
UPMC
PARIS UNIVERSITAS

CHAPITRE 2

DATALOG ET SA SÉMANTIQUE

2.1 Datalog

Un programme DATALOG est un ensemble fini de clauses de Horn sans fonctions. Les relations correspondant aux prédicats sont de deux types

- relations explicitement données dans la base de données relationnelle appelées BDE (base de données extentionnelle),
- relations implicitement définies par les clauses de Horn appelées BDI (base de données intentionnelle).

Les prédicats de la BDI permettent de décrire des requêtes récursives non exprimables en calcul relationnel comme par exemple la clôture transitive.

Illustrons ces notions sur un exemple simple (clôture transitive). Soit la relation arc définie par : arc(Paris,Lyon), arc(Paris,Bordeaux), arc(Lyon,Nice), arc(Bordeaux,Nice). Posons

$$P : \begin{cases} chem(X,Y) \longleftarrow arc(X,Y) \\ chem(X,Y) \longleftarrow chem(X,Z), chem(Z,Y) \end{cases}$$

Ici, BDE=$\{arc\}$ et BDI=$\{chem\}$.

On associe à l'ensemble BDE + BDI un unique programme DATALOG, encore noté P formé des clauses :

$$\begin{aligned}
r_1 : \quad & chem(X,Y) \longleftarrow arc(X,Y) \\
r_2 : \quad & chem(X,Y) \longleftarrow chem(X,Z), chem(Z,Y) \\
r_3 : \quad & arc(\text{Paris,Lyon}) \longleftarrow \\
r_4 : \quad & arc(\text{Paris,Bordeaux}) \longleftarrow \\
r_5 : \quad & arc(\text{Lyon,Nice}) \longleftarrow \\
r_6 : \quad & arc(\text{Bordeaux,Nice}) \longleftarrow
\end{aligned}$$

Les termes de l'univers de Herbrand associé sont : $U_H = \{\text{Paris, Bordeaux, Lyon, Nice}\}$.

Les atomes de la base de Herbrand associée sont : $B_H = \{arc(X,Y) | X,Y \in U_H\} \cup \{chem(X,Y) | X,Y \in U_H\}$.

Des interprétations de Herbrand de P sont par exemple :

- $I_0 = \emptyset$, i.e. pour tous $X, Y \in U_H$, $arc(X,Y)$ et $chem(X,Y)$ sont faux ; cette interprétation n'est pas un modèle de P car les règles r_3 à r_6 ne sont pas satisfaites dans I_0.
- $I' = B_H$, i.e. pour tous $X, Y \in U_H$, $arc(X,Y)$ et $chem(X,Y)$ sont vrais ; cette interprétation est un modèle de P car toutes les règles r_1 à r_6 sont trivialement satisfaites dans I', puisque la tête de la règle est vraie. Toutefois, cette interprétation n'est pas très intéressante en tant que modèle.
- $I_1 = \{arc(\text{Paris,Lyon}), arc(\text{Paris,Bordeaux}), arc(\text{Lyon,Nice}), arc(\text{Bordeaux,Nice})\}$, i.e. $arc(\text{Paris,Lyon})$, $arc(\text{Paris,Bordeaux})$, $arc(\text{Lyon,Nice})$, $arc(\text{Bordeaux,Nice})$ sont vrais et tous les autres atomes sont faux ; cette interprétation n'est pas un modèle de P car la règle r_1 n'est pas satisfaite dans I_1 (toutes les autres règles sont satisfaites).
- $I_2 = \{arc(\text{Paris,Lyon}), arc(\text{Paris,Bordeaux}), arc(\text{Lyon,Nice}), arc(\text{Bordeaux,Nice}), chem(\text{Paris,Lyon}), chem(\text{Paris,Bordeaux}), chem(\text{Lyon,Nice}), chem(\text{Bordeaux,Nice})\}$, i.e. $arc(\text{Paris,Lyon})$, $arc(\text{Paris, Bordeaux})$, $arc(\text{Lyon,Nice})$, $arc(\text{Bordeaux,Nice})$, $chem(\text{Paris,Lyon})$, $chem(\text{Paris,Bordeaux})$, $chem(\text{Lyon,Nice})$, $chem(\text{Bordeaux,Nice})$ sont vrais et tous les autres atomes sont faux ; cette interprétation n'est pas un modèle de P car la règle r_2 n'est pas satisfaite dans I_2 (toutes les autres règles sont satisfaites).
- $I_3 = \{arc(\text{Paris,Lyon}), arc(\text{Paris,Bordeaux}), arc(\text{Lyon,Nice}), arc(\text{Bordeaux,Nice}), chem(\text{Paris,Lyon}), chem(\text{Paris,Bordeaux}), chem(\text{Lyon,Nice}), chem(\text{Bordeaux,Nice}), chem(\text{Paris,Nice})\}$; cette interprétation est un modèle de P car toutes les règles r_1 à r_6 sont satisfaites dans I_3, de plus c'est LE modèle intéressant de P puisqu'il consiste, intuitivement, de tous les atomes qui sont conséquence de P, et uniquement de ces atomes, i.e. I_3 est le plus petit modèle de Herbrand de P et on peut considérer que I_3 est engendré par P.

2.2 Ensembles complets. Treillis

2.2.1 Ensembles complets et fonctions continues

Définition 2.1 *Un ensemble ordonné $(E, \leq)$ est dit complet (ou plus précisément sup-complet), si toute partie de E admet une borne supérieure.*

Définition 2.2 *Un ensemble ordonné $(E, \leq)$ est un treillis (resp. treillis complet), si toute partie finie de E (resp. toute partie de E) admet une borne supérieure.*

Exemple 2.3 $\mathcal{P}(E)$ ordonné par inclusion est un treillis complet, et un ensemble complet.

Si un ensemble ordonné est complet, la partie vide a aussi une borne supérieure $\sup(\emptyset)$ que l'on notera $\bot$, prononcé "bottom". Puisque l'ensemble $\mathrm{Maj}(\emptyset)$ des majorants de la partie vide est E tout entier, $\bot$ est le minimum de E. Le fait qu'un ensemble soit complet implique donc qu'il contienne un élément minimum.

De façon analogue, on notera habituellement $\top$, prononcé "top", l'élément maximum d'un ensemble ordonné, s'il existe.

Définition 2.4 *Une application f d'un ensemble ordonné $(E_1, \leq_1)$ dans un ensemble ordonné $(E_2, \leq_2)$ est dite continue, (ou plus précisément sup-continue), si elle préserve les bornes supérieures des suites croissantes non vides : si la partie $E' \neq \emptyset$ a une borne supérieure $e = \sup(E')$, alors $f(E') = \{f(x) \,/\, x \in E'\}$ a aussi une borne supérieure qui est égale à $f(e)$.*

Remarque 2.5 Comme la borne supérieure de la partie vide est $\bot$, la condition que f préserve la borne supérieure de la partie vide s'écrit tout simplement $f(\bot_1) = \bot_2$. C'est une condition très forte que nous n'exigerons pas d'une fonction continue.

Puisque dans un ensemble complet les bornes supérieures existent toujours, la continuité d'une application s'exprime alors simplement par :

$$f(\sup(E)) = \sup(f(E))\,.$$

Exercice 2.1 Montrer que toute fonction continue est monotone. ♢

2.2.2 Points fixes de fonctions monotones

Soit f une application d'un ensemble E dans lui-même. Un *point fixe* de f est un élément x de E tel que $f(x) = x$.

Si E est un ensemble ordonné, l'ensemble des points fixes de f est un sous-ensemble ordonné de E, éventuellement vide. Si ce sous-ensemble admet un élément minimum, on l'appellera le *plus petit point fixe* de f, et s'il admet un élément maximum, on l'appellera le *plus grand point fixe* de f.

Théorème 2.6 *Si f est une application monotone d'un treillis complet dans lui-même, alors f a un plus petit point fixe.*

Démonstration. Vérifions que f a un plus petit point fixe. Soit

$$X = \{x \in E \,/\, x \geq f(x)\}$$

et soit $z = \inf(X)$. Par définition de z, on a $\forall x \in X,\ x \geq z$ d'où, puisque f est monotone, $f(x) \geq f(z)$. Et comme $x \geq f(x)$, $f(z)$ est un minorant de X, d'où $z \geq f(z)$. On en déduit que $f(z) \geq f(f(z))$, d'où $f(z) \in X$ et donc $f(z) \geq z$. Il en résulte que z est un point fixe de f. Si z' est un autre point fixe, $z' \in X$ et donc $z' \geq z$. □

Théorème 2.7 *Si f est une application continue d'un ensemble ordonné complet dans lui-même, alors f a un plus petit point fixe. Ce plus petit point fixe est égal à*

$$\sup(\{f^n(\bot) \,/\, n \in \mathbb{N}\}).$$

Démonstration. Soit $x = \sup(\{f^n(\bot) \,/\, n \in \mathbb{N}\})$. Comme f est continue,

$$f(x) = \sup(\{f^{n+1}(\bot) \,/\, n \in \mathbb{N}\})$$

et puisque $\bot = f^0(\bot)$ est le minimum de E,

$$\sup(\{f^{n+1}(\bot) \,/\, n \in \mathbb{N}\}) = \sup(\{f^n(\bot) \,/\, n \in \mathbb{N}\}) = x$$

qui est donc un point fixe de f. Si y est un autre point fixe de f, on montre d'abord par induction que $\forall n \in \mathbb{N},\ f^n(\bot) \leq y$; puisque $\bot$ est le minimum de E, $\bot = f^0(\bot) \leq y$; si $f^n(\bot) \leq y$ alors $f^{n+1}(\bot) \leq f(y) = y$. D'où il découle que $x = \sup(\{f^n(\bot) \,/\, n \in \mathbb{N}\}) \leq y$. □

Théorème 2.8 *Si E est un ensemble ordonné fini admettant un élément minimum $\bot$, pour toute fonction monotone f de E dans lui-même il existe $k \leq card(E)$ tel que le plus petit point fixe de f est $f^k(\bot)$.*

Démonstration. Considérons la suite

$$\bot\,,\ f(\bot)\,,\ f^2(\bot)\,,\ \ldots\,,\ f^n(\bot)\,,\ \ldots$$

qui est croissante, puisque f est monotone. Si elle a deux éléments consécutifs égaux elle est stationnaire : $f^i(\bot) = f^{i+1}(\bot) \implies f^{i+1}(\bot) = f^{i+2}(\bot)$ et donc, par induction, $\forall j \geq i,\ f^i(\bot) = f^j(\bot)$. Dans les $|E| + 1$ premiers éléments de cette suite, il y en a forcément deux consécutifs qui sont égaux. On a donc $f^k(\bot) = f^{k+1}(\bot)$ pour $k \leq |E|$. Le fait que $f^k(\bot)$ est plus petit que tout autre point fixe de f se démontre comme dans le cas du théorème précédent. □

Exercice 2.2 1) Que vaut $f(\bot)$ si f préserve *toutes* les bornes supérieures ?
2) Si $f(\bot) = \bot$, quel est le plus petit point fixe de f ? ♢

2.2.3 Application à la sémantique déclarative de DATALOG

Rappels

Soit $C : \quad A_1, \ldots, A_n \longleftarrow B_1, \ldots, B_p$ et v une valuation, alors $v \models C$, ou, en identifiant 0 à faux et 1 à vrai, $\bar{v}(C) = 1$ si et seulement si

- soit $\exists i \in \{1, \ldots, n\} \quad \bar{v}(A_i) = 1$,
- soit $\exists j \in \{1, \ldots, p\} \quad \bar{v}(B_j) = 0$.

Donc, en particulier,

- si $C : \quad A \longleftarrow$, alors $\bar{v}(C) = 1$ si et seulement si $\bar{v}(A) = 1$, et on parle de clause positive,
- si $C : \quad \longleftarrow B$, alors $\bar{v}(C) = 1$ si et seulement si $\bar{v}(B) = 0$, et on parle de clause négative,
- si $C : \quad \longleftarrow$, est la clause vide, alors $\bar{v}(C) = 0$ pour toute valuation v, c'est-à-dire que la clause vide est insatisfaisable.

Une clause de Horn est de l'une des trois formes

- $C : \quad B \longleftarrow B_1, \ldots, B_p$
- $C' : \quad B \longleftarrow$
- $C'' : \quad \longleftarrow B_1, \ldots, B_p$

Les clauses de type C ou C' sont appelées des clauses définies, et les clauses de type C'' sont appelées des clauses négatives.

Soient r une clause de Horn, P un ensemble de clauses de Horn. M est un modèle de r (resp. P) si et seulement si pour toute valuation ν $M, \nu \models r$ (resp. $\forall r \in P$, $M, \nu \models r$) ; notation : $M \models r$ (resp. $M \models P$). Un programme est un ensemble de clauses de Horn définies.

Une interprétation de Herbrand I_H est une interprétation ayant pour domaine l'univers de Herbrand formé des termes $\mathcal{T}$, et qui est entièrement définie par la donnée d'un sous ensemble I_H de la base de Herbrand B_P : pour tout atome A, $I_H \models A$ si et seulement si $A \in I_H$. Pour une interprétation de Herbrand, une valuation se réduit à une substitution $s: X \longrightarrow T$ qui remplace chaque variable par un terme de T.

Un modèle de Herbrand est une interprétation de Herbrand qui est un modèle de P. Si P est formé de clauses de Horn, P a un modèle si et seulement si P a un modèle de Herbrand, il nous suffit donc de considérer les modèles de Herbrand.

Remarque 2.9 Cela est faux si P contient des clauses non Horn : par exemple $P = \{p(a) \longleftarrow \quad , \quad \exists x \neg p(x)\}$ admet un modèle mais pas de modèles de Herbrand. $M = \{0, 1\}$ avec $a_M = 0$ et $p_M(0) = \mathit{tt}$, $p_M(1) = \mathit{ff}$ est un modèle ; il n'y a pas de modèles de Herbrand car les seules interprétations de Herbrand sont $I_0 = \emptyset$ et $I_1 = \{p(a)\}$ et aucune des deux n'est un modèle de P.

Définition 2.10 *Si P est formé de clauses de Horn, la sémantique dénotationnelle ou déclarative de P est par définition l'ensemble des atomes clos de la base de Herbrand qui sont des conséquences logiques de P, i.e. $M(P) = \{A \in B_P | P \models A\}$.*

Nous verrons que cet ensemble $M(P) = \{A \in B_P | P \models A\}$ forme le plus petit modèle de Herbrand de P.

Définition 2.11 *Opérateur de conséquence immédiate T_P. Soit $\mathcal{P}(B_P)$ l'ensemble des parties de B_P. L'opérateur de conséquence immédiate $T_P : \mathcal{P}(B_P) \longrightarrow \mathcal{P}(B_P)$ est défini par : $T_P(I) = \{A \in B_P | \exists r = (B \longleftarrow B_1, \ldots, B_n) \in P, \exists s$ une substitution telle que $\forall i = 1, \ldots, n$, $s^*(B_i) = A_i \in I$, $s^*(B) = A\}$, où $s^*(B) = B[x_1 := s(x_1)] \cdots [x_p := s(x_p)]$ (resp. $s^*(B_i) = B_i[x_1 := s(x_1)] \cdots [x_p := s(x_p)]$) désigne l'atome clos obtenu en substituant le terme $s(x_k)$ à x_k dans B (resp. B_i), pour toute variable $x_k \in X$.*

En d'autres termes, $T_P(I)$ est l'ensembles des atomes A tel que $A \longleftarrow A_1, \ldots, A_n$ est une instance close (sans variable, ou ground instance) d'une clause r de P et que de plus $A_1, \ldots, A_n$ sont dans I.

Proposition 2.12 *Soit I une interprétation de Herbrand ; alors I est un modèle de P si et seulement si $T_P(I) \subset I$.*

Démonstration. I est un modèle de P si et seulement si pour toute valuation (ici substitution s), et pour toute règle r de P, $I, s \models r$, soit encore, si et seulement si pour toute instance close $A \longleftarrow A_1, \ldots, A_n$ d'une clause r de P, $I \models [A \longleftarrow A_1, \ldots, A_n]$, c'est-à-dire si et seulement si pour toute instance close $A \longleftarrow A_1, \ldots, A_n$ d'une clause r de P, $I \models A_1$, $\ldots$, $I \models A_n$ implique que $I \models A$, ce qui équivaut exactement à $T_P(I) \subset I$. □

Lemme 2.13 *1) $\mathcal{P}(B_P)$ muni de l'inclusion est un treillis complet, son plus petit élément est $\emptyset$, son plus grand élément B_P, $\sup_i K_i = \cup_i K_i$, $\inf_i K_i = \cap_i K_i$.*

2) $T_P : \mathcal{P}(B_P) \longrightarrow \mathcal{P}(B_P)$ est une application monotone.

Corollaire 2.14 *Le plus petit point fixe de T_P est le plus petit modèle de Herbrand de P.*

Démonstration. Il suffit de combiner le théorème 2.6 et la proposition 2.12. □

Lemme 2.15 *$T_P : \mathcal{P}(B_P) \longrightarrow \mathcal{P}(B_P)$ est une application continue.*

Démonstration. Soit $\{K_i\}_{i \in N}$ une suite croissante de parties de $\mathcal{P}(B_P)$. Comme T_P est monotone, il est clair que : $\sup_i T_P(K_i) \subset T_P(\sup_i K_i)$. Pour l'inclusion inverse, i.e. : $\sup_i T_P(K_i) \supset T_P(\sup_i K_i)$, soit $A \in T_P(\sup_i K_i)$, alors $\exists (A \longleftarrow A_1, \ldots, A_n)$ une instance close d'une clause r de P, avec $A_1 \in \sup_i K_i, \ldots, A_n \in \sup_i K_i$, soit $A_1 \in K_{i_1}, \ldots, A_n \in K_{i_n}$; comme K_i est une suite croissante, si $l = \sup\{i_1, \ldots, i_n\}$, $K_{i_1} \subset K_l, \ldots, K_{i_n} \subset K_l$ et donc $A_1 \in K_l, \ldots, A_n \in K_l$; d'où $A \in T_P(K_l) \subset \sup_i T_P(K_i)$. □

Remarque 2.16 Si $\{K_i\}_{i \in N}$ est un sous ensemble quelconque de parties de $\mathcal{P}(B_P)$ ne formant pas une suite croissante, il est faux que : $\sup_i T_P(K_i) \supset T_P(\sup_i K_i)$.

Corollaire 2.17 *Le plus petit point fixe $\mu(T_P)$ de T_P est calculé par*

$$\mu(T_P) = \sup(\{T_P^n(\emptyset) \,/\, n \in \mathbb{N}\}).$$

Démonstration. Il suffit de combiner le lemme précédent et le théorème 2.7. □

REMARQUE 2.18 Si le langage comporte un nombre fini de symboles de constantes et pas de symboles de fonctions, alors $\mathcal{P}(B_P)$ est fini et le plus petit point fixe de T_P s'obtient en un nombre fini d'étapes. Si par contre l'ensemble des symboles de constantes est non vide et il y a au moins un symbole de fonction d'arité ≥ 1, alors $\mathcal{P}(B_P)$ est infini et le plus petit point fixe de T_P s'obtient en un nombre infini d'étapes.

EXEMPLE 2.19 Soit P défini par

$$r_1 : \quad chem(X,Y) \longleftarrow arc(X,Y)$$
$$r_2 : \quad chem(X,Y) \longleftarrow chem(X,Z), chem(Z,Y)$$

avec la BDE $arc(a,aa), arc(aa,aaa), arc(aa,aab)$. On lui associe le programme DATALOG

$$r_1 : \quad chem(X,Y) \longleftarrow arc(X,Y)$$
$$r_2 : \quad chem(X,Y) \longleftarrow chem(X,Z), chem(Z,Y)$$
$$r_3 : \quad arc(a,aa) \longleftarrow$$
$$r_4 : \quad arc(aa,aaa) \longleftarrow$$
$$r_5 : \quad arc(aa,aab) \longleftarrow$$

Alors le plus petit modèle $\mu(T_P)$ de P est donné par :

$$\mu(T_P) = \begin{array}{l}\{arc(a,aa), arc(aa,aaa), arc(aa,aab),\\ chem(a,aa), chem(aa,aaa), chem(aa,aab), chem(a,aab)\}\end{array}$$

EXEMPLE 2.20 1) Soit P défini par

$$r_1 : \quad q(a) \longleftarrow p(X)$$
$$r_2 : \quad p(f(X)) \longleftarrow p(X)$$
$$r_3 : \quad p(a) \longleftarrow$$

Alors $B_P = \{p(X), q(X) \mid X = f^n(a), \text{pour un } n \in \mathbb{N}\}$, B_P est le plus grand modèle de P, le plus petit modèle de P est $\mu(T_P) = \{q(a)\} \cup \{p(X) \mid X = f^n(a), \text{pour un } n \in \mathbb{N}\}$

2) Soit P défini par

$$r_1 : \quad q(a) \longleftarrow p(X)$$
$$r_2 : \quad p(f(X)) \longleftarrow p(X)$$
$$r_3 : \quad q(f(X)) \longleftarrow q(X)$$

Alors $B_P = \{p(X), q(X) \mid X = f^n(a), \text{pour un } n \in \mathbb{N}\}$, B_P est le plus grand modèle de P, le plus petit modèle de P est $\mu(T_P) = \emptyset$ qui coincide avec le plus petit point fixe de T_P et aussi avec le plus grand point fixe de T_P.

REMARQUE 2.21 Posons

$$T_P \uparrow \omega = \sup(\{T_P^n(\emptyset) \,/\, n \in \mathbb{N}\})$$

et de manière duale,

$$T_P \downarrow \omega = \inf(\{T_P^n(B_P) \,/\, n \in \mathbb{N}\})\,;$$

soient $\mu(T_P)$ le plus petit point fixe de T_P et $\nu(T_P)$ le plus grand point fixe de T_P. On a alors $\mu(T_P) = T_P \uparrow \omega$, et $\mu(T_P)$ le plus petit modèle de P. Par contre, on n'a pas $\nu(T_P) = T_P \downarrow \omega$ et $T_P \downarrow \omega$ n'est pas le plus grand modèle de P. Par exemple, on a vu dans l'exemple 2.20 2) un exemple P tel que $\nu(T_P) = \emptyset$, $T_P \downarrow \omega = \{q(X) \mid X = f^n(a), \text{pour un } n \in \mathbb{N}\}$, et le plus grand modèle de P est B_P.

Le plus grand modèle d'un programme P formé de clauses de Horn est toujours B_P. On peut remarquer de plus que les $T_P^n(B_P)$ forment une suite décroissante donc sont tous des modèles de P.

EXERCICE 2.3 Calculer $T_P \downarrow \omega$ pour le programme P de l'exemple 2.20 1). ♢

EXERCICE 2.4 Soit P le programme suivant

r_1 : $q(a) \longleftarrow$
r_2 : $p(X) \longleftarrow q(X)$
r_3 : $r(b) \longleftarrow$

Calculer $T_P \downarrow \omega$, le plus petit point fixe $\mu(T_P)$ de T_P et le plus grand point fixe $\nu(T_P)$ de T_P. ♢

EXERCICE 2.5 Soit le programme P suivant

r_1 : $q(a) \longleftarrow p(X)$
r_2 : $p(f(X)) \longleftarrow p(X)$
r_3 : $q(f(X)) \longleftarrow q(X)$
r_4 : $q(a) \longleftarrow q(X)$

Calculer $T_P \downarrow \omega$, le plus petit point fixe $\mu(T_P)$ de T_P et le plus grand point fixe $\nu(T_P)$ de T_P. ♢

2.2.4 Sémantique procédurale de DATALOG

Les sémantiques procédurales de DATALOG sont les sémantiques basées sur la notion de calculs ou d'opérations faite par le programme. On les appelle aussi sémantiques opérationnelles. On distingue deux grandes classes de techniques de calcul ou d'évaluation pour DATALOG :

- les méthodes par chaînage avant, ou bottom-up, qui appliquent les règles du programme à la manière de l'opérateur T_P : par exemple les méthodes naives et semi-naives ; ces méthodes sont faciles à implanter, mais peu efficaces en général (calculs inutiles et redondants).
- les méthodes par chaînage arrière, ou top-down, qui appliquent les règles du programme en sens inverse de leur écriture, pour essayer de déduire les sous buts à vérifier en fonction du but initial donné : par exemple les méthodes par SLD résolution, Alexandre ou magic sets ; ces méthodes sont plus difficiles à implanter, mais plus efficaces en général (calculs inutiles et redondants sont évités).

Méthode naive

Elle correspond à l'évaluation de T_P. Pour le programme P :

r_1 : $chem(X,Y) \longleftarrow arc(X,Y)$

r_2 : $chem(X,Y) \longleftarrow arc(X,Z), chem(Z,Y)$

la méthode naive peut se décrire par l'algorithme suivant :

$T = \emptyset$

`TANTQUE` T croit

`FAIRE` $T := arc \cup (arc \bowtie T)$

`FINTANTQUE`

$chem := T$

On voit que cette méthode fait beaucoup de calculs inutiles et redondants puisque elle recalcule à chaque itération la relation arc ainsi que tous les couples de la relation $chem$, y compris ceux déjà calculés (à l'itération n on recalcule tout $T_P^n(\emptyset)$).

Méthode semi-naive

Elle correspond à une légère optimisation de la méthode précédente et peut se décrire, pour le même programme P que ci-dessus, par :

$T = \emptyset$; $\Delta = arc$

`TANTQUE` $\Delta \neq \emptyset$

`FAIRE`

$T := \Delta \cup T$

$\Delta := arc \bowtie \Delta$

`FINFAIRE`

`FINTANTQUE`

$chem := T$

On voit que cette méthode fait moins de calculs inutiles et redondants puisque elle calcule à chaque itération uniquement les nouveaux couples de la relation $chem$ (à l'itération n on calcule $T_P^n(\emptyset) - T_P^{n-1}(\emptyset)$ qu'on met dans Δ).

Exercice 2.6 Décrire les algorithmes par les méthodes naive et semi-naive qui correspondent au programme de "same generation" donné ci dessous :

$$P : \begin{cases} sg(X,Y) \longleftarrow eq(X,Y) \\ sg(X,Y) \longleftarrow up(X,X_1), sg(X_1,Y_1) down(Y_1,Y) \end{cases}$$

$\diamondsuit$

Remarquons que la méthode semi-naive décrite ci-dessus n'est correcte que pour des programmes *linéaires* c'est-à-dire des programmes qui comportent dans chaque corps de

règle au plus un IDB. Dans le cas de programme non linéaires, la méthode semi-naive donne un résultat un peu plus complexe comme illustré ci dessous.

Considérons le progamme non linéaire suivant qui calcule la cloture transitive :

$$P : \begin{cases} chem(X,Y) \longleftarrow arc(X,Y) \\ chem(X,Y) \longleftarrow chem(X,Z), chem(Z,Y) \end{cases}$$

Un programme correct donné par la méthode naive est écrit ci-dessous : $T = \emptyset$

`TANTQUE` T croit

`FAIRE` $T := arc \cup (T \bowtie T)$

`FINTANTQUE`

$chem := T$ Le programme suivant, construit selon la méthode semi-naive correspondant au cas linéaire n'est pas correct :

$T = \emptyset$; $\Delta = arc$

`TANTQUE` $\Delta \neq \emptyset$

`FAIRE`

$T := \Delta \cup T$

$\Delta := \Delta \bowtie \Delta$

`FINFAIRE`

`FINTANTQUE`

$chem := T$

Un programme semi-naif correct est donné par :

$T = \Delta = arc$

`TANTQUE` $\Delta \neq \emptyset$

`FAIRE`

$temp = (T \bowtie \Delta) \cup (\Delta \bowtie T)$

$\Delta := temp \setminus T$

$T := \Delta \cup T$

`FINFAIRE`

`FINTANTQUE`

$chem := T$

L'idée sous-jacente à ce programme est que, à l'étape i, T^i contienne les anciens faits calculés jusqu'à l'étape i, Δ^i contienne les nouveaux faits calculés à l'étape i exactement, et $temp^i$ contienne le calcul intermédiaire destiné à trouver les nouveaux faits.

EXERCICE 2.7 Pouvez-vous améliorer le programme précédent ? ♢

On recherche donc une méthode réellement efficace, qui calcule rapidement exactement l'ensemble minimum de faits utiles pour déduire un but donné, et cela se fait par (SLD-)résolution. On étudiera dans la suite la résolution d'abord dans le cas des clauses sans variables, où elle se réduit à la méthode des coupures, puis dans le cas des clauses avec variables.

L'idée de la méthode de résolution est la suivante : rappelons que $P \models A$ si et seulement si $P \cup \{\neg A\}$ est insatisfaisable. On montre le

Théorème 2.22 *$P \cup \{\neg A\}$ est insatisfaisable si et seulement si on peut déduire par résolution la clause vide à partir de $P \cup \{\neg A\}$.*

Par conséquent, on définit la sémantique procédurale de P comme l'ensemble des atomes clos A de la base de Herbrand tels que l'on puisse déduire par résolution la clause vide à partir de $P \cup \{\neg A\}$.

2.2.5 Sémantique procédurale de DATALOG : résolution pour les clauses sans variable (cas du calcul propositionnel)

La résolution se réduit ici à la méthode des coupures, que nous illustrons sur deux exemples avant de la définir formellement.

EXEMPLE 2.23 1) Soit le programme P_0 (cf. exemple 2.19)

$$
\begin{array}{ll}
1 : & arc(a, aa) \longleftarrow \\
2 : & arc(aa, aab) \longleftarrow \\
3 : & chem(a, aa) \longleftarrow arc(a, aa) \\
4 : & chem(aa, aab) \longleftarrow arc(aa, aab) \\
5 : & chem(a, aab) \longleftarrow chem(a, aa), chem(aa, aab)
\end{array}
$$

Pour montrer que $P_0 \models chem(a, aab)$, on montre que l'on peut déduire la clause vide $\square$ de $P_0 \cup \{\neg chem(a, aab)\}$. On remarque que $\neg chem(a, aab)$ peut s'écrire

$$6 \qquad \longleftarrow chem(a, aab)$$

et on vérifie que

$$
\begin{array}{lll}
7 : & 1 + 3 \Longrightarrow & chem(a, aa) \longleftarrow \\
8 : & 2 + 4 \Longrightarrow & chem(aa, aab) \longleftarrow \\
9 : & 5 + 6 \Longrightarrow & \longleftarrow chem(a, aa), chem(aa, aab) \\
10 : & 7 + 9 \Longrightarrow & \longleftarrow chem(aa, aab) \\
10 : & 8 + 10 \Longrightarrow & \longleftarrow \square
\end{array}
$$

2) Soit le programme $P_1 = \{A \longleftarrow \quad , \quad C \longleftarrow A, B \quad , \quad B \longleftarrow A\}$.

Pour montrer que $P_1 \models C$, on montre que l'on peut déduire la clause vide $\square$ de $P_1 \cup \{\neg C\}$, ce qui se fait comme suit.

De $\longleftarrow C$ et $C \longleftarrow A, B$ on tire $\longleftarrow A, B$;

de $\longleftarrow A, B$ et $B \longleftarrow A$ on tire $\longleftarrow A$;

de $\longleftarrow A$ et $A \longleftarrow$ on tire la clause vide $\square$;

Enonçons maintenant formellement la règle de coupure.

Définition 2.24 *Soient* $C : \quad A, A_1, \ldots, A_n \longleftarrow B_1, \ldots, B_p$ *et* $C' : \quad A'_1, \ldots, A'_{n'} \longleftarrow A, B'_1, \ldots, B'_{p'}$ *deux clauses où le même atome* A *se trouve une fois à gauche de la flèche et une fois à droite, alors on peut en déduire la clause* $R : A_1, \ldots, A_n, A'_1, \ldots, A'_{n'} \longleftarrow B_1, \ldots, B_p, B'_1, \ldots, B'_{p'}$ *appelée résolvent de* C *et* C'*, en mettant ensemble d'une part les hypothèses de* C *et* C'*, d'autre part les conclusions de* C *et* C'*, et en supprimant l'atome commun* A *des deux côtés, i.e.*

$$\frac{A, A_1, \ldots, A_n \longleftarrow B_1, \ldots, B_p \qquad A'_1, \ldots, A'_{n'} \longleftarrow A, B'_1, \ldots, B'_{p'}}{A_1, \ldots, A_n, A'_1, \ldots, A'_{n'} \longleftarrow B_1, \ldots, B_p, B'_1, \ldots, B'_{p'}}$$

Notation : $R = res(C, C')$.

On peut démontrer que la règle de coupure est valide (ou consistente, i.e. on ne peut faire que des déductions correctes à l'aide de cette règle) et complète (i.e.toutes les déductions correctes peuvent être faites à l'aide de cette règle). Pour la validité, c'est une conséquence du lemme suivant.

Lemme 2.25 *Soient* $C : \quad A, A_1, \ldots, A_n \longleftarrow B_1, \ldots, B_p$ *et* $C' : \quad A'_1, \ldots, A'_{n'} \longleftarrow A, B'_1, \ldots, B'_{p'}$ *deux clauses,* $R : A_1, \ldots, A_n, A'_1, \ldots, A'_{n'} \longleftarrow B_1, \ldots, B_p, B'_1, \ldots, B'_{p'}$ *leur résolvent, et* v *une valuation.*

1) si $v(C) = v(C') = 1$ *alors* $v(R) = 1$,

2) si $v(R) = 0$ *alors* $v(C) = 0$ *ou* $v(C') = 0$.

Démonstration. Montrons le 2) et le 1) s'en déduit aussitôt. Supposons $v(R) = 0$, alors

- $\forall i \in \{1, \ldots, n\} \quad v(A_i) = 0$, et $\forall i \in \{1, \ldots, n'\} \quad v(A'_i) = 0$,
- $\forall j \in \{1, \ldots, p\} \quad v(B_j) = 1$, et $\forall j \in \{1, \ldots, p'\} \quad v(B'_j) = 1$.

On distingue les 2 cas possibles pour $v(A)$

- soit $v(A) = 1$, et alors $v(C') = 0$,
- soit $v(A) = 0$, et alors $v(C) = 0$.

D'où le 2) ; le 1) est la contraposée, donc en découle aussitôt. $\square$

Remarque 2.26 Les réciproques sont fausses, par exemple soit :

$$\frac{A \longleftarrow B \quad B \longleftarrow A}{A \longleftarrow A}$$

$P' : A \longleftarrow A$ est le résolvant de $P = \{A \longleftarrow B \quad , \quad B \longleftarrow A\}$, P' est valide mais pas P, c'est-à-dire que, pour toute valuation $v(A \longleftarrow A) = 1$, alors qu'il existe des valuations qui ne satisfont pas $v(A \longleftarrow B) = v(B \longleftarrow A) = 1$.

Corollaire 2.27 *Soit $P = \{C_1, C_2, C_3, \ldots, C_n\}$ un ensemble de clauses, $R = Res(C_1, C_2)$ le résolvent de C_1 et C_2 et soit $P' = \{R, C_1, C_2, C_3, \ldots, C_n\}$ l'ensemble de clauses déduit de P par une coupure portant sur C_1 et C_2. Alors*

1) P est valide si et seulement si P' est valide,

2) P' est insatisfaisable si et seulement si P est insatisfaisable.

Démonstration. Montrons le 2) ($\Longrightarrow$, car la partie $\Longleftarrow$ est claire). Si P' est insatisfaisable, alors pour toute valuation v,

(i) soit $\exists i \in \{3, \ldots, n\}$ tel que $v(C_i) = 0$,

(ii) soit $v(R) = 0$.

Dans le cas (i), v ne satisfait pas P trivialement, et dans le cas (ii) par le lemme précédent, soit $v(C_1) = 0$ soit $v(C_2) = 0$ et v ne satisfait pas P non plus.

Montrons le 1) ($\Longrightarrow$, car la partie $\Longleftarrow$ est claire). Si toute valuation v satisfait P, alors $\forall i = 1, \ldots, n$, $v(C_i) = 1$, de plus par le lemme précédent, $v(R) = 1$: donc $v(R) = v(C_3) = v(C_4) = \cdots = v(C_n) = 1$, et donc toute valuation v satisfait P' qui est aussi valide. □

On peut déduire du corollaire 2.27 la validité de la méthode de coupure pour le calcul propositionnel.

Définition 2.28 *Soit $P = \{C_1, C_2, C_3, \ldots, C_p\}$ un ensemble de clauses et soit C une clause, on dit que P se dérive en C par résolution si et seulement si il existe une suite de clauses $C'_0, C'_1, \ldots, C'_n = C$ telle que $\forall i = 0, \ldots, n$,*

- *soit $C'_i \in P$*
- *soit il existe j et k tels que $j < i$, $k < i$, et $C'_i = Res(C'_j, C'_k)$ se déduit de C'_j et C'_k par une seule coupure (on dira une étape de résolution).*

Notation : $P \vdash^* C$ ou bien si on veut préciser le nombre d'étapes de résolution, et si k est ce nombre –k est donc le nombre de C'_i qui sont des résolvants obtenus par coupure dans la suite $C'_0, C'_1, \ldots, C'_n$ – on écrit $P \vdash^k C$.

Remarquons qu'une clause intermédiaire (et en particulier une clause de P peut être utilisée plusieurs fois).

Théorème 2.29 *(Validité) Si on peut déduire la clause vide de P par résolution, i.e. si $P \vdash^* \square$, alors P est insatisfaisable.*

Démonstration. Par induction sur le nombre d'étapes de coupures, i.e. par induction sur k tel que $P \vdash^k \square$.

- Si $k = 1$, alors $P = \{\longleftarrow \quad , \quad \longleftarrow A\}$ et P est clairement insatisfaisable ;
- Si l'hypothèse d'induction est vraie pour $k - 1$ étapes de coupures, et si $P \vdash^k \square$, alors $P \vdash P_1 \vdash^{k-1} \square$: par l'hypothèse d'induction, P_1 est insatisfaisable, mais

comme P_1 s'obtient à partir de P par une seule coupure, par le corollaire 2.27 2) , P est aussi insatisfaisable. □

Le théorème de complétude est un peu plus complexe : nous en donnons ici une preuve simple mais non constructive.

Théorème 2.30 *(Complétude) Si P est insatisfaisable, alors on peut déduire la clause vide de P par résolution, i.e. $P \vdash^* \square$.*

C'est une conséquence immédiate du

Lemme 2.31 *Soit P insatisfaisable et $C : L_1 \vee \cdots \vee L_n$ une clause $C \in P$. Alors il existe $D : L_{i_1} \vee \cdots \vee L_{i_p} \subsetneq C$, avec $\{i_1, \ldots, i_p\} \subsetneq \{1, \ldots, n\}$, $p < n$, (i.e. D est une clause strictement incluse dans C) telle que l'on puisse dériver D par résolution à partir de P.*

Démonstration. Par induction sur le nombre n d'atomes dans P.

- Si $n = 1$, alors $P = \{A \longleftarrow \quad , \quad \longleftarrow A\}$ et clairement $P \vdash \square \subsetneq C$ pour C l'une des 2 clauses de P.
- Supposons le résultat vrai si P' comporte au plus $n-1$ atomes et soit P comportant n atomes. $C : L \vee L_2 \vee \cdots \vee L_k$ une clause $C \in P$ où l'on distingue un littéral L. On construit P' ayant un atome de moins que P (l'atome correspondant au littéral L) comme suit :
  - on supprime toutes les clauses de P contenant $\neg L$,
  - on enlève L dans toutes les clauses de P contenant L,
  - toutes les clauses de P ne contenant ni $\neg L$, ni L sont inchangées.

Alors

- la clause C' correspondant à $C : L \vee L_2 \vee \cdots \vee L_k$ dans P' est $C' : L_2 \vee \cdots \vee L_k$
- P' a au plus $n-1$ atomes,
- P' est insatisfaisable (soit il existe dans P une clause qui ne contient que L et alors P' contient la clause vide, soit sinon, si une valuation v' satisfaisait P', alors la valuation v égale à v' sur les atomes autres que L et telle que $v(L) = 0$ satisfairait P, une contradiction).

Par l'hypothèse d'induction, il existe donc une dérivation δ' par résolution $\delta' : P' \vdash^q D' \subsetneq C'$; à chaque étape de δ' on remet L dans chacune des clauses de la dérivation δ' provenant d'une clause de P où L figurait, on obtient ainsi une dérivation par résolution δ de P, $\delta : P \vdash^q D \subsetneq C$ avec soit $D = D'$ soit $D = D' \vee L$, i.e. dans les 2 cas $D \subsetneq C$. □

Le théorème 2.30 découle immédiatement du lemme 2.31 par induction sur la longueur de la clause C ; on choisit une clause C dans P, on en déduit par résolution une clause $D \subsetneq C$; on remarque que $P \cup \{D\}$ est aussi insatisfaisable, et on itère le processus avec $P \cup \{D\}$ et la clause D qui est strictement plus petite que C ; on finira donc par obtenir la clause vide.

2.2.6 Sémantique procédurale de DATALOG : résolution pour les clauses avec variables

La résolution se décompose en la méthode des coupures à laquelle on ajoute un algorithme d'unification.

EXEMPLE 2.32 1) Soit le programme P (cf. exemple 2.19)

$$1 : \quad arc(a, aa) \longleftarrow$$
$$2 : \quad arc(aa, aab) \longleftarrow$$
$$3 : \quad chem(x, y) \longleftarrow arc(x, y)$$
$$4 : \quad chem(x, y) \longleftarrow arc(x, z), chem(z, y)$$

On veut montrer $P \models chem(a, aab)$, on montre que l'on peut déduire la clause vide $\square$ de $P \cup \{\neg chem(a, aab)\}$. On remarque que $\neg chem(a, aab)$ peut s'écrire

$$5 : \quad \longleftarrow chem(a, aab)$$

et on vérifie que

$$6 : \quad 5 + 4 + \sigma = [x/a, y/aab] \Longrightarrow \quad \longleftarrow arc(a, z), chem(z, aab)$$
$$7 : \quad 6 + 1 + \sigma = [z/aa] \Longrightarrow \quad \longleftarrow chem(aa, aab)$$
$$8 : \quad 3 + 7 + \sigma = [x'/aa, y'/aab] \Longrightarrow \quad \longleftarrow arc(aa, aab)$$
$$9 : \quad 8 + 2 \Longrightarrow \quad \longleftarrow \square$$

Les substitutions σ de l'exemple ci-dessus sont des unifications.

Unification

Une *substitution* σ est une application d'un ensemble de variables $x_1, \ldots, x_n$ dans l'ensemble des termes ; on note $\sigma: \{x_1, \ldots, x_n\} \longrightarrow T$ ou bien $\sigma = [x_1/t_1, \ldots, x_n/t_n]$. Toute substitution admet un prolongement unique en un homomorphisme, encore noté $\sigma: T \longrightarrow T$ on note par $t\sigma$ le résultat de la substitution σ appliquée au terme t.

Deux termes t et t' sont des variantes ssi il existe 2 substitutions σ et θ telles que $t\sigma = t'$ et $t'\theta = t$: par exemple $t = h(f(x, y), g(z), a)$ et $t' = h(f(y, x), g(u), a)$.

On dit que σ est une substitution de renommage ssi tous les t_i sont des variables deux à deux distinctes. On montre que si t et t' sont des variantes, alors il existe des substitutions de renommage σ et θ telles que $t\sigma = t'$ et $t'\theta = t$.

Définition 2.33 *Un unificateur d'un ensemble S fini de termes est une substitution σ telle que $S\sigma$ soit réduit à un singleton.*

Un unificateur σ d'un ensemble S est appelé unificateur le plus général ou principal (m.g.u. ou most general unifier) si pour tout autre unificateur θ il existe une substitution γ telle que $\theta = \gamma\sigma$.

EXEMPLE 2.34 Par exemple, soit $S = \{f(h(x), z)\,,\ f(y, a)\}$; $\sigma_1 = [y/h(a)\,,\ x/a\,,\ z/a]$ et $\sigma_2 = [y/h(b)\,,\ x/b\,,\ z/a]$ sont des unificateurs de S.

$\eta = [y/h(x)\,,\ z/a]$ est un unificateur le plus général de S : $\sigma_1 = \eta\gamma_1$ avec $\gamma_1 = [x/a]$ et $\sigma_2 = \eta\gamma_2$ avec $\gamma_2 = [x/b]$.

Proposition 2.35 *L'unificateur le plus général est unique à un renommage près des variables.*

Démonstration. Soient σ_1, σ_2 deux unificateurs principaux ; soient $\bar{X}_1$ (resp. $\bar{X}_2$) l'ensemble des variables apparaissant dans les termes $\sigma_1(x)$ (resp. $\sigma_2(x)$). σ_1 étant principal, on a $\sigma_2 = \gamma\sigma_1$, et de même $\sigma_1 = \gamma'\sigma_2$. Posons $t_1 = \sigma_1(x)$; on a donc $\sigma_2(x) = \gamma(\sigma_1(x)) = \gamma(t_1) = t_1(\gamma(X_1))$ et $t_1 = \sigma_1(x) = \gamma'(\sigma_2(x)) = \gamma'(t_1(\gamma(X_1))) = t_1(\gamma'(\gamma(X_1)))$; d'où l'on déduit que pour toute variable z de X_1, $\gamma'(\gamma(z)) = z$, ce qui implique que $\gamma(z)$ soit une variable ; γ est donc un renommage sur les variables de X_1, et donc partout puisque γ n'est définie que sur les variables de X_1. De même on montre que γ' est donc un renommage sur les variables de X_2. □

Soient t et t' deux termes, et soit l'ensemble des couples correspondants $CORR(t, t')$ défini comme étant le plus petit ensemble de sous termes de t, t' tel que

- $(t, t') \in CORR(t, t')$
- si $(u, u') \in CORR(t, t')$ avec $u = f(t_1, \ldots, t_n)$ et $u' = f(t'_1, \ldots, t'_n)$ alors $(t_i, t'_i) \in CORR(t, t')$.

Un couple $(t_i, t'_i) \in CORR(t, t')$ est dit irréductible (resp. fortement irréductible) ssi les premiers symboles de t_i et t'_i sont distincts (resp. les premiers symboles de t_i et t'_i sont distincts et ni t_i ni t'_i ne sont réduits à une variable).

L'algorithme suivant calcule l'unificateur le plus général de t et t' s'il existe, et s'arrête avec échec sinon.

```
PROGRAMME unification
DÉBUT
LIRE t, t'
σ := ε ; i := 0 ;
TANT QUE tσ ≠ t'σ FAIRE
   SI CORR(tσ, t'σ) contient un couple fortement irréductible
   ALORS échec
   FINSI
   SI ∃(s, s') ∈ CORR(tσ, t'σ) tel que
      s ≠ s' et soit s soit s' est une variable ALORS i := i + 1 ;
      SI s est une variable ALORS x_i = s ; t_i = s' ;
      SINON {s' est une variable} x_i = s' ; t_i = s ;
      FINSI
   SINON échec
   FINSI
   SI x_i figure dans t_i ; ALORS échec
   SINON σ := σ[x_i/t_i]
   FINSI
FINTQ
AFFICHER σ qui est l'unificateur le plus général
FIN
```

EXEMPLE 2.36 Appliquons cet algorithme à l'exemple suivant :

$$S = \{p(a, x, h(g(z))), p(z, h(y), h(y))\} = \{t, t'\}.$$

- $\sigma := \varepsilon$; $i := 0$ et $CORR(t, t')) = \{< p(a, x, h(g(z))), p(z, h(y), h(y)) >, < a, z >, < x, h(y) >, < h(g(z)), h(y) >, < g(z), y >\}$.
- $i := 1$; $x_1 = z$; $t_1 = a$; $\sigma := [z/a]$ et $CORR(t\sigma, t'\sigma) = \{< p(a, x, h(g(a))), p(a, h(y), h(y)) >, < a, a >, < x, h(y) >, < h(g(a)), h(y) >, < g(a), y >\}$.
- $i := 2$; $x_2 = x$; $t_2 = h(y)$; $\sigma := [z/a, x/h(y)]$ et $CORR(t\sigma, t'\sigma) = \{< p(a, h(y), h(g(a))), p(a, h(y), h(y)) >, < a, a >, < h(y), h(y) >, < h(g(a)), h(y) >, < g(a), y >\}$.
- $i := 3$; $x_3 = y$; $t_3 = g(a)$; $\sigma := [z/a, x/h(y), y/g(a)]$ et $CORR(t\sigma, t'\sigma) = \{< p(a, h(g(a)), h(g(a))), p(a, h(g(a)), h(g(a))) >, < a, a >, < h(g(a)), h(g(a)) >, < h(g(a)), h(g(a)) >, < g(a), g(a) >\}$.
- $\sigma := [z/a, x/h(y), y/g(a)]$ est l'unificateur le plus général.

Résolution pour les clauses avec variables

Nous pouvons maintenant énoncer la règle de résolution générale pour les clauses avec variables

Définition 2.37 *Soient*

$$C : \quad A_1, \ldots, A_n \longleftarrow B_1, \ldots, B_p, E_1, \ldots, E_r$$

et

$$C' : \quad D_1, \ldots, D_m, A'_1, \ldots, A'_{n'} \longleftarrow B'_1, \ldots, B'_{p'}$$

deux clauses ; soit $S = \{E_1, \ldots, E_r, D_1, \ldots, D_m\}$ *tels que*

- *S est unifiable et* σ *est un m.g.u. de S avec* $S\sigma = A$,
- $\{E_1, \ldots, E_r\} \subset \{B_1, \ldots, B_p, E_1, \ldots, E_r\}$,
- $\{D_1, \ldots, D_m\} \subset \{A'_1, \ldots, A'_{n'}, D_1, \ldots, D_m\}$, *alors on peut en déduire la clause* R : $A_1\sigma, \ldots, A_n\sigma, A'_1\sigma, \ldots, A'_{n'}\sigma \longleftarrow B_1\sigma, \ldots, B_p\sigma, B'_1\sigma, \ldots, B'_{p'}\sigma$ *appelée résolvent de* C *et* C' *pour le m.g.u.* σ, *et notée par* $R = Res(C, C', \sigma)$, *en mettant ensemble d'une part les hypothèses de* $C\sigma$ *et* $C'\sigma$, *d'autre part les conclusions de* $C\sigma$ *et* $C'\sigma$, *et en supprimant l'atome commun* A *des deux côtés, i.e.*

$$\frac{A, A_1\sigma, \ldots, A_n\sigma \longleftarrow B_1\sigma, \ldots, B_p\sigma \qquad A'_1\sigma, \ldots, A'_{n'}\sigma \longleftarrow A, B'_1\sigma, \ldots, B'_{p'}\sigma}{A_1\sigma, \ldots, A_n\sigma, A'_1\sigma, \ldots, A'_{n'}\sigma \longleftarrow B_1\sigma, \ldots, B_p\sigma, B'_1\sigma, \ldots, B'_{p'}\sigma}$$

EXEMPLE 2.38 Soient par exemple

C : $\quad q(x, y) \longleftarrow p(x), p(f(y))$

C' : $\quad p(z), r(z) \longleftarrow m(z)$

Alors $\sigma = [x/f(y), z/f(y)]$ est un m.g.u. de $S = \{p(z)\,,\, p(x)\,,\, p(f(y))\}$ avec $S\sigma = p(f(y))$ et le résolvent de C et C' est

$$q(f(y), y)\,,\, r(f(y)) \longleftarrow m(f(y))$$

Z On ne peut pas couper plusieurs ensembles d'atomes à la fois. 1) Par exemple soient

C : $\quad q(x, y) \longleftarrow p(x)\,,\, p(f(y))$

C' : $\quad p(z)\,,\, r(y) \longleftarrow q(z, y)$

Une résolution correcte consiste à poser $\sigma = [x/f(y), z/f(y)]$ et à en déduire

$$q(f(y), y)\,,\, r(y) \longleftarrow q(f(y), y)$$

On peut aussi poser $\sigma = [x/z]$ et en déduire

$$p(z)\,,\, r(y) \longleftarrow p(z)\,,\, p(f(z))$$

Par contre la résolution consistant à poser $\sigma = [x/f(y), z/f(y)]$ et à en déduire, en coupant à la fois $p(f(y))$ et $q(f(y), y)$

$$r(y) \longleftarrow$$

est fausse.

2) Un autre exemple est :

C : $\quad r(x) \longleftarrow p(x)\,,\, q(y)$

C' : $\quad p(x)\,,\, q(y) \longleftarrow$

en coupant à la fois $p(x)$ et $q(y)$ on obtient la déduction erronnée $r(x) \longleftarrow$.

Définition 2.39 *Soit $P = \{C_1, C_2, C_3, \ldots, C_p\}$ un ensemble de clauses et soit C une clause, on dit que P se dérive en C par résolution si et seulement si il existe une suite de clauses $C'_0, C'_1, \ldots, C'_n = C$ telle que $\forall i = 0, \ldots, n-1$,*

- *soit $C'_i \in P$*
- *soit il existe j et k tels que $j < i$, $k < i$, et $C'_i = Res(C'_j, C'_k, \sigma)$ se déduit de C'_j et C'_k par une étape de résolution.*

Notation : $P \vdash^n C$, si on veut préciser le nombre d'étapes de résolution, ou bien $P \vdash^* C$. Remarquons qu'une clause intermédiaire (et en particulier une clause de P peut être utilisée plusieurs fois).

On peut prouver que la méthode de résolution est valide et complète soit :

Théorème 2.40 *Soit P un ensemble de clauses.*

1) (Validité) Si on peut déduire la clause vide de P par résolution, i.e. si $P \vdash^* \square$, alors P est insatisfaisable.

2) (Complétude) Si P est insatisfaisable, alors on peut déduire la clause vide de P par résolution, i.e. $P \vdash^* \square$.

La validité se prouve comme pour le cas du calcul propositionnel. Nous donnerons plus loin une idée d'une preuve constructive de la complétude pour la stratégie de SLD résolution pour les clauses de Horn.

Remarque 2.41 La méthode de résolution est complète pour réfuter, pas pour prouver : soit par exemple

$$P : \begin{cases} p(X) \longleftarrow \\ q(X) \longleftarrow \end{cases}$$

et soit le but $C = \{p(x), q(x) \longleftarrow\}$. On ne peut pas déduire C de P par résolution, par contre on peut déduire la clause vide de $P \cup \neg C$ par résolution.

La méthode de résolution est NP-complète, même dans le cas le plus simple du calcul propositionnel où elle se réduit aux coupures : d'où l'importance de stratégies efficaces pour implanter la méthode de résolution. Une telle stratégie est la SLD résolution dont nous verrons qu'elle reste complète pour les clauses de Horn (mais pas pour les clauses générales).

Stratégies de résolution

Une *stratégie de résolution* est une méthode permettant de restreindre à chaque étape le nombre de choix possibles sur les clauses utilisées par la règle de résolution.

Soit $P = \{C_1, C_2, C_3, \ldots, C_p\}$ un ensemble de clauses et soit C une clause, et $C'_0, C'_1, \ldots, C'_n = C$ une dérivation par résolution de C à partir de P.

Un *arbre de preuve* correspondant à la résolution $P \vdash^* C$ est un arbre binaire dont les noeuds sont étiquetés par des clauses et tel que :

- sa racine est C,
- ses feuilles sont étiquetées par des clauses de P, et
- si un noeud est étiqueté par la clause $C'_i = Res(C'_j, C'_k, \sigma)$, alors le fils gauche de ce noeud est étiqueté par la clause C'_j, et le fils droit de ce noeud est étiqueté par la clause C'_k.

Soit $P = P_1 \cup \{\longleftarrow C\} = \{A \longleftarrow \quad , \quad C \longleftarrow A, B \quad , \quad B \longleftarrow A \quad , \quad \longleftarrow C\}$, où P_1 est le programme de l'exemple 2.23 2). L'arbre ci-dessous est un arbre de preuve correspondant à la résolution $P \vdash^* \square$

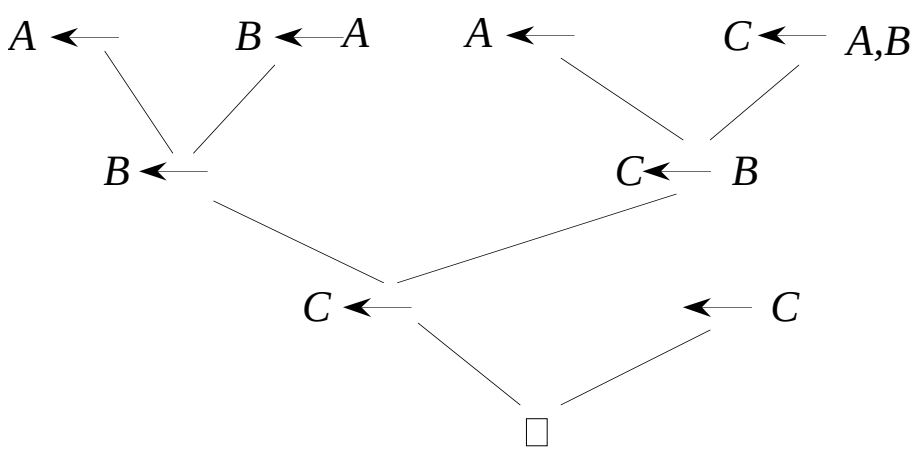

FIGURE 2.1

Résolution linéaire

Une résolution est dite *linéaire* si à chaque étape l'une des deux clauses choisies est la dernière clause obtenue par la règle de résolution. L'arbre correspondant est un arbre en "peigne", voir figure 2.2. Une réfutation linéaire est une preuve par résolution linéaire de la clause vide.

La réfutation de la figure 2.1 n'est pas linéaire. Une réfutation linéaire du même ensemble de clauses est représentée ci-dessous.

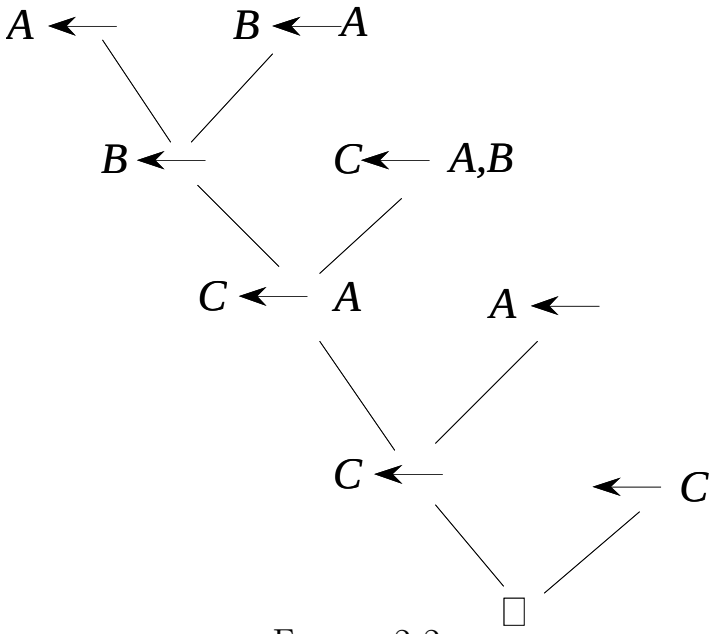

FIGURE 2.2

Résolution par entrée et SLD

Soit $P = \{C_1, C_2, C_3, \ldots, C_p\}$ un ensemble de clauses définies (clauses de Horn ayant au moins un atome positif), soit C une clause, soit G une clause négative, appelée *but*, et $C'_0, C'_1, \ldots, C'_n = C$ une dérivation par résolution de C à partir de P.

Une résolution est appelée une

– résolution *par entrée* si à chaque étape l'une des deux clauses choisies est dans P et si la première clause choisie est G. Une réfutation par entrée est une preuve par résolution par entrée de la clause vide.

– résolution SLD si elle est linéaire et par entrée.

Une stratégie st de résolution est dite complète si pour tout ensemble de clauses définies P et toute clause négative G tels que $P \cup \{G\}$ est insatisfaisable, $P \cup \{G\}$ admet une réfutation selon la stratégie st.

Théorème 2.42 *Les stratégies linéaire, par entrée et SLD sont complètes pour les clauses de Horn.*

La stratégie SLD ne fournit pas un algorithme, car à chaque étape le choix de l'atome dans la clause négative reste libre. Par exemple, on a ci-dessous deux arbres SLD différents pour le même ensemble P de clauses.

$$P : \begin{cases} (1) & chem(X,Y) \longleftarrow chem(Z,Y), arc(X,Z) \\ (2) & chem(X,X) \longleftarrow \\ (3) & arc(b,c) \longleftarrow \\ (4) & \longleftarrow chem(X,c) \end{cases}$$

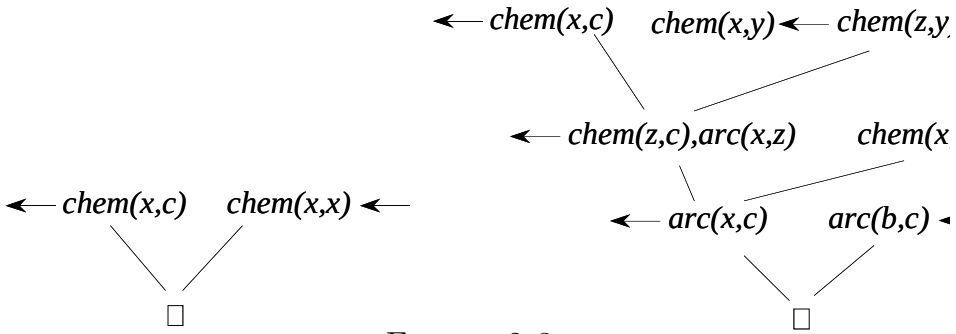

FIGURE 2.3

Pour avoir un algorithme, il faut imposer de plus le choix de l'atome de la clause négative à unifier à chaque étape, ainsi que le choix de la clause de P avec lequel on l'unifie.

Stratégie PROLOG de résolution SLD

L'implantation PROLOG de la résolution SLD unifie à chaque étape l'atome le plus à gauche de la clause négative avec la tête de la première règle de P (dans l'ordre où ces règles sont données) qui est unifiable avec cet atome. Cette stratégie est incomplète (cf. exemple 2.44).

Sémantique procédurale de DATALOG : SLD résolution pour les clauses de Horn avec variables

La SLD résolution (Selection rule driven Linear resolution for Definite clauses)

– s'applique aux clauses définies (ou clauses de Horn) pour lesquelles elle est complète, alors qu'elle n'est pas complète pour les clauses générales,

– unifie à chaque étape au plus un atome de la clause négative avec la tête de la clause choisie (linéarité),

– l'atome qui est choisi pour unification à chaque étape est choisi par une fonction de sélection qui dépend de l'implantation de la méthode de SLD résolution ; diverses fonctions de sélection sont possibles, certaines donnent des stratégies complètes, et d'autres non (par exemple la stratégie de PROLOG).

Définition 2.43 *Une SLD dérivation du but $N = \{\longleftarrow A_1, \ldots, A_n\}$ à partir du programme P est une suite de buts négatifs $N_0 = N$, $N_1, \ldots, N_p$ tels que $\forall i > 0$, N_i est un résolvent de N_{i-1} et d'une clause C_i de P (à un renommage près), résolvent qui est obtenu par une étape de résolution linéaire, c'est-à-dire que au plus un atome de N_{i-1} est unifié et coupé avec l'atome de tête de C_i ; les clauses de P peuvent être réutilisées plusieurs fois.*

Une SLD dérivation de $N = \{\longleftarrow A_1, \ldots, A_n\}$ sera donc une suite :

– $N_0 = N = \{\longleftarrow A_1, \ldots, A_{j_1}, \ldots, A_n\}$,

– $N_1 = \{\longleftarrow \sigma_1(A_1, \ldots, B_1, \ldots, B_p, \ldots, A_n)\}$ où $C_1 = \{A \longleftarrow B_1, \ldots, B_p\}$ est une clause de P et σ_1 un m.g.u. de A et A_{j_1},

– $\vdots$

– $N_i = \{\longleftarrow \sigma_i(A'_1, \ldots, B'_1, \ldots, B'_p, \ldots, A'_n)\}$ où $C_i = \{A' \longleftarrow B'_1, \ldots, B'_p\}$ est une clause de P et σ_i un m.g.u. de A' et A'_{j_i},

– $\vdots$

– $N_p = \{\longleftarrow A'_1, \ldots, A'_k\}$.

Une SLD dérivation de $N = \{\longleftarrow A_1, \ldots, A_n\}$

– échoue si $N_p \neq \square$ et on ne peut plus la prolonger en unifiant un atome de N_p avec une tête de clause de P,

– est une SLD réfutation réussie de N si $N_p = \square$, et alors la substitution $\sigma = \sigma_p \cdots \sigma_1$ est telle que $P \models \sigma(A_1, \ldots, A_n)$,

– s'enlise (floundering en anglais) si la suite N_i est infinie sans qu'aucun des N_i soit égal à $\square$.

Les SLD dérivations de $N = \{\longleftarrow A_1, \ldots, A_n\}$ sont représentées sous forme d'un arbre.

Exemple 2.44 Considérons par exemple le programme

$$P : \begin{cases} (1) & chem(X, Z) \longleftarrow chem(Y, Z), arc(X, Y) \\ (2) & chem(X, X) \longleftarrow \\ (3) & arc(b, c) \longleftarrow \end{cases}$$

Diverses SLD dérivations de $\longleftarrow chem(x, c)$ sont représentés dans la figure figure 2.4.

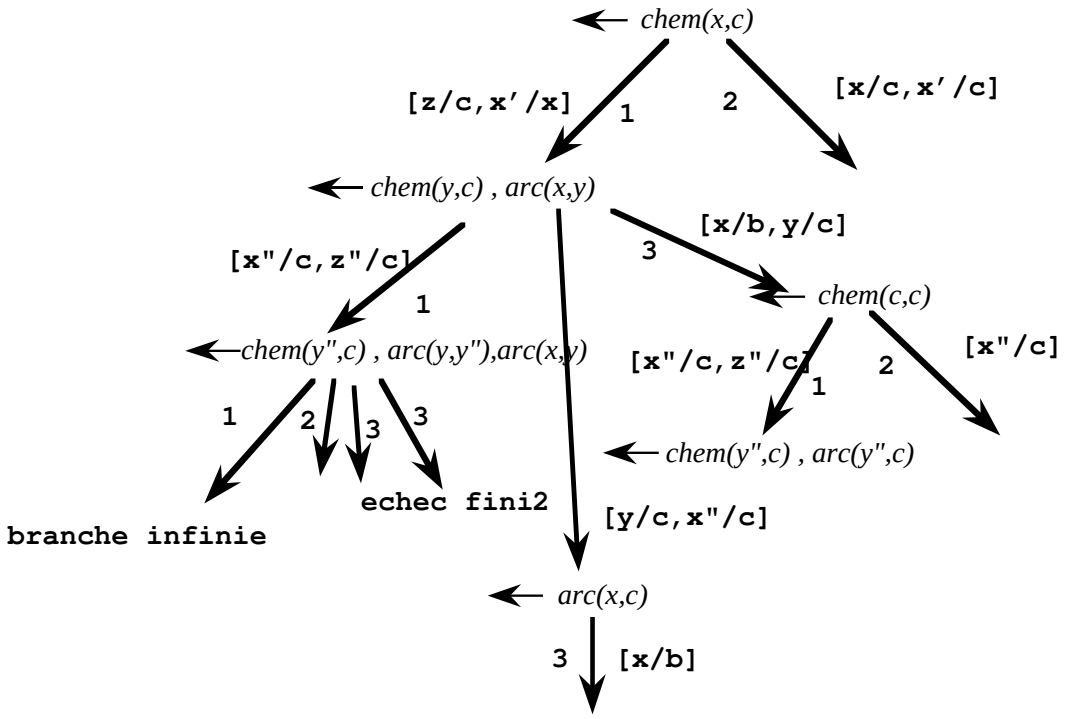

FIGURE 2.4

Soit P un programme formé de clauses de Horn et $N = \{\longleftarrow A_1, \ldots, A_n\}$ une clause négative. On peut avoir des

(i) arbres SLD réussis : une des feuilles est étiquetée par la clause vide $\square$,

(ii) arbres SLD d'échec fini : l'arbre est fini (i.e. on ne peut plus prolonger aucune des dérivations représentées par cet arbre) et aucun noeud n'est étiqueté par la clause vide $\square$,

(iii) arbres SLD infinis : toutes les branches sont infinies et aucun noeud n'est étiqueté par la clause vide $\square$, on parle alors d'enlisement.

Pour P un programme formé de clauses de Horn, dans le cas (i) $P \models \{A_1, \ldots, A_n\}$, dans le cas (ii) $P \not\models \{A_1, \ldots, A_n\}$, et dans le cas (iii)

- $P \not\models \{A_1, \ldots, A_n\}$ pour la SLD résolution générale ou les stratégies complètes de SLD résolution,
- on ne peut pas conclure pour les stratégies incomplètes de SLD résolution, comme par exemple la stratégie PROLOG.

REMARQUE 2.45 La méthode de SLD résolution est complète pour les clauses de Horn, pas pour les clauses générales : soit par exemple

$$P : \quad p(X), p(Y) \longleftarrow$$

et le but $N : \quad \longleftarrow p(U), p(Z)$. On obtient la clause vide à partir de $P \cup \neg N$ par une étape de résolution générale avec la substitution $\sigma(U) = \sigma(Y) = \sigma(Z) = X$. Par contre la SLD résolution donne un cas d'enlisement dans un arbre infini.

REMARQUE 2.46 La stratégie PROLOG de SLD résolution (unifier l'atome le plus à gauche de la première règle unifiable) peut conduire à des cas d'enlisement inutiles, voir

exemple 2.44 où la sratégie PROLOG choisit la branche infinie la plus à gauche. Cette stratégie n'est donc pas complète.

Complétude de la SLD résolution pour les clauses de Horn avec variables

La validité de la SLD résolution découle de la validité de la résolution générale, puisque la SLD résolution en est un cas particulier. Nous esquissons une preuve constructive de la complétude de la SLD résolution pour les clauses de Horn avec variables.

Théorème 2.47 *La SLD résolution est complète pour les clauses de Horn.*

Démonstration. Rappelons que pour P formé de clauses de Horn, la sémantique dénotationnelle de P est par définition l'ensemble des atomes clos de la base de Herbrand qui sont des conséquences logiques de P, i.e. $M(P) = \{A \in B_P | P \models A\}$. Pour prouver la complétude de la SLD résolution il suffit donc de démontrer que $\forall A \in M(P)$ on peut trouver une SLD réfutation de $P \cup \{\longleftarrow A\}$; puis pour réfuter $N = \{\longleftarrow A_1, \ldots, A_n\}$ on regroupera les réfutations de $P \cup \{\longleftarrow A_i\}$ pour $i = 1, \ldots, n$.

Pour trouver une SLD réfutation de $P \cup \{\longleftarrow A\}$ on fait une récurrence sur le n tel que $A \in T_P^n(\emptyset)$.

- Si $A \in T_P^1(\emptyset)$, alors nécessairement il y a dans P une clause de la forme $A \longleftarrow$, et donc $P \cup \{\longleftarrow A\} = \{A \longleftarrow\} \cup \{\longleftarrow A\}$ admet une SLD réfutation en une étape.
- Supposons le résultat vrai si $A \in T_P^{n-1}(\emptyset)$, et soit $A \in T_P^n(\emptyset)$; alors par définition de T_P, $\exists r = (B \longleftarrow B_1, \ldots, B_n) \in P$, $\exists s$ une substitution telle que $\forall i = 1, \ldots, n$, $s(B_i) = A_i \in T_P^{n-1}(\emptyset)$, $s(B) = A\}$. Par l'hypothèse d'induction, $P \cup \{\longleftarrow A_i\}$ pour $i = 1, \ldots, n$ admet une SLD réfutation D_i ; par conséquent, la SLD réfutation D qui consiste
- en une première étape qui est la résolution de $\longleftarrow A$ avec $B \longleftarrow B_1, \ldots, B_n$ donnant le résolvent $\{\longleftarrow s(B_1, \ldots, B_n)\} = \{A_1, \ldots, A_n\}$,
- puis en n blocs de résolution obtenus en mettant en séquence les SLD réfutations $D_1, \ldots, D_i, \ldots, D_n$ de $A_1, \ldots, A_i, \ldots, A_n$ pour $i = 1, \ldots, n$

est une SLD réfutation de $\longleftarrow A$.

Ceci prouve l'hypothèse d'induction et le théorème. □

Corollaire 2.48 *Pour les programmes formés de clauses de Horn, la sémantique procédurale définie par la SLD résolution coïncide avec la sémantique déclarative définie par le plus petit modèle de Herbrand.*

2.3 Extensions de DATALOG

Diverses extensions de DATALOG sont possibles en ajoutant :

- des fonctions,
- des négations,
- de fonctions d'aggrégat,
- des updates.

Ces extensions augmentent plus ou moins le pouvoir d'expression de DATALOG.

Notons d'abord qu'on ne considère en général que le sous-ensemble "range-restricted" de DATALOG défini ci-dessous, et ce afin d'avoir des requêtes "safe" c'est-à-dire n'ayant qu'un nombre fini de réponses.

Définition 2.49 *Un programme DATALOG est "range-restricted" si et seulement si pour chacune des règles du programme, toutes les variables de la tête de la règle se retrouvent comme argument d'un prédicat IDB ou EDB du corps de la règle et tous les prédicats EDB ont un domaine fini.*

Proposition 2.50 *Pour un programme "range-restricted" et une BDE donnée par des relations finies toute requête a un nombre fini de réponses.*

EXEMPLE 2.51 Avec la requête $salaire(?)$, le programme P suivant n'est pas "range-restricted", alors que P' est "range-restricted".

$$P : \begin{cases} salaire(Y) \longleftarrow smig(X)\ ,\ X \leq Y \\ smig(4800) \longleftarrow \end{cases}$$

$$P' : \begin{cases} salaire(Y) \longleftarrow smig(X)\ ,\ X \leq Y\ ,\ raisonnable(Y) \\ smig(4800) \longleftarrow \\ raisonnable(Y) \longleftarrow Y \leq 50000 \end{cases}$$

2.3.1 DATALOG + Fonctions

On permet des termes généraux T qui ne sont plus restreints à constantes C ou variables V, mais peuvent être des termes quelconques formés à partir d'un ensemble F de symboles de fonctions et des constantes et variables (i.e. $C \cup V \subset T$, et $\forall f \in F$, f d'arité n, $\forall t_1, \ldots, t_n \in T\ ,\ f(t_1, \ldots, t_n) \in T$).

Les pouvoirs d'expression de DATALOG et de DATALOG + Fonctions sont identiques, mais on n'est plus assuré que toute reqête a un nombre fini de réponses lorsque les fonctions sont permises.

En efffet, on peut simuler la fonction f par la relation

$$R_f(X, Y) \Longleftrightarrow \big(Y = f(X)\big)$$

en remplaçant toute règle $A \longleftarrow B\ ,\ R(\ldots, f(X), \ldots)$ par la règle

$$A \longleftarrow B\ ,\ R(\ldots, Y, \ldots)\ ,\ R_f(X, Y),$$

et en remplaçant toute règle

$$R(\ldots, f(X), \ldots) \longleftarrow B$$

par la règle

$$R(\ldots, Y, \ldots) \longleftarrow B\ ,\ R_f(X, Y).$$

Le programme P suivant est "range-restricted" mais la requête $entier(?)$ a une infinité de réponses.

$$P : \quad entier(s(X)) \longleftarrow entier(X)$$

2.3.2 DATALOG + Négation

DATALOG + Négation consiste à prendre la syntaxe de Datalog (sans fonctions) et à permettre des négations mais uniquement dans les corps des règles.

La sémantique se complique considérablement car

- T_P n'est plus monotone.
- P n'admet plus un plus petit modèle de Herbrand.

Exemple 2.52 Le programme suivant admet 2 modèles de Herbrand minimaux incomparables $\{p\}$ et $\{q\}$, mais pas de plus petit modèle de Herbrand.

$$P : \begin{cases} p \longleftarrow \neg q \\ q \longleftarrow \neg p \end{cases}$$

Toutefois lorsqu'on se restreint aux programmes stratifiés, c'est-à-dire sans négation "à travers" la récursion, on peut construire un modèle de Herbrand minimal (mais non minimum) canonique correspondant au programme.

Le programme de l'exemple 2.52 n'est pas stratifié, alors que le programme de l'exemple 2.53 est stratifié.

Exemple 2.53 Le programme suivant est stratifié ;

$$P : \begin{cases} r(a) \longleftarrow \\ q(x) \longleftarrow \neg r(x) \\ s(b) \longleftarrow \\ \\ p(x) \longleftarrow r(x) \\ p(x) \longleftarrow \neg q(x) \\ \\ t(x) \longleftarrow \neg r(x) \\ t(x) \longleftarrow \neg p(x) \end{cases}$$

Son modèle de Herbrand minimal (mais non minimum) canonique est

$$r(a), q(b), s(b), p(a), t(b).$$

Tous les atomes clos de la base de Herbrand qui ne sont pas dans la liste ci-dessus sont bien sûr faux dans le modèle minimal.

Formellement, un programme P est dit stratifié si on peut le mettre sous la forme $P = P_1 \cup \cdots \cup P_n$, où P_1 est un programme DATALOG sans négation, et $\forall i \leq n$,

- tous les prédicats BDI qui se trouvent dans P_i sont définis (i.e. occurrent en tête de règle) uniquement dans les $P_j,\ j \leq i$,
- tous les prédicats BDI qui sont niés dans P_i (i.e. occurrent sous forme d'un littéral négatif dans un corps de règle de P_i) sont définis (i.e. occurrent en tête de règle) uniquement dans les $P_j,\ j < i$.

On peut aussi définir les notions de programmes stratifiables (et montrer que la sémantique est indépendante de la stratification choisie) et localement stratifiables.

2.3.3 DATALOG + Aggrégats (ou ensembles)

DATALOG + Aggrégats sonsiste à prendre la syntaxe de Datalog (sans fonctions) et à permettre

- des termes de la forme $< y >$ dont la signification intuitive est "l'ensemble des y tels que..."
- des prédicats $\in$, $\cup$, $\cap$, $-$ sur les ensembles
- des fonctions d'aggrégats : min, max, average ... prenant respectivement le minimum, le maximum, la moyenne, ... d'un ensemble de nombres.

On montre que DATALOG + Aggrégats (ou ensembles) a le même pouvoir d'expression que DATALOG + négation + fonctions. On aura donc autant de difficultés pour donner la sémantique de DATALOG + Aggrégats que pour donner la sémantique de DATALOG + Négation ; les mêmes techniques de stratification s'appliqueront.

2.3.4 DATALOG + Updates

DATALOG + Aggrégats consiste à permettre des négations en tête de règles qui correspondent à des mises à jour où l'on supprime des éléments. La sémantique pose encore plus de problèmes et on ne peut guère donner que des sémantiques opérationnelles.

2.4 La méthode d'Alexandre

Il s'agit d'une méthode d'évaluation des requêtes récursives qui consiste, étant donné un programme DATALOG P et une requête récursive Q, à construire un nouveau programme DATALOG P' tel que l'évaluation de P' en chaînage avant simule l'évaluation de P en chaînage arrière sur la requête Q.

Nous illustrons cette méthode sur des exemples. Considérons le programme

$$P : \begin{cases} A(X,Y) \longleftarrow p(X,Y) \\ A(X,Y) \longleftarrow A(X,Z), A(Z,Y) \end{cases}$$

avec la requête $A(a,?)$. L'idée est de simuler l'implantation PROLOG du chaînage arrière par la SLD résolution, en générant, par exemple pour la seconde règle une nouvelle règle pour chaque sous-but correspondant à un prédicat BDI du corps de la règle : pour trouver des Y tels que $A(a,Y)$ on commence par trouver des Z tels que $A(a,Z)$ puis des Y tels que $A(Z,Y)$; formellement on introduit de nouveaux prédicats BDE et BDI $PbA^{bf}(X)$ (l'exposant bf indiquant que la requête porte sur le second argument de A qui est libre alors que le premier argument de A est lié et donné par X) et $solA(X,Y)$ ainsi que l'ensemble P' de règles suivantes :

$$P' : \begin{cases} PbA^{bf}(a) \longleftarrow \\ PbA^{bf}(X), p(X,Y) \longrightarrow solA(X,Y) \\ PbA^{bf}(X) \longrightarrow PbA^{bf}(X) \\ PbA^{bf}(X), solA(X,Z) \longrightarrow PbA^{bf}(Z) \\ PbA^{bf}(X), solA(X,Z), solA(Z,Y) \longrightarrow solA(X,Y) \end{cases}$$

La première règle $PbA^{bf}(a) \longleftarrow$ de P' est l'initialisation en fonction de la requête donnée, la règle suivante provient de la première règle de P et les trois dernières règles de P' proviennent de la seconde règle de P.

Remarquons que P' ne dépend que de P et de la requête de départ, et pas de la BDE. Illustrons sur un exemple de BDE le fonctionnement de P et P'. Soit la relation p donnée par :

$$p : \begin{cases} p(a,aa) \longleftarrow \\ p(aa,aab) \longleftarrow \\ p(b,aab) \longleftarrow \\ p(c,cc) \longleftarrow \\ p(cc,ccb) \longleftarrow \end{cases}$$

Considérons la requête $A(a,?)$. En chaînage arrière, P va calculer les réponses $A(a,aa)$, $A(a,aab)$, avec le calcul intermédiaire de $A(aa,aab)$. Les faits engendrés en chaînage

avant par P' sont les mêmes ; de fait les calculs de P' sont comme suit :

$$\begin{aligned}
&\longrightarrow PbA^{bf}(a)\\
PbA^{bf}(a), p(a,aa) &\longrightarrow solA(a,aa)\\
PbA^{bf}(a), solA(a,aa) &\longrightarrow PbA^{bf}(aa)\\
PbA^{bf}(aa), p(aa,aab) &\longrightarrow solA(aa,aab)\\
PbA^{bf}(a), solA(a,aa), solA(aa,aab) &\longrightarrow solA(a,aab)
\end{aligned}$$

Toutefois, comme la stratégie d'évaluation de PROLOG sur laquelle elle est basée, la méthode d'Alexandre peut donner des résultats catastrophiques selon l'ordre des atomes dans les corps des règles ; par exemple pour le programme P, avec la requête $A(?, aab)$ le programme P' engendré par la méthode d'Alexandre sera donné par :

$$P' : \begin{cases}
PbA^{fb}(aab) \longleftarrow\\
PbA^{fb}(Y), p(X,Y) \longrightarrow solA(X,Y)\\
PbA^{fb}(Y) \longrightarrow PbA^{ff}\\
PbA^{ff}, p(X,Y) \longrightarrow solA(X,Y)\\
PbA^{bb}(X,Y), p(X,Y) \longrightarrow solA(X,Y)\\
PbA^{bf}(X), p(X,Y) \longrightarrow solA(X,Y)\\
PbA^{ff}, solA(X,Z) \longrightarrow PbA^{bf}(Z)\\
PbA^{fb}(Y), solA(X,Z) \longrightarrow PbA^{bb}(Z,Y)\\
PbA^{bb}(X,Y), solA(X,Z) \longrightarrow PbA^{bb}(Z,Y)\\
PbA^{bf}(X), solA(X,Z) \longrightarrow PbA^{bf}(Z)\\
PbA^{fb}(Y), solA(X,Z), solA(Z,Y) \longrightarrow solA(X,Y)\\
PbA^{bf}(X), solA(X,Z), solA(Z,Y) \longrightarrow solA(X,Y)\\
PbA^{bb}(X,Y), solA(X,Z), solA(Z,Y) \longrightarrow solA(X,Y)\\
PbA^{ff}, solA(X,Z), solA(Z,Y) \longrightarrow solA(X,Y)
\end{cases}$$

Le programme P' est catastrophique à cause du prédicat BDI PbA^{ff} qu'il génére et qui lui engendre le calcul de la relation A toute entière. Il aurait suffit pour éviter ce problème d'assurer la propagation des liaisons sur les variables de la requête en changeant l'ordre des atomes de la seconde règle de P et en remplaçant cette seconde règle par :

$$A(X,Y) \longleftarrow A(Z,Y), A(X,Z)$$

La méthode dite des *magic sets* fait cette optimisation en se permettant de permuter l'ordre des atomes dans les règles pour profiter des propagations des liaisons de variables.

Remarque 2.54 L'étiquetage en exposant des nouveaux prédicats PbA donne les variables libres et liées de ces nouveaux prédicats, et chaque prédicat a une arité égale au nombre de ses variables liées ; par exemple $PbQ^{bbfbfb}(X,Y,Z,U)$ représentera la requête $A(X,Y,?,Z,?,U)$.

Exercice 2.8 Que donne la méthode d'Alexandre sur le programme

$$P : \begin{cases} A(X,Y) \longleftarrow p(X,Y) \\ A(X,Y) \longleftarrow A(X,Z), p(Z,Y) \end{cases}$$

1) Avec la requête $A(a,?)$.

1) Avec la requête $A(?,b)$. Comment modifier l'écriture de P pour améliorer ce dernier résultat ? ♢

Avec la requête $A(a,?)$, nous obtenons P' :

$$P' : \begin{cases} PbA^{bf}(a) \longleftarrow \\ PbA^{bf}(X), p(X,Y) \longrightarrow solA(X,Y) \\ PbA^{bf}(X), solA(X,Z), p(Z,Y) \longrightarrow solA(X,Y) \end{cases}$$

Avec la requête $A(?,b)$, nous obtenons P'' :

$$P'' : \begin{cases} PbA^{fb}(b) \longleftarrow \\ PbA^{fb}(Y), p(X,Y) \longrightarrow solA(X,Y) \\ PbA^{fb}(Y) \longrightarrow PbA^{ff} \\ PbA^{ff}, p(X,Y) \longrightarrow solA(X,Y) \\ PbA^{fb}(Y), solA(X,Z), p(Z,Y) \longrightarrow solA(X,Y) \\ PbA^{ff}, solA(X,Z), p(Z,Y) \longrightarrow solA(X,Y) \end{cases}$$

Pour améliorer ce dernier résultat il suffit de récrire la seconde règle de P

$$A(X,Y) \longleftarrow p(Z,Y), A(X,Z)$$

en mettant d'abord les prédicats extensionnels.