Programmation logique

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Programmation logique / Logique des prédicats

Sommaire

Programmation logique / Logique des prédicats

Introduction

Basée sur le principe de la démonstration automatique de théorèmes(utilisant la logique du premier ordre)

Un programme en Logique = ensemble d'énoncés.

Les énoncés = formules du calcul du premier ordre

Tout problème calculable peut être formulé dans ce langage.

Programmation logique / Logique des prédicats

Introduction

Programmer en logique consiste :

à définir les hypothèses (énoncés definis dans un programme exprimant la connaissance relative au problème à résoudre).

à introduire la conclusion (poser le problème).

L'interpréteur tente ensuite de réaliser la preuve du but en utilisant l'*inférence logique*.

Le langage le plus représentatif est PROLOG(1971) basé sur deux mécanismes : *unification* et *résolution*.

Programmation logique / Logique des prédicats

Syntaxe : Éléments de base

Constantes : a, b, c,...

Variables : x, y, z,...

Fonctions : f, g, h, ...

Relations(prédicats ou fonctions booléennes): P, Q, R

Syntaxe : définition d'un terme

Toute constante est un terme.
Toute variable est un terme.
Si t1, t2, ..tn sont des termes et si f est une fonction n-aire, alors f(t1, t2, ..,tn) est un terme.

Syntaxe : définition d'un atome

Si t1, t2, ..., tn sont des termes et si P est un prédicat n-aires alors P(t1, t2, .., tn) est un atome.

Programmation logique / Logique des prédicats

Syntaxe : définition d'une formule

. Tout atome est une formule.

. Si P et Q sont deux formules alors :

Non P,

P & Q,

P V Q,

P ==> Q,

P <==> Q,

$\forall$ x P(x),

$\exists$x P(x)

sont des formules

Syntaxe : exemples

$\forall$x NOMBRE(x) ==> $\exists$y PLUSGRANDQUE(y, x)

$\forall$x, $\exists$y (P(x) & Q(y)) ==> (R(a) V Q(b))

On notera FL1 une formule logique du premier ordre

Programmation logique / Logique des prédicats

Sémantique

Interprétation : attribuer des valeurs à chaque terme et à chaque symbole de prédicat dans une FL1

Formules équivalentes : même interprétation.

Exemple : Evaluation

E : $\forall$ x ((A(a, x) V B(f(x)) & C(x)) ==> D(x)

A, B, C et D sont des atomes.

Soit l'interprétation suivante de domaine D = {1, 2}

a = 2

f(1) = 2 ; f(2) = 1

A(2, 1) = vrai ; A(2, 2) = faux

B(1) = vrai ; B(2) = faux

C(1) = vrai ; C(2) = faux

D(1) = faux ; D(2) = vrai

Cas x = 1

A(2, 1) vrai et B(2) faux donc A(2, 1) V B(2) vrai

C(1) vrai donc (A(2, 1) V B(2)) & C(1) vrai

D(1) faux l'implication est donc fausse

Cas x = 2

On montre de la même façon que E est vraie

Puisque E n'est pas vraie pour tout x, l'expression est alors fausse.

Programmation logique / Logique des prédicats

F, G et H sont des FL1 ne contenant pas de variable
F(x) et G(x) sont des FL1 contenant la variable x.

Propriétés

L'évaluation de FL1 complexes peut être facilitée par une phase de substitution (transformation syntaxique)

On utilisera alors des propriétés.

Double négation :

NON (NON F) = F

Commutativité :

F & G = G & F F V G = G V F

Associativité :

F & (G & H) = (F & G) & H

F V (G V H) = (F V G) V H

Distributivité

F V(G & H) = (F V G) & (F V H)

F &(G V H) = (F & G) V (F & H)

De Morgan

NON (F & G) = NON F V NON G NON (F V G) = NON F & NON G

F ==> G = NON F V G

F <==> G = (NON F V G)&(NON G V F)

NON (∀x F(x)) = ∃x NON F(x)

NON (∃x F(x)) = ∀x NON F(x)

∀x F(x) & ∀x G(x) = ∀x (F(x) & G(x))

∃x F(x) V ∃x G(x) = ∃x (F(x) V G(x))

Programmation logique / Logique des prédicats

Inconsistance et validité d'une formule

FL1 inconsistante (insatisfiable) : fausse pour toute interprétation.

FL1 consistance (satisfiable): Il existe une interpretation pour laquelle elle est vraie

FL1 valide : vraie pour toute interprétation.

FL1 non valide : fausse pour une interpétation

Q est une conséquence logique de P1, P2, ...Pn] <==>

[Si P1 & p2 ...& Pn est vraie pour toute interprétation, alors Q est aussi vraie].

Exemples :

P V NON P : valide (vraie pour toute interprétation) : Tautologie

P & NON P : inconsistante (fausse pour toute interprétation) : Antilogie

Variables liées et libres

Dans l'expression

$\forall x (P(x) ==> Q(x, y))$

x est une variable liée et y est une variable libre.

Une expression ne peut être évaluée que lorsque toutes les variables sont liées.

Aussi, nous exigeons que toutes les FL1 ne contiennent que les variables liées.

Forme normale conjonctive/disjonctive

Soient F1, F2, ...des formules :

Forme normale conjonctive : F1 & F2 &.....Fn avec Fi forme disjonctive

Exemple : (NON P V Q V R) & (NON P V NON Q) & NON R est une forme conjonctive.

Forme normale disjonctive : F1 V F2 V.....Fn avec Fi forme conjonctive

Exemple: (P & Q & R) V (Q & R) V P est une forme disjonctive.

Toute formule peut être transformée en une forme normale disjonctive ou conjonctive.

Programmation logique / Logique des prédicats

Règles d'inférence

Fournissent un moyen de faire des démonstrations (déductions logiques)

Problème : étant donné un ensemble d'énoncés {s1, s2, ...sn} (les prémices), prouver la vérité de s (conclusion).

L'utilisation de la table de vérité est un moyen de démonstration(sémantique) : processus très lent.

Règles d'inférences : procède par transformations sémantiques (simplifie la démonstration)

Programmation logique / Logique des prédicats

Règles-clés utilisées dans les démonstrateurs

P
P-->Q
---------Modus Ponens
Q

Non Q
P ==> Q
---------Modus Tollens
Non P

P-->Q
Q-->R
--------- Chaînage
P-->R

Non P V Q
Non Q V R
---------- Chaînage
Non P V R

P & Q
----------Substitution
P

P
Q
----------Conjonction
P & Q

P --> Q
---------- Contraposée
Non Q --> Non P

Programmation logique / Démonstrateur de théorèmes

Substitution

C'est un ensemble {ti/xi} où xi sont des variables et ti sont des termes qui peuvent être des constantes, des variables ou des fonctions.

Ca revient à remplacer toutes les occurrences des variables xi par ti.

Soit la substitution S = {a/x , g(b)/y } et soit la clause C = P(x, y) V Q(x, f(y))

C'=C.S=P(a,g(b)) V Q(a, f(g(b)))

Programmation logique / Démonstrateur de théorèmes

Unification

Deux termes t1 et t2 sont unifiables s'il existe une substitution S telle que S.t1 = S.t2

Un ensemble de termes {t1, t2, ...tn} est unifiable s'il existe une substitution S telle que S.t1 = S.t2 = ... = S.tn

Exemple

t1 = f(a, g(b))

t2 = f(x, y)

t1 et t2 sont unifiable car il existe S = {a/x , g(b)/y } telle que S.t1 = S.t2 = f(a, g(b)).

Programmation logique / Démonstrateur de théorèmes

Unification : exemples

Prédicats parents p(5) et p(5)
Unificateur : μ = {}
Instance commune p(5)

Prédicats parents p(x) et p(5)
Unificateur : μ = { x:=5}
Instance commune p(5)

Prédicats parents p(x) et p(y)
Unificateur : μ = {x:=y}
Instance commune p(y)

Prédicats parents p(x, x) et p(5, y)
Unificateur : μ = {x:=5; y:=5}
Instance commune p(5, 5)

Prédicats parents p(f(x), f(5), x) et p(z, f(y), y)
Unificateur : μ = {z:= f(5); y:=5; x:=5)}
Instance commune p(f(5), f(5), 5)

Programmation logique / Démonstrateur de théorèmes

Unification : Couple de désaccord

Le couple de désaccord D(t1, t2) entre deux termes t1 et t2 est le couple le plus à gauche et ne débutant pas par un symbole identique.

Exemple :

t1 = f(g(u), z)

t2 = f(g(h(t)), r(a))

--> D(t1, t2) =(u, h(t))

Programmation logique / Démonstrateur de théorèmes

Unification : algorithme

Soient deux termes t1 et t2.
L'algorithme qui suit cherche l'unificateur μ s'il existe.

1. $S_0 = \{\}$; k := 0
2. SI $S_k.t1 = S_k.t2$: $\mu := S_k$; fin avec succès
3. Calculer $D(S_k.t1, S_k,t2)$. Soit (d1, d2)

   SI d1 est une variable :

   $S_{k+1} = \{ d2/d1\} \cup S_k$

   Allera 2

   SI d2 est une variable

   $S_{k+1} = \{d1/d2\} \cup S_k$

   Allera 2

   Sinon "Echec"

Exemple

t1 = f(g(u), z)
t2 = f(g(h(t)), r(a))

S0 = {}
D(t1, t2) = (u,h(t))
S1 = S0 U { h(t)/u } = {h(t)/u}
D(S1.t1, S1.t2) = (z, r(a))
S2 = S1 U { r(a)/z } = { h(t)/u, r(a)/z}
S2 est l'unificateur.

Remarques

Deux termes ne sont pas toujours unifiables.
Exemple f(a), f(b).

Deux termes peuvent avoir une infinité d'unificateurs, mais il existe un qui est le plus général(donné par l'algorithme).

x et f(x) donne une boucle infinie.

Programmation logique / Démonstrateur de théorèmes

Principe de résolution (Robinson 65)

C'est une procédure d'inférence syntaxique qui, appliquée à un ensemble de clauses, détermine si l'ensemble est inconsistant

(il n y a pas d'interprétation pour laquelle l'expression est vraie).

Cette procédure est similaire au processus de démonstration par l'absurde.

Programmation logique /

Démonstrateur de théorèmes

Formules utilisées :
p-->q = non p V q
non (p V q) = non p & non q

Principe de résolution (Robinson 65)

Supposons que nous voulons démontrer { C1, C2, ...Cn} --> D ou encore

C1 & C2 ... & Cn --> D

Cà revient à démontrer l'inconsistance de C1 & C2 ....& Cn & non D

En effet :

Démontrer que (C1 & C2 .... & Cn --> D) est valide revient à démontrer que sa négation est inconsistante c'est à dire

non(C1 & C2 & .... & Cn --> D) est inconsistante.

= non(non(C1 & C2 .....Cn)V D))

= C1 & C2 ..... & Cn & non D.

Programmation logique / Démonstrateur de théorèmes

Principe de résolution (Robinson 65)

En utilisant

- la résolution (définie après) et
- l'unification

nous pouvons montrer que l'ensemble { C1, C2, ...Cn, Non D) est inconsistant (par la déduction d'une contradiction).

Si l'ensemble des clauses est inconsistant, la résolution et l'unification génère toujours la clause vide(Complétude ou semi-décidabilité de la logique du premier ordre).

Programmation logique / Démonstrateur de théorèmes

Détermination des résolvantes

Soient deux clauses

C1 = A1 V A2 ..... V An

C2 = B1 V B2..... V Bm

S'il existe i et j tels que Ai et Non Bj sont unifiables par S, alors on obtient une nouvelle clause appelée résolvante et est donnée par

S(A1 V .... V A_{i-1} V A_{i+1} .....An V B1 V ....V B_{j-1} V B_{j+1} V ....Bm)

La résolvante est une conséquence logique de C1 et C2.

Démonstration

C1 de la forme X V Ai (Commutativité du V)

C2 de la forme Bj V Y (Commutativité du V)

S.C1 = S.X V S.Ai (1)

S.C2 = S.Bj V S.Y (2)

de(1) : Non S.X ==> S.Ai

de(2) : Non S.Bj ==> S.Y

Comme S.Ai = S.Non Bj (S unificateur), par transitivité on : Non S.X ==> S.Y

ou : S.X V S.Y

ou encore S.(X V Y)

Programmation logique / Démonstrateur de théorèmes

Algorithme général de résolution

Soit à démontrer C1, C2, ..., Cn --> But

C1, C2, ...Cn et but sont des clauses.

1. A_0 ={C1, C2, ...Cn}U{ Non but}

2. A_{i+1} = A_i U { Toutes les résolvantes de clauses prises dans A_i}

3. SI A_{i+1} contient la clause vide : Arrêt, But est une conséquence logique de C1,..Cn

4. Si Ai+1 = Ai :Arrêt, But n'est pas une conséquence logique de C1,..Cn

5. Allera 2

Résolution dans un cas particulier

Soient deux clauses C1 et C2 avec aucune variable en commun.

S'il existe un littéral L1 dans C1 qui est un complément d'un littéral L2 dans C2, L1 et L2 sont supprimés et la disjonction de C1 et C2 est formée à partir des restes des clauses. La nouvelle clause est appelée résolvante de C1 et C2.

Exemple : la résolution des deux clauses (Non p) V q et (Non q) V r donne (Non p) V r

En effet :

(Non p) V q c'est $p \rightarrow q$

(Non q) V r c'est $q \rightarrow r$

Par transitivité $p \rightarrow r$ qui est (Non p) V r

L'unificateur est {}

Programmation logique / Démonstrateur de théorèmes

Résolution dans un cas particulier : Exemple

Prouver A1 & A2 avec les hypothèses suivantes :

C1 : Non B1 V Non B2 V A1

C2 : B1

C3 : Non B1 V Non A2 V B2

C4 : A2

Il s'agit donc de réfuter l'ensemble

A0 = { C0, C1, C2, C3, C4}

avec

C0 : Non A1 V Non A2

Construction de A1 :

de C0 et C1 on déduit : C5 = Non B1 V Non B2 V Non A2

de C0 et C4 on déduit : C6 = Non A1

de C1 et C2 on déduit : C7 = Non B2 V A1

de C1 et C3 on déduit : C8 = Non B1 V Non A2 V A1

de C2 et C3 on déduit : C9 = Non A2 V B2

de C3 et C4 on déduit : C10 = Non B1 V B2

A1 = { C1, C2, C3, C4, C5, C6, C7, C8, C9, C10}

Programmation logique /

Démonstrateur de théorèmes

C0: Non A1 V Non A2
C1: Non B1 V Non B2 V A1
C2: B1
C3: Non B1 V Non A2 V B2
C4: A2
C5 = Non B1 V Non B2 V Non A2
C6 = Non A1
C7 = Non B2 V A1
C8 = Non B1 V Non A2 V A1
C9 = Non A2 V B2
C10 = Non B1 V B2

Résolution dans un cas particulier : Exemple

Construction de A2 :

De C0 et C7 on déduit : C11 = Non B2 V Non A2

De C0 et C8 on déduit : C12 = Non B1 V Non A2

De C1 et C6 on déduit : C13 = Non B1 V Non B2

De C1 et C9 on déduit : C14 = Non B1 V Non A2 V A1

De C1 et C10 on déduit : C15 = Non B1 V A1

Ect...

A la prochaine étape on aura

C2 et C12 ont comme résolvante s = Non A2.

Ce qui mène vers une clause vide en la combinant avec C4.

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog : clauses de Horn

Dans Prolog, les énoncés sont restreints à deux types de formules(Logique des clauses de HORN) : assertion et implication.

- Assertion : un atome(littéral positif)
- Implication : A <-- B1 & B2 & ... &Bn

A désigne le conséquent et B1 & B2 &.. l'antécédent.

Cette dernière écriture est équivalente à : (Non B1) V (Non B2) V .... (Non Bn) V A.

Une clause de Horn est une clause qui contient au plus un littéral positif.

Prolog utilise un type particulier de résolution appelée "résolution descendante"

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog (Les différents cas que l'on peut avoir)

(a)

Dénégation : Non A

Implication : A <-- B

Résolvante : Non B

(b)

Dénégation : Non A

Implication : A <--

Résolvante : s = clause vide.

On retrouve les résultats précédents en mettant les implications sous forme disjonctive.

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog (Les différents cas que l'on peut avoir)

(c)

Dénégation :

Non (A1, A2, ..., An)

Implication :

Ak <-- B1, B2, ...,Bm

Résolvante :

s = Non (A1, A2, ..., Ak-1, B1, B2, ...,Bm, Ak+1, ... An)

(d)

Dénégation :

Non (A1, A2, ..., An)

Implication :

Ak <--

Résolvante :

s = Non (A1, A2, ..., Ak-1, Ak+1, ... An)

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog (Pas d'inférence dans le cas de la résolution descendante)

1. S'assurer qu'il n'existe pas de variables communes dans la dénégation D et dans l'implication S. Un changement de variable peut être fait.

2. Choisir dans D, un Ai qui est conséquent de S. Si ceci est possible, continuer.

3. Déterminer l'unificateur le plus général μ de Ai et du conséquent de S. Si ceci est possible, continuer.

4. Remplacer dans D, Ai par les antécédents de S. Si S est une assertion, effacer tout simplement Ai de D. On obtient ainsi une nouvelle dénégation.

5. appliquer μ à la dénégation tout entière.

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog : Exemple sans variable

Prouver A1, A2. (, désigne le "et logique")

avec comme implications :

f0 : A1 <-- B1, B2.

f1 : B1 <--

f2 : B2 <-- B1, A2

f3 : A2 <--

Il suffit donc de réfuter Non(A1 & A2)

Non(A1 et A2)	
Non(B1 et B2 et A2)	f0
Non(B2 et A2)	f1
Non(B1 et A2 et A2)	f2
Non(B1 et A2)	simplification
Non(A2)	f1
Faux	f3

Noter l'utilisation du Backtracking.

Programmation logique / Démonstrateur de théorèmes

Résolution dans Prolog :
Exemple avec variables

Prouver Inf(2, 3) avec

S0: Inf(0, Y) <-- Y > 0

S1: Inf(X, Y) <-- X <> 0, Inf(X-1, Y-1)

Il faut donc réfuter D = Non(Inf(2, 3)).
Inf(2, 3) s'unifie avec Inf(X, Y) (application de S1)
μ = { X:=2 ; Y :=3 }
D devient Non (2<>0, Inf(1, 2))

Inf(1, 2) s'unifie avec Inf(X, Y) (application de S1)
μ = { X:=1 ; Y :=2 }
D devient Non (2<>0, 1<>0, Inf(0, 1))

Inf(0, 1) s'unifie avec Inf(0, Y) (application de S0)
μ = { Y :=1 }
D devient Non (2<>0, 1<>0, 1>0)

D'ou l'inconsistance de D.