Logică și structuri discrete

Logica predicatelor

Marius Minea
marius@cs.upt.ro

http://www.cs.upt.ro/~marius/curs/lsd/

21 noiembrie 2016

Logica apare în programare!

… pentru că orice program trebuie *gândit*

În logică, putem exprima *declarativ* proprietăți în structuri de date.

Un dicționar conține două valori egale? E o permutare?

```
module M = Map.Make(String)
let is_inj m = M.for_all (fun k v ->
    not (M.exists (fun k2 v2 -> k2 <> k && v2 = v) m)) m
let is_perm m =
    M.for_all (fun _ v -> M.exists (fun k _ -> k = v) m) m

let im = M.(singleton "a" "1" |> add "b" "2" |> add "c" "1")
let pm = M.(singleton "a" "b" |> add "b" "c" |> add "c" "a")

let res1 = is_inj im (* false *)
let res2 = is_perm pm (* true *)
```

Funcțiile `for_all` și `exists` pe colecții *fac ele parcurgerea*, trebuie doar să indicăm proprietatea dorită

Cât de generală e o demonstrație?

$\forall x(boy(x) \vee girl(x) \to child(x))$

$\quad \neg boy(x) \vee \neg girl(x) \vee child(x)$

...

$\neg\exists x(child(x) \wedge getstrain(x)) \to \neg\exists x(boy(x) \wedge good(x))$

negata: $\quad (\neg child(y) \vee \neg getstrain(y)) \wedge boy(c) \wedge good(c)$

Demonstrăm prin *rezoluție*: $\dfrac{\neg boy(x) \vee A \qquad boy(c) \vee B}{A[x \mapsto c] \vee B}$

Demonstrația e făcută *fără* a ține cont (sau înțelege) *semnificația* predicatelor *boy*, *child*, *good*, etc.

$\Rightarrow$ puteau fi $P(x), Q(x), R(x), \ldots$

Demonstrația e bună pentru *orice predicate* care satisfac ipotezele.

La ce e bună o demonstrație generală?

Exemplu de teoremă:
O *relație de echivalență* definește o *partiție* a mulțimii de definiție.

$\forall x\, R(x,x)$	reflexivitate
$\wedge \forall x \forall y (R(x,y) \rightarrow R(y,x))$	simetrie
$\wedge \forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$	tranzitivitate
$\rightarrow \forall x \forall y (\neg \exists z (R(x,z) \wedge R(y,z))$	clase disjuncte
$\vee \forall z (R(x,z) \leftrightarrow R(y,z)))$	sau clase identice

Teorema e adevărată indiferent de *semnificația* atribuită lui $R(x,y)$ (indiferent de *interpretare*). R ar putea fi (printre altele):

$R(x,y) \stackrel{def}{=} x \equiv y \mod d$ (același rest la împărțirea cu d)

$R(x,y) \stackrel{def}{=} length(x) = length(y)$ liste/șiruri de aceeași lungime

Definim mai precis ce înseamnă o *demonstrație*.

Recapitulăm: sintaxa logicii predicatelor

Formulele logicii predicatelor sunt definite *structural recursiv*:

Termenii

variabilă v sau constantă c

$f(t_1, \cdots, t_n)$ cu f funcție n-ară și $t_1, \cdots, t_n$ *termeni*

Formule (well-formed formulas, formule bine formate):

$P(t_1, \cdots, t_n)$	cu P predicat n-ar; $t_1, \cdots, t_n$ termeni
$\neg\alpha$	unde α este o formulă
$\alpha \to \beta$	unde α, β sunt formule
$\forall v\, \alpha$	cu v variabilă, α formulă: *cuantificare universală*
$t_1 = t_2$	cu t_1, t_2 termeni (în logica de ord. I cu egalitate)

Față de logica propozițională, în loc de propoziții avem *predicate* (peste *termeni*).

Logica se numește *de ordinul I*, deoarece *cuantificatorii logici* se pot aplica doar variabilelor.

Sintaxă și semantică

Ca în logica propozițională (și pentru orice limbaj), deosebim

sintaxa = *forma, regulile* după care construim ceva (aici, formule)

semantica = *înțelesul* construcțiilor de limbaj

La fel ca în logic propozițională, lucrăm cu

deducția (demonstrația): procedeu pur sintactic

implicația / consecința logică (consecința semantică): *interpretăm* formula (*înțelesul* ei, valoarea de adevăr)

Ne interesează corespondența dintre aceste două aspecte.

Axiomele calculului predicatelor

Definim: putem *substitui* variabila x cu termenul t în $\forall y \varphi$ dacă:

x nu apare liber în φ (substituția nu are efect) sau

x se poate substitui cu t în φ și y nu apare în t

(nu putem substitui variabile legate)

A1: $\alpha \to (\beta \to \alpha)$ (A1-A3 din logica propozițională)

A2: $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

A3: $(\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$

A4: $\forall x(\alpha \to \beta) \to (\forall x\alpha \to \forall x\beta)$

A5: $\forall x\alpha \to \alpha[x \leftarrow t]$ dacă x poate fi substituit cu t în α

A6: $\alpha \to \forall x\alpha$ dacă x nu apare liber în α

În logica cu egalitate, adăugăm și

A7: $x = x$

A8: $x = y \to \alpha = \beta$

unde β se obține din α înlocuind oricâte din aparițiile lui x cu y.

Regula de inferență: e suficient *modus ponens*: $\dfrac{A \qquad A \to B}{B}$

Deducție

Fie H o mulțime de formule. O *deducție* (demonstrație) din H e un șir de formule $A_1, A_2, \cdots, A_n$, astfel ca $\forall i \in \overline{1, n}$

1. A_i este o *axiomă*, sau
2. A_i este o *ipoteză* (o formulă din H), sau
3. A_i rezultă prin *modus ponens* din A_j, A_k anterioare $(j, k < i)$

Spunem că A_n *rezultă* din H (e *deductibil*, e o *consecință*). Notăm:

$$H \vdash A_n$$

Alte reguli de inferență

$$\frac{\forall x\ \varphi(x)}{\varphi(c)}$$ *instanțiere universală* (*vezi A*5)

unde c e o constantă arbitrară (nu apare anterior în demonstrație)

Dacă φ e valabil pentru orice x, atunci și pentru o valoare arbitrară c.

$$\frac{\varphi(c)}{\forall x\ \varphi(x)}$$ *generalizare universală* (*vezi A*6)

unde c e o valoare arbitrară (nu apare în ipoteze)

Dacă φ e valabilă pentru o valoare arbitrară, e valabilă pentru orice x.

$$\frac{\exists x\ \varphi(x)}{\varphi(c)}$$ *instanțiere existențială*

Dacă există o valoare cu proprietatea φ, o instanțiem (cu un nume nou).

$$\frac{\varphi(c)}{\exists x\ \varphi(x)}$$ *generalizare existențială*

Dacă φ e adevărată pentru o anumită valoare, există o valoare care o face adevărată.

Cum interpretăm o formulă ?

Intuitiv, găsim un *înțeles* pentru fiecare simbol din formulă:

O *interpretare* (*structură*) I în logica predicatelor constă din:

- o mulțime nevidă U numită *universul* sau *domeniul* lui I
  (mulțimea valorilor pe care le pot lua variabilele)
- pentru orice simbol de constantă c, o valoare $c_I \in U$
- pentru orice simbol de funcție n-ară f, o funcție $f_I : U^n \to U$
- pentru orice simbol de predicat n-ar P, o submulțime $P_I \subseteq U^n$.

(o *relație* n-ară pe U)

Deci, dăm o *interpretare* fiecărui simbol din formulă.

O interpretare *nu* dă valori variabilelor (vezi ulterior: atribuire).

$\forall x \forall y \forall z . P(x, y) \wedge P(y, z) \to P(x, z)$ tranzitivitate

De exemplu: universul U = numere reale; predicatul P: relația $\leq$

$\exists e \forall x \neg A(x, e)$ existența mulțimii vide. $A(x, y)$ e $x \in y$

$\forall x . \forall y . P(x, y) \to \exists z . P(x, z) \wedge P(z, y)$

Găsiți o interpretare în care e adevărată și una în care e falsă ?

Interpretări, atribuiri, valori de adevăr

Fie I o *interpretare* cu univers U
și fie V mulțimea tuturor simbolurilor de variabile.

O *atribuire* este o funcție $s : V \to U$
(dă fiecarei *variabile libere* o *valoare* din *univers*)
$\Rightarrow$ din atribuirea s se poate obține valoarea pentru orice *termen*
(știm valoarea fiecărei variabile și înțelesul fiecărei funcții)

Interpretarea I dă și înțelesul fiecărui *predicat*
$\Rightarrow$ putem calcula *valoarea de adevăr* a unei formule
$\neg$, $\to$ etc. au înțelesul cunoscut din logica propozițională
trebuie definit înțelesul (semantica) lui $\forall$

Spunem că $\forall x \varphi$ e adevărată în interpretarea I cu atribuirea s dacă φ e adevărată înlocuind x cu *orice* valoare $d \in U$ din univers.

Modele și tautologii

Un *model* pentru o formulă φ e o interpretare în care formula e adevărată *pentru orice atribuire* a variabilelor.
Spunem că φ e *adevărată* în interpretarea I, și notăm $I \models \varphi$

Obs: Dacă o formulă nu are variabile libere, valoarea ei de adevăr depinde doar de interpretare, nu și de vreo atribuire.

Def: O *tautologie* e o formulă adevărată în *orice* interpretare.

Spre deosebire de logica propozițională, în logica predicatelor, numărul interpretărilor e *infinit*
$\Rightarrow$ nu mai putem construi exhaustiv tabelul de adevăr.

E *esențial* deci să putem *demonstra* o formulă
(pornind de la axiome și reguli de inferență)

Implicația logică (consecința semantică)

Fie H o mulțime de formule și C o formulă.
Notăm $I \models H$ dacă I e un model pentru fiecare formulă din H.

Spunem că H *implică* C ($H \models C$) dacă pentru orice interpretare I,

$$I \models H \text{ implică } I \models C$$

(C e adev. în orice interpretare care satisface toate ipotezele din H)

Consistență și completitudine

La fel ca în logica propozițională
deducția (demonstrația) se face pur sintactic
consecința/implicația logică e o noțiune semantică,
considerând *interpretări* și valori de adevăr.

Calculul predicatelor de ordinul I este *consistent* și *complet*
(la fel ca și logica propozițională):

$$H \vdash C \text{ dacă și numai dacă } H \models C$$

Concluzia C se poate *deduce* (demonstra) $\vdash$ din ipotezele H dacă și numai dacă ea e o *consecință semantică* $\models$ a ipotezelor H
(e adevărată în orice *interpretare* care satisface toate ipotezele)

Dar: relația de implicație logică e doar *semidecidabilă*
dacă o formulă e o tautologie, ea poate fi demonstrată
dar dacă nu e, încercarea de a o demonstra (sau o refuta) poate continua la nesfârșit

Programare logică: Prolog

Programatorul specifică *declarativ* ce se știe despre problemă.
Interpretorul *găsește soluțiile* căutând demonstrații.

```
fiu(ion, petre).
fiu(george, ion).
fiu(radu, ion).
fiu(petre, vasile).
desc(X, Y) :- fiu(X, Y).
desc(X, Z) :- fiu(X, Y), desc(Y, Z).
```

fapte (predicate adevărate)

reguli: `:-` e implicație $\leftarrow$
scrisă: stânga *dacă* dreapta
virgula e conjuncție $\wedge$

X e descendentul lui Y dacă X e fiul lui Y.

Variabilele din *stânga* sunt cuantificate *universal* (în ambele părți)
Variabilele care apar doar în *dreapta* sunt cuantificate *existențial*.

X e descendent al lui Z dacă *există* Y astfel încât X e fiul lui Y
și Y e descendent al lui Z

Regula 2: $\forall X \forall Z . desc(X, Z) \leftarrow \exists Y . fiu(X, Y) \wedge desc(Y, Z)$
$\forall X \forall Z . desc(X, Z) \vee \forall Y \neg(fiu(X, Y) \wedge desc(Y, Z))$

Eliminând cuantificatorii universali rezultă:

$$desc(X, Z) \vee \neg fiu(X, Y) \vee \neg desc(Y, Z)$$

Rezoluția în Prolog

Fie ca întrebare/scop/țintă (engl. *goal*) `desc(X, vasile).`
O *soluție* = o valoare pentru X care face predicatul adevărat.
O formulă e *realizabilă* dacă *negația* e o *contradicție*.

Scriem negația întrebării: $\neg$ `desc(X, vasile).`
adică: `desc(X, vasile)` e *fals* pentru orice `X`.
Încercăm să derivăm o contradicție cu ipotezele folosind *rezoluția*.

Alegem pentru unificare prima regulă (redenumind cu variabile noi):
`desc(X1, Y1)` $\vee$ $\neg$ `fiu(X1, Y1).`
Obținem ca rezolvent $\neg$ `fiu(X, vasile).` `X1=X, Y1=vasile`

Se poate unifica cu faptul nr. 4 `fiu(petre, vasile).`
Obținem ca rezolvent clauza vidă (contradicție) `X=petre`

Deci, `desc(X, vasile)` *NU e fals* pentru orice `X`.
`desc(petre, vasile)` e *adevărat*. `X=petre` e o *soluție*.

Găsim în continuare și alte soluții...

Exemplu Prolog (cont.)

Pornim tot cu negația întrebării: $\neg$`desc(X, vasile)`.

Unificăm cu regula 2 (redenumind din nou variabilele):
`desc(X2, Z2)` $\vee \neg$ `fiu(X2, Y2)` $\vee \neg$ `desc(Y2, Z2)`
Obținem: $\neg$ `fiu(X, Y2)` $\vee \neg$ `desc(Y2, vasile)` `X2=X`, `Z2=vasile`

Unificăm cu `fiu(ion, petre)` (nr. 1) `X=ion`, `Y2=petre`
Obținem rezolventul $\neg$ `desc(petre, vasile)`.

Am obtinut înainte `desc(petre, vasile)` $\Rightarrow$ derivăm clauza vidă.
$\Rightarrow$ `X=ion` e încă o soluție pentru întrebarea inițială.

Dacă întrebarea are variabile, interpretorul Prolog va genera toate soluțiile posibile (toate unificările/substituțiile pentru variabile).

Altfel, determină dacă predicatul dat (fără variabile) e adevărat.

Exemplu Prolog: inversarea listelor

Noțiunile recursive se scriu similar în logică și programare funcțională.
Pentru liste folosim constanta `nil` (lista vidă) și constructorul `cons`.
Inversarea devine *predicat* cu 3 argumente: lista, acumulator, rezultat.

```
rev3(nil, R, R).
rev3(cons(H, T), Ac, R) :- rev3(T, cons(H, Ac), R).
rev(L, R) :- rev3(L, nil, R)
```

Când lista e vidă, rezultatul e acumulatorul.
Altfel, adăugând capul listei la acumulator, obținem același rezultat.
Inversarea se obține luând acumulatorul (auxiliar) lista vidă.

Dând ca întrebare $rev(c(1, c(2, c(3, nil)))), X)$ obținem
$X = c(3, c(2, c(1, nil)))$, derivarea (raționamentul) fiind:

$rev(c(1, c(2, c(3, nil))), X)$	`L1=c(1,c(2,c(3,nil)))`, `R1=X`
$\leftarrow rev3(c(1, c(2, c(3, nil))), nil, X)$	`H1=1`, `T1=c(2,c(3,nil))`, `Ac1=nil`
$\leftarrow rev3(c(2, c(3, nil)), c(1, nil), X)$	`H2=2`, `T2=c(3,nil)`, `Ac2=c(1,nil)`
$\leftarrow rev3(c(3, nil), c(2, c(1, nil)), X)$	`H3=3`, `T3=nil`, `Ac3=c(2,c(1,nil))`
$\leftarrow rev3(nil, c(3, c(2, c(1, nil))), X)$	`X=c(3,c(2,c(1,nil)))`

Rezoluție și programare logică

Prolog folosește un caz particular de rezoluție: *clauzele Horn*
= clauze cu *cel mult un literal pozitiv*

clauze definite: $p_i \leftarrow h_1 \wedge \ldots \wedge h_k$ (implicație)
se transformă în $p_i \vee \neg h_1 \vee \ldots \vee \neg h_k$ doar p_i pozitiv
fapte: p_j (se afirmă, ipoteze) p_j pozitiv
întrebare/scop/țintă (*goal*): *false* $\leftarrow g_1 \wedge \ldots \wedge g_m$
(arată prin contradicție că $g_1 \wedge \ldots \wedge g_m$ e adevărată)
se transformă în $\neg g_1 \vee \ldots \vee \neg g_m$

Pentru astfel de clauze, există o metodă de rezoluție mai simplă:
se pornește de la țintă
se caută rând fiecare scop parțial g_j în concluziile clauzelor și se înlocuiește cu conjuncția premiselor din clauză
privit ca rezoluție: se unifică un scop $\neg g_j$ cu o concluzie p_i, rezultă înlocuirea cu $\neg h_1 \vee \ldots \vee \neg h_k$

Cum se face unificarea și ce putem învăța de aici?

În logica predicatelor, rezoluția *unifică* un literal cu negatul lui:

$$A \vee P(t_1, t_2, ...\neg t_n) \quad \text{și} \quad B \vee \neg P(t'_1, t'_2, ...t'_n)$$

dacă putem unifica ("potrivi") argumentele lui P și $\neg P$: t_1 cu t'_1,...

O *substituție* e o *funcție* care asociază unor *variabile* niște *termeni*:

$$\{x_1 \mapsto t_1,\ \ldots, x_n \mapsto t_n\}$$

Doi termeni se pot *unifica* dacă există o substituție care îi face egali (o asemenea substituție se numește *unificator*)

$f(x, g(y, z), t)\{x \mapsto h(z), y \mapsto h(b), t \mapsto u\} = f(h(z), g(h(b), z), u)$

Termenul T cu substituția σ se notează uzual postfix: $T\sigma$

Substituția găsită se aplică predicatelor care rămân (rezolventul):

$$\frac{A \vee P(t_1, t_2, ...\neg t_n) \qquad B \vee \neg P(t'_1, t'_2, ...t'_n)}{A\sigma \vee B\sigma}$$

Reguli de unificare

O *variabilă* x poate fi unificată cu orice *termen* t (substituție)
dacă x *nu apare* în t dar nu: x cu $f(g(y), h(x, z))$
pentru că altfel, substituția ar duce la un termen infinit
(Dar putem unifica o variabilă cu ea însăși, x cu x)

Doi *termeni* $f(...)$ pot fi unificați doar dacă au funcții identice, și *argumentele* (termeni) pot fi unificate unul câte unul

două *constante* pot fi unificate doar dacă sunt *identice*
caz particular, funcții cu zero argumente

$\Rightarrow$ cu aceste reguli, se poate scrie un algoritm recursiv de unificare care determină *cel mai general unificator*
(orice alt unificator se poate obține din el printr-o altă substituție)

Dacă unificăm pe x cu y și apoi pe y cu $f(z, a)$, atunci si x e unificat cu $f(z, a)$ $\Rightarrow$ *relație de echivalență*

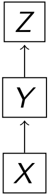
Union-Find

Structură de date+algoritm pentru lucru cu clase de echivalență.

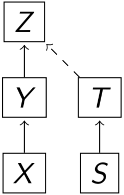
Operații:
find(element): găsește reprezentantul clasei de echivalență
union(elem1, elem2): declară elementele ca fiind echivalente
(și rămân așa mai departe)

Implementare: pădure de *arbori* cu legături de la fiu la părinte
find: returnează rădăcina (chiar nodul, dacă e singur)
union: leagă rădăcina unuia de rădăcina celuilalt

Exemplu pentru Union-Find



$find(X) = find(Y)$
$= find(Z) = Z$



$union(Y, S)$
leagă $find(Y)$ și $find(S)$

$union(T_1, T_2)$ produce *substituția* care unifică T_1 cu T_2
un *dicționar* de la variabile (șiruri) la termeni

$find(X)$ dă (căutând recursiv) reprezentantul lui X

$subst(D, T)$ face substituțiile din dicționarul D în termenul T

Principiul inducției matematice

Logica de ordinul I nu e legată de un anumit univers.
dar în matematică, folosim universul numerelor (întregi sau reali)

Principiul *inducției* matematice e (în ciuda numelui)
o *regulă de deducție* în *teoria aritmetică* a numerelor naturale

S-ar putea formula:

$$\forall P[P(0) \wedge \forall k \in \mathbb{N}.P(k) \rightarrow P(k+1)] \rightarrow \forall n \in \mathbb{N}\ P(n)$$

dar formula e în logică de *ordinul 2* (cuantificare peste predicate)

În aritmetica lui Peano, e definit ca o *schemă de axiome* (o axiomă pentru fiecare predicat) $\Rightarrow$ nu necesită cuantificare peste predicate

$$\forall \bar{x}[P(0, \bar{x}) \wedge \forall n(P(n, \bar{x}) \rightarrow P(S(n), \bar{x})) \rightarrow \forall n P(n, \bar{x})]$$

$\bar{x}$: toate celelalte variabile de care depinde predicatul P

Principiul inducției matem.: echivalent cu *principiul bunei ordonări*:
Orice mulțime nevidă de nr. naturale are un cel mai mic element

Mai general: *inducția structurală*: demonstrăm proprietăți despre obiecte tot mai complexe (pt. obiecte definite inductiv/recursiv)

Teoremele de incompletitudine ale lui Gödel

Prima teoremă de incompletitudine:
Orice sistem logic capabil să exprime aritmetica elementară nu poate fi și consistent și complet
  i.e., se poate scrie o afirmație adevărată dar care nu poate fi demonstrată în acel sistem

Demonstrație: codificând formule și demonstrații ca numere construim un număr care exprimă că formula sa e nedemonstrabilă

A doua teoremă de incompletitudine:
Consistența unui sistem logic capabil să exprime aritmetica elementară nu poate fi demonstrată în cadrul acelui sistem.

  dar ar putea fi eventual demonstrată în alt sistem logic