Logică și structuri discrete

Logică propozițională

Marius Minea
marius@cs.upt.ro

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

31 octombrie 2016

Logica stă la baza informaticii

circuite logice: descrise în algebra booleană

calculabilitate: ce se poate calcula algoritmic?

metode formale: demonstrarea corectitudinii programelor

inteligența artificială: cum reprezentăm și deducem cunoștințe?

baze de date relaționale

etc.

Exemple de specificații

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;
```

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

Logică și gândire computațională

Computational logic

folosirea logicii pentru a *efectua* sau *raționa despre* calcule

programare logică: descrie declarativ *ce*, rezultă automat *cum*

logica ne permite să *verificăm* programele scrise

Algorithm = Logic + Control R. Kowalski, 1979

componenta logică: cunoștințele folosite (înțelesul algoritmului)

+ componenta de control: strategiile de execuție ⇒ eficiența

Computational thinking

"Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out."

"... computational thinking will be a fundamental skill [...] used by everyone by the middle of the 21st Century."

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

Operatori logici uzuali

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

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

Tabele de adevăr:

p	$\neg p$
F	T
T	F

negație $\neg$ NU

C: `!`   ML: `not`

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

disjuncție $\vee$ SAU

C/ML: `||`

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

conjuncție $\wedge$ ȘI

C/ML: `&&`

Logica propozițională

Unul din cele mai simple *limbaje* (limbaj $\Rightarrow$ putem *exprima* ceva)
așa cum codificăm numere, etc. în *biți*
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

Cum *folosim* logica pentru a lucra cu alte noțiuni din informatică?
(mulțimi, relații, automate)

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

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 matematică ne permite să raționăm *precis*.
$\Rightarrow$ trebuie să definim *precis* ce e *logica propozițională*
sintaxa (cum arată/e formată) și *semantica* (ce înseamnă)

Sintaxa logicii propoziționale

Un *limbaj* e definit prin *simbolurile* sale și prin *regulile* după care le combinăm corect.

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*,
o formulă complexă e construită din formule mai simple:
orice *propoziție* (numită și formulă atomică)
$(\neg\alpha)$ dacă α este o formulă
$(\alpha \rightarrow \beta)$ dacă α și β sunt formule
(α, β numite *subformule*)

Omitem parantezele redundante, considerând $\neg$ mai prioritar ca $\rightarrow$

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)

Sintaxa (concretă și abstractă) vs. semantică

Sintaxa: o mulțime de reguli care definește construcțiile unui limbaj
aici: *formulele* logicii propoziționale
NU spune *ce înseamnă* o formulă

Semantica: definește înțelesul unei construcții (unui limbaj)

Sintaxa *concretă* precizează modul *exact* de scriere. O formulă e:
prop $\neg$ *formulă* *formulă* $\wedge$ *formulă* sau *formulă* $\vee$ *formulă*

Sintaxa *abstractă*: interesează *structura* formulei din subformule:
propoziție, negația unei formule, conjuncția/disjuncția a 2 formule

În ML, putem definim un tip recursiv urmărind *sintaxa abstractă*:

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

Numele de *constructori* `And`, `Or`, etc. sunt alese de noi.
Tipul reprezintă *structura* formulelor, nu simboluri concrete ($\wedge$, $\vee$),
nici scrierea infix sau prefix, etc.

Implicația logică $\rightarrow$

$p \rightarrow q$ numită și *condițional(ă)*
p: *antecedent* (sau *ipoteză*)
q: *consecvent* (sau *concluzie*)

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
(dacă p, atunci q ar trebui să fie adevărat)

Tabelul de adevăr:

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

Exprimat cu alți conectori: $\boxed{p \rightarrow q \; = \; \neg p \vee q}$

Implicația în vorbirea curentă și în logică

În limbajul natural, "dacă ... atunci" denotă adesea *cauzalitate*
dacă plouă, iau umbrela

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.

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

Despre implicație

Fiind dată o implicație $p \rightarrow q$, definim:

reciproca: $q \rightarrow p$

inversa: $\neg p \rightarrow \neg q$

contrapozitiva: $\neg q \rightarrow \neg p$

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

$p \rightarrow q$ ~~NU e echivalent~~ cu $q \rightarrow p$ (reciproca)

Calculul în logică: funcții de adevăr

Formula e o noțiune *sintactică*.
Valoarea de adevăr e o noțiune *semantică*.

Definim riguros cum calculăm valoarea de adevăr a unei formule.

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

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

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

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

Exemplu: fie $v(a) = T$, $v(b) = F$, $v(c) = T$
putem calcula v pentru orice formulă cu propoziții din $\{a, b, c\}$

$v((a \to b) \to c)$:
avem $v(a \to b) = F$ pentru că $v(a) = T$ și $v(b) = F$ (cazul 1)
și atunci $v((a \to b) \to c) = T$ (cazul 2: $v(a \to b) = F$)

Interpretări ale unei formule

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

O intrepretare *satisface* o formulă dacă o evaluează la T.
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 $v(a) = \text{T}, v(b) = \text{F}, v(c) = \text{T}$ o satisface
interpretarea $v(a) = \text{T}, v(b) = \text{T}, v(c) = \text{T}$ 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
- *contingentă*: adevărată în unele interpretări, falsă în altele

(nici tautologie, nici contradicție)

Tabela de adevăr

Tabela 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 →(b →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 →b) →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

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

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

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

$$(p \rightarrow q) \wedge p \ \rightarrow \ q$$

$$(p \rightarrow q) \wedge \neg q \ \rightarrow \ \neg p$$

$$p \wedge q \ \rightarrow \ p$$

$$(p \rightarrow q) \ \rightarrow \ q$$

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

$$(p \rightarrow q) \wedge (q \rightarrow r) \ \rightarrow \ (p \rightarrow r)$$

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: $A \vee (A \wedge B) = A \qquad A \wedge (A \vee B) = A$
$\neg A \vee (A \wedge B) = \neg A \vee B$ (din distributivitate și complement)

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)

Descompunerea după o variabilă

Fixând valoarea unei variabile într-o formulă, aceasta se simplifică:
Fie $f = (a \vee b) \wedge (a \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$.
$f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c \qquad f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$

Descompunerea Boole (sau *Shannon*) exprimă o funcție booleană f în raport cu o variabilă x: $\qquad f = x \wedge f|_{x=T} \vee \neg x \wedge f|_{x=F}$

În program (ML), am scrie

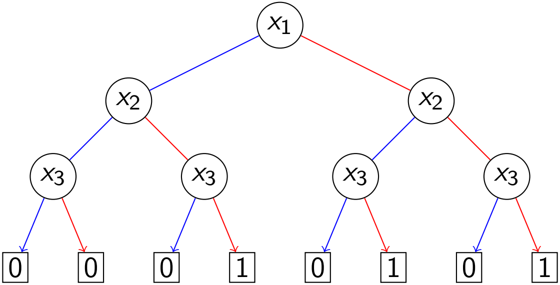
```
let f = if a then not b || c else b && not c
```

Continuând pentru cele două subformule, obținem în final un *arbore de decizie*: dând valori la variabile ($a = T$, $b = F$, $c = T$) și *urmând ramurile* respective, ajungem la valoarea funcției (T / F)

E o reprezentare echivalentă *mai compactă* decât tabelul de adevăr, ceea ce e util în practică

Simplificând *arborele* de decizie obținem o *diagramă* de decizie (graf)

Arbori de decizie binari



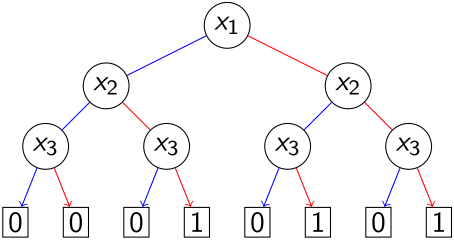
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
de ex. $f(T, F, T) = T$, $f(F, T, F) = F$, etc.

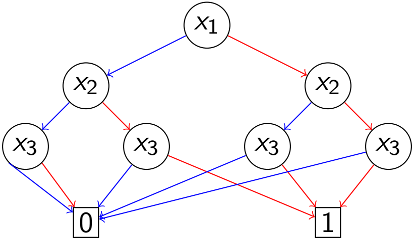
noduri *terminale*: valoarea funcției (0 sau 1, adică F sau T)
noduri *neterminale: variabile* x_i (de care depinde funcția)
ramuri: *low*(*nod*) / *high*(*nod*) : atribuire F/T a variabilei din nod

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

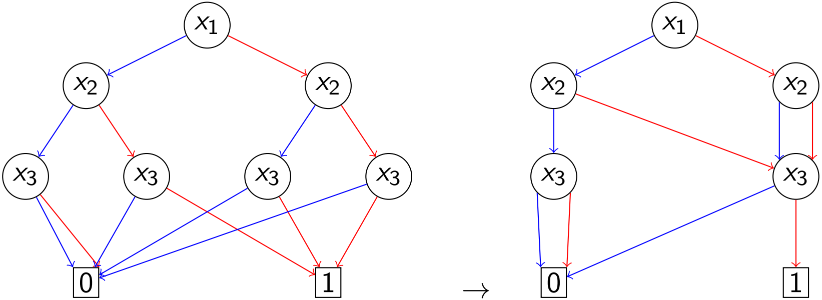
păstrăm o singură copie pentru nodurile 0 și 1





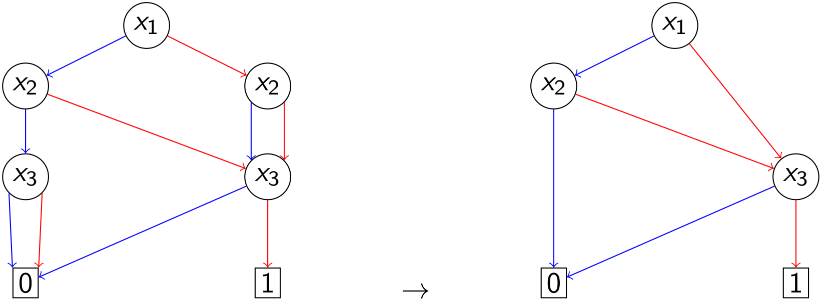
Reducerea nr. 2: Comasarea nodurilor izomorfe

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)

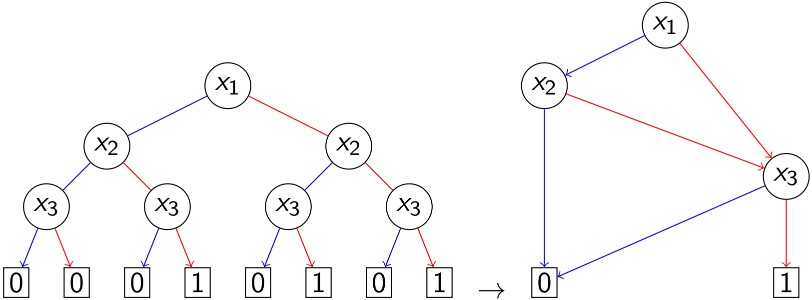


Reducerea nr. 3: Eliminarea testelor inutile

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



De la arbore la diagramă de decizie binară



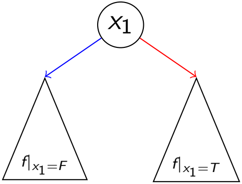
arbore de decizie binar

diagramă de decizie binară

Diagrame de decizie binare

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

Putem s-o construim recursiv, descompunând după o variabilă:



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

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

apoi comasând eventuale noduri comune între cele două părți

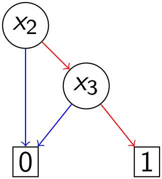
A devenit standard în industria circuitelor integrate digitale
toate companiile și programele de proiectare o folosesc

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

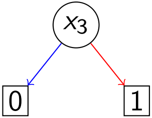
Construim mai simplu o BDD pentru o funcție

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

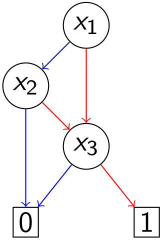
Construim BDD-urile pentru cele două funcții (direct, dacă sunt simple, altfel continuăm recursiv, alegând următoarea variabilă)



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



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



Adăugăm nodul cu x_1 deasupra.
Remarcăm că diagrama cu x_3 e comună și păstrăm o singură copie

Forma normală conjunctivă (conjunctive normal form)

formula = *conjuncție* $\wedge$ de *clauze*
clauză = *disjuncție* $\vee$ de *literali*
literal = propoziție sau negația ei

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

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

Transformarea unei formule în formă normală conjunctivă

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

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

ducem (repetat) disjuncția înăuntru (*distributivitate*)

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

Abordarea naivă poate *crește exponențial* dimensiunea formulei:

$(p_1 \wedge p_2 \wedge p_3) \vee (q_1 \wedge q_2 \wedge q_3) =$
$(p_1 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_2 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_3 \vee (q_1 \wedge q_2 \wedge q_3))$
$= (p_1 \vee q_1) \wedge (p_1 \vee q_2) \wedge (p_1 \vee q_3) \wedge (p_2 \vee q_1) \wedge (p_2 \vee q_2) \wedge (p_2 \vee q_3)$
$\wedge (p_3 \vee q_1) \wedge (p_3 \vee q_2) \wedge (p_3 \vee q_3)$

Transformarea Tseitin

Dă o formulă realizabilă *dacă și numai dacă* cea inițială e realizabilă
echirealizabilă (en. *equisatisfiable*)

Dimensiunea formulei rezultante e *liniară* în cea a formulei inițiale

Pentru fiecare operator introducem *o nouă propoziție*
reprezintă subformula calculată de acel operator

Scriem (în CNF) că noua propoziție e *echivalentă* cu subformula
(implicație în ambele sensuri)

Obținem regulile de transformare pentru cei trei operatori

formula	$p \leftrightarrow$ formula	rescriere in CNF
$\neg A$	$(\neg A \rightarrow p) \wedge (p \rightarrow \neg A)$	$(A \vee p) \wedge (\neg A \vee \neg p)$
$A \wedge B$	$(A \wedge B \rightarrow p) \wedge (p \rightarrow A \wedge B)$	$(\neg A \vee \neg B \vee p) \wedge (A \vee \neg p) \wedge (B \vee \neg p)$
$A \vee B$	$(p \rightarrow A \vee B) \wedge (A \vee B \rightarrow p)$	$(A \vee B \vee \neg p) \wedge (\neg A \vee p) \wedge (\neg B \vee p)$

Rezultă o formulă cu mai multe propoziții $\Rightarrow$ *nu e echivalentă*
dar e realizabilă dacă și numai dacă formula inițială e realizabilă
deci o putem folosi în verificarea realizabilității

Transformarea Tseitin: exemplu

Numerotăm operatorii din formulă: $(a \overset{1}{\wedge} \neg b) \vee \neg(c \overset{2}{\wedge} d)$

pentru simplitate, nu e nevoie să numerotăm negațiile
și nici conectorii la nivelul final $(...) \vee (...)$

Introducem propozițiile: $p_1 \leftrightarrow a \overset{1}{\wedge} \neg b$, $p_2 \leftrightarrow c \overset{2}{\wedge} d$.

Scriem relațiile intrare-ieșire pentru fiecare operator în parte
și adăugăm formula pentru operatorul care dă rezultatul.
Legăm toate acest relații prin conjuncție:

$(\neg a \vee b \vee p_1) \wedge (a \vee \neg p_1) \wedge (\neg b \vee \neg p_1)$ p_1 reprezintă $a \wedge \neg b$

$\wedge (\neg c \vee \neg d \vee p_2) \wedge (c \vee \neg p_2) \wedge (d \vee \neg p_2)$ p_2 reprezintă $c \wedge d$

$\wedge (p_1 \vee \neg p_2)$ vrem ca toată formula să fie adevărată

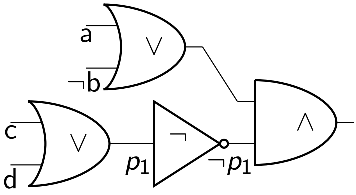
Putem transforma direct conjuncții/disjuncții multiple

ȘI $(\neg A_1 \vee \neg A_2 \vee \neg A_3 \vee p) \wedge (A_1 \vee \neg p) \wedge (A_2 \vee \neg p) \wedge (A_3 \vee \neg p)$

ȘAU $(A_1 \vee A_2 \vee A_3 \vee \neg p) \wedge (\neg A_1 \vee p) \wedge (\neg A_2 \vee p) \wedge (\neg A_3 \vee p)$

Transformarea Tseitin: formula ca circuit logic

$(a \vee \neg b) \wedge \neg(c \overset{1}{\vee} d)$
intrarea fiecărei porți: propoziție nouă
nu trebuie numerotat $\vee$ sub $\wedge$
și nici operatorul final $\wedge$
Avem un singur operator de exprimat:
$(a \vee \neg b) \wedge \neg p_1$ formula
$\wedge(p_1 \leftrightarrow c \vee d)$ ce înseamnă p_1



Scriem fiecare echivalență în CNF, sau direct după regulile dinainte; punem conjuncție între ele
$(a \vee \neg b) \wedge p_1$ (nivelul final cu rezultatul)
$\wedge(c \vee d \vee \neg p_1) \wedge (\neg c \vee p_1) \wedge (\neg d \vee p_1)$

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* (pozitive sau negate)

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

Cum stabilim dacă o formulă e realizabilă ?

Reguli de simplificare:

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

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

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

R2a) Dacă un literal e T, *pot fi șterse clauzele* în care apare
(ele sunt adevărate, și nu mai influențează formula)
R2b) Dacă un literal e F, *el poate fi șters* din clauzele în care apare
(nu ajută în a face clauza adevărată)

Exemplele de mai sus se simplifică:

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

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

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

Cum stabilim dacă o formulă e realizabilă ?

R3) Dacă *nu mai sunt clauze*, am terminat (și avem o atribuire)
Dacă se ajunge la o *clauză vidă*, formula *nu e realizabilă*

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

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

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

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

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

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

- cu valoarea F
- cu valoarea T

O soluție pentru oricare caz e bună (nu căutăm o soluție anume).
Dacă nici un caz nu are soluție, formula nu e realizabilă.