Logică și structuri discrete

Logică propozițională

Casandra Holotescu

casandra@cs.upt.ro

https://tinyurl.com/lecturesLSD

În cursul de azi

Cum determinăm dacă o formulă e *realizabilă*?
algoritm folosit în rezolvarea multor probleme

Ce înseamnă o *demonstrație* logică?

Realizabilitatea unei formule în logică propozițională (SAT-problem/*satisfiability*)

Realizabilitatea unei formule propoziționale (satisfiability)

Se dă o formulă în *logică propozițională*.

Există vreo atribuire de valori de adevăr care o face adevărată ?
= e *realizabilă* (engl. *satisfiable*) formula ?

$$\begin{aligned}
&(a \vee \neg b \vee \neg d)\\
\wedge\ &(\neg a \vee \neg b)\\
\wedge\ &(\neg a \vee c \vee \neg d)\\
\wedge\ &(\neg a \vee b \vee c)
\end{aligned}$$

Găsiți o atribuire care satisface formula?

Formula e în *formă normală conjunctivă* (conjunctive normal form)
= conjuncție de disjuncții de *literali* (pozitiv sau negat)

Fiecare conjunct (linie de mai sus) se numește *clauză*

Reguli în determinarea realizabilității

Simplificăm problema, știind că vrem **formula adevărată**
(NU se aplică la simplificarea formulelor în formule echivalente!)

R1) Un literal *singur într-o clauză* are o singură valoare utilă:

în $a \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$ — a trebuie să fie T

în $(a \vee b) \wedge \neg b \wedge (\neg a \vee \neg b \vee c)$ — b trebuie să fie F

(altfel formula are valoarea F)

Reguli pentru determinarea realizabilității (cont.)

R2a) Dacă un literal e T, *pot fi șterse clauzele* în care apare
(ele sunt adevărate, le-am rezolvat)

R2b) Dacă un literal e F, *el poate fi șters* din clauzele în care apare
(nu poate face clauza adevărată)

Exemplele anterioare se simplifică:

$$a \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c) \quad \stackrel{a=\mathsf{T}}{\rightarrow} \quad (b \vee c) \wedge (\neg b \vee \neg c)$$

$$(a \vee b) \wedge \neg b \wedge (\neg a \vee \neg b \vee c) \quad \stackrel{b=\mathsf{F}}{\rightarrow} \quad a$$

(și de aici $a = T$, deci formula e realizabilă)

Reguli pentru determinarea realizabilității (cont.)

R3) Dacă *nu mai sunt clauze*, formula e realizabilă
(cu atribuirea construită)

Dacă obținem o *clauză vidă*, formula *nu e realizabilă*
(fiind vidă, nu putem s-o facem T)

$(a \vee b) \wedge a \wedge (a \vee \neg b \vee c) \quad \stackrel{a=\mathrm{T}}{\rightarrow} \quad (\mathrm{T} \vee b) \wedge \mathrm{T} \wedge (\mathrm{T} \vee \neg b \vee c) \xrightarrow{R2a}$
ștergem toate clauzele (conțin T, le-am rezolvat)
$\Rightarrow$ formulă realizabilă (cu $a = \mathrm{T}$)

$$a \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$$
$$\stackrel{a=\mathrm{T}}{\rightarrow} b \wedge (\neg b \vee c) \wedge (\neg b \vee \neg c)$$
$\stackrel{b=\mathrm{T}}{\rightarrow} c \wedge \neg c \quad \stackrel{c=\mathrm{T}}{\rightarrow} \quad \emptyset \quad$ ($\neg c$ devine clauza vidă $\Rightarrow$ nerealizabilă)

Reguli pentru determinarea realizabilității (cont.)

Dacă *nu mai putem face reduceri* după aceste reguli ?

$a \wedge (\neg a \vee b \vee c) \wedge (\neg b \vee \neg c) \quad \stackrel{a=\mathrm{T}}{\rightarrow} \quad (b \vee c) \wedge (\neg b \vee \neg c) \quad ??$

R4) Alegem o variabilă și *despărțim pe cazuri* (încercăm):

- cu valoarea F
- cu valoarea T

O soluție pentru *oricare* caz e bună (nu căutăm o soluție anume).

Dacă *niciun caz* nu are soluție, formula *nu e realizabilă*.

Un algoritm de rezolvare

Problema are ca date:

- lista clauzelor (formula)
- mulțimea variabilelor deja atribuite (inițial vidă)

Regulile 1 și 2 ne *reduc problema la una mai simplă*
(mai puține necunoscute sau clauze mai puține și/sau mai simple)

Regula 3 spune când ne oprim (avem răspunsul).

Regula 4 reduce problema la rezolvarea a *două probleme mai simple*
(cu o necunoscută mai puțin)

Reducerea problemei la *aceeași problemă cu date mai simple*
(una sau mai multe instanțe) înseamnă că problema e *recursivă*.

Obligatoriu: trebuie să avem și o *condiție de oprire*

Algoritmul Davis-Putnam-Logemann-Loveland (1962)

function solve(truelit: lit set, clauses: clause list)
(truelit, clauses) = simplify(truelit, clauses) (* R1, R2 *)
if clauses = lista vidă **then**
  return truelit; (* R3: realizabila, returneaza atribuirile *)
if clauses conține clauza vidă **then**
  raise Unsat; (* R3: nerealizabila *)
if clauses conține clauză cu unic literal *a* **then**
  solve (truelit $\cup\{a\}$, clauses) (* R1: *a* trebuie să fie T *)
else
  try solve (truelit $\cup\{\neg a\}$, clauses); (* R4: încearcă a=F *)
  with Unsat $\rightarrow$ solve (truelit $\cup\{a\}$, clauses); (* încearcă T *)

Rezolvitoarele (*SAT solvers/checkers*) moderne pot rezolva formule cu milioane de variabile (folosind optimizări)

Complexitatea realizabilității

O formulă cu n propoziții are 2^n atribuiri
$\Rightarrow$ timp *exponențial* încercând toate

O atribuire dată se verifică în timp *liniar* (în dimensiunea formulei)
parcurgem formula o dată și obținem valoarea

În general, a *verifica* o soluție e (mult) mai simplu decât a o *găsi*.

P = clasa problemelor care pot fi *rezolvate* în timp polinomial (relativ la dimensiunea problemei)
căutare în tablou nesortat: liniar, $O(n)$
sortare $O(n \log n)$ (eficient), $O(n^2)$ (nu folosiți)
toate drumurile minime în graf $O(n^3)$

NP (nondeterministic polynomial time) = clasa problemelor pentru care o soluție ("ghicită", dată) poate fi *verificată* în timp polinomial

Probleme NP-complete

Unele probleme din clasa *NP* sunt mai dificile decât altele.

Probleme *NP-complete*: cele mai dificile probleme din clasa NP
dacă una din ele s-ar rezolva în timp polinomial,
orice altă problemă din NP s-ar rezolva în timp polinomial

$\Rightarrow$ am avea P = NP (se crede P $\neq$ NP)

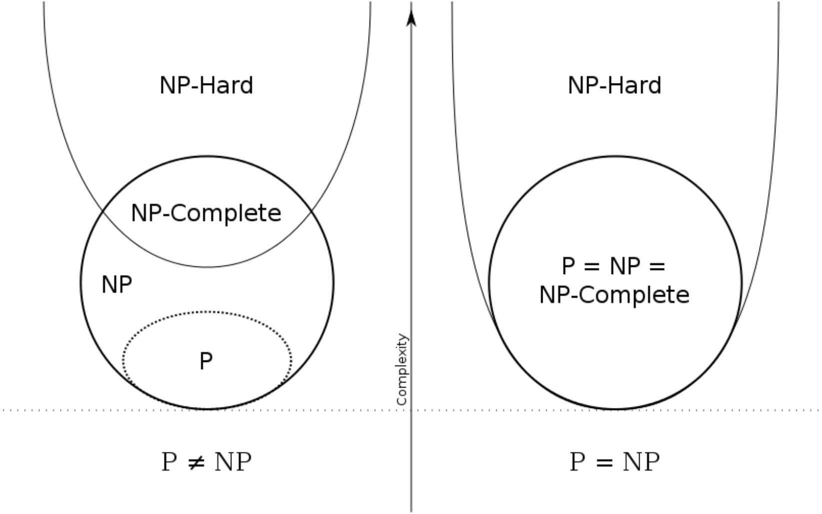
Realizabilitatea (SAT) e prima problemă demonstrată a fi *NP-completă* (Cook, 1971). Sunt multe altele (21 probleme clasice: Karp 1972).

Cum demonstrăm că o problemă e NP-completă (grea) ?
reducem o problemă cunoscută din NP la problema studiată

$\Rightarrow$ dacă s-ar putea rezolva în timp polinomial problema nouă,
atunci ar lua timp polinomial problema cunoscută

P = NP?

Una din problemele fundamentale în informatică



Se crede că $P \neq NP$, dar nu s-a putut (încă) demonstra

Imagine: `http://en.wikipedia.org/wiki/File:P_np_np-complete_np-hard.svg`

Demonstrație vs consecință logică

Sintaxă și semantică

Pentru logica propozițională, am discutat:

Sintaxa: o formulă are *forma*:
propoziție sau ($\neg$ *formulă*) sau (*formulă* $\rightarrow$ *formulă*)

Semantica: calculăm *valoarea de adevăr* (înțelesul), pornind de la cea a propozițiilor

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

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

Deducții logice

Deducția ne permite să demonstrăm o formulă în mod *sintactic* (folosind doar structura ei)

E bazată pe o *regulă de inferență* (de deducție)

$$\frac{A \qquad A \rightarrow B}{B} \qquad \textit{modus ponens}$$

(din A și $A \rightarrow B$ deducem/inferăm B; A, B formule oarecare)

și un set de *axiome* (formule care pot fi folosite ca premise/ipoteze)
A1: $\alpha \rightarrow (\beta \rightarrow \alpha)$
A2: $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$
A3: $(\neg\beta \rightarrow \neg\alpha) \rightarrow (\alpha \rightarrow \beta)$
în care α, β etc. pot fi înlocuite cu *orice* formule

Exercițiu: arătați că A1 - A3 sunt tautologii

Deducție (demonstrație)

Informal, o deducție (demonstrație) e o *înșiruire de afirmații* în care fiecare *rezultă* (poate fi derivată) din cele *anterioare*.

Riguros, definim:

Fie H o mulțime de formule (ipoteze).

O *deducție* (demonstr.) 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: $H \vdash A_n$

Exemplu de deducție

Demonstrăm că $A \to A$ pentru orice formulă A

(1) $A \to ((A \to A) \to A))$ A1 cu $\alpha = A, \beta = A \to A$

(2) $A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$
A2 cu $\alpha = \gamma = A, \beta = A \to A$

(3) $(A \to (A \to A)) \to (A \to A)$ MP(1,2)

(4) $A \to (A \to A)$ A1 cu $\alpha = \beta = A$

(5) $A \to A$ MP(3,4)

Verificarea unei demonstrații e un proces simplu, mecanic
(verificăm motivul indicat pentru fiecare afirmație;
o simplă comparație de șiruri de simboluri).

Găsirea unei demonstrații e un proces mai dificil.

Alte reguli de deducție

Modus ponens e suficient pentru a formaliza logica propozițională dar sunt și alte reguli de deducție care simplifică demonstrațiile

$$\frac{p \to q \qquad \neg q}{\neg p}$$ *modus tollens (reducere la absurd)*

$$\frac{p}{p \vee q}$$ *generalizare (introducerea disjuncției)*

$$\frac{p \wedge q}{p}$$ *specializare (simplificare)*

$$\frac{p \vee q \qquad \neg p}{q}$$ *eliminare (silogism disjunctiv)*

$$\frac{p \to q \qquad q \to r}{p \to r}$$ *tranzitivitate (silogism ipotetic)*

Deducția (exemplu)

Fie $H = \{a,\ \neg b \vee d,\ a \to (b \wedge c),\ (c \wedge d) \to (\neg a \vee e)\}$.
Arătați că $H \vdash e$.

(1) a	ipoteză, H_1
(2) $a \to (b \wedge c)$	ipoteză, H_3
(3) $b \wedge c$	modus ponens (1, 2)
(4) b	specializare (3)
(5) d	eliminare (4, H_2)
(6) c	specializare (3)
(7) $c \wedge d$	(5) și (6)
(8) $\neg a \vee e$	modus ponens (7, H_4)
(9) e	eliminare (1, 8)

Consecința logică (semantică)

Interpretare = atribuire de adevăr pentru propozițiile unei formule.
O formulă poate fi adevărată sau falsă într-o interpretare.

Def.: O mulțime de formule $H = \{H_1, \ldots, H_n\}$ *implică* o formulă C dacă *orice interpretare* care satisface (formulele din) H satisface C

Notăm: $H \models C$

(C e o *consecință logică* / consecință semantică a ipotezelor H)

Ca să stabilim consecința semantică trebuie să *interpretăm* formule (cu valori/funcții de adevăr)

⇒ lucrăm cu *semantica* (înțelesul) formulelor

Exemplu: arătăm $\{A \vee B, C \vee \neg B\} \models A \vee C$ Fie interpretarea v.
Cazul 1: $v(B) = \mathrm{T}$. Atunci $v(A \vee B) = \mathrm{T}$ și $v(C \vee \neg B) = v(C)$.
Dacă $v(C) = T$, atunci $v(A \vee C) = T$, deci afirmația e adevărată.
Cazul 2: $v(B) = \mathrm{F}$. La fel, reducem la $\{A\} \models A \vee C$ (adevărat).

Consistență și completitudine

$H \vdash C$: *deducție* (pur sintactică, din axiome și reguli de inferență)
$H \models C$: *implicație, consecință semantică* (valori de adevăr)

Consistență: Dacă H e o mulțime de formule, și C este o formulă astfel ca $H \vdash C$, atunci $H \models C$

(Orice teoremă e *validă*; orice afirmație obținută prin deducție e *întotdeauna adevărată*).

Completitudine: Dacă H e o mulțime de formule, și C e o formulă astfel ca $H \models C$, atunci $H \vdash C$.

(Orice tautologie e o teoremă, orice consecință semantică poate fi *dedusă* din *aceleași ipoteze*).

Logica propozițională e *consistentă și completă*:

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