Logică și structuri discrete

Logică propozițională

Marius Minea
marius@cs.upt.ro

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

3 noiembrie 2014

Unde *aplicăm* verificarea realizabilității?

probleme de *căutare* și *planificare*

Cum *reprezentăm eficient* formulele boolene ?

diagrame de decizie binare

Ce înseamnă o *demonstrație* logică?

Unde aplicăm logica booleană?

Calculatoarele sunt construite din *circuite logice*
$\Rightarrow$ realizează aceleași *funcții* ca în logică (ȘI, SAU, NU)

Numerele sunt reprezentate în calculator *în baza 2*
$\Rightarrow$ valori *boolene* (F sau T, 0 sau 1)

Aritmetica pe numere e implementată prin circuite logice

```
let rec add a b =
  if b = 0 then a else add (a lxor b) ((a land b) lsl 1)
```

Mulțimile pot fi reprezentate prin șiruri de valori boolene
pentru fiecare element: face sau nu parte din mulțime ?

Orice noțiune din matematică sau realitate e reprezentată pe biți

Aplicație: Planificarea

= găsirea unei secvențe de acțiuni care duc la o țintă dorită

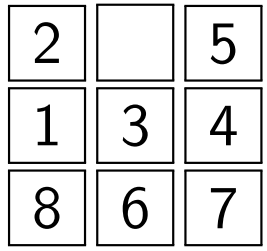
Exemple:

- deplasările unor roboți inteligenți
- comportamentul sistemelor autonome (sonde spațiale)
- rezolvarea de probleme (de tip puzzle, jocuri, etc.)

În general: într-un sistem descris prin *stări* și *acțiuni* (tranziții), cum găsim o cale de la o *stare inițială* la o *stare țintă* (finală) ?

Exemplu: ordonarea a 3x3 piese

Se poate reface ordinea? Din câte mutări?



starea jocului e dată de numărul de pe fiecare din cele 9 poziții:
un *vector de stare* $\bar{v} = (p_1, p_2, \ldots, p_9)$, $p_i \in [0..8]$ (0 = liber)

fiecare valoare p_i poate fi reprezentată boolean (cu 4 biți)

sau direct: b_{ij} = piesa j (0..8) e pe poziția i (1..9)
cu constrângeri: un singur b_{ij} adevărat pentru fiecare i
(un singur număr în fiecare loc)

O *stare* e descrisă printr-o *formulă logică*

peste variabilele de stare (elementele vectorului de stare)

Starea inițială din figură: $S_0(\bar{v}) = p_1 = 2 \wedge p_2 = 0 \wedge p_3 = 5 \wedge p_4 = 1$
$\wedge\, p_5 = 3 \wedge p_6 = 4 \wedge p_7 = 8 \wedge p_8 = 6 \wedge p_9 = 7$

Reprezentarea unei mutări

Indexăm stările: dacă $\bar{v}_k$ e starea curentă, $\bar{v}_{k+1}$ e cea următoare.

O *mutare* e schimbarea între două piese de pe poziții vecine s și d

toate celealte piese rămân la fel: $p_{i\,k+1} = p_{ik}$ pt. $i \neq s, d$

mutarea e posibilă doar dacă una din pozițiile s, d e liberă (0)

$$m_k(s,d) = (p_{s\,k+1} = p_{dk}) \wedge (p_{d\,k+1} = p_{sk}) \wedge \bigwedge_{i \neq s,d} (p_{i\,k+1} = p_{ik})$$

$$\wedge (p_{sk} = 0 \vee p_{dk} = 0)$$

Sunt 12 perechi de poziții vecine: $P = \{(1,2),(1,4),\ldots(8,9)\}$

123
456
789

Toate mutările posibile definesc o *relație* între starea curentă și cea următoare: *relația de tranziție* a sistemului

$$R(\bar{v}_k, \bar{v}_{k+1}) = \bigvee_{(s,d)\in P} m_k(s,d)$$

(avem *disjuncție* pentru că facem (doar) *una* din mutările posibile)

Reprezentarea unui șir de mutări

Prima mutare: $R(\bar{v}_0, \bar{v}_1) =$

$(p_{10} = 0 \vee p_{20} = 0) \wedge p_{11} = p_{20} \wedge p_{21} = p_{10} \wedge p_{31} = p_{30} \wedge ... \wedge p_{91} = p_{90}$

$\vee (p_{10} = 0 \vee p_{40} = 0) \wedge p_{11} = p_{40} \wedge p_{41} = p_{10} \wedge p_{21} = p_{20} \wedge ... \wedge p_{91} = p_{90}$

...

$\vee (p_{80} = 0 \vee p_{90} = 0) \wedge p_{91} = p_{80} \wedge p_{81} = p_{90} \wedge p_{31} = p_{30} \wedge ... \wedge p_{71} = p_{70}$

Un lanț de k mutări: $R(\bar{v}_0, \bar{v}_1) \wedge R(\bar{v}_1 \bar{v}_2) \wedge \ldots \wedge R(\bar{v}_{k-1}, \bar{v}_k)$

O *tranziție* și un *șir de tranziții* se pot reprezenta ca formule logice

Starea finală e tot o formulă peste elementele vectorului de stare $\bar{v}$:

$S_f(\bar{v}) = p_1 = 1 \wedge p_2 = 2 \wedge \ldots \wedge p_7 = 7 \wedge p_8 = 8 \wedge p_9 = 0$

Există o soluție în k pași dacă și numai dacă

$$S_0(\bar{v}_0) \wedge R(\bar{v}_0, \bar{v}_1) \wedge R(\bar{v}_1 \bar{v}_2) \wedge \ldots \wedge R(\bar{v}_{k-1}, \bar{v}_k) \wedge S_f(\bar{v}_k)$$

Găsirea unui plan

Fie $S_0(\bar{v})$ și $S_f(\bar{v})$ formulele ce exprimă stările inițiale și finale
A ajunge la S_f din S_0 în *1 mutare* $\Leftrightarrow$ e realizabilă formula

$$S_0(\bar{v}_0) \wedge R(\bar{v}_0, \bar{v}_1) \wedge S_f(\bar{v}_1)$$

($\bar{v}_0$ e o stare inițială și $\bar{v}_1$ o stare finală și e o tranziție între ele)

A ajunge la S_f din S_0 în *k mutări* $\Leftrightarrow$ e realizabilă formula

$$S_0(\bar{v}_0) \wedge R(\bar{v}_0, \bar{v}_1) \wedge ... \wedge R(\bar{v}_{k-1}, \bar{v}_k) \wedge S_f(\bar{v}_k)$$

$\Rightarrow$ Găsim un plan de lungime minimă căutând succesiv soluții pentru formule tot mai complexe

Există și alți algoritmi dedicați planificării.
Aici am redus problema la o exprimare *simplă*, fundamentală:
determinarea realizabilității unei formule boolene (problema SAT)

Reprezentarea formulelor boolene

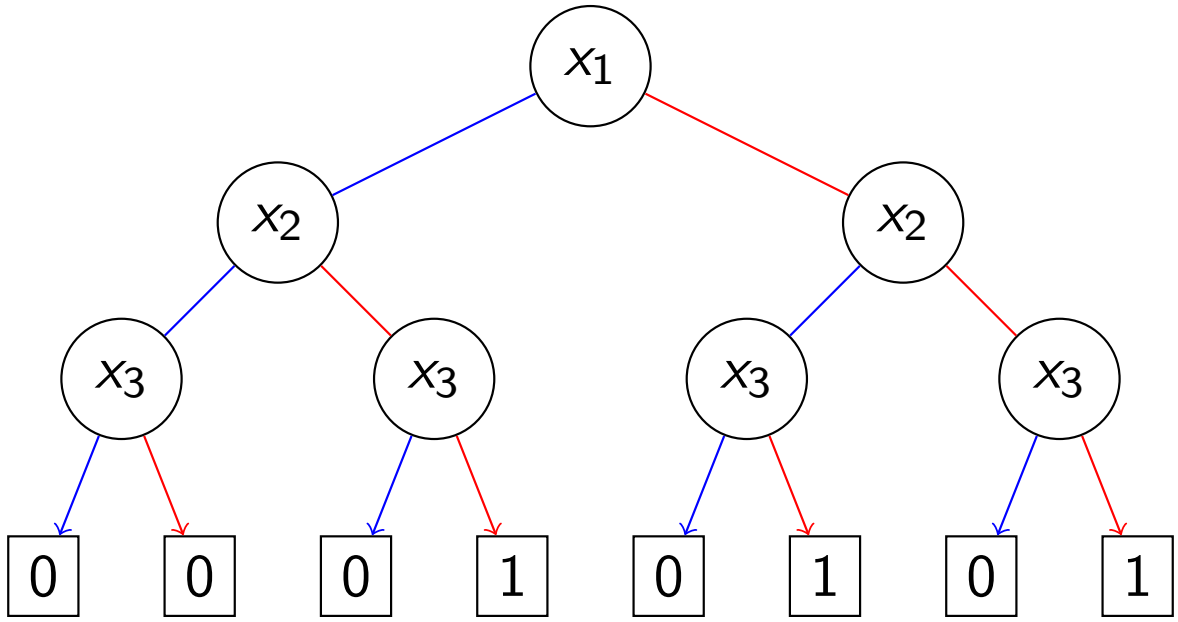
Forma normală conjunctivă e utilă în determinarea realizabilității

dar există formule a căror reprezentare în CNF crește exponențial

În general, avem nevoie de o reprezentare
- ușor manipulabilă
- compactă pentru *majoritatea* formulelor tipice

O astfel de reprezentare: *diagrame de decizie binare* (Bryant, 1986)

Arborele de decizie binar



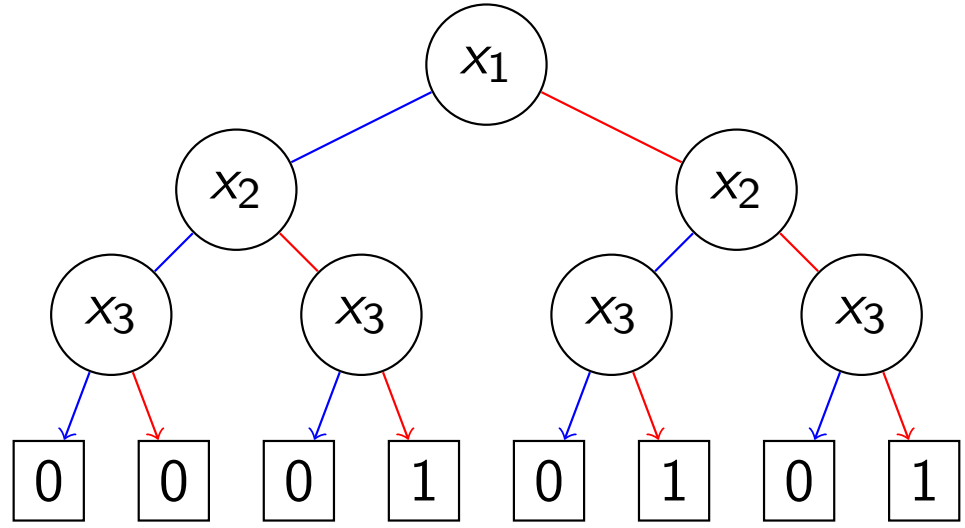
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$

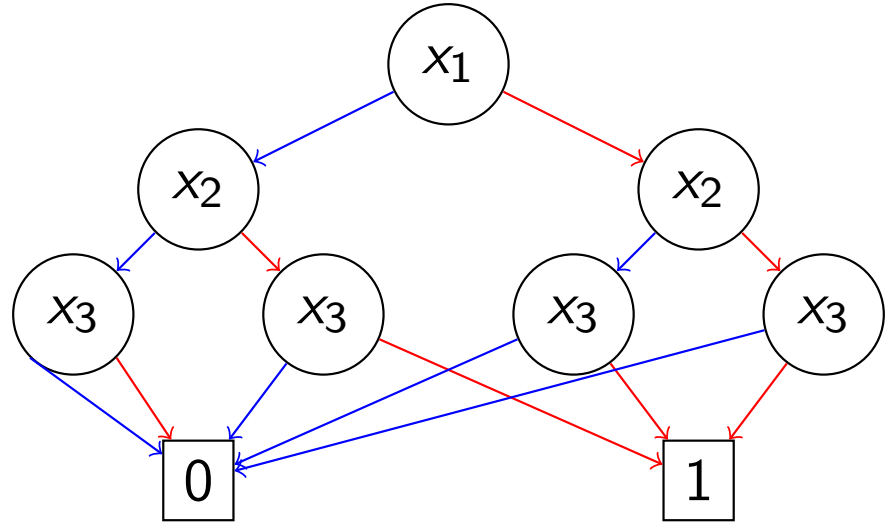
noduri *terminale*: valoarea funcției (0 sau 1)
noduri *neterminale: variabile* x_i (de care depinde funcția)
ramuri: *low*(*nod*) / *high*(*nod*) : atribuire F/T a variabilei din nod

Fixând ordinea variabilelor, arborele e unic (canonic), dar *ineficient*:
2^n combinații posibile, la fel ca tabela de adevăr

Reducerea nr. 1: Comasarea nodurilor terminale

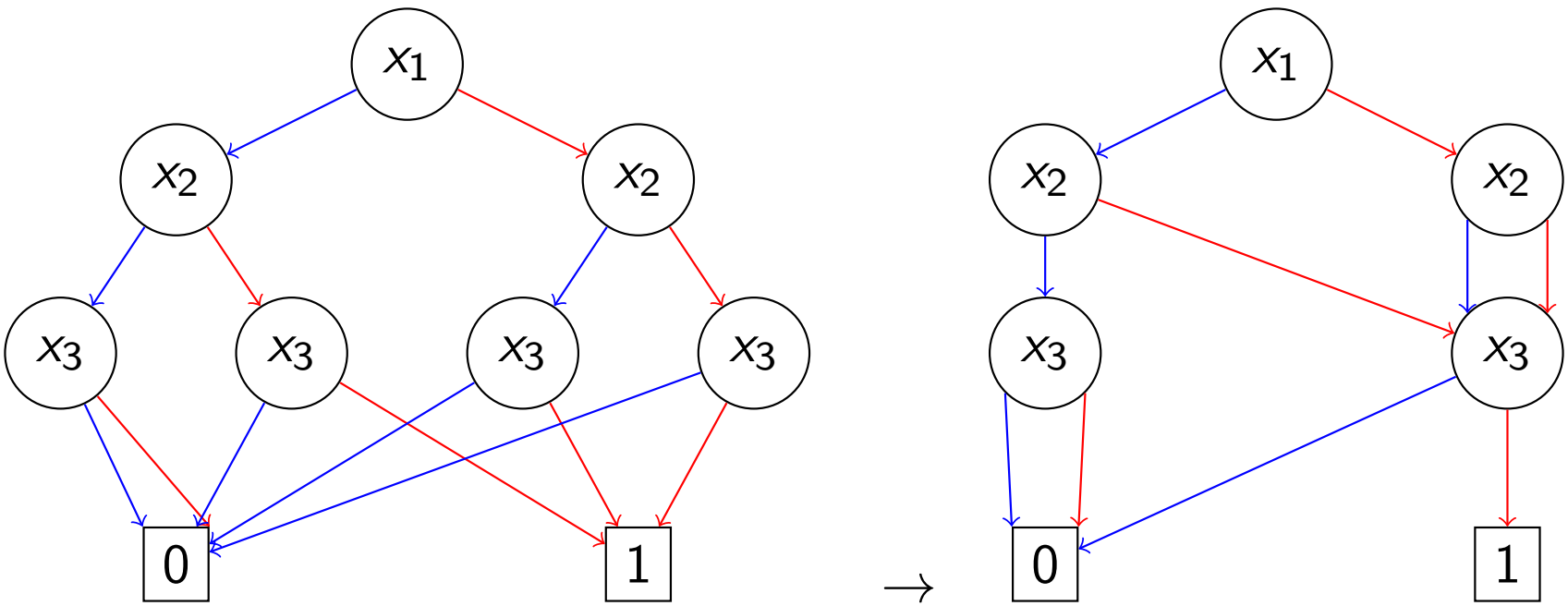
păstrăm o singură copie pentru nodurile 0 și 1





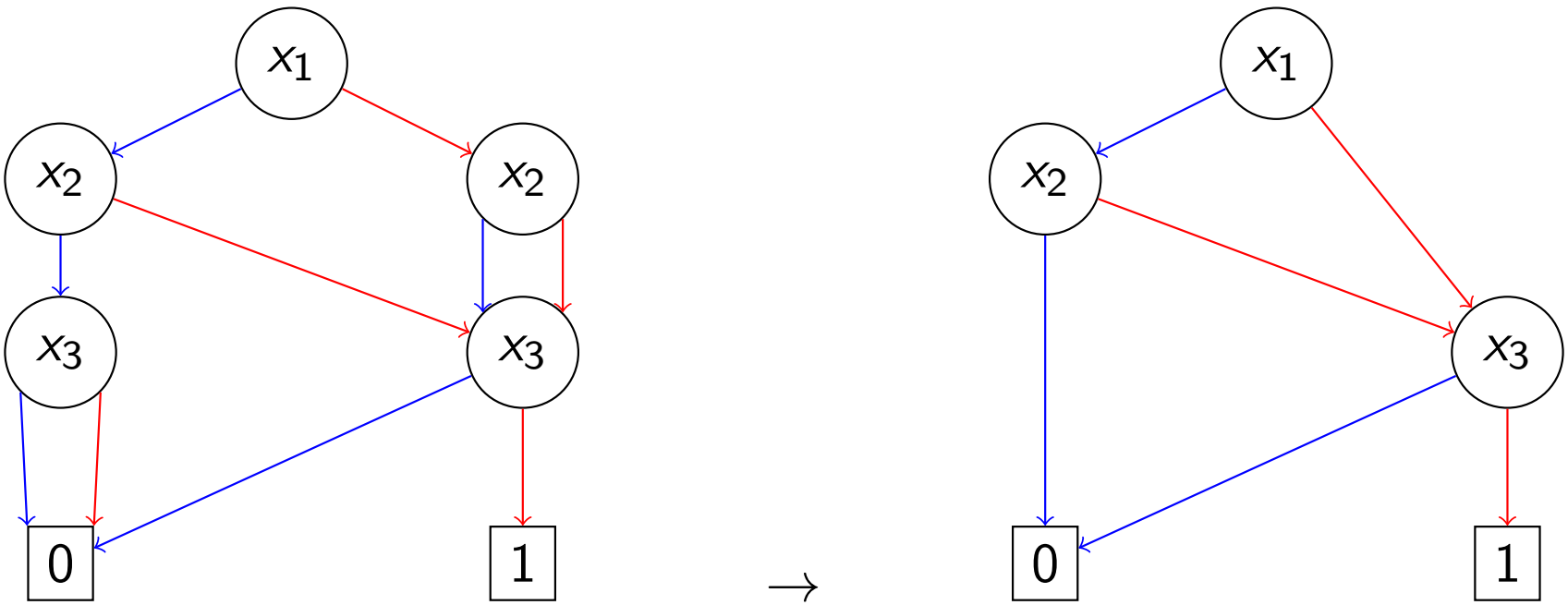
Reducerea nr. 2: Comasarea nodurilor izomorfe

Dacă $low(n_1) = low(n_2)$ și $high(n_1) = high(n_2)$, comasăm n_1 și n_2 (dacă nodurile au același rezultat pe ramura fals, și același rezultat pe ramura adevărat, ele se comportă la fel și le comasăm)



Reducerea nr. 3: Eliminarea testelor inutile

Dacă un nod dă același rezultat pe ramurile fals și adevărat, nodul poate fi eliminat



Diagrame de decizie binare

Rezultatul obținut: *binary decision diagram* (BDD)
(reprezentare introdusă de R. Bryant în 1986)

A devenit standard în industria circuitelor integrate digitale
toate companiile și programele de proiectare o folosesc

Pentru a verifica egalitatea a două funcții:
se construiesc BDD-uri pentru cele două funcții
dacă funcțiile sunt egale, se obține *același obiect* BDD
$\Rightarrow$ se verifică direct și eficient egalitatea funcțiilor

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)

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

(din φ_1 și $\varphi_1 \to \varphi_2$ deducem/inferăm 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: $(\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$
în care α, β etc. pot fi înlocuite cu *orice* formule

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)*

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: $H \vdash A_n$

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.

Consecința semantică

Reamintim: o *interpretare* e o atribuire de adevăr pentru propozițiile unei formule.
O formulă poate fi adevărată sau falsă într-o interpretare.

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

$$H \models \varphi$$

dacă orice interpretare care satisface (formulele din) H satisface φ

Pentru a stabili consecința semantică trebuie să *interpretăm* formulele (cu valori/funcții de adevăr)

$\Rightarrow$ lucrăm cu *semantica* (înțelesul) formulelor

Exemplu: $\{\alpha \vee \beta, \gamma \vee \neg\beta\} \models \alpha \vee \gamma$ Fie o interpretare v.
Cazul 1: $v(\beta) = \mathsf{T}$. Atunci $v(\alpha \vee \beta) = \mathsf{T}$ și $v(\gamma \vee \neg\beta) = v(\gamma)$.
Dacă $v(\gamma) = T$, atunci $v(\alpha \vee \gamma) = T$, deci afirmația e adevărată.
Cazul 2: $v(\beta) = \mathsf{F}$. La fel, reducem la $\{\alpha\} \models \alpha \vee \gamma$ (adevărat).

Consistență și completitudine

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

Deci, 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*.

Rezoluția (în calculul propozițional)

Rezoluția e o *regulă de inferență* care produce o nouă clauză din două clauze cu literale complementare (p și $\neg p$).

$$\frac{p \vee \alpha \qquad \neg p \vee \beta}{\alpha \vee \beta} \qquad \textit{rezoluție}$$

Clauza obținută = *rezolventul* celor două clauze în raport cu p
Exemplu: $rez_p(p \vee q \vee \neg r, \neg p \vee s) = q \vee \neg r \vee s$

Modus ponens poate fi privit ca un *caz particular de rezoluție*:

$$\frac{p \vee \mathit{false} \qquad \neg p \vee q}{\mathit{false} \vee q}$$

Rezoluția e o regulă de inferență *validă*: am văzut că

$$\{p \vee \alpha, \neg p \vee \beta\} \models \alpha \vee \beta$$

Corolar: dacă $\alpha \vee \beta$ e contradicție, la fel și $(p \vee \alpha) \wedge (\neg p \vee \beta)$.

Folosim rezoluția pentru a arăta că o formulă e o *contradicție*.
e o metodă de *refutație* (respingere a unei afirmații)

Putem *demonstra* o teoremă (tautologie) prin *reducere la absurd* arătând prin rezoluție că negația ei e o contradicție (nerealizabilă).

Cum folosim rezoluția (în calculul propozițional)

Pornim de la o formulă în formă normală conjunctivă (CNF).

Adăugăm rezolvenți, încercând să *obținem clauza vidă:*

Se formează clauze mai simple "anulând" un literal cu negația lui.
$\Rightarrow$ *reducem numărul de propoziții* simplificând formula,
și păstrând realizabilitatea (dar nu echivalența)

Alegem un literal (propoziție) l, formăm și adăugăm toți rezolvenții
dacă avem m clauze cu l și n clauze cu $\neg l$, creăm $m \cdot n$ rezolvenți
ștergem cele $m+n$ clauze inițiale (doar în logica propozițională!)

Dacă vreun rezolvent e *clauza vidă*, formula e nerealizabilă
Dacă nu mai putem crea rezolvenți (literalele au polaritate unică),
formula e realizabilă (facem T toate literalele rămase)

Problemă: numărul de clauze poate crește exponențial.
DPLL: rezoluție doar la clauze unitare + despărțire pe cazuri
(formula nu crește, dar poate încerca nr. exponențial de cazuri)

Metoda rezoluției: exemplu 1

$$
\begin{array}{ll}
\quad (a \vee \neg b \vee \neg d) & b \text{ negat} \\
\wedge\, (\neg a \vee \neg b) & b \text{ negat} \\
\wedge\, (\neg a \vee c \vee \neg d) & \\
\wedge\, (\neg a \vee b \vee c) & b \text{ pozitiv}
\end{array}
$$

Luăm o propoziție cu ambele polarități (b) și construim rezolvenții

$rez_b(a \vee \neg b \vee \neg d,\ \neg a \vee b \vee c) = a \vee \neg d \vee \neg a \vee c = \mathrm{T}$

$rez_b(\neg a \vee \neg b,\ \neg a \vee b \vee c) = \neg a \vee \neg a \vee c = \neg a \vee c$

Adăugăm noii rezolvenți (ignorăm T); eliminăm vechile clauze cu b

$$
\begin{array}{l}
\quad (\neg a \vee c \vee \neg d) \\
\wedge\, (\neg a \vee c)
\end{array}
$$

Nu mai putem crea rezolvenți. Nu avem clauza vidă.

$\Rightarrow$ formula e realizabilă, de exemplu cu $a = \mathrm{F}$. Sau cu $c = \mathrm{T}$.

Obs: nici $a = \mathrm{F}$ nici $c = \mathrm{T}$ nu sunt suficiente în formula inițială, trebuie să mai dăm o valoare și lui b (F)

Metoda rezoluției: exemplu 2

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

Aplicăm rezoluția după c, avem o singură pereche de clauze:
$rez_c(\neg b \vee c,\ \neg a \vee \neg b \vee \neg c) = \neg b \vee \neg a \vee \neg b = \neg a \vee \neg b$

Eliminăm cele două clauze cu c și adăugăm clauza nouă:

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

Aplicăm rezoluția după b:
$rez_b(\neg a \vee b, \neg a \vee \neg b) = \neg a \vee \neg a = \neg a$

Eliminăm cele două clauze cu b, adăugăm clauza nouă:

$$
\begin{aligned}
& a \\
\wedge\ & \neg a
\end{aligned}
$$

Aplicăm rezoluția după a: $rez_a(a, \neg a) = \mathrm{F}$ (clauza vidă)
Deci formula inițială e o contradicție (e nerealizabilă).