Logique du premier ordre
4ème leçon

Jean-Pierre Jouannaud
École Polytechnique
91400 Palaiseau, France

email: jouannaud@lix.polytechnique.fr
http: //w^3.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud

Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

March 14, 2005

Outline

Feuille de route

- Clauses de Horn et programmes logiques
- Résolution et factorisation binaire
- Résolution SLD
- Calcul des réponses
- Sémantique des programmes logiques

Definition

Une clause de Horn comporte au plus un littéral positif. Un *programme logique* est un ensemble fini de clauses de Horn notées
$A : -B_1, \ldots, B_p$
comportant exactement un littéral positif. Une clause est appellée *fait* si $p = 0$, et *règle* sinon. Une *requête* est une clause formée de littéraux tous négatifs, notée
$: -R_1, \ldots, R_n$

Tout programme logique possède des modèles, puisqu'il suffit d'interpréter tous les atomes en T.

Résolution et Factorisation binaires

La règle de résolution unifie des termes en nombre arbitraire plutôt que des couples de termes. Nous allons donc donner une nouvelle version de la résolution à l'aide de règles d'inférences dites binaires, appellées *RFB*, pouvant coder la résolution générale.

$$\text{Résolution Binaire : } \frac{A \vee C \qquad \neg B \vee D}{C\sigma \vee D\sigma}$$

$$\text{Factorisation Binaire : } \frac{A \vee B \vee C}{B\sigma \vee C\sigma}$$

où σ est l'unificateur principal, suivant le cas, des atomes ou des littéraux A, B.

Résolution et Factorisation binaires, suite

Theorem

Résolution et factorisation binaires sont correctes et complètes : Un ensemble de clauses S est insatisfiable ssi $\square \in RFB^*(S)$.

On montre que toute résolution se code comme une séquence de résolutions et factorisations binaires. Ce codage utilise de façon essentielle, d'une part le fait que les clauses à résoudre ne partagent pas de variables, d'autre part la présentation de l'unification sous forme de règles de transformation.

Résolution et Factorisation binaires, suite

On part d'une inférence par résolution :

$$\frac{P_1 \vee \ldots \vee P_p \vee C \qquad \neg N_1 \vee \ldots \vee \neg N_n \vee D}{C\sigma \vee D\sigma}$$

où σ désigne l'unificateur principal du problème

$$P_1 = P_2 \;\&\; \ldots \;\&\; P_1 = P_p \;\&\; P_1 = N_1 \;\&\; N_1 = N_2$$

et on résoud le problème d'unification avec une stratégie adéquate d'application des règles d'unification de manière à faire apparaître les factorisations binaires successives terminées par une résolution binaire. Les clauses à résoudre doivent être renommées pour ne pas partager pas de variables.

Résolution et Factorisation binaires, fin

On commence par unifier $P_1 = P_2$, d'où l'unificateur principal σ_2, et l'on remplace les variables de $\mathcal{D}om(\sigma_2)$ dans $P_3, \ldots, P_p$. On recommence jusqu'à résolution complète du problème $P_1 = P_2 \ \& \ \ldots \ \& \ P_1 = P_p$, d'où les unificateurs partiels $\sigma_2, \ldots, \sigma_p$. On recommence ensuite avec le problème
$N_1 = N_2 \ \& \ \ldots \ \& \ N_1 = N_n$, d'où les unificateurs partiels $\tau_2, \ldots, \tau_n$. On termine avec le problème $P_1\sigma_2 \ldots \sigma_n = N_1\tau_2 \ldots \tau_n$ qui donne la résolution binaire voulue.
Le lecteur se demandera quel est l'impact de cette stratégie sur la complexité des opérations d'unification qui doivent être effectuées.

Cas des clauses de Horn

La règle de factorisation devient inutile dans le cas des clauses de Horn (la preuve utilise à nouveau une stratégie d'unification particulière qui va permettre de résoudre autant de fois que nécessaire avec la même clause du programme). Les détails de cette preuve sont laissés au lecteur.
Ces codages étant indépendant de la forme des clauses, on en déduit la complétude de la résolution binaire négative dans le cas des clauses de Horn.

Résolution négative input

Une remarque essentielle est que le résultat d'une résolution binaire mettant en jeu des clauses de Horn est une clause de Horn. Dans le cas de programmes logiques, il est facile de voir que toute résolution négative met en jeu la requête et une clause du programe pour produire une nouvelle requête, une stratégie appellée *negative input*.

Theorem

La résolution negative input est complète pour les programmes en logique de Horn.

Résolution SLD

Étant donnée une requête $R_1, \ldots, R_p$, la stratégie négative consiste à unifier l'un des R_i avec le littéral positif H d'une clause $H : -A_1, \ldots, A_n$ du programme. Si σ est l'unificateur principal de H et R_i, on infère $A_1\sigma, \ldots, A_n\sigma$ qui peuvent être considérés comme de nouvelles requêtes à rajouter aux requêtes restantes, le but R_i étant donc effacé. Ceci revient à réécrire un des littéraux du but en utilisant une des clauses du programme, vu comme une réécriture de la tête (littéral positif) dans le corps (littéraux négatifs).

Résolution SLD

On peut facilement montrer que le choix d'un littéral particulier du but n'a pas d'influence sur l'obtention de la clause vide. Il suffit pour cela de faire commuter les résolutions successives.
On parle à ce propos de non-déterminisme "don't care", et on va donc adopter un ordre particulier pour le choix du littéral à éliminer à chaque étape.
Par contre, toutes les clauses permettant de réécrire ce but devront être considérées : on parle cette fois de non-déterminisme "don't know".

Résolution SLD

On appelle *fonction de sélection* un ordre total sur les littéraux négatifs qui spécifie le but à réécrire en priorité : on résoud toujours sur le but minimal. Cette stratégie est complète : à chaque étape, un seul but est à considérer.
La règle de résolution avec une fonction de sélection et pour les programmes logiques est appelée *SLD-résolution*.

Theorem

La SLD-résolution est complète.

Calcul des réponses

Le *résultat* d'une requête est l'ensemble des valeurs des variables de la requête calculé au cours de la résolution. Pour des raisons d'efficacité, on n'applique pas les substitutions, mais on considère des *clauses contraintes* :

Definition

Une clause contrainte est une paire formée d'une clause C et d'un problème d'unification ϕ, notée $C \mid \phi$, qui représente l'ensemble des clauses $\{C\sigma \mid \sigma \text{ close et } \sigma \models_{T(F)} \phi\}$.

En général, on évite le test d'occurrence dans l'unification en utilisantt $\models_{RT(F)}$.

Résolution contrainte

La règle de résolution peut alors s'écrire:

$$\frac{A(t) \vee C \qquad \neg B(u) \vee D \mid \phi}{C \vee D \mid \phi \wedge A(t) = B(u)}$$

Les variables de t n'apparaîssant pas dans u.
Il faut rajouter des règles pour les contraintes.
Si $R \mid \phi$ désigne la requête contrainte courante :

$$\begin{array}{rcll} R \mid \phi & \rightarrow & R \mid \phi' & \text{si } \phi \longrightarrow_{UNIF} \phi' \\ R \mid \bot & \rightarrow & \text{échec} & \\ \square \mid \phi & \rightarrow & \text{succès}(\phi) & \phi \text{ est en forme résolue} \end{array}$$

Résolution contrainte

La *réponse* à une requête $\neg B$ est alors la contrainte ψ telle que l'on ait inféré la clause $\square \mid \psi$. La contrainte ψ est donnée en forme résolue. Ses solutions sont des substitutions σ telles que $P \models B\sigma$.

Le fait que les contraintes soient des équations ne joue de rôle que pour la mise en forme résolue et on peut donc employer d'autres systèmes de contraintes plus expressifs (CHIP, ILOG-solver). Les clauses du programme P peuvent bien-sûr être contraintes elles-aussi.

Exemple

Soit le programme logique

$$
\begin{aligned}
& l(x, [], x.[]). \\
& l(x, x.l, x.l). \\
& l(x, y.l, y.l') \leftarrow l(x, l, l') \mid x \neq y
\end{aligned}
$$

Qui a pour but d'insérer un élément dans une liste sans doublons.
Soit la requête $l(0, s(0).(x.[]), y)$. On obtient deux dérivations possibles conduisant à $\square \mid \psi$ avec une contrainte ψ satisfiable: la première est (par souci de simplicité, nous n'écrivons la contrainte complète que dans cette séquence, puis nous écrirons une contrainte simplifiée).

Exemple, suite

$$
\begin{aligned}
& I(0, s(0).(x.[]), y) \\
& \quad \rightarrow I(x_1, I, I') \mid \\
0 = x_1 \wedge y_1.I = s(0).(x.[]) \wedge y = y_1.I' \wedge x_1 \neq y_1 & \\
& \quad \rightarrow \square \mid \\
\exists x_2, I', I'', \quad \wedge y = s(0).I' \wedge x_2 = 0 \wedge x_2.I'' = x.[] \wedge x_2.I'' & \\
& \quad \rightarrow \square \mid x = 0 \wedge y = s(0).(0.[])
\end{aligned}
$$

Exemple, fin

$$
\begin{gathered}
I(0, s(0).(x.[]), y) \\
\rightarrow I(x_1, l, l') \mid x_1 = 0 \wedge l = x.[] \wedge y = s(0).l' \rightarrow \\
I(x_2, l_1, l_1') \mid \exists y_2, l, l',\ l = x.[] \wedge y = s(0).l' \\
\wedge x_2 = 0 \wedge l = y_2.l_1 \wedge l' = y_2.l_1' \wedge x_2 \neq y_2 \\
\rightarrow \square \mid \exists x_2, l_1, l_1', y_2, l, l',\ l_1 = [] \wedge l_1' = x_2.[] \\
\wedge l = x.[] \wedge y = s(0).l' \wedge x_2 = 0 \wedge l = y_2.l_1 \\
\wedge l' = y_2.l_1' \wedge x_2 \neq y_2 \\
\rightarrow \square \mid \wedge y = s(0).(0.(x.[])) \wedge x \neq 0
\end{gathered}
$$

Complétude de la résolution contrainte

Lemma

Si la réponse ψ à une requête R pour un programme logique P a pour mgu σ, alors, pour toute substitution close θ, $R\sigma\theta, P$ est insatisfiable.

Lemma

Si R est une requête pour un programme logique P et si $\mathcal{R}\theta, P$ est insatisfiable, alors il existe une réponse ψ à R et une substitution τ telles que ψ a pour mgu σ et $\theta = \sigma\tau$.

Plus petit modèle de Herbrand

Soit P est un programme logique. On note $\mathcal{B}$ la base de Herbrand et $2^{\mathcal{B}}$ l'ensemble des parties de $\mathcal{B}$, ou interprétations de Herbrand ($I \in 2^{\mathcal{B}}$ définit les atomes de la base qui s'interprètent en vrai, , les autres s'inteprétant en faux). On note M_P l'intersection de tous les modèles de Herbrand de P, qui sera le "plus petit modèle de Herbrand" du programme :

$$M_P = \bigcap_{I \in 2^{\mathcal{B}}, I \models P} I$$

Lemma

Pour tout programme logique P, $M_P \models P$.

Preuve

Il suffit de montrer que l'intersection d'un nombre quelconque de modèles de Herbrand de P est aussi un modèle de Herbrand de P.
On fait le raisonnement sur des clauses closes pour simplifier.
Soit $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ une clause C. Il y a deux cas.
$A \in M_P$. Alors $M_P \models C$.
$A \notin M_P$. Alors, par définition de M_P, il existe un modèle de Herbrand I de P tel que $A \notin I$. Mais comme $I \models C$, il existe j tel que $B_j \notin I$, et donc $B_j \notin M_P$ par définition de M_P, d'où $M_P \models C$.

Exemple

Soit P l'ensemble de clauses

$$Q(s(x)) \vee \neg Q(x)$$
$$P(0) \vee \neg Q(s(x))$$

Alors $M_P = \emptyset$ car l'interprétation vide (tout est faux) satisfait bien P.

Exercice

Soit P le programme:

$$
\begin{aligned}
& l(x, [], x.[]) :- \\
& l(x, x.l, x.l) :- \\
& l(x, y.l, y.l') \quad :- \quad l(x, l, l') \mid x \neq y
\end{aligned}
$$

Quel est le plus petit modèle de Herbrand de ce programme ?

Opérateur de conséquence immédiate

Si P est un programme logique, on note T_P l'*opérateur de conséquence immédiate* défini sur $2^{\mathcal{B}}$ par

$$T_P(I) = \{A \in \mathcal{B} \mid \left.\begin{array}{l} A \vee \neg A_1 \vee \ldots \vee \neg A_n \\ \text{ est une instance close} \\ \text{d'une clause de } P \\ \text{et } A_1, \ldots, A_n \in I \end{array}\right\}$$

Exemples

Avec $P = \{Q(s(x)) \vee \neg Q(x), P(0) \vee \neg Q(s(x))$.
$T_P(\emptyset) = \emptyset$,
$T_P(\mathcal{B}) = \{P(0)\} \cup \{Q(s(t)) \;\; t \in T(\mathcal{F})\}$
$T_P(T_P(\mathcal{B})) = \{P(0)\} \cup \{Q(s(s(t))) \;\; t \in T(\mathcal{F})\}$
etc.

Exercice : Qu' obtient-on avec les clauses du programme

$$
\begin{array}{lll}
l(x, [], x.[]) & :- & \\
l(x, x.l, x.l) & :- & \\
l(x, y.l, y.l') & :- & l(x, l, l') \mid x \neq y
\end{array}
$$

Croissance

$2^{\mathcal{B}}$ est muni d'une structure de treillis complet pour l'ordre $\subseteq$ (i.e. l'ordre d'inclusion des interprétations). On va donc pouvoir caractériser le plus petit point fixe de T_P grâce au théorème de Tarski, comme la limite de ses approximations successives, pourvu que l'opérateur T_P soit croissant et continu.

Lemma

T_P est croissant sur les interprétations de Herbrand.

La preuve est laissée en exercice.

Continuité

Soit une famille croissante de parties de $\mathcal{B}$, $\{I_k\}_{k\in\mathbb{N}}$, dont le sup, le treillis des parties étant complet pour l'ordre, est noté $S = \bigcup_k I_k$.
T_P étant croissante, $\{T_P(I_k)\}_{k\in\mathbb{N}}$, est une autre famille croissante dont le sup est noté $\bigcup_k T_P(I_k)$.
L'équation $T_P(\bigcup_k I_k) = \bigcup_k T_P(I_k)$ exprime la continuité de l'opérateur T_P.

Lemma

T_P est continu sur les interprétations de Herbrand.

Continuité, suite

$$T_P(\bigcup_{I \in S} I) = \{A \in \mathcal{B} \mid A \vee \neg A_1 \vee \ldots \vee \neg A_n$$
est une instance close d'une clause de$P$ et
$$\forall i \in [1..n]\ A_i \in \bigcup_{I \in S} I\}$$
$$= \{A \in \mathcal{B} \mid \forall i \exists I_j \in S,\ A \vee \neg A_1 \vee \ldots \vee \neg A_n$$
est une instance close d'une clause de P et $A_i \in I_j\}$
(Notons que I_j dépend de i.)
$$= \{A \in \mathcal{B} \mid \exists I \in S,\ A \vee \neg A_1 \vee \ldots \vee \neg A_n$$
est un instance close d'une clause de P et $\forall i A_i \in I\}$
Notons ici que I ne dépend plus de i.
Cela utilise la propriété déjà montrée
de croissance de la famille d'interprétations.

Modèle des approximations finies d'un programme

Par le théorème de Tarski, le plus petit point fixe de T_P existe (monotonie de T_P) et est le sous-ensemble (continuité de T_P)

$$F \stackrel{d}{=} T_P^{\omega}(\emptyset) = \bigcup_k T_P^k(\emptyset)$$

Lorsque T_P est monotone, T_P^{α} pour un ordinal α est défini par récurrence transfinie par: $T_P^0 = \emptyset$, si $T_P^{\alpha+1} = T_P(T_P^{\alpha})$ et, T_P^{α} est la borne supérieure des T_P^{β} pour $\beta < \alpha$ si α est un ordinal limite. La continuité implique $\alpha = \omega$, le premier ordinal transfini.

Lemme d'inclusion

Nous montrons maintenant que le modèle minimal est point fixe de T_P, et que tout point fixe de T_P est modèle de P. Cela assurera que le modèle minimal coincide avec le plus petit point fixe.

Lemma

Soit I une interprétation de Herbrand et P un programme logique. I est un modèle de P si et seulement si $T_P(I) \subseteq I$.

$I \models P$ ssi pour toute instance close $A \vee \neg A_1 \vee \ldots \vee \neg A_n$ d'une clause de P, $I \models A_1, \ldots, A_n$ implique $I \models A$. $\square$

Plus petit point fixe d'un programme logique

Theorem

Si P est un programme logique, M_P est le plus petit point fixe de T_P.

Tout point fixe de T_P est modèle de P par le lemme. Il suffit donc de montrer que M_P est point fixe de T_P pour conclure par minimalité. M_P étant modèle de P, $T_P(M_P) \subseteq M_P$ par le lemme. D'après la propriété de monotonie, $T_P^2(M_P) \subseteq T_P(M_P)$ et donc $T_P(M_P)$ est un modèle de P par le lemme. Par minimalité de M_P on a donc $M_P \subseteq T_P(M_P)$ et donc $T_P(M_P) = M_P$.

Les approximations finies du modèle minimal

Exemple

Il est possible de généraliser la définition de T_P aux clauses arbitraires :

$$T_P(I) = \{A \in \mathcal{B} \mid A \vee C \text{ est une instance}$$

$$\text{close d'une clause de } P \text{ et } I \models \neg C\}$$

Mais dans ce cas T_P n'est plus nécessairement continu ni même croissant :

Exemple : Supposons P réduit à la seule clause $A \vee B$. $T_P(\emptyset) = \{A, B\}$ et $T_P(\{A, B\}) = \emptyset$, $T_P(\{A\}) = \{A\}$ et $T_P(\{B\}) = \{B\}$. Par exemple pour la partie dirigée $S = \{\emptyset, \{A\}\}$, $T_P(\bigcup_{I \in S} I) = T_P(\{A\}) = \{A\} \neq \{A, B\} = T_P(\emptyset) = \bigcup_{I \in S} T_P(I)$.

Exercice

Donner un ensemble de clauses tel que T_P^ω n'est pas un point fixe alors que $T_P^{2\times\omega}$ est un point fixe.

Exercice

La stratégie *unitaire* considére des résolutions dont une prémisses est réduite à un littéral.

1. Montrer que la stratégie unitaire est incomplète : donner un ensemble de clauses closes insatifiable tel que factorisation + résolution unitaire ne permet pas de dériver la clause vide.
2. Montrer qu'il existe une réfutation d'un ensemble S de clauses par la résolution unitaire si et seulement si il existe une réfutation de S par la stratégie "input".
3. Montrer que la stratégie unitaire est complète pour les clauses de Horn. Est-ce une stratégie intéressante ?

Bibliographie essentielle

kowalski70mi,kowalski76jacm,apt82,colmerauer82,co