Logică și structuri discrete

Logica predicatelor

Marius Minea
marius@cs.upt.ro

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

20 noiembrie 2017

În cursul de azi

Un algoritm de unificare

Programare logică
- specificăm *ce* vrem să obținem
- caută automat *soluția* (decide *cum*)
- bazată pe rezoluție

Ce e o “demonstrație” și “totdeauna adevărat”
- și limitele a ce putem demonstra

Cum se face unificarea?

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

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

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):

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

Reguli de unificare

O *variabilă* x poate fi unificată cu orice *termen* t (substituție)
dacă x *nu apare* în t (altfel, substituind obținem un termen infinit)
deci nu: x cu $f(h(y), g(x, z))$; dar trivial, putem unifica x cu x

Doi *termeni* $f(...)$ pot fi unificați doar dacă au aceeași funcție,
și *argumentele* (termeni) pot fi unificate (poziție cu poziție)

Două *constante* (funcții cu 0 arg.) $\Rightarrow$ unificate dacă sunt identice

$\Rightarrow$ cu aceste reguli, putem găsi *cel mai general unificator*
(orice alt unificator se poate obține din el printr-o altă substituție)

Clase de echivalență și Union-Find

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*

Trebuie să reținem ce variabile sunt *echivalente*.

Union-Find:
structură de date+algoritm pentru lucru cu clase de echivalență.

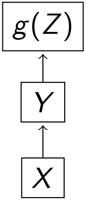
Operații:
find(element): găsește reprezentantul clasei de echivalență
union(elem1, elem2): declară elementele ca fiind echivalente
(rămân echivalente mai departe)

Exemplu pentru Union-Find

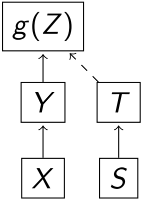
O implementare: pădure de *arbori* cu legături de la fiu la părinte

find: returnează rădăcina (chiar nodul, dacă e singur)

union: leagă rădăcina unuia de rădăcina celuilalt



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



$union(Y, S)$

leagă $find(Y)$ și $find(S)$

Union-Find cu dicționare

Substituție = *dicționar* de la variabile (string) la termeni

union(D, T_1, T_2) produce *o nouă substituție* care unifică T_1 cu T_2
pornind de la o substituția / dicționarul D existent
construim substituția pas cu pas, unificând succesiv termeni

find(D, X) caută recursiv reprezentantul lui X ("ce înseamnă")
adică aplică lui X substituțiile din D

Exemplu: unificăm $f(x, g(x, s(z)), t)$ cu $f(h(z), g(h(b), u), z)$
x cu $h(z) \Rightarrow \{x \mapsto h(z)\}$
$g(h(x), z)$ cu $g(h(b), u) \Rightarrow$
x cu $h(b) \Rightarrow h(z)$ cu $h(b) \Rightarrow \{x \mapsto h(z), z \mapsto b\}$
$s(z)$ cu $u \Rightarrow \{x \mapsto h(z), z \mapsto b, u \mapsto s(z)\}$
t cu $z \Rightarrow t$ cu $b \Rightarrow \{x \mapsto f(z), z \mapsto b, u \mapsto s(z), t \mapsto b\}$
Substituind până la capăt:
$\{x \mapsto f(b), z \mapsto b, u \mapsto s(b), t \mapsto b\}$

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) `fiu(ion, petre).`

reguli: `:-` e implicație $\leftarrow$ (stânga *dacă* dreapta)

virgula e conjuncție $\wedge$

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

X e descendentul lui Z dacă X e fiul lui Y și Y e descendentul lui Z

Ce înseamnă clauzele

```
(R1)   desc(X, Y) :- fiu(X, Y).
(R2)   desc(X, Z) :- fiu(X, Y), desc(Y, Z).
```

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

R1: $\forall X \forall Y. desc(X, Y) \leftarrow fiu(X, Y)$

R2: X e descendent al lui Z dacă *există* Y
astfel încât X e fiul lui Y și Y e descendent al lui Z

R2: $\forall X \forall Z. desc(X, Z) \leftarrow \exists Y. fiu(X, Y) \wedge desc(Y, Z)$
$\forall X \forall Z. desc(X, Z) \vee \neg \exists Y. fiu(X, Y) \wedge desc(Y, Z)$
$\forall X \forall Z. desc(X, Z) \vee \forall Y \neg (fiu(X, Y) \wedge desc(Y, Z))$

Eliminând cuantificatorii universali rezultă:

R1: $desc(X, Y) \vee \neg fiu(X, Y)$

R2: $desc(X, Z) \vee \neg fiu(X, Y) \vee \neg desc(Y, Z)$

Rezoluția în Prolog

Fie ca întrebare/scop/țintă (engl. *goal*) `desc(X, vasile).`
O *soluție* = o valoare `xs` pentru `X` care face predicatul adevărat.
⇒ negația $\neg$ `desc(xs, vasile)` va da o contradicție.

Folosim *rezoluția* încercând să găsim o contradicție.

Unificăm cu regula R1 (redenumind-o cu variabile noi):

	$\neg$ `desc(X, vasile)`	
(R1)	`desc(X1, Y1)` $\vee \neg$ `fiu(X1, Y1)`	
	$\neg$ `fiu(X, vasile)`	`X1=X, Y1=vasile`
faptul (4)	`fiu(petre, vasile)`	
	$\emptyset$ (clauza vidă)	`X=petre`

Pentru `X=petre` am obținut o contradicție.
`desc(petre, vasile)` e *adevărat*. `X=petre` e o *soluție*.

Continuând, putem găsi și alte soluții.

Soluții multiple în Prolog

Pornim tot cu negația întrebării: `¬desc(X, vasile)`.

Unificăm cu regula 2 (redenumind din nou variabilele):

$$\frac{\begin{array}{l}\neg\texttt{desc(X,vasile)} \\ \qquad\texttt{desc(X2,Z2)} \lor \neg\texttt{fiu(X2,Y2)} \lor \neg\texttt{desc(Y2,Z2)}\end{array}}{\neg\texttt{fiu(X, Y2)} \lor \neg\texttt{desc(Y2,vasile)} \qquad \texttt{X2=X,Z2=vasile}}$$

$$\frac{\text{faptul (1)} \qquad \texttt{fiu(ion,petre)}}{\neg\texttt{desc(petre,vasile)} \qquad \texttt{X=ion,Y2=petre}}$$

$$\frac{\text{obținut anterior} \qquad \texttt{desc(petre,vasile)}}{\emptyset \text{ (clauza vidă)}}$$

⇒ `X=ion` e încă o soluție pentru întrebarea inițială.

Dacă întrebarea are variabile, interpretorul Prolog generează *toate soluțiile posibile* (toate substituțiile pentru variabile).

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

Exemplu Prolog: inversarea listelor (1)

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.

```
let rec rev2 acc = function
  | [] -> acc
  | h :: t -> rev2 (h::acc) t
```

Exemplu Prolog: inversarea listelor (2)

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

Cu întrebarea $rev(c(1, c(2, c(3, nil)))), X)$ obținem derivarea:

$rev(c(1, c(2, c(3, nil))), X)$

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

Pentru clauze Horn, rezoluția se aplică mai simplu:
se pornește de la țintă
se caută pe rând fiecare scop parțial g_j în concluziile regulilor
și se înlocuiește cu conjuncția premiselor din regulă (clauză)
până când în toate cazurile se ajunge la *fapte*

Privit ca rezoluție: se unifică un scop $\neg g_j$ cu o concluzie p_i, rezultă înlocuirea cu $\neg h_1 \vee ... \vee \neg h_k$

Aplicații ale programării logice

Gestiune de dependențe/configurații/compilare în sisteme software
reguli de tip Makefile

```
main:  main.c file.c lib.c
    gcc -o main main.c file.c lib.c
```

Sisteme expert: reguli cu cunoștințe dintr-un domeniu (medical,...)

Datalog: un subset de Prolog
 interogări în baze de date
 specificări de politici de securitate

Programare logică cu constrângeri (*constraint logic programming*)
 planificare (orare, zboruri, personal)
 gestiune de portofolii financiare
 optimizări de circuite

Cât de generală e o demonstrație?

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

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

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

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

rezoluție: $$\frac{\neg boy(x) \vee \neg girl(x) \vee child(x) \qquad boy(c)}{\neg girl(c) \vee child(c)}$$

Demonstrația e făcută *fără* a ține cont (sau înțelege) *semnificația* predicatelor *boy*, *child*, *good*, etc.: puteau fi $P(x), Q(x), R(x), \ldots$

Demonstrația e valabilă 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.

$$
\begin{array}{ll}
\ \forall x\, R(x,x) & \text{reflexivitate} \\
\wedge\ \forall x \forall y (R(x,y) \to R(y,x)) & \text{simetrie} \\
\wedge\ \forall x \forall y \forall z (R(x,y) \wedge R(y,z) \to R(x,z)) & \text{tranzitivitate} \\
\to \forall x \forall y (\neg \exists z (R(x,z) \wedge R(y,z)) & \text{clase disjuncte} \\
\qquad\qquad \vee\ \forall z (R(x,z) \leftrightarrow R(y,z))) & \text{sau clase identice}
\end{array}
$$

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)

În loc de propoziții avem *predicate* (peste *termeni*).

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.

Reguli și ce înseamnă aplicarea lor

Regulile discutate sunt *sintactice*: manipulează forma (*simboluri*, nu înțelesul lor).

Regulile lui deMorgan: $\neg(a \vee b) = \neg a \wedge \neg b, \quad \neg(a \wedge b) = \neg a \vee \neg b$
Înlocuim o formă cu alta.
Rezultatul: formulele sunt echivalente

Regulă: Dacă un literal L e singur într-o clauză:
ștergem toate clauzele din care apare
ștergem $\neg L$ din toate clauzele

Rezultatul: dacă formula era realizabilă, rămâne realizabilă

Rezoluția: $$\frac{p \vee A \qquad \neg p \vee B}{A \vee B}$$

Rezultatul: dacă formula era realizabilă, rămâne realizabilă

Axiomele calculului predicatelor

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.

*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)

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* $A5$)

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 valoare, există o valoare care o face adevărată

Semantica

Definim noțiunile:

- univers
- interpretare
- model
- consecință semantică

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).

Exemple de interpretări

$\forall x \forall y \forall z . P(x, y) \wedge P(y, z) \rightarrow P(x, z)$ 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 $x \in y$

$\forall x . \forall y . P(x, y) \rightarrow \exists z . P(x, z) \wedge P(z, y)$
Găsiți o interpretare în care e adevărată și una în care e falsă ?

Atribuiri și 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

Există logici mai bogate decât logica predicatelor

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

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

această formă e în logică de *ordinul 2* (cuantificare peste predicate)

Poate fi definit ca o *schemă de axiome* (o axiomă pentru fiecare predicat), fără a cuantifica 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)

Logica are limitări

Teoria numerelor naturale cu adunare (aritmetica Presburger) e *decidabilă*
(orice putem exprima despre adunarea numerelor naturale e demonstrabil).
Dar: nu putem exprima divizibilitate, numere prime, etc.

Aritmetica lui Peano (cu adunare și înmulțire) e mai bogată dar e *nedecidabilă*: sunt afirmații despre care nu se poate decide dacă sunt adevărate sau nu.

Teoremele de incompletitudine ale lui Gödel

Prima teoremă de incompletitudine:
Orice sistem logic *consistent* care poate exprima aritmetica elementară e incomplet

i.e., se pot scrie afirmații care nu pot fi nici demonstrate nici infirmate 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