Programmation Logique Inductive

Fabien Torre

Université de Lille

Mercredi 25 novembre et 2 décembre 2009

Constantes et variables [Lloyd, 1987]

Définition : constante

Une *constante* est le nom d'un objet précis, par convention, en minuscules.
Exemples : `0`, `zéro`, `jean` ou encore `train42`.

Définition : variable

Une *variable* permet de nommer un objet sans l'identifier précisément comme avec une constante, par convention le nom d'une variable commence par une lettre majuscule.
Exemples : *X*, *A*, *Personne*.

Fonctions et termes [Lloyd, 1987]

Définition : fonction

Une *fonction* est un symbole et une arité (une constante peut être vue comme une fonction d'arité nulle).
! seulement une représentation, pas de calcul effectif !
Exemple : `s` d'arité 1, pourra représenter la fonction qui à un entier n associe son successeur, c'est-à-dire $n+1$.

Définition : terme

Un *terme* est une constante, une variable, ou un symbole de fonction associé à ses arguments qui sont aussi des termes (dont le nombre correspond à l'arité de la fonction).
Exemples : `0`, `s(s(0))`, `s(V)`.

Prédicats et atomes [Lloyd, 1987]

Définition : prédicat

La définition d'un *prédicat* nécessite un symbole et une arité. Il fournira une valeur booléenne fonction des arguments, une fois ceux-ci instanciés.
Exemple : `add /3` est vrai si le dernier argument est la somme des deux premiers.

Définition : atome

Un *atome* est un symbole de prédicat avec des termes comme arguments (en nombre correspondant à l'arité).
une valeur de vérité pourra lui être associée
Exemple : L'atome `add(0,X,X)` indique ainsi que somme de zéro et d'une valeur `X` quelconque, vaut `X`.

Littéraux [Lloyd, 1987]

Définition : littéral

Un *littéral* est un atome nié ou non : s'il est nié le littéral est dit *négatif*, *positif* sinon.
Exemples : `add(0,X,X)` est un littéral, ¬`add(0,X,X)` aussi.

Clauses et clauses de Horn [Lloyd, 1987]

Définition : clause

Une *clause* est une disjonction de littéraux.

Définition : clause de Horn

Une *clause de Horn* est une clause dont au plus un des littéraux est positif, les autres négatifs.

Clauses définies [Lloyd, 1987]

Définition : clause définie

Une *clause définie* est une clause dont exactement un des littéraux est positif, les autres négatifs.
Exemple : $A_0 \vee \overline{A_1} \vee \overline{A_2} \vee \ldots \vee \overline{A_n}$

Une clause définie peut se récrire sous forme de règle :

$$A_0 \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

où A_0 est la tête de la clause, les autres atomes formant son corps.

Les variables de la tête sont quantifiées universellement, les autres sont quantifiées existentiellement : male(X) ← pere(X,Y).

Substitution [Lloyd, 1987]

Définition : substitution

Une substitution est un ensemble de couples terme/variable. L'application d'une substitution θ à une clause C fournit une clause, notée $C\theta$, obtenue en remplaçant les variables de C apparaissant dans θ par les termes correspondants.

$$C = pere(X, Y), pere(Y, Z)$$
$$\theta = \{abraham/X; homer/Y; bart/Z\}$$
$$C\theta = pere(abraham, homer), pere(homer, bart)$$

Vue relationnelle

Lien avec les bases de données : un prédicat est une relation (ou une table).

père	
abraham	homer
clancy	marge
homer	bart
homer	lisa
homer	maggie
...	...

mère	
mona	homer
jacqueline	marge
marge	lisa
marge	bart
marge	maggie
...	...

Exemples de programmes

un programme Prolog

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mere(marge,lisa). mere(marge,bart).
mere(marge,maggie).
pere(homer,lisa). pere(homer,bart).
pere(homer,maggie).

parent(X,Y) :- pere(X,Y).
parent(X,Y) :- mere(X,Y).

frere_ou_soeur(X,Y) :- parent(Z,X), parent(Z,Y), X \= Y.
```

Hypothèse du monde clos et résolution... Prolog !

Inductive Logic Programming [Muggleton, 1991, Muggleton and De Raedt, 1994]

ILP : apprentissage de programmes logiques.

- $\mathcal{E}$ et $\mathcal{H}$ sont des clauses dont la tête a pour symbole de prédicat le concept que l'on cherche à caractériser ;
- les clauses de $\mathcal{E}$ n'utilisent pas de variable ;
- cible : souvent une seule clause, pas récursive...
- possible existence d'une *théorie du domaine* à travers des prédicats déjà définis ; prise en compte ?
- quels sont les tests de subsomption possibles ? et le moindre généralisé (unicité, complexité) ?

VC dimension ?

L'implication logique

- La relation la plus naturelle ;
- mais indécidable entre clauses [Schmidt-Schauß, 1988] ;
- et indécidable entre clauses de Horn [Marcinkowski and Pacholski, 1992].

Nota bene : le moindre généralisé est défini, son calcul est de complexité exponentielle [Nienhuys-Cheng and de Wolf, 1996].

La θ-subsomption [Plotkin, 1970]

Définition : θ-subsomption

On dit que C θ-subsume D ($C \succeq_\theta D$) si et seulement s'il existe une substitution θ telle que $C\theta \subseteq D$.
C et D sont θ-équivalentes ($C \sim_\theta D$) ssi $C \succeq_\theta D$ et $D \succeq_\theta C$.

NP-difficile [Kapur and Narendran, 1986], équivalente à l'implication logique sur les clauses non récursives [Gottlob, 1987].

Pistes

- implémentations *efficaces* [Kietz and Lübbe, 1994, Scheffer et al., 1996, Maloberti and Sebag, 2004] ;
- subsomption *stochastique* [Sebag and Rouveirol, 1997] ;
- clauses *déterminées* [Muggleton and Feng, 1990].

θ-subsomption : exemple

grand-père(A,B) ← père(A,C), père(C,B) $\succeq_\theta$
grand-père(abraham,bart) ← père(abraham,homer), père(homer,bart) ?

grand-père(A,B)	grand-père(abraham,bart)	$A/abraham$, $B/bart$
père(A,C)	père(abraham,homer) père(homer,bart)	$A/abraham$, $C/homer$ $A/homer$, $C/bart$
père(C,B)	père(abraham,homer) père(homer,bart)	$C/abraham$, $B/homer$ $C/homer$, $B/bart$

Oui, avec $\theta = \{A/abraham, B/bart, C/homer\}$.

Autre exemple : q ← p(X,Y), p(Y,X) et q ← p(X,X) ?

Réduction sous θ-subsomption

Définition : clause non réduite

Une clause C est dite *non réduite* si C contient un littéral L tel que $C \sim_\theta C \setminus L$ ou, autrement dit, s'il existe une substitution θ telle que $C\theta \subsetneq C$.

Trois clauses

C_0 : $c(X) \leftarrow p(X, s(0))$
C_1 : $c(X) \leftarrow p(X, s(0)), p(A, B)$
C_2 : $c(X) \leftarrow p(X, s(0)), p(C, s(D)), p(X, E)$

- C_0 est réduite ;
- C_1 n'est pas réduite car $C_1\theta_1 \subset C_1$ avec $\theta_1 = \{X/A; s(0)/B\}$;
- C_2 non plus car $C_2\theta_2 \subset C_2$ avec $\theta_2 = \{X/C; 0/D; s(0)/E\}$;
- les trois clauses sont équivalentes.

Subsomption stochastique [Sebag and Rouveirol, 1997]

- Idée : construire progressivement et aléatoirement une substitution entre les deux clauses à tester ;
- deux substitutions sont *compatibles* si elles n'assignent pas une même variable à deux objets différents ;
- au final :
  - soit on est parvenu à trouver une substitution compatible pour chaque littéral, auquel cas il y a réellement θ-subsomption,
  - soit le nombre maximal de tirages aléatoires est atteint et on conclut qu'il n'y a pas θ-subsomption (et pourtant il peut exister une substitution entre les deux clauses) ;
- *méthode d'apprentissage qui tienne le choc ?*

Subsomption stochastique [Sebag and Rouveirol, 1997]

Algorithme Ssubsomption(C, D, η)

```
1: for i = 1 to η do
2:     θ = ∅
3:     for all l in C do
4:         tirer σ un appariement de l avec un littéral de D
5:         if θ et σ sont compatibles then
6:             θ = θ ∪ σ
7:         else
8:             reprendre la boucle externe
9:         end if
10:    end for
11:    return true
12: end for
13: return false
```


Opérateurs

- Algorithme ascendant : on part d'un exemple et on généralise pour couvrir tous les positifs ;
- algorithme descendant : on parle de la clause avec un corps vide et la tête correspondant au concept cible, puis on spécialise pour écarter les négatifs ;
- opérateurs de généralisation : enlever un littéral, remplacer une constante ou un terme en variable, transformer une variable en deux, etc. ;
- opérateurs de spécialisation : ajouter un littéral, transformer une variable en constante, unifier deux variables, etc. ;

Moindre généralisé sous θ-subsomption

Il y a unicité et un algorithme polynomial [Plotkin, 1970] :

- généralisation de deux termes : on garde ce qui est commun, on remplace ce qui diffère par une nouvelle variable ;
- généralisation de deux littéraux avec même symbole de prédicat : on généralise les arguments terme à terme ;
- généralisation de deux clauses : on généralise les littéraux deux à deux (la taille du moindre généralisé est de l'ordre du produit des tailles des clauses de départ) ;
- idée clef : on garde la même variable si l'on retrouve les deux mêmes termes ;
- la clause construite est souvent non réduite ;
- l'algorithme de réduction proposé par [Plotkin, 1970] utilise des tests de θ-subsomption...

Moindre généralisé : exemple 1

le concept grand-père

- Calcul de MG $\left(\begin{array}{l}\text{grand-père(a,b)} \leftarrow \text{père(a,c)}, \text{père(c,b)}.\\ \text{grand-père(d,e)} \leftarrow \text{père(d,f)}, \text{mère(f,e)}.\end{array}\right)$
- Appariements : $\begin{array}{l|l|l}\text{grand-père(a,b)} & \text{père(a,c)} & \text{père(c,b)}\\ \text{grand-père(d,e)} & \text{père(d,f)} & \text{père(d,f)}\end{array}$
- Première étape : $\begin{array}{l|l|l}\text{grand-père(A,b)} & \text{père(A,c)} & \text{père(c,b)}\\ \text{grand-père(A,e)} & \text{père(A,f)} & \text{père(d,f)}\end{array}$
- Résultat : grand-père(A,B) $\leftarrow$ père(A,C), père(D,E).

Le résultat est une clause non réduite...

Moindre généralisé : exemple 2

maisons avec fenêtres (petites ou grandes, blanches ou noires)

m(m1)←f(m1,g,b), f(m1,p,n).
m(m2)←f(m2,g,n), f(m2,p,b).

m(m1)←f(m1,g,b),	f(m1,p,n),	f(m1,g,b),	f(m1,p,n).
m(m2)←f(m2,g,n),	f(m2,p,b),	f(m2,p,b),	f(m2,g,n).
m(M) ←f(M,g,C1),	f(M,p,C2),	f(M,T1,b),	f(M,T2,n).

Clause réduite ?

Casser la taille de $\mathcal{H}$

Étude intensive des biais

- biais de langage [Torre, 2000] :
  - bornes sur le nombre de littéraux, sur le nombre de variables existentielles, sur la profondeur des termes, sur le degré d'une variable ;
  - utilisation des variables de tête dans le corps, variables connectées ;
  - typage des arguments, contraintes sur les entrées/sorties ;
- biais de validation ;
- biais de recherche.

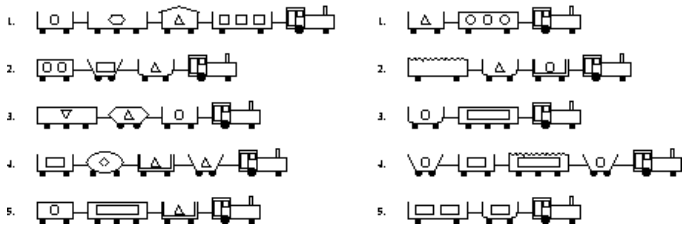
Liens de parenté

```
pere(abraham,homer).
mere(mona,homer).
pere(clancy,marge).
mere(jacqueline,marge).

grand_pere(clancy,bart).
grand_pere(abraham,bart).
grand_pere(abraham,lisa).
:- grand_pere(mona,bart).
:- grand_pere(jacqueline,lisa).
```

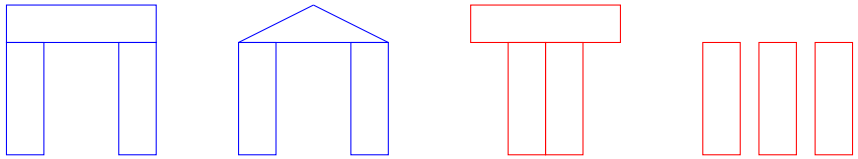
Cible : grand_pere(X,Y) ← pere(X,Z), parent(Z,Y).

Les trains de Michalski



```
eastbound(t1) :- car(t1,c1),car(t1,c2),car(t1,c3),car(t1,c4),
  infront(t1,c1),infront(c1,c2),infront(c2,c3),infront(c3,c4),
  long(c1),short(c2),long(c3),short(c4),
  shape(c1,rect),shape(c2,rect),shape(c3,rect),shape(c4,rect),
  open(c1),closed(c2),open(c3),open(c4),
  load(c1,rect,3),load(c2,triangle,1),load(c3,hexagon,1),load(c4,circle,1),
  wheels(c1,2),wheels(c2,2),wheels(c3,3),wheels(c4,2).
```

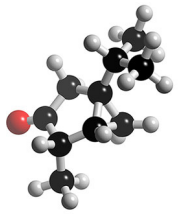
Les arches de Winston



```
arch(b11,b12,b13) :-  brick(b11), brick(b12), brick(b13),
   left_of(b12,b13), supports(b12,b11),supports(b13,b11).

:- arch(b31,b32,b33), brick(b31), brick(b32), brick(b33),
   left_of(b32,b33), supports(b32,b31), supports(b33,b31),
   touches(b32,b33), touches(b33,b32).
```


Mutagénèse [Srinivasan et al., 1994]



active(M) ← atom(M,A1,carbon,22),
atom(M,A2,carbon,10),
bond(M,A1,A2,1).

Plan du cours

Aplatissement, saturation et troncation

Démarche [Rouveirol, 1994]

- *saturation* : enrichir un exemple à l'aide d'une théorie du domaine ;
- *aplatissement* : supprimer préalablement foncteurs et constantes de l'exemple et de la théorie du domaine ;
- *troncation* : généraliser l'exemple saturé par suppression de littéraux.

Une théorie du domaine sur les entiers

```
fois_deux(0,0).
fois_deux(s(X),s(s(Y))) :- fois_deux(X,Y).

plus_trois(X,s(s(s(X)))).
```

Théorie aplatie

```
zerop(0).                                          /* new predicates */
succp(X,s(X)).

fois_deuxf(X,Y) :- zerop(X), zerop(Y).                      /* theory */
fois_deuxf(B,Z) :- succp(A,B), succp(Y,Z), succp(X,Y), fois_deuxf(A,X).

plus_troisf(A,D) :- succp(A,B), succp(B,C), succp(C,D).

                                                            /* tests */
test1(Y) :- X=0,          fois_deux(X,Y),  fois_deuxf(X,Y).
test2(Y) :- X=0,          plus_trois(X,Y), plus_troisf(X,Y).
test3(Y) :- X=s(s(0)), fois_deux(X,Y),  fois_deuxf(X,Y).
test4(Y) :- X=s(s(0)), plus_trois(X,Y), plus_troisf(X,Y).
```


Recherche de op tel que $op(1) = 5$

Exemple du concept cible :

```
op(s(0),s(s(s(s(s(0)))))).
```

Exemple aplati :

```
op(U,C) :- zerop(Z), succp(Z,U), succp(U,D), succp(D,T),
           succp(T,Q), succp(Q,C).
```


Saturation (à profondeur 4) de l'exemple

```
op(U,C) :- zerop(Z), succp(Z,U), succp(U,D), succp(D,T),
           succp(T,Q), succp(Q,C), succp(C,V6), succp(V6,V7),
           succp(V7,V8), succp(V8,V9), fois_deuxf(Z,Z),
           plus_troisf(Z,T), plus_troisf(U,Q), plus_troisf(D,C),
           fois_deuxf(U,D), plus_troisf(T,V6), fois_deuxf(D,Q),
           plus_troisf(Q,V7), fois_deuxf(T,V6), plus_troisf(C,V8).
```

solutions possibles par troncation

Dans cette clause saturée, on trouve :

- `plus_troisf(U,Q), succp(Q,C)` ;
- `fois_deuxf(U,D), fois_deuxf(D,Q), succp(Q,C)` ;
- `succp(U,D), succp(D,T), fois_deuxf(T,V6), succp(C,V6)`.

FOIL [Quinlan, 1990] : boucle externe

Caractéristiques de FOIL

- Algorithme de la même famille que DLG ;
- contre-exemples de A^- : fournis ou construits par hypothèse du monde clos.

Algorithme FOIL(A^+, A^-)

1: $P = A^+$
2: $H = \emptyset$
3: **while** $P \neq \emptyset$ **do**
4: $\quad h = \text{FOILnewRule}(P, A^-)$
5: $\quad$ ajouter h à H
6: $\quad P = \{e \in P : h \not\geq_\theta e\}$
7: **end while**
8: **return** H

FOIL [Quinlan, 1990] : construction d'une clause

Algorithme FOILnewRule(P, A^-)

1: $N = A^-$
2: $h = (\text{cible} \leftarrow \emptyset)$
3: **while** $N \neq \emptyset$ **do**
4: $\quad L = \text{FOILcandidates}(h)$
5: $\quad l = \text{argmax}_{l \in L}(\text{FOILgain}(l, h, P, N))$
6: $\quad h = h \cup l$
7: $\quad N = \{e \in N : h \succeq_\theta e\}$
8: **end while**
9: **return** h

FOIL [Quinlan, 1990] : littéraux candidats

FOILcandidates(h)

- pour chaque prédicat p d'arité n, on ajoute les littéraux de la forme $p(X_1, \ldots, X_n)$ où au moins l'une des variables X_i est déjà présente dans la clause h ;
- pour tout couple (X_i, X_j) de variables déjà existantes dans h, on ajoute les inégalités $X_i \neq X_j$.

Le résultat sera une *clause liée* !

FOIL [Quinlan, 1990] : mesure de gain

Apport de l'ajout d'un littéral l à une clause h vis-à-vis de P (exemples du concept à apprendre) et de N (contre-exemples).

FOILgain(l, h, P, N)

- $h' = h \cup l$;
- soit p (respectivement p') le nombre de substitutions possibles entre h (respectivement h') et les exemples de P ;
- soit n (respectivement n') le nombre de substitutions possibles entre h (respectivement h') et les exemples de N ;
- on considère les exemples de P couverts par h et h', soit t le nombre de substitutions entre h et ces exemples.

$$\text{FOILgain}(l, h, P, N) = t \times \left(\log_2 \frac{p'}{p' + n'} - \log_2 \frac{p}{p + n} \right)$$

Comparable aux mesures utilisées pour les arbres de décision ?

Combinaisons : discussions

Leçons de [Plotkin, 1970]

- existence d'un moindre généralisé unique sous θ-subsomption ;
- apparition possible d'hypothèses non réduites ;
- réduction et tests de subsomption coûteux.

Manipulation de clauses déterminées

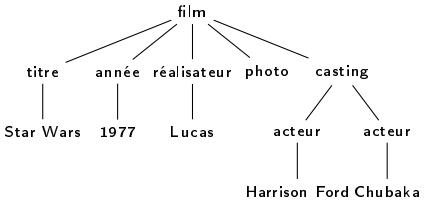
Exemple : codage déterminé d'arbres basé sur `root`, `first-child` et `next-sibling`.

Que donne un calcul de `MG` sur des exemples ainsi codés ?
Il faut en tout cas choisir un codage approprié...

Un codage ILP pour la classification d'arbres

Proposition de Jean Decoster

- coder les arbres à l'aide de clauses déterminées pour casser la complexité du test de θ-subsomption ;
- utiliser le calcul de moindre-généralisé de [Plotkin, 1970] ;
- intégration dans volata.



ciné(sw) ← p_1(film), $p_{1.1}$(titre), $p_{1.2}$(année), $p_{1.3}$(réalisateur), $p_{1.4}$(photo), $p_{1.5}$(casting), $p_{1.5.1}$(acteur), $p_{1.5.2}$(acteur).

Bibliographie I

Gottlob, G. (1987).
Subsumption and implication.
Information Processing Letters, 24(2) :109–111.

Kapur, D. and Narendran, P. (1986).
Np-completeness of the set unification and matching problems.
In *Proceedings of 8th Conference on Automated Deduction*, volume 230, pages 489–495. Springer-Verlag.

Kietz, J.-U. and Lübbe, M. (1994).
An efficient subsumption algorithm for inductive logic programming.
In Cohen, W. W. and Hirsh, H., editors, *Proceedings 11th International Conference on Machine Learning*, pages 130–138. Morgan Kaufmann.

Bibliographie II

Lloyd, J. W. (1987).
Foundations of Logic Programming.
Springer, Berlin, 2 edition.

Maloberti, J. and Sebag, M. (2004).
Fast theta-subsumption with constraint satisfaction algorithms.
Machine Learning, 55(2) :137–174.

Marcinkowski, J. and Pacholski, L. (1992).
Undecidability of the horn-clause implication problem.
In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 354–362, Pittsburgh, PN. IEEE Computer Society Press.

Bibliographie III

Muggleton, S. (1991).
Inductive logic programming.
New Generation Computing Journal, 8(4) :295–317.

Muggleton, S. and De Raedt, L. (1994).
Inductive logic programming : Theory and methods.
Journal of Logic Programming, 19 :629–679.

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

Bibliographie IV

Nienhuys-Cheng, S. and de Wolf, R. (1996).
Least generalizations and greatest specializations of sets of clauses.
Journal of Artificial Intelligence Research, 4 :341–363.

Plotkin, G. (1970).
A note on inductive generalization.
In Meltzer, B. and Mitchie, D., editors, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press.

Quinlan, J. R. (1990).
Learning logical definitions from relations.
Mach. Learn., 5(3) :239–266.

Bibliographie V

Rouveirol, C. (1994).
Flattening and saturation : Two representation changes for generalization.
Machine Learning, 14(1) :219–232.

Scheffer, T., Herbrich, R., and Wysotzki, F. (1996).
Efficient θ-subsumption based on graph algorithms.
In Muggleton, S., editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 312–329. Stockholm University, Royal Institute of Technology.

Schmidt-Schauß, M. (1988).
Implication of clauses is undecidable.
TCS : Theoretical Computer Science, 59(3) :287–296.

Bibliographie VI

Sebag, M. and Rouveirol, C. (1997).
Tractable induction and classification in FOL via stochastic matching.
In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892.

Srinivasan, A., Muggleton, S., King, R. D., and Sternberg, M. J. E. (1994).
Mutagenesis : ILP experiments in a non-determinate biological domain.
In Wrobel, S., editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

Bibliographie VII

Torre, F. (2000).
Intégration des biais de langage à l'algorithme générer-et-tester - Contributions à l'apprentissage disjonctif.
PhD thesis, Laboratoire de Recherche en Informatique Université Paris-Sud France.