Logică și structuri discrete

Logica predicatelor. Calculabilitate

Marius Minea
marius@cs.upt.ro

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

12 ianuarie 2016

Recapitulare: 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 limbaje 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 (în general, 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: variabila x se poate *substitui* cu termenul t în $\forall y\varphi$ dacă:
x nu apare liber în φ (substituția nu are efect) sau
y nu apare în t și x se poate substitui cu t în φ
(nu putem substitui variabile legate)

A1: $\alpha \to (\beta \to \alpha)$
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

Pentru 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: tot *modus ponens* (e suficient)

Alte reguli de inferență

$$\frac{\forall x\ \varphi(x)}{\varphi(c)}$$ *instanțiere universală* (*vezi A5*)

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 A6*)

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 $P_I \subseteq U^n$.
  (*interpretăm* fiecare simbol din formulă)

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

Exemplu:

$\forall x.P(x,x)$ reflexivitate

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

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

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

găsiți două interpretări în care e adevărat / fals ?

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 regiuli de inferență)

Consistență și completitudine

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

Implicația logică (consecința semantică)
Fie H o mulțime de formule și φ o formulă.
Notăm $I \models H$ dacă I e un model pentru fiecare formulă din H.
Spunem că H implică φ ($H \models \varphi$) dacă pentru orice interpretare I,

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

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

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

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

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

Rezoluția în calculul predicatelor

Folosim rezoluția pentru *refutație* (reducere la absurd):
Luăm mulțimea ipotezelor H și concluzia C și arătăm că mulțimea $H \cup \{\neg C\}$ e *inconsistentă*
Cum: derivând *clauza vidă* prin rezoluție:

Fie două clauze A și B și predicatul P care apare pozitiv și negat:

Alegem literale $P_1, ..., P_k \in A$ și $\neg P_{k+1}, ...\neg P_{k+l} \in B$ cu pred. P

Unificăm termenii $\{P_1, \ldots, P_k, P_{k+1}, \ldots P_{k+l}\}$
Fie σ substituția (unificatorul cel mai general) rezultat.

Ștergem $\{P_1, \ldots P_k, P_{k+1}, \ldots P_{k+l}\}$ din clauza $A \vee B$, îi aplicăm substituția σ (unificatorul), și o adăugăm la mulțimea clauzelor.

Dacă repetând obținem *clauza vidă*, formula inițială nu e realizabilă. (e o contradicție, deci $H \vdash C$ e o teoremă, validă).
Dacă nu mai putem crea rezolvenți noi, formula inițială e realizabilă, $H \cup \{\neg C\}$ nu e o contradicție, și $H \vdash C$ nu e tot timpul adevărată

Rezoluția e completă pentru refutație

Metoda rezoluției e corectă, și *completă* relativ la refutație:
pentru orice formulă nerealizabilă, va ajunge la clauza vidă
dar nu poate *determina* realizabilitatea *oricărei formule*
(există formule pentru care rulează la infinit)

E folosită în practică în *demonstratoare de teoreme*

Principiul inducției matematice

Logica de ordinul I nu e legată de un anumit univers.
însă frecvent, 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

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

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 suficient de bogat pentru a exprima aritmetica 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 teoreme și construind un număr care exprimă că formula sa e nedemonstrabilă

A doua teoremă de incompletitudine:
Dacă un sistem logic care poate exprima aritmetică și demonstrații conține o afirmație despre propria consistență, e inconsistent.

Logică și calcul (programare)

Prin logică, putem *formaliza* cerințele programelor

În limbaje *declarative* (de nivel înalt), putem specifica *ce* să facă programul, iar interpretorul/compilatorul efectuează calculul.

În încheiere: Calculabilitate

Ce se poate *calcula*, și cum putem defini această noțiune ?

Teza Church-Turing

o afirmație despre noțiunea de *calculabilitate*:
următoarele modele de calcul sunt echivalente:

- lambda-calculul
- mașina Turing
- funcțiile recursive

Lambda-calcul

Definit de Alonzo Church (1932); poate fi privit ca fiind cel mai simplu limbaj de programare

O expresie în lambda-calcul e fie:

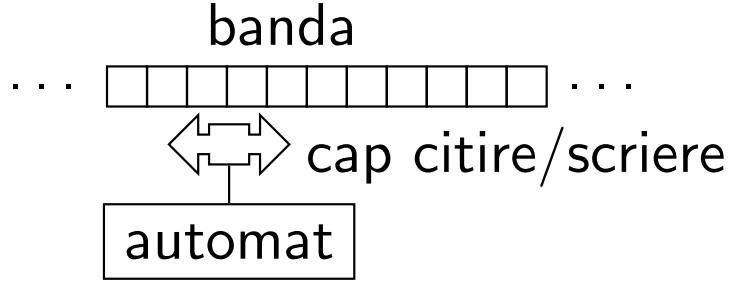
- o *variabilă* — x
- o *funcție* — $\lambda x\,.\,e$ (funcție de variabilă x)
  în ML: `fun x -> e`
- o *evaluare* de functie — $e_1\ e_2$ (funcția e_1 aplicată argumentului e_2)
  la fel în ML: `f 3` fără paranteze
  asociativă la stânga: `f x y` = `(f x) y`

Toate noțiunile fundamentale (numere naturale, booleni, perechi, etc.) pot fi exprimate în lambda-calcul.

Mașina Turing

Mașina Turing e compusă din:

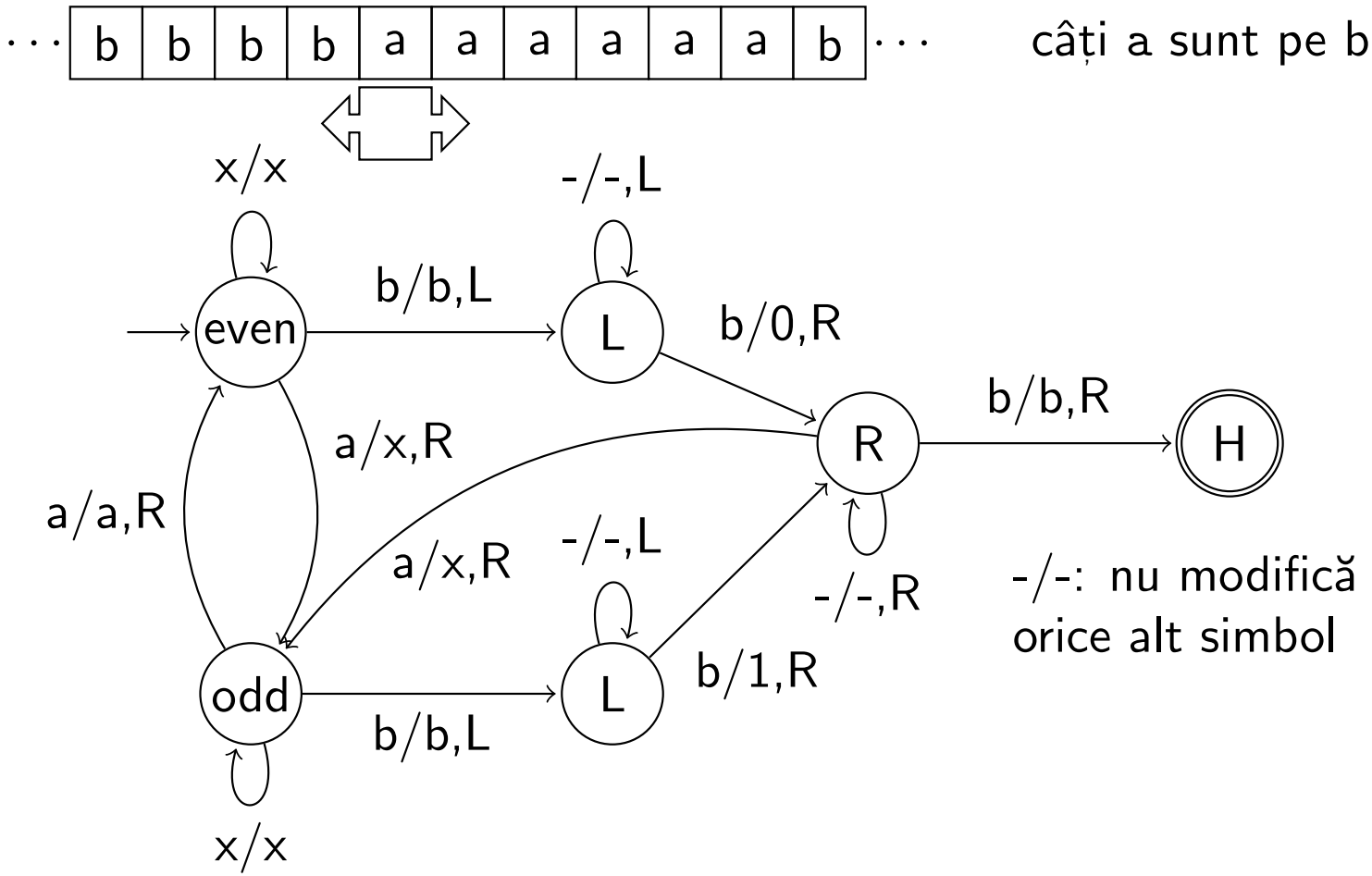
- o *bandă* cu un număr infinit de *celule*; fiecare conține un *simbol*
  (banda poate fi infinită la unul/ambele capete, e echivalent)
- un *cap* de citire/scriere, controlat de un *automat cu stări finite*



Automatul și conținutul benzii determină comportarea.
După 1) starea curentă și 2) simbolul aflat sub cap, mașina:
1) trece în starea următoare, 2) scrie un (alt) simbol sub cap
3) mută capul la stânga sau la dreapta

Inițial, banda are un șir finit de simboluri, capul e pe cel din stânga;
restul celulelor conțin un simbol special (numit vid sau blanc).

Exemplu: numără simboluri și scrie în binar



obține fiecare bit din numărul de a
$\rightarrow$: schimbă a cu x din doi în doi
$\leftarrow$: scrie 0 sau 1 după paritate
repetă până nu mai sunt a: Halt

```
   bbbbaaaaaab → bbbbxaxaxab
← bbb0xaxaxab → bbb0xxxaxxb
← bb10xxxaxxb → bb10xxxxxxb
← b110xxxxxxb → b110xxxxxxb
Halt
```

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)
Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)
Γ: mulțimea simbolurilor de pe bandă; $\Sigma \subset \Gamma$
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{l, r\}$: funcția de tranziție:
dă starea următoare, simbolul cu care e înlocuit cel curent, și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate și rămâne pe loc)
$q_0 \in Q$: starea inițială a automatului de control
$b \in \Gamma \setminus \Sigma$: simbolul vid (blanc): toate celulele cu excepția unui număr finit sunt inițial vide
$F \subseteq Q$: mulțimea stărilor finale, automatul se oprește (halt)

Poate descrie *orice calcul* (implementabil prin program)
Nu există algoritm care să decidă pentru orice automat și intrare dacă se oprește (*halting problem*) – la fel pentru programe