Logică și structuri discrete

Logică propozițională

Casandra Holotescu

casandra@cs.upt.ro

https://tinyurl.com/lecturesLSD

Introducere

Logica stă la baza informaticii

circuite logice: descrise în algebra booleană
Logica digitală, sem. 2

calculabilitate: ce se poate calcula algoritmic?

metode formale: demonstrarea corectitudinii programelor
eroare în sortarea Java (Timsort) corectată (2015)

inteligența artificială: cum reprezentăm și deducem cunoștințe?

testare și *securitate*: găsirea unor intrări și căi de eroare, exploatarea automată de vulnerabilități

etc.

Din istoria logicii

Aristotel (sec.4 î.e.n.): primul sistem de *logică formală* (riguroasă)

Gottfried Wilhelm Leibniz (1646-1714): *logică computațională*
raționamentele logice pot fi reduse la *calcul matematic*

George Boole (1815-1864): *The Laws of Thought*:
logica modernă, algebră booleană (*logică* și *mulțimi*)

Gottlob Frege (1848-1925): *logica simbolică clasică*
Begriffsschift: formalizare a logicii ca fundament al matematicii

Bertrand Russell (1872-1970): *Principia Mathematica*
(cu A. N. Whitehead)
formalizare încercând să elimine paradoxurile anterioare

Kurt Gödel (1906-1978): *teoremele de incompletitudine* (1931):
nu există axiomatizare consistentă și completă a aritmeticii
limitarea logicii: fie paradoxuri, fie afirmații nedemonstrabile

Exemple de specificații logice

RFC822: Standard for ARPA Internet Text Messages (e-mail)

If the "Reply-To" field exists, **then** the reply **should** go to the addresses indicated in that field **and not** to the address(es) indicated in the "From" field.

Contracte pentru funcții în limbaje de programare
Bertrand Meyer, Eiffel, 1986; acum în mai toate limbajele
incl. pentru C în anul 1, `http://c0.typesafety.net`

```
int log(int x)
//@requires x >= 1;
//@ensures \result >= 0;
//@ensures (1 << \result) <= x;
```

Logică și gândire computațională (algoritmică)

Computational logic

folosirea logicii pentru *calcule* sau *raționamente despre* calcule

Logică și gândire computațională (algoritmică)

Computational logic

folosirea logicii pentru *calcule* sau *raționamente despre* calcule

legată de *programare logică*: descrie declarativ *ce* se calculează, ordinea operatiilor (*cum*) rezultă automat

Algorithm = Logic + Control R. Kowalski, 1979

"logica" algoritmului: definiții, relații, reguli $\Rightarrow$ înțelesul
+ "control": strategiile de execuție $\Rightarrow$ eficiența

Logică și gândire computațională (algoritmică)

Computational logic

folosirea logicii pentru *calcule* sau *raționamente despre* calcule

legată de *programare logică*: descrie declarativ *ce* se calculează, ordinea operatiilor (*cum*) rezultă automat

Algorithm = Logic + Control   R. Kowalski, 1979

"logica" algoritmului: definiții, relații, reguli ⇒ înțelesul
+ "control": strategiile de execuție ⇒ eficiența

Computational thinking

"Computational thinking is the thought process involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry it out."
"... **computational thinking will be a fundamental skill** [...] used by everyone by the middle of the 21st Century."

J. Wing, VP Microsoft Research `http://socialissues.cs.toronto.edu/?p=279.html`

Logica și calculatoarele

Demonstrațiile logice se reduc la *calcule*
(*algoritmi*, *programe*)

Multe *probleme* din *informatică*
se pot reduce la *logică*
și rezolva apoi *automat*

Știm deja: Operatorii logici uzuali

NU ($\neg$), SAU ($\vee$), ȘI ($\wedge$)

```
let bisect an =
   an mod 4 = 0 && not (an mod 100 = 0) || an mod 400 = 0
```

Tabele de adevăr:

p	$\neg p$
F	T
T	F

negație $\neg$ NU
C: `!`   ML: `not`

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

disjuncție $\vee$ SAU
C/ML: `||`

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

conjuncție $\wedge$ ȘI
C/ML: `&&`

Logica propozițională

Unul din cele mai simple *limbaje* (limbaj $\Rightarrow$ putem *exprima* ceva)
putem exprima probleme prin *formule* în logică

Logica propozițională

Unul din cele mai simple *limbaje* (limbaj $\Rightarrow$ putem *exprima* ceva)
putem exprima probleme prin *formule* în logică

Discutăm:

Cum definim o *formulă logică*:
forma ei (*sintaxa*) vs. înțelesul ei (*semantica*)

Cum *reprezentăm* o formulă? pentru a opera *eficient* cu ea

Ce sunt *demonstrațiile* și *raționamentul logic* ?
cum putem demonstra? se poate demonstra (sau nega) orice?

Cum *folosim* logica pentru a opera cu alte noțiuni din informatică?
(mulțimi, relații, etc.)

Propoziții logice

O *propoziție* (logică) e o afirmație care e fie *adevărată*, fie *falsă*, dar nu ~~ambele simultan~~.

Propoziții logice

O *propoziție* (logică) e o afirmație care e fie *adevărată*, fie *falsă*, dar nu ~~ambele simultan~~.

Sunt sau nu propoziții?

$2 + 2 = 5$

$\mathrm{x} + 2 = 4$

Toate numerele prime mai mari ca 2 sunt impare.

$x^n + y^n = z^n$ nu are soluții întregi nenule pentru niciun $n > 2$

Dacă $x < 2$, atunci $x^2 < 4$

Propoziții logice

O *propoziție* (logică) e o afirmație care e fie *adevărată*, fie *falsă*, dar nu ~~ambele simultan~~.

Sunt sau nu propoziții?

$2 + 2 = 5$

$\mathrm{x} + 2 = 4$

Toate numerele prime mai mari ca 2 sunt impare.

$x^n + y^n = z^n$ nu are soluții întregi nenule pentru niciun $n > 2$

Dacă $x < 2$, atunci $x^2 < 4$

Logica ne permite să raționăm *precis*.
$\Rightarrow$ pentru aceasta trebuie să o definim *precis*
sintaxa (cum arată/e formată) și *semantica* (ce înseamnă)

Sintaxa

Sintaxa logicii propoziționale

Un *limbaj* e definit prin
- *simbolurile* sale
- și *regulile* după care combinăm corect simbolurile (*sintaxa*)

Sintaxa logicii propoziționale

Un *limbaj* e definit prin
- *simbolurile* sale
- și *regulile* după care combinăm corect simbolurile (*sintaxa*)

Simbolurile logicii propoziționale:
- *propoziții*: notate deobicei cu litere p, q, r, etc.
- *operatori* (conectori logici): negație $\neg$, implicație $\rightarrow$, paranteze ( )

Sintaxa logicii propoziționale

Un *limbaj* e definit prin
simbolurile sale
și *regulile* după care combinăm corect simbolurile (*sintaxa*)

Simbolurile logicii propoziționale:
propoziții: notate deobicei cu litere *p*, *q*, *r*, etc.
operatori (conectori logici): negație $\neg$, implicație $\rightarrow$, paranteze ( )

Formulele logicii propoziționale: definite prin *inducție structurală*
(construim formule complexe din altele mai simple)

O formulă e:
orice *propoziție* (numită și formulă atomică)
$(\neg\alpha)$ dacă α este o formulă
$(\alpha \rightarrow \beta)$ dacă α și β sunt formule

$(\alpha, \beta$ numite *subformule*)

Alți operatori (conectori) logici

Deobicei, dăm definiții *minimale* (cât mai puține cazuri)
(orice raționament ulterior trebuie făcut pe toate cazurile)

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

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

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

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

Alți operatori (conectori) logici

Deobicei, dăm definiții *minimale* (cât mai puține cazuri)
(orice raționament ulterior trebuie făcut pe toate cazurile)

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

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

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

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

Omitem parantezele redundante, definind precedența operatorilor.

Ordinea precedenței: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

Implicația e asociativă *la dreapta*! $p \rightarrow q \rightarrow r = p \rightarrow (q \rightarrow r)$

Sintaxa nu definește ce *înseamnă* o formulă. Definim *semantica* ulterior.

Sintaxa (concretă și abstractă) vs. semantică

Sintaxa: o mulțime de *reguli* care definește construcțiile unui limbaj

Sintaxa (concretă și abstractă) vs. semantică

Sintaxa: o mulțime de *reguli* care definește construcțiile unui limbaj (dacă ceva nu e construit corect nu putem să-i definim înțelesul)

Sintaxa *concretă* precizează modul *exact* de scriere.

prop $\quad \neg$ *formulă* $\quad$ *formulă* $\wedge$ *formulă* $\quad$ *formulă* $\vee$ *formulă*

Sintaxa (concretă și abstractă) vs. semantică

Sintaxa: o mulțime de *reguli* care definește construcțiile unui limbaj
(dacă ceva nu e construit corect nu putem să-i definim înțelesul)

Sintaxa *concretă* precizează modul *exact* de scriere.

prop $\quad \neg$ *formulă* $\quad$ *formulă* $\wedge$ *formulă* $\quad$ *formulă* $\vee$ *formulă*

Sintaxa *abstractă*: interesează *structura* formulei din subformule
(propoziție, negația unei formule, conjuncția/disjuncția a 2 formule)
nu contează simbolurile concrete ($\wedge$, $\vee$), scrierea infix / prefix,...

Sintaxa (concretă și abstractă) vs. semantică

Sintaxa: o mulțime de *reguli* care definește construcțiile unui limbaj
(dacă ceva nu e construit corect nu putem să-i definim înțelesul)

Sintaxa *concretă* precizează modul *exact* de scriere.

prop $\quad \neg$ *formulă* $\quad$ *formulă* $\wedge$ *formulă* $\quad$ *formulă* $\vee$ *formulă*

Sintaxa *abstractă*: interesează *structura* formulei din subformule
(propoziție, negația unei formule, conjuncția/disjuncția a 2 formule)
nu contează simbolurile concrete ($\wedge$, $\vee$), scrierea infix / prefix,...

ML: definim un tip recursiv urmărind structura (*sintaxa abstractă*):

```
type boolform = V of string | Neg of boolform
  | And of boolform * boolform | Or of boolform * boolform
```

Numele de *constructori* `V`, `And`, `Or`, etc. sunt alese de noi.

Implicația logică $\rightarrow$

$p \rightarrow q$ numită și *condițional(ă)*
p: *antecedent* (în raționamente: *ipoteză*, *premisă*)
q: *consecvent* (în raționamente: *concluzie*)

Implicația logică $\rightarrow$

$p \rightarrow q$ numită și *condițional(ă)*
p: *antecedent* (în raționamente: *ipoteză*, *premisă*)
q: *consecvent* (în raționamente: *concluzie*)

Înțelesul: *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 când p e adevărat, dar q e fals

Implicația logică $\to$

$p \to q$ numită și *condițional(ă)*
p: *antecedent* (în raționamente: *ipoteză*, *premisă*)
q: *consecvent* (în raționamente: *concluzie*)

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

Deci, $p \to q$ e fals doar când p e adevărat, dar q e fals

Tabelul de adevăr:

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{q} \\ & p \to q & \mathrm{F} & \mathrm{T} \\ \hline p & \mathrm{F} & \mathrm{T} & \mathrm{T} \\ \cline{2-4} & \mathrm{T} & \mathrm{F} & \mathrm{T} \end{array}$$

Exprimat cu conectorii uzuali: $\boxed{p \to q \;=\; \neg p \vee q}$

Negația: $\neg(p \to q) = p \wedge \neg q$

Implicația în vorbirea curentă și în logică

În limbajul natural, "dacă ... atunci" denotă adesea *cauzalitate*
dacă plouă, iau umbrela (din *cauză* că plouă)

În logica matematică, $\rightarrow$ *NU înseamnă cauzalitate*
3 e impar $\rightarrow$ 2 e număr prim implicație adevărată, T $\rightarrow$ T
(dar faptul că 2 e prim *nu e din cauză că* 3 e impar)

În demonstrații, vom folosi ipoteze *relevante* (legate de concluzie)

Implicația în vorbirea curentă și în logică

În limbajul natural, "dacă ... atunci" denotă adesea *cauzalitate*
dacă plouă, iau umbrela (din *cauză* că plouă)

În logica matematică, $\rightarrow$ *NU înseamnă cauzalitate*
3 e impar $\rightarrow$ 2 e număr prim implicație adevărată, T $\rightarrow$ T
(dar faptul că 2 e prim *nu e din cauză că* 3 e impar)

În demonstrații, vom folosi ipoteze *relevante* (legate de concluzie)

Vorbind, spunem adesea "dacă" gândind "dacă și numai dacă"
(echivalență, o noțiune mai puternică!)
Exemplu: Dacă depășesc viteza, iau amendă. (dar dacă nu?)

Implicația în vorbirea curentă și în logică

În limbajul natural, "dacă ... atunci" denotă adesea *cauzalitate*
dacă plouă, iau umbrela (din *cauză* că plouă)

În logica matematică, $\rightarrow$ *NU înseamnă cauzalitate*
3 e impar $\rightarrow$ 2 e număr prim implicație adevărată, T $\rightarrow$ T
(dar faptul că 2 e prim *nu e din cauză că* 3 e impar)

În demonstrații, vom folosi ipoteze *relevante* (legate de concluzie)

Vorbind, spunem adesea "dacă" gândind "dacă și numai dacă"
(echivalență, o noțiune mai puternică!)
Exemplu: Dacă depășesc viteza, iau amendă. (dar dacă nu?)

ATENȚIE: *fals implică orice!* (vezi tabelul de adevăr)
$\Rightarrow$ un raționament cu o verigă falsă poate duce la *orice concluzie*
$\Rightarrow$ un paradox ($A \wedge \neg A$) distruge încrederea într-un sistem logic

Implicație: contrapozitiva, inversa, reciproca

Fiind dată o implicație $A \to B$, definim:

reciproca: $B \to A$

inversa: $\neg A \to \neg B$

contrapozitiva: $\neg B \to \neg A$

Implicație: contrapozitiva, inversa, reciproca

Fiind dată o implicație $A \rightarrow B$, definim:

reciproca: $B \rightarrow A$

inversa: $\neg A \rightarrow \neg B$

contrapozitiva: $\neg B \rightarrow \neg A$

Contrapozitiva e *echivalentă* cu formula inițială (directa).

$$A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A$$

Inversa e echivalentă cu reciproca.

$$B \rightarrow A \Leftrightarrow \neg A \rightarrow \neg B$$

$A \rightarrow B$ ~~NU e echivalent~~ cu $B \rightarrow A$ (reciproca)

Semantica

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

O *funcție de adevăr* v atribuie oricărei formule
o *valoare de adevăr* $\in \{\mathrm{T}, \mathrm{F}\}$ astfel încât:

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

O *funcție de adevăr* v atribuie oricărei formule
o *valoare de adevăr* $\in \{\mathrm{T}, \mathrm{F}\}$ astfel încât:

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

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

O *funcție de adevăr* v atribuie oricărei formule
o *valoare de adevăr* $\in \{\mathsf{T}, \mathsf{F}\}$ astfel încât:

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

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

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

O *funcție de adevăr* v atribuie oricărei formule
o *valoare de adevăr* $\in \{\mathrm{T}, \mathrm{F}\}$ astfel încât:

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

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

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

Semantica unei formule: funcții de adevăr

Definim riguros cum calculăm valoarea de adevăr a unei formule
= dăm o *semantică* (înțeles) formulei (formula=noțiune *sintactică*)

O *funcție de adevăr* v atribuie oricărei formule
o *valoare de adevăr* $\in \{\mathrm{T}, \mathrm{F}\}$ astfel încât:

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

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

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

Exemplu: $v((a \rightarrow b) \rightarrow c)$ pentru $v(a) = \mathrm{T}$, $v(b) = \mathrm{F}$, $v(c) = \mathrm{T}$
avem $v(a \rightarrow b) = \mathrm{F}$ pentru că $v(a) = \mathrm{T}$ și $v(b) = \mathrm{F}$ (cazul 1)
și atunci $v((a \rightarrow b) \rightarrow c) = \mathrm{T}$ (cazul 2: premisă falsă)

Interpretări ale unei formule

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

Interpretări ale unei formule

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

O interpretare *satisface* o formulă dacă o evaluează la T.

Spunem că interpretarea e un *model* pentru formula respectivă.

Exemplu: pentru formula $a \wedge (\neg b \vee \neg c) \wedge (\neg a \vee c)$

Interpretări ale unei formule

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

O interpretare *satisface* o formulă dacă o evaluează la T.

Spunem că interpretarea e un *model* pentru formula respectivă.

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

Interpretări ale unei formule

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

O interpretare *satisface* o formulă dacă o evaluează la T.

Spunem că interpretarea e un *model* pentru formula respectivă.

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

Interpretări ale unei formule

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

O interpretare *satisface* o formulă dacă o evaluează la T.

Spunem că interpretarea e un *model* pentru formula respectivă.

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

O formulă poate fi:
tautologie (*validă*): adevărată în *toate* interpretările
realizabilă (en. *satisfiable*): adevărată în *cel puțin o* interpretare
contradicție (nerealizabilă): nu e adevărată în *nicio* interpretare
contingență: adevărată în unele interpretări, falsă în altele
(nici tautologie, nici contradicție)

Tabelul de adevăr

Tabelul de adevăr prezintă valoarea de adevăr a unei formule în *toate interpretările posibile*

2^n interpretări dacă formula are n propoziții

a	b	c	a $\rightarrow$(b $\rightarrow$c)
F	F	F	T
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

a	b	c	(a $\rightarrow$b) $\rightarrow$c
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

Două formule sunt *echivalente* dacă au *același tabel de adevăr*

Două formula ϕ și ψ sunt echivalente dacă $\phi \leftrightarrow \psi$ e o tautologie

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Asociativitate: $(A \vee B) \vee C = A \vee (B \vee C)$ și
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Asociativitate: $(A \vee B) \vee C = A \vee (B \vee C)$ și
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Distributivitate: $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ și
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Asociativitate: $(A \vee B) \vee C = A \vee (B \vee C)$ și
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Distributivitate: $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ și
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Identitate: există două valori (aici F și T) astfel ca:
$$A \vee \mathrm{F} = A \qquad A \wedge \mathrm{T} = A$$

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Asociativitate: $(A \vee B) \vee C = A \vee (B \vee C)$ și
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Distributivitate: $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ și
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Identitate: există două valori (aici F și T) astfel ca:
$$A \vee \mathrm{F} = A \qquad A \wedge \mathrm{T} = A$$

Complement: $A \vee \neg A = \mathrm{T} \qquad A \wedge \neg A = \mathrm{F}$

Algebra Booleană

Pe mulțimi, $\cup$, $\cap$ și complementul formează o algebră booleană.
Tot o algebră booleană formează în logică și $\wedge$, $\vee$ și $\neg$:

Comutativitate: $A \vee B = B \vee A \qquad A \wedge B = B \wedge A$

Asociativitate: $(A \vee B) \vee C = A \vee (B \vee C)$ și
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Distributivitate: $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ și
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Identitate: există două valori (aici F și T) astfel ca:

$$A \vee \mathrm{F} = A \qquad A \wedge \mathrm{T} = A$$

Complement: $A \vee \neg A = \mathrm{T} \qquad A \wedge \neg A = \mathrm{F}$

Alte proprietăți (pot fi deduse din cele de mai sus):

Idempotență: $A \wedge A = A \qquad A \vee A = A$

Absorbție: $A \vee (A \wedge B) = A \qquad A \wedge (A \vee B) = A$
$\neg A \vee (A \wedge B) = \neg A \vee B$ *simplifică formula!*

Exemple de tautologii

$a \vee \neg a$ $\qquad$ $\neg\neg a \leftrightarrow a$

Regulile lui de Morgan

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

$$(a \to b) \wedge (\neg a \to c) \;\leftrightarrow\; (a \wedge b) \vee (\neg a \wedge c)$$

$$a \to (b \to c) \;\leftrightarrow\; (a \wedge b) \to c$$

$(p \to q) \wedge p \;\to\; q$ $\qquad$ $(p \to q) \wedge \neg q \;\to\; \neg p$

$p \wedge q \;\to\; p$ $\qquad$ $(p \vee q) \wedge \neg p \;\to\; q$

$$(p \to q) \;\to\; q$$

$$(p \to q) \wedge (q \to r) \;\to\; (p \to r)$$

Reprezentarea formulelor boolene

E bine ca o reprezentare să fie:

- *canonică* (un obiect să fie reprezentat într-un *singur* fel)
  avem egalitate dacă și numai dacă au aceeași reprezentare
- *simplă* și *compactă* (ușor de implementat / stocat)
- *ușor de prelucrat* (algoritmi simpli / eficienți)

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

Diagrame de decizie binară

Descompunerea după o variabilă

Fixând valoarea unei variabile într-o formulă, aceasta se simplifică.

Fie $f = (a \vee b) \wedge (a \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$.

Dăm valori lui a:

$f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c$
$f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$

Descompunerea după o variabilă

Fixând valoarea unei variabile într-o formulă, aceasta se simplifică.

Fie $f = (a \vee b) \wedge (a \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$.

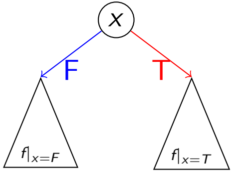
Dăm valori lui a:

$f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c$
$f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$

Descompunerea Boole (sau *Shannon*)

$$\boxed{f = x \wedge f|_{x=T} \vee \neg x \wedge f|_{x=F}}$$

exprimă o funcție booleană f
în raport cu o variabilă x



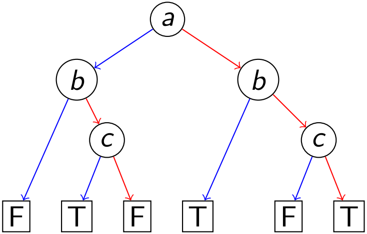
În program: if-then-else după variabila respectivă

```
let f a b c = if a then not b || c else b && not c
```

Arbore de decizie binar

Continuând pentru subformule, obținem un *arbore de decizie*: dând valori la variabile ($a = T$, $b = F$, $c = T$) și *urmând ramurile* respective, obținem valoarea funcției (T/F, sau $0/1$)

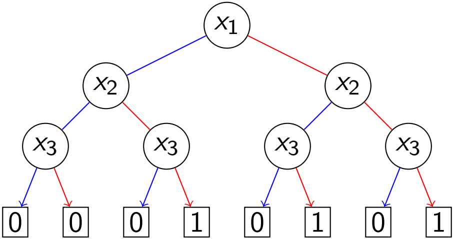
$f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c$
$f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$




```
if a then if b then c
          else true
else
  if b then if c then false
            else true
  else false
```

Fixând ordinea variabilelor, arborele e unic (canonic), dar *ineficient*: 2^n combinații posibile, ca tabelul de adevăr (deși e mai compact)

De la arbore la diagramă de decizie



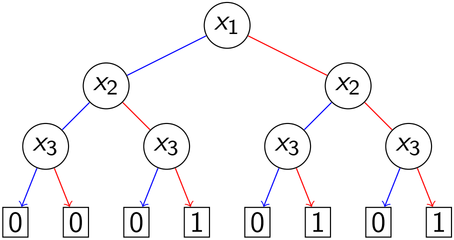
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
de ex. $f(T, F, T) = T$, $f(F, T, F) = F$, etc.

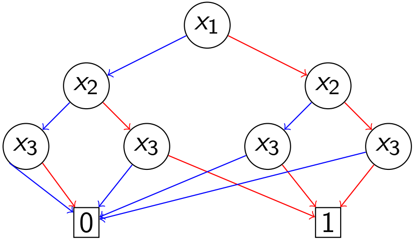
noduri *terminale*: valoarea funcției (0 sau 1, adică F sau T)
noduri *neterminale: variabile* x_i (de care depinde funcția)
ramuri: *low*(*nod*) / *high*(*nod*) : atribuire F/T a variabilei din nod

Definim 3 reguli de transformare pentru o formă mai compactă, *diagrama de decizie binară*.

Reducerea nr. 1: Comasarea nodurilor terminale

Păstrăm o singură copie pentru nodurile 0 și 1

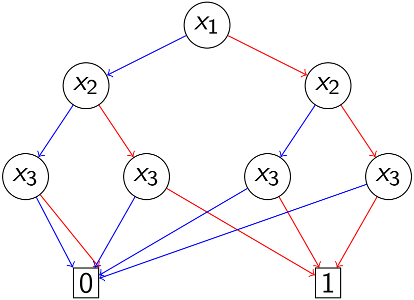




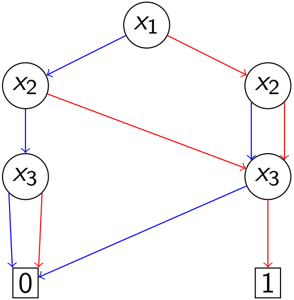
Reducerea nr. 2: Comasarea nodurilor izomorfe

Dacă $low(n_1) = low(n_2)$ și $high(n_1) = high(n_2)$, comasăm n_1 și n_2

dacă au același rezultat pe ramura fals, și același rezultat pe ramura adevărat, nodurile dau aceeași valoare

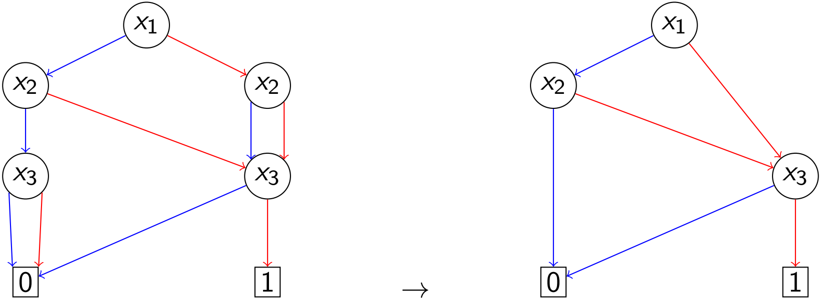


$\rightarrow$

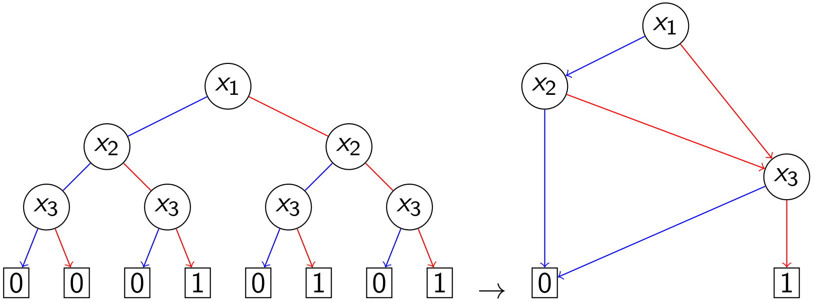


Reducerea nr. 3: Eliminarea testelor inutile

Eliminăm nodurile cu același rezultat pe ramurile fals și adevărat



De la arbore la diagramă de decizie binară



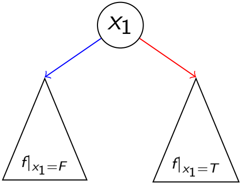
arbore de decizie binar *diagramă de decizie binară*

Cele trei transformări sunt folosite pentru a *defini* o BDD.
În practică, vrem să *evităm* arborele de decizie, fiind prea mare.
Aplicăm *direct* descompunerea funcției după o variabilă.

Cum construim practic o BDD

În practică, NU ~~pornim de la arborele binar complet~~.

Construim o BDD direct *recursiv*, *descompunând după o variabilă*:



$f = x_1 \wedge f|_{x_1=T} \vee \neg x_1 \wedge f|_{x_1=F}$

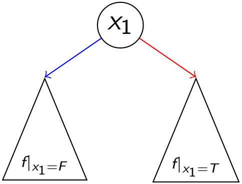
construim $f|_{x_1=T}$ și $f|_{x_1=F}$

apoi *comasăm* eventuale noduri *comune* între cele două părți

Cum construim practic o BDD

În practică, NU ~~pornim de la arborele binar complet~~.

Construim o BDD direct *recursiv*, *descompunând după o variabilă*:



$f = x_1 \wedge f|_{x_1=T} \vee \neg x_1 \wedge f|_{x_1=F}$

construim $f|_{x_1=T}$ și $f|_{x_1=F}$

apoi *comasăm* eventuale noduri *comune* între cele două părți

BDD-urile sunt folosite în practic toate programele de proiectare pentru circuite integrate

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

Exemplu pentru construirea de BDD

$f(x_1, x_2, x_3) = (\neg x_1 \land x_2 \land x_3) \lor (x_1 \land \neg x_2 \land x_3) \lor (x_1 \land x_2 \land x_3)$

Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Exemplu pentru construirea de BDD

$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$),

Exemplu pentru construirea de BDD

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

Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$), altfel continuăm *recursiv*, alegând *o nouă variabilă*:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$ $\qquad$ $f|_{x_1=T} = x_3$

$f_1|_{x_2=F} = \mathrm{F}$ $\qquad$ $f_1|_{x_2=T} = x_3$

$\boxed{0}$

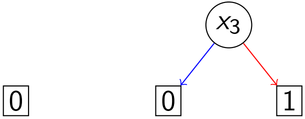
Exemplu pentru construirea de BDD

$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$), altfel continuăm *recursiv*, alegând *o nouă variabilă*:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$ $\qquad\qquad$ $f|_{x_1=T} = x_3$

$f_1|_{x_2=F} = \mathrm{F}$ $\qquad$ $f_1|_{x_2=T} = x_3$



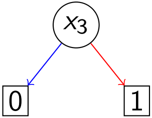
Exemplu pentru construirea de BDD

$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

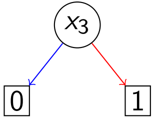
Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$), altfel continuăm *recursiv*, alegând *o nouă variabilă*:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$ $\qquad$ $f|_{x_1=T} = x_3$

$f_1|_{x_2=F} = \text{F}$ $\qquad$ $f_1|_{x_2=T} = x_3$



$\boxed{0}$



$\downarrow$

Exemplu pentru construirea de BDD

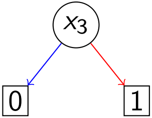
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$), altfel continuăm *recursiv*, alegând *o nouă variabilă*:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$

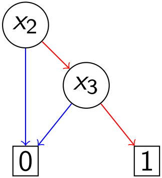
$f_1|_{x_2=F} = \text{F}$ $\qquad$ $f_1|_{x_2=T} = x_3$

$\boxed{0}$

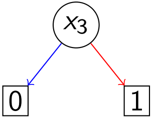


Adăugăm nodul cu decizia după x_2

$\downarrow$



$f|_{x_1=T} = x_3$



$\downarrow$

Exemplu pentru construirea de BDD

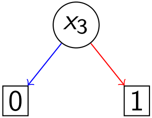
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$

Alegem o variabilă: x_1. Calculăm $f|_{x_1=F}$ și $f|_{x_1=T}$

Construim BDD pentru cele două funcții: direct, dacă sunt simple (T, F, p, $\neg p$), altfel continuăm *recursiv*, alegând *o nouă variabilă*:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$

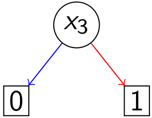
$f|_{x_1=T} = x_3$



$\downarrow$

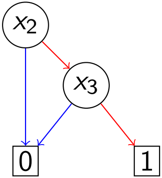
$f_1|_{x_2=F} = \text{F}$

$f_1|_{x_2=T} = x_3$



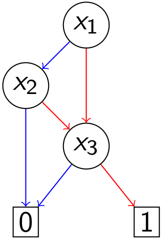
Adăugăm nodul cu decizia după x_2

$\downarrow$



Adăugăm decizia după x_1

$\longrightarrow$



Remarcăm că diagrama cu x_3 e comună și păstrăm o singură copie

Forma normală conjunctivă

Forma normală conjunctivă (conjunctive normal form)

folosită pentru a determina dacă o formulă e *realizabilă* (poate fi T)

Def: *Forma normală conjunctivă*
= *conjuncție* $\wedge$ de *clauze*
clauză = *disjuncție* $\vee$ de *literali*
literal = propoziție sau negația ei (p sau $\neg p$)

$(a \vee \neg b \vee \neg d)$	clauză
$\wedge (\neg a \vee \neg b)$	clauză
$\wedge (\neg a \vee c \vee \neg d)$	...
$\wedge (\neg a \vee b \vee c)$	clauză

Similar: forma normală *disjunctivă* (disjuncție de conjuncții)

Forma normală conjunctivă (conjunctive normal form)

folosită pentru a determina dacă o formulă e *realizabilă* (poate fi T)

Def: *Forma normală conjunctivă*	$(a \vee \neg b \vee \neg d)$	clauză
$=$ *conjuncție* $\wedge$ de *clauze*	$\wedge\, (\neg a \vee \neg b)$	clauză
clauză $=$ *disjuncție* $\vee$ de *literali*	$\wedge\, (\neg a \vee c \vee \neg d)$	...
literal $=$ propoziție sau negația ei (p sau $\neg p$)	$\wedge\, (\neg a \vee b \vee c)$	clauză

Similar: forma normală *disjunctivă* (disjuncție de conjuncții)

Transformarea în formă normală conjunctivă

1) ducem (repetat) negația înăuntru *regulile lui de Morgan*

$$\neg(A \vee B) = \neg A \wedge \neg B \qquad \neg(A \wedge B) = \neg A \vee \neg B$$

2) ducem (repetat) disjuncția înăuntru *distributivitate*

$$(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C)$$

Exemplu: forma normală conjunctivă

Lucrăm *din exterior* – evităm muncă inutilă

1) ducem *negațiile înăuntru* până la propoziții r. *de Morgan*

   dubla negație dispare $\neg\neg A = A$

   înlocuim implicațiile dinspre exterior când ajungem la ele

   $p \to q = \neg p \vee q \qquad \neg(p \to q) = p \wedge \neg q$

Exemplu: forma normală conjunctivă

Lucrăm *din exterior* – evităm muncă inutilă

1) ducem *negațiile înăuntru* până la propoziții r. *de Morgan*

   dubla negație dispare $\neg\neg A = A$

   înlocuim implicațiile dinspre exterior când ajungem la ele

   $p \rightarrow q = \neg p \vee q \qquad \neg(p \rightarrow q) = p \wedge \neg q$

2) ducem disjuncția $\vee$ înăuntrul conjuncției $\wedge$ *distributivitate*

$$
\begin{aligned}
& \neg(\ (r \vee \neg(p \rightarrow (q \wedge r))) \vee (p \wedge q)\ ) \\
= \ & \neg(r \vee \neg(p \rightarrow (q \wedge r))) \ \wedge \ \neg(p \wedge q) \\
= \ & \neg r \wedge (p \rightarrow (q \wedge r)) \ \wedge \ (\neg p \vee \neg q) \\
= \ & \neg r \wedge (\neg p \vee (q \wedge r)) \ \wedge \ (\neg p \vee \neg q) \\
= \ & \neg r \wedge (\neg p \vee q) \wedge (\neg p \vee r) \ \wedge \ (\neg p \vee \neg q)
\end{aligned}
$$

Exemplu 2: forma normală conjunctivă

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

Transformarea poate *crește exponențial* dimensiunea formulei:

$$\begin{aligned}
& (a \wedge b \wedge c) \vee (p \wedge q \wedge r) \\
&= (a \vee (p \wedge q \wedge r)) \wedge (b \vee (p \wedge q \wedge r)) \wedge (c \vee (p \wedge q \wedge r)) \\
&= (a \vee p) \wedge (a \vee q) \wedge (a \vee r) \wedge (b \vee p) \wedge (b \vee q) \wedge (b \vee r) \\
&\wedge (c \vee p) \wedge (c \vee q) \wedge (c \vee r)
\end{aligned}$$

În practică, se introduc propoziții auxiliare $\Rightarrow$ crește doar liniar