Fundamente de informatică

Logică propozițională

Marius Minea
marius@cs.upt.ro

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

17 octombrie 2012

De ce logică

logică = rațiune
"Nu are logică!" = e irațional (sau: nu știe să gândească)

E necesară dincolo de programare și știința calculatoarelor
în raționamente, argumente, etc.

Bază pentru

inteligența artificială
(cum deducem ? cum reprezentăm cunoștiințe?)

metode formale în inginerie software
cum exprimăm și *raționăm* despre ce face un program?
cum *demonstrăm* că un program e corect?
cum scriem automat *teste* care urmăresc căile printr-un program?

Logica propozițională

Unul din cele mai simple *limbaje* (limbaj ⇒ putem *exprima* ceva)
așa cum codificăm numere, etc. în *biți*
putem exprima probleme prin *formule* în logică

Probleme de discutat:

Cum reprezentăm o formulă
ca să putem opera eficient cu ea

Fiind dată o formulă, poate fi adevărată ?
(realizabilitate, engl. satisfiability) ⇒ SAT checking

Ce știm despre logică?

Știm deja: operatorii logici ȘI ($\wedge$), SAU ($\vee$), NU ($\neg$)

p	$\neg p$
F	T
T	F

negație $\neg$ NU

	$p \wedge q$	q: F	q: T
p	F	F	F
p	T	F	T

conjuncție $\wedge$ ȘI

	$p \vee q$	q: F	q: T
p	F	F	T
p	T	T	T

disjuncție $\vee$ SAU

Implicația logică $\rightarrow$

$p \rightarrow q$

Semnificație: dacă p e adevărat, atunci q e adevărat (if-then)
dacă p nu e adevărat, nu știm nimic despre q (poate fi oricum)

Deci, $p \rightarrow q$ e fals doar dacă p e adevărat și q e fals
(q ar trebui să fie adevărat)

	$p \rightarrow q$	q = F	q = T
p	F	T	T
p	T	F	T

Atenție! *fals* implică orice!
$\Rightarrow$ un raționament cu o verigă falsă poate duce la orice concluzie

Implicația nu înseamnă cauzalitate
un fapt adevărat implică orice fapt adevărat (fără legătură)
fals implică orice

Logica propozițională, mai riguros

Limbajul logicii propoziționale: format din *simboluri* pentru

propoziții: p, q, r, etc.

operatori (conectori logici): $\neg$, $\rightarrow$

paranteze ( )

Formulele logicii propoziționale:

orice propoziție atomică este o formulă

dacă α este o formulă, atunci $(\neg\alpha)$ este o formulă.

dacă α și β sunt formule, atunci $(\alpha \rightarrow \beta)$ este o formulă.

Operatorii cunoscuți pot fi definiți folosind $\neg$ și $\rightarrow$:

$\alpha \wedge \beta \stackrel{def}{=} \neg(\alpha \rightarrow \neg\beta)$

$\alpha \vee \beta \stackrel{def}{=} \neg\alpha \rightarrow \beta$

$\alpha \leftrightarrow \beta \stackrel{def}{=} (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

(am renunțat la parantezele redundante)

Calculul în logică: funcții de adevăr

O *funcție de adevăr* v: atribuie la orice formulă o *valoare de adevăr* $\{\mathsf{T}, \mathsf{F}\}$ astfel încât:

$v(p)$ e definită pentru fiecare propoziție atomică p.

$$v(\neg\alpha) = \begin{cases} \mathsf{T} & \text{dacă } v(\alpha) = \mathsf{F} \\ \mathsf{F} & \text{dacă } v(\alpha) = \mathsf{T} \end{cases}$$

$$v(\alpha \to \beta) = \begin{cases} \mathsf{F} & \text{dacă } v(\alpha) = \mathsf{T} \text{ și } v(\beta) = \mathsf{F} \\ \mathsf{T} & \text{în caz contrar} \end{cases}$$

Exemplu: dacă mulțimea de propoziții atomice e $\{a, b, c\}$
și alegem $v(a) = \mathsf{T}$, $v(b) = \mathsf{F}$, $v(c) = \mathsf{T}$
atunci v poate fi calculat pentru orice formulă:

$(a \to b) \to c$:
avem $v(a \to b) = \mathsf{F}$ pentru că $v(a) = \mathsf{T}$ și $v(b) = \mathsf{F}$
și atunci $v((a \to b) \to c) = \mathsf{T}$ pentru că $v(a \to b) = \mathsf{F}$.

Interpretări ale unei formule

O *interpretare* a unei formule = o evaluare pentru propozițiile ei

O intrepretare *satisface* o formulă dacă o evaluează la T.
(interpretarea e un *model* pentru formula respectivă).

Exemplu: interpretarea $v(a) = \mathrm{T}, v(b) = \mathrm{F}, v(c) = \mathrm{T}$
satisface formula $a \wedge (\neg b \vee \neg c) \wedge (\neg a \vee c)$.
Interpretarea $v(a) = \mathrm{T}, v(b) = \mathrm{T}, v(c) = \mathrm{T}$ nu o satisface.

O formulă poate fi:
validă (*tautologie*): adevărată în toate interpretările
realizabilă (satisfiable): adevărată în cel puțin o interpretare
nerealizabilă (*contradicție*): falsă în orice interpretare

Exemple de tautologii

$a \vee \neg a$

$\neg\neg a \leftrightarrow a$

$\neg(a \vee b) \ \leftrightarrow \ \neg a \wedge \neg b$

$\neg(a \wedge b) \ \leftrightarrow \ \neg a \vee \neg b$ (regulile lui de Morgan)

$(a \rightarrow b) \wedge (\neg a \rightarrow c) \ \leftrightarrow \ (a \wedge b) \vee (\neg a \wedge c)$

$a \rightarrow (b \rightarrow c) \ \leftrightarrow \ (a \wedge b) \rightarrow c$

Implicația logică (adevărul logic)

O mulțime de formule $H = \{\varphi_1, \ldots, \varphi_n\}$ *implică* o formulă φ

$$H \models \varphi$$

dacă orice funcție de adevăr care satisface H (formulele din H) satisface φ

Pentru a stabili implicația logică trebuie să *interpretăm* formulele
(cu valori/funcții de adevăr)
$\Rightarrow$ lucrăm cu *semantica* (înțelesul) formulelor

Deducții logice

O variantă de a stabili adevărul unei formule în mod *sintactic* (folosind doar structura ei)

bazată pe o *regulă de deducție*

$$\frac{\varphi_1 \qquad \varphi_1 \to \varphi_2}{\varphi_2} \qquad \textit{modus ponens}$$

(din φ_1 și $\varphi_1 \to \varphi_2$ deducem φ_2)

și un set de *axiome* (formule care pot fi folosite ca premise/ipoteze)
A1: $\alpha \to (\beta \to \alpha)$
A2: $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
A3: $(\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$

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:

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

Spunem că A_n rezultă din H (e deductibil, e consecință): $H \vdash A_n$

Exemplu: demonstrăm că $\varphi \to \varphi$

(1) $\varphi \to ((\varphi \to \varphi) \to \varphi))$	A1
(2) $\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	A2
(3) $(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	MP(1,2)
(4) $\varphi \to (\varphi \to \varphi)$	A1
(5) $\varphi \to \varphi$	MP(3,4)

Verificarea unei demonstrații e un proces simplu, mecanic (cele 3 reguli de mai sus), chiar dacă găsirea demonstrației poate fi dificilă.

Consistență și completitudine

$H \vdash \varphi$: *deducție* (pur sintactică, din axiome și reguli de deducție)
$H \models \varphi$: *implicație* (semantică, bazat pe tabele de adevăr)
Care e legătura între ele ?

Consistență: Dacă H e o mulțime de formule, și α este o formulă astfel ca $H \vdash \alpha$, atunci $H \models \alpha$.
(Orice teoremă în logica propozițională este o tautologie).

Completitudine: Dacă H e o mulțime de formule, și α este o formulă astfel ca $H \models \alpha$, atunci $H \vdash \alpha$.
(Orice tautologie este o teoremă).

Ca să demonstrăm o formulă, putem arăta că a *validă*. Pentru aceasta, verificăm că *negația ei nu e realizabilă*

Forma normală conjunctivă

Conjunctive normal form (CNF)

Formula scrisă ca o *conjuncție* de *clauze*
Fiecare clauză e o *disjuncție* de *literali*
(propoziție atomică sau negația ei)

$$\begin{aligned} &(a \lor \neg b \lor \neg d) \\ \land\ &(\neg a \lor \neg b) \\ \land\ &(\neg a \lor c \lor \neg d) \\ \land\ &(\neg a \lor b \lor c) \end{aligned}$$

Problemă: cum convertim o funcție în CNF?

Reprezentare: tipuri sumă (cu variante) în ML

ML permite definirea simplă de reprezentări cu alternative (inclusiv recursive)

```
type boolexp =
    Id of string
  | Not of boolexp
  | And of boolexp * boolexp
  | Or of boolexp * boolexp
```

Identificatorii (*constructorii de tip*) Id, And, etc. sunt aleși de utilizator
(ca niște etichete care identifică varianta).

Prelucrarea se face prin potrivire de tipare:

```
match e with
  Not e -> ...
| And (e1, e2) -> ...
...
```

Compilatorul *avertizează* dacă nu tratăm toate variantele.

Citirea cu `scanf` în Ocaml

modulul `Scanf`, funcția `scanf`

`open Scanf` (scriem simplu `scanf`, nu `Scanf.scanf`)

Folosirea: `scanf` *format funcție-de-valori-citite*

Formate (similar cu limbajul C)

%c caracter %0c fără să-l consume (pentru test)

%d întreg

%f real

%s șir

%[caractere] șir din caracterele permise (sau ^ pt. excluse)

spațiu în format: ignoră 0 sau oricâte spații albe

număr după %: max. atâtea caractere

Valorile citite sunt *folosite direct* în funcția dată:

```
let f () = scanf "%d %c %d" (fun x op y ->
                 if op = '+' then x + y else 0)
```

Gramatica pentru o formulă

Formula ::= Term | Formula '+' Term
Term ::= Factor | Term '*' Factor
Factor ::= id | '~' id | '(' Formula ')'

Pentru a citi efectiv o formulă, modificăm gramatica așa încât să putem decide după primul caracter ce variantă trebuie urmată:

Formula ::= Term RestForm
RestForm ::= ϵ | '+' Term RestForm
Term ::= Factor RestTerm
RestTerm ::= ϵ | '*' Factor RestTerm

În program, funcțiile pentru RestForm/RestTerm au ca parametru termenul/factorul citit înainte, pentru a-l putea grupa cu următorul