Cours 6
Prolog

Jean-Jacques.Levy@inria.fr

http://jeanjacqueslevy.net

secrétariat de l'enseignement:

Catherine Bensoussan

cb@lix.polytechnique.fr

Laboratoire d'Informatique de l'X

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique

Références

- The Art of Prolog, Leon Sterling & Ehud Shapiro, MIT Press, 1986
- Programming Languages, Concepts and Constructs, Ravi Sethi, 2nd edition, 1997.
  `http://cm.bell-labs.com/who/ravi/teddy/`
- Theories of Programming Languages, J. Reynolds, Cambridge University Press, 1998.
  `http://www.cs.cmu.edu/afs/cs.cmu.edu/user/jcr/www/`

Plan

1. Datalog
2. Prolog sur des termes structurés
3. Syntaxe de Prolog
4. Termes, Clauses
5. Principe de résolution
6. Unification
7. Unification rapide

Programmation

=

Logique

Un peu de Prolog

Faits

```
pere(charles,napoleon).
pere(charles,lucien).
pere(charles,joseph).
pere(napoleon,aiglon).
pere(lucien,charlotte).
pere(lucien,charles_lucien).
pere(lucien,christine).

mere(josephine,aiglon).
```

```
male(charles).
male(napoleon).
male(lucien).
male(joseph).
male(charles_lucien).

femelle(charlotte).
femelle(christine).
femelle(josephine).
```

Requêtes

```
femelle(napoleon)?
pere(josephine,aiglon)?
```

La prèmière répondra `oui`, la deuxième `non`.

Un peu de Prolog

Requêtes existentielles $\exists X.pere(lucien, X)$

```
pere(lucien,X)?
X=charlotte
X=charles_lucien
X=christine
```

Réversibilité $\exists X.pere(X, charles\_lucien)$

```
pere(X,charles_lucien)?
X=lucien
```

Un peu de Prolog

Requêtes avec conjonction

`pere(charles,napoleon),femelle(josephine)?`

Le système répond `oui`.

Requête conjonctive plus intéressante $\exists X. pere(lucien, X) \wedge male(X)$

```
pere(lucien,X), male(X)?
X=charles_lucien
```

Autre Requête conjonctive $\exists X. \exists Y. pere(charles, X) \wedge pere(X, Y)$

```
pere(charles,X), pere(X,Y)?
X=napoleon, Y=aiglon
X=lucien, Y=charlotte
X=lucien, Y=charles_lucien
X=lucien, Y=christine
```

Règles – Clauses

On peut définir des règles grâce aux clauses suivantes

```
fils(X,Y) <- pere(Y,X), male(X).
fille(X,Y) <- pere(Y,X), femelle(X).
grandpere(X,Y) <- pere(X,Z), pere(Z,Y).
```

signifiant

$\forall X.\forall Y.pere(Y, X) \wedge male(X) \supset fils(X, Y)$

$\forall X.\forall Y.pere(Y, X) \wedge femelle(X) \supset fille(X, Y)$

$\forall X.\forall Y.\forall Z.pere(X, Z) \wedge pere(Z, Y) \supset grandpere(X, Y)$

et lancer les requêtes

```
fils(napoleon,X)?
fille(X,charlotte)?
grandpere(charles,X)?
```

Un programme Prolog est un ensemble fini de clauses.

Récursivité

La disjonction de deux clauses permet d'écrire

```
parent(X,Y) <- pere(X,Y).
parent(X,Y) <- mere(X,Y).
```

Rien de plus que l'application de la tautologie

$(B \vee C) \supset A \ \equiv \ (B \supset A) \vee (C \supset A)$

Inductivement, on a

$\forall X.ancetre(X, X)$

$\forall X.\forall Y.\forall Z.parent(X, Z) \wedge ancetre(Z, Y) \supset ancetre(X, Y)$

D'où le programme récursif

```
ancetre(X,X) <-.
ancetre(X,Y) <- parent(X,Z), ancetre(Z,Y).
```

Structures de données et termes

Les prédicats peuvent porter sur des objets structurés. Par exemple des listes

```
append([],Y,Y) <-.
append(X::Xs,Y::Ys,Z::Zs) <- append(Xs,Ys,Zs).
```

Requêtes possibles

```
append(3::4::[],Y) ?
append(X, 3::4::Y, 1::2::3::4::Z) ?
```

Exercice 1 **Écrire** *member*, *reverse*.

De même sur les entiers unaires, on peut écrire

```
plus(X,0,X) <-.
plus(X,s(Y),s(Z) <- plus(X,Y,Z).
times(0,X,0) <-.
times(s(X),Y,Z) <- times(X,Y,W), plus(W,Y,Z).
```

Exercice 2 **Écrire factorielle, Fibonacci, Ackermann.**

Syntaxe de Prolog

$\mathcal{P}$	$::=$	$C_1, C_2, \ldots C_n$	**programme**
C	$::=$	$P \leftarrow P_1 \wedge P_2 \wedge \ldots P_n$	**clause de Horn**
P	$::=$	$p(M_1, M_2, \ldots M_n)$	**littéral** (n **arité de** p)
M	$::=$	$f(M_1, M_2, \ldots M_n)$	**terme** (n **arité de** f)
	$\mid$	X	**variable**
		$P \leftarrow$	**Fait**

$FV(M)$, $FV(P)$, $FV(C)$ **sont les ensembles des variables de** C, P, C.

Exercice 3 **En donner la définition récursive.**

Faits et Clauses

Les faits sont donc des clauses sans prémisses. Les clauses sont des assertions universelles. Si $\vec{X} = FV(C)$, la clause C représente le prédicat $\forall \vec{X}.C$.

Buts

Les buts sont existentiels. Si $\vec{X} = FV(P_1 \wedge P_2 \wedge \ldots P_n)$, le but à montrer est $\exists \vec{X}.P_1 \wedge P_2 \wedge \ldots P_n$.

Raisonnement par contradiction

On essaie de montrer que $\neg \exists \vec{X}.P_1 \wedge P_2 \wedge \ldots P_n$ aboutit à une contradiction.

On rajoute donc le fait $\forall \vec{X}.\neg P_1 \vee \neg P_2 \vee \ldots \neg P_n$

C'est à dire la clause $\forall \vec{X}. \leftarrow P_1 \wedge P_2 \wedge \ldots P_n$

Montrer des buts consiste à rajouter une clause dont les prémisses sont les buts et la conclusion est vide, et montrer que le tout aboutit à une contradiction.

Résolution propositionnelle

En logique, la règle de coupure suivante est valide

$$\frac{\vdash P \vee Q \qquad \vdash \neg P \vee R}{\vdash Q \vee R}$$

Soit une clause $P \leftarrow P_1 \wedge P_2 \wedge \ldots P_n$, alors on peut réduire les ensembles de buts comme suit

$$\{P, Q_2, \ldots Q_m\} \longrightarrow \{P_1, P_2, \ldots P_n, Q_1, Q_2, \ldots Q_m\}$$

Exercice 4 Montrer que ce remplacement est valide.

Comme $n \geq 0$, après un certain nombre de coupures on peut se retrouver avec un ensemble vide de buts. Equivalent au prédicat faux. Donc contradiction.

Cette méthode [Robinson] est le principe de résolution.

Substitutions

Une substitution σ est une fonction des variables dans les termes. On écrira $\sigma = [X_1 \backslash M_1, X_2 \backslash M_2, \cdots X_n \backslash M_n]$ quand son graphe est fini. On l'étend naturellement comme fonction des termes dans les termes. On écrit $M\sigma$ pour le terme obtenu en appliquant σ à M. De même, pour la composition $\sigma \circ \sigma'$ de deux substitutions σ et σ'.

Posons $M \leq M'$ ss'il existe σ tel que $M' = M\sigma$
et $\sigma \leq \sigma'$ ss'il existe ϕ tel que $\sigma' = \phi \circ \sigma$.

On peut remarquer que les termes ont une structure de treillis vis à vis de $\leq$.

De même, tout système d'équations sur les termes admet une solution minimale.

Un terme sans variable est un terme clos. Une substitution close remplace des variables par des termes clos.

Deux formules $\mathcal{F}$ et $\mathcal{F}'$ sont disjointes si $FV(\mathcal{F}) \cup \mathcal{FV}(\mathcal{F}') = \emptyset$

Résolution

On peut avoir à tenir compte des variables pour arriver à une réfutation. La règle de coupure correspondante est

$$\frac{\vdash P \vee Q \qquad \vdash \neg P' \vee R \qquad P\sigma = P'\sigma \qquad P \vee Q \text{ et } \neg P' \vee R \text{ disjointes}}{\vdash Q\sigma \vee R\sigma}$$

Résolution SLD

Soit un ensemble de buts $\{Q_1, Q_2, \ldots Q_n\}$. Les règles de calcul de Prolog sont les suivantes

Succès $\{\,\} \longrightarrow \texttt{succes}$

SLD $\{Q_1, Q_2 \ldots Q_m\} \longrightarrow \{P_1\sigma, P_2\sigma, \ldots P_n\sigma, Q_2\sigma, \ldots Q_m\sigma\}$
où $P \leftarrow P_1 \wedge P_2 \wedge \ldots P_n$ est une clause disjointe des buts telle que $Q_1\sigma = P\sigma$

Echec $\{Q_1, Q_2 \ldots Q_m\} \longrightarrow \texttt{echec}$ sinon

Exemples

```
fils(napoleon,X)
pere(X,napoleon),male(napoleon)
male(napoleon)    [X=charles]
succes    [X=charles]
```

```
append(X,2::Y,1::2::Z)
append(Xs,2::Y,2::Z)   [X=1::Xs]
succes    [X=1::[], Y=Z]
```

```
append(X,2::Y,1::2::Z)
append(Xs,2::Y,2::Z)   [X=1::Xs]
append(X1s,Y,Z)   [X=1::2::X1s]
succes    [X=1::2::[], Y=Z]
```

Non déterminisme

Plusieurs solutions sont possibles car on peut choisir arbitrairement les buts à remplacer, et les règles à utiliser.

Théorème [correction] L'ensemble des solutions trouvées est correct.

Théorème [complétude] Par résolution, on trouve toutes les solutions.

Retour en arrière et stratégie de Prolog

Un des choix peut aboutir à une impasse. On doit revenir en arrière (*bactracking*) pour en essayer un autre.

L'ensemble des solutions trouvées dépend de la stratégie de réduction. Prolog fait une stratégie en profondeur d'abord. Il peut louper des solutions.

Par exemple, deviner l'ensemble des solutions trouvées pour

```
ancetre(X,X).
ancetre(X,Y) <- ancetre(X,Z),parent(Z,Y).
```

Unification sur les termes

C'est le même problème que celui déjà vu pour l'inférence de types. Seule l'algèbre des termes à considérer diffère.

$\mathtt{mgu}(E)$ **est la plus petite solution du système d'équations E si elle existe. Le résultat est donc** `echec` **ou une substitution. (***most general unifier***)**

Algorithme de Robinson

$$
\begin{array}{rcll}
\mathtt{mgu}(\emptyset) & = & \mathtt{identite} & \\
\mathtt{mgu}(\{X = X\} \cup E) & = & \mathtt{mgu}(E) & \\
\mathtt{mgu}(\{X = M\} \cup E) & = & \mathtt{mgu}(E[X\backslash M]) \circ [X\backslash M] & \textbf{si } X \notin FV(t) \\
\mathtt{mgu}(\{M = X\} \cup E) & = & \mathtt{mgu}(E[X\backslash M]) \circ [X\backslash t] & \textbf{si } X \notin FV(t) \\
\mathtt{mgu}(\{F(\vec{M}) = F(\vec{N})\} \cup E) & = & \mathtt{mgu}(\{M_1 = N_1, \ldots M_n = N_n\} \cup E) & \\
\textbf{sinon} \quad \mathtt{mgu}(E) & = & \mathtt{echec} &
\end{array}
$$

Unification rapide

Algorithme [Martelli-Montanari; Huet-Kahn]

Equations sur les arbres rationnels (sans *occur-check* pour les variables) et on évite l'élimination des variables qui fait augmenter la taille des termes.

Exemple: $M = f(g(y, h(t, y)), x)$, $N = f(x, g(h(h(t, t), h(a, b))))$

On génère un ensemble de classes d'équivalence sur les sous-termes de M et de N.

$$\begin{array}{ll} n_1 = f(n_2, x) & n_3 = f(x, n_5) \\ n_2 = g(y, n_4) & n_5 = g(n_6, y) \\ n_4 = h(t, u) & n_6 = h(n_7, n_8) \\ & n_7 = h(t, t) \\ & n_8 = h(a, b) \end{array}$$

En TD

- continuer évaluateur, interpréteur, vérificateur de types, synthétiseur de types.
- mettre les listes dans PCF
- écrire le programme des cousins en Prolog. Se servir de la magnifique base généalogique dynamique de Daniel de Rauglaudre `loupiac.inria.fr` ou `geneweb.inria.fr`.

A la maison et prochaine fois

- Allocation de l'espace-mémoire