ajout : si un biais privé échoue sur des littéraux, on n'a pas besoin d'essayer les ensembles qui contiennent ces littéraux.

Nous donnons un algorithme plus précis de ce nouvel opérateur ("NouvelAjout") à la table 3.1.

TABLE 3.1 – Nouvel ajout

`NouvelAjout(ClauseDepart,Candidats,NouvelleClause)`

1. `Reste = Candidats ; Ajoute =` $\emptyset$
2. `Répéter`
   (a) `Choisir` L_c `dans Reste =` $\{L_1, \ldots, L_c, \ldots, L_n\}$ (*point de backtrack*)
   (b) `Reste =` $\{L_{c+1}, \ldots, L_n\}$
   (c) `Ajoute = Ajoute` $\cup$ $\{L_c\}$
   (d) `valider l'ajout : trouver une règle qui valide l'ajout de Ajoute à ClauseDepart pour chaque biais`
   (e) `Si la validation échoue à cause d'un biais privé alors backtracker`
3. `Jusqu'à ce que la validation réussisse`
4. `NouvelleClause = ClauseDepart` $\cup$ `Ajoute`
5. `Candidats = Reste`

On peut indifféremment se servir de cet opérateur dans les deux types d'approches que nous avons étudiés. Les méthodes exhaustives réappliquent NouvelAjout sur ses propres résultats autant de fois que possible. Le nouvel opérateur peut aussi s'intégrer dans un algorithme transformationnel à ceci près qu'il n'y a plus aucun test par rapport aux biais et qu'à chaque hypothèse, on associe l'en-

semble des littéraux que l'on peut encore lui ajouter (pour pouvoir réappliquer notre opérateur).

A noter encore une fois que l'implémentation en PROLOG autorise deux utilisations : si l'on appelle NouvelAjout en instanciant NouvelleClause et non plus ClauseDepart, on va travailler en vérificateur.

La version exhaustive a déjà été testée sur le problème des arches (section 2.1.1) : en utilisant tous les biais, les clauses candidates sont générées en 30 secondes CPU (contre plusieurs centaines d'heures avec un opérateur classique).

3.1.4 Avenir de cette approche

Le défaut de cet opérateur est qu'il repose complètement sur l'opérateur habituel d'ajout et qu'il nous cantonne donc dans les approches descendantes. L'idée est alors d'utiliser des règles d'inférence plus générales : on ne se contente plus d'ajouter des littéraux, on peut aussi en enlever.

$$\frac{[(\{\text{Plus}_\text{i}\}, \{\text{Moins}_\text{j}\}) \cup \Sigma, \text{I}_{\mathcal{B}\text{iais}}, \Gamma] \ , \ \text{Condition} \vdash \text{Tete} \leftarrow \text{Corps} \cup \{\text{Moins}_\text{j}\}}{[\Sigma', \text{I}'_{\mathcal{B}\text{iais}}, \Gamma'] \vdash \text{Tete} \leftarrow \text{Corps} \cup \{\text{Plus}_\text{i}\}}$$

On espère pouvoir améliorer n'importe quel opérateur et cela, en fixant de manière adéquat la fac'on d'obtenir les littéraux ajoutés et ceux remplacés.

La difficulté majeure consistera, dans les méthodes ascendantes, à trouver une hypothèse maximalement spécifique qui vérifie tous les biais.

3.2 Vers de Nouveaux Biais

Il s'agit maintenant d'étudier une approche opposée : trouver de nouveaux biais, significatifs, clos et privés pour au moins certains des opérateurs habituels.

La mise sous forme de règles décrite à la section 3.1.2 met en évidence une similarité entre les biais, si bien qu'il faut s'interroger sur ce que pourrait être ce nouveau biais.

3.2.1 Limitations numériques

On considère ici les biais qui font appel à une fonction comme décrit à la section 1.4.1.

— Si la fonction est croissante par rapport à la taille de la clause et le biais est une borne supérieure à sa valeur, alors ce biais est non-clos et privé pour l'ajout, clos et non-privé pour l'abandon.

— Si la fonction est croissante par rapport à la taille de la clause et le biais est une borne inférieure à sa valeur, alors ce biais est clos et non-privé pour l'ajout, non-clos et privé pour l'abandon.
— On retrouve les cas symétriques pour une fonction décroissante.
— Si elle est non monotone (c'est le cas du degré), a priori le biais n'est ni clos ni privé pour l'ajout ou l'abandon de littéraux.

3.2.2 Conditions sur les variables

Ces conditions s'expriment toujours en termes d'intersection et d'inclusion d'ensembles de variables que l'on a récupérées dans les littéraux de la clause. De la même manière que pour les limitations numériques, on peut montrer que pour ces tests les propriétés *clos* et *privé* sont incompatibles.

3.2.3 Espoir de cette approche

Nous venons de montrer que si un biais de langage clos et privé par rapport à plusieurs opérateurs existe, il opèrera de manière radicalement différente des biais que nous avons étudiés. Aussi, on peut émettre des doutes sur la sémantique d'un tel biais.

Bilan

Le problème de la taille de l'espace des hypothèses est connu depuis longtemps. Du fait de sa haute expressivité, ILP retrouve ce problème à une échelle plus importante et il apparaît que ce problème doit impérativement être résolu de manière efficace si l'on veut travailler sur des problèmes raisonnables. Dans ce cadre, l'utilisation de biais de langage est reconnue comme une solution satisfaisante.

Cependant, nul n'était capable de dire formellement comment il convenait d'utiliser ces biais, et encore moins de prévoir leur influence sur l'apprentissage en fonction de l'opérateur utilisé.

Nous avons donc commencé par étudier la force brute des biais. Cela nous a permis de vérifier l'intuition générale à savoir que les biais sont extrêmement puissants dans l'élagage de l'espace des hypothèses. A un tel point que l'association de plusieurs biais semble sans intérêt.

Ensuite, nous avons défini des propriétés des biais de langage par rapport aux opérateurs : nous avons été contraints de constater que les opérateurs n'étaient pas en mesure d'utiliser au mieux les biais de langage dans les approches transformationnelles. Il nous faut finalement admettre qu'une utilisation optimale des biais dans ces approches n'est pas envisageable.

Pour dépasser ce résultat négatif, nous avons imaginé des opérateurs dirigés par les biais de langage et, en particulier, nous avons construit un opérateur descendant à l'aide de règles d'inférence, ces règles traduisant les contraintes imposés pas les biais. Cet opérateur n'a pas pu encore être testé mais les résultats déjà obtenus semblent très prometteurs.

Dans l'avenir, il faudra tenter de généraliser cette méthode et de la comparer tout de même à celles que nous avons jugé inefficaces.

D'autre part, il sera intéressant de mesurer quantitativement l'élagage des biais dans les approches transformationnelles.

Bibliographie

[AB92] Hilde Adé and Maurice Bruynooghe. A comparative study of declarative and dynamically adjustable language bias in concept learning. In *Proceedings of the ML-92 workshop on Biases in Inductive Learning*, 1992.

[ADRB94] H. Ade, L. De Raedt, and M. Bruynooghe. Declarative bias for specific-to-general ilp systems. *Machine Learning*, 1994. Special Issue on Declarative Bias, submitted.

[BG93] F. Bergadano and D. Gunetti. An interactive system to learn functional logic programs. In *Proceedings of IJCAI-93*, pages 1044–1049. Morgan Kaufmann, 1993.

[BG95] F. Bergadano and D. Gunetti. Learning clauses by tracing derivations. In *ILP : from ML to Software Engineering*. MIT Press, 1995.

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

[Coh94] W.W. Cohen. Grammatically biased learning : Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68 :303–366, 1994.

[DRB92] L. De Raedt and M. Bruynooghe. An overview of the interactive concept-learner and theory revisor CLINT. In S. Muggleton, editor, *Inductive Logic Programming*, pages 163–192. Academic Press, 1992.

[MF90] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[NR94] Claire Nédellec and Céline Rouveirol. Specifications of the HAIKU system. Technical Report 928, Laboratoire de Recherche en Informatique, Université Paris Sud, August 1994.

[Plo70] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.

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

[RB93] Luc De Raedt and Maurice Bruynooghe. A theory of clausal discovery. In Ruzena Bajcsy, editor, *Proceedings of IJCAI93*, pages 1058–1063. Morgan Kaufmann, 1993.

[Rou92] Céline Rouveirol. Extensions of inversion resolution applied to theory completion. In Stephen Muggleton, editor, *Inductive Logic Programming*, chapter 3. Academic Press, London, 1992.

[SB86] Claude Sammut and Ranan B. Banerji. Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : An Artificial Intelligence Approach*, volume 2, chapter 7, pages 167–192. Morgan Kaufmann, 1986.

[ST94] Irene Stahl and Birgit Tausend. MILES - a modular inductive logic programming experimentation system. Technical Report 6020 (ESPRIT BRA ILP), 1994.

[Tau94] B. Tausend. Representing biases for inductive logic programming. In F. Bergadano and L. De Raedt, editors, *European Conference on Machine Learning 94*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 427–430. Springer-Verlag, April 1994.