Universitatea „Al. I. Cuza”, Iaşi

Facultatea de Informatică

Logica pentru Informatică

2017 - 2018

1-2

- Prof. dr. Cristian Masalagiu
- mcristy@info.uaic.ro
  https://profs.info.uaic.ro/~masalagiu/

1-3

Structura anului universitar 2017 - 2018, Semestrul I (decizie Rectorat)

- **Luni, 2 octombrie 2017: festivități deschidere**
- 2 octombrie – 24 decembrie 2017 (**12** săptămâni): **activitate didactică**
- 25 decembrie 2017 – 7 ianuarie 2018 (2 săptămâni): vacanță
- 8 ianuarie 2018 – 21 ianuarie 2018 (**2** săptămâni): **activitate didactică**
- 22 ianuarie 2018 – 4 februarie 2018 (**2** săptămâni): **sesiune curentă/ evaluare**
- 5 februarie 2018 – 18 februarie 2018 (2 săptămâni): 1 de vacanță și **1** de **sesiune suplimentară**: **restanțe/ măriri** (există niște reguli ...)

1-4

- Rezumat: **14 s** - activitate didactică, **4/ 3 s** – vacanță (din care **2/ 1 s** între semestre), **2/ 3 s** – sesiune (alte „calcule” ...)
- De fapt, la **FII**, în **săptămâna a 8-a** (20-26 noiembrie 2017) nu se va face activitate didactică, ci se vor da (unele) **lucrări de evaluare** (parțială)
- Recuperările de cursuri, zilele libere suplimentare etc., vor fi anunţate în timp util
- **Consultați permanent** (măcar săptămânal) paginile web personale ale profesorilor cu care vă „intersectați”
- Consultați cu regularitate pagina facultății: Regulamente, schimbări orar, alte informații up-to-date, ...

Structura și evaluarea la acest curs

- Cursul este de fapt <u>împărțit în 2</u>
- Prima parte (<u>**L***ogica* **P***ropozițională*</u> – <u>**LP**</u>) este alcătuită din 7 cursuri (<u>Curs 1 – Curs 7</u>) și 6-7 seminarii: <u>**C. Masalagiu**</u>
- A doua parte (<u>**L***ogica cu* **P***redicate de ordinul* **întâi**</u> – <u>**LP1**</u>) are 6 cursuri (<u>Curs 8/ 9 – Curs 13/ 14</u>): <u>**Ș. Ciobâcă**</u>
- <u>Fiecare parte</u> se va încheia cu o lucrare de evaluare („examene parțiale” - 1 respectiv 2): prima va fi în săptămâna a 8-a, iar a doua în sesiunea curentă

1-6

- Nota finală se obţine în urma acumulării unui număr de puncte (maxim **100 p** + **10 p** bonus) şi în urma aplicării unei ajustări de tip Gauss conform regulamentelor interne (ale Universității și **FII**) aflate în vigoare (de ex. **primii 5% vor avea nota 10**, etc.)
- Cele **100 de puncte** se pot obţine astfel:
  - **10 p** prezenţa la seminarii
  - **45 p** examen parțial 1 (în săptămâna a 8-a)
  - **45 p** examen parțial 2 (în sesiunea curentă)
  - **Promovarea** este asigurată de un punctaj minim total de **45 p**

  Observație. Se pot obține cele (maxim) **10 p** bonus pentru activitate ***deosebită*** efectuată în cadrul seminariilor.

Structura primelor 7 cursuri

Curs 1: **Logica și Informatica** (*problem solving*; **definiții** și **demonstrații prin inducție structurală**; *introducere* în **LP**)

Curs 2: **Sintaxa** și **semantica** *formală* a **LP** (concepte și rezultate importante)

Curs 3: **Problema satisfiabilității** (**SAT**) pentru **LP**; algoritmi de rezolvare bazați pe semantică/ tabele de adevăr și sintaxă (algoritmul lui **D***avis*-**P***utnam* (**L***ogemann*-**L***oveland*); prescurtat, **DP**(**LL**))

Curs 4: Alternative (tot sintactice) pentru rezolvarea **SAT**; **rezoluție** și **programare logică/ PROLOG**

1-8

<u>Curs 5</u>: Complemente de sintaxă și semantică: **algebre booleene** și moduri de reprezentare a **clasei funcțiilor booleene** (textual: *mulțimi complete*; *baze*; **forme normale**; grafic: **diagrame de decizie binare**); <u>intuitiv</u>: câte ceva despre *gramatici* și *automate*, *compilare* și *interpretare* ...

<u>Curs 6-7</u>: Tratarea *globală* (sintactică și semantică) a claselor de formule, cu exemplificare pentru **LP** (**teorii logice** și **sisteme deductive**; **teoreme de corectitudine și completitudine**); ca particularizare, **deducția naturală** pentru **LP**; <u>intuitiv</u>: **demonstrare automată**, *model checking* + *reasoning with ontologies* în **inteligența artificială**

MATERIALE SUPORT (electronic, specifice) **ȘI BIBLIOGRAFIE** (cărți generale; unele se găsesc (și) tipărite; vezi și sfârșitul fiecărui curs)

- **Suportul electronic** specific va cuprinde, în mare:

-slide-urile care au fost prezentate la orele de curs

-fișiere cu conținutul sugerat în slide-uri, sau accesibile prin link direct

-fișiere conținând exerciții pentru seminar (destinate lucrului individual, inclusiv aplicații implementate)

-fișiere suplimentare destinate auto-ghidării învățării individuale (+ suplimente curs)

1-10

- **Cărțile/ articolele** sugerate se pot accesa (majoritatea) în format electronic și sunt de fapt opționale (doar **<u>cursul este de neînlocuit</u>**):

<u>1.</u> **C. Masalagiu** - *Fundamentele logice ale Informaticii*, Ed. Universităţii „Al. I. Cuza", Iasi, 2004, ISBN 973-703-015-X.

<u>2.</u> **M. Huth, M. Ryan** - *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, England, 2000, ISBN 0-521-65200-6.

3. **R. Fagin**, et al. – *Reasoning about Knoledge*, M. I. T. Press, 2003.

4. **M. Mezghiche** – *Notes de cours: Logique Mathématique*.

5. **C. Cazacu, V. Slabu** – *Logica matematică*, Ed. „Ștefan Lupașcu", Iași, 1999, ISBN 973-99044-0-8.

- Alte site-uri/ link-uri de urmărit: a se vedea pagina mea web

1-11

- **Comentarii**: universitate vs școală; consultații; cum colaborăm (**NU** prin *social media*); voi – eu (curs ...)
- Principiu: ***Libertatea ta încetează atunci când începe libertatea altuia*** (întrebări; exerciții vs teste ...)
- Cum se învață: învățarea *nu este liniară* (individual ...)
- A căpăta/ obține informații punctuale (gen **Google**, **Wiki**, etc.) nu înseamnă și a **asimila** cunoștințe; e bine să aveți câte un *learning journal* (pagina web ...)
- Winston Churchill: ***Success is not final, failure is not fatal: it is the courage to continue that counts***
- ***There is no elevator to success. You have to take the stairs!*** Esențial: **CONCENTRARE !!**
- Predăm cuvinte: *limbă*, *limbaj*, comunicare (ambiguitate)

1-12

- **Realitate** = <u>Mulțimea tuturor *problemelor*</u>, oricât de „ciudate”, dar pe care *vrem să le rezolvăm cu ajutorul calculatorului*: ***„... angels ...”***
- **Calculator** = <u>dispozitiv</u> care implementează **algoritmi** (= texte scrise în **limbaje de programare** *comerciale*)
- **Logica** este/ poate fi (transformată într-) un/ văzută ca un/ asemenea limbaj
- Totul va însemna familiarizarea/ stăpânirea domeniului (mai mult sau mai puțin *automatizat*) numit ***Problem Solving*** (discutat în curs mai <u>întâi</u>)
- Noțiunea esențială va fi cea de ***mulțime definită constructiv/ structural*** (<u>după</u>)

PROBLEM SOLVING

- **Problemă** (pe scurt, intuitiv) =

-o mulțime de **date**/ **informații** (= conform *descrierii* problemei) +

-o mulțime de **operații** (= **mutări**/ **acțiuni** *permise*; evenimente apărute – tot din descriere) +

-un **scop** (= **ce se cere**, ce înseamnă o **soluție** *dorită* ...)

- Cum se **rezolvă** o problemă – metodologia (unanim acceptată) propusă de **George Polya** =

1-14

1. Se *înțelege* problema
2. Se *concepe* un *plan* (**algoritm** ...)
3. Se *execută* planul (limbaj de programare, **execuție**, implementare ...)
4. *Se „privește înapoi”* și *se verifică rezultatul* (ideal: **verificarea** apriorică a **corectitudinii algoritmului**)

- Înțelegerea problemei =

-Se citește cu atenție descrierea problemei; dacă nu se „vede” *exact* **ce** *este nevoie* să fie rezolvat, desigur că se va propune o soluție incorectă/ nedorită/ nereală; oricum, este bine să se enunțe încă o dată problema (dar cu alte cuvinte) și/ sau să fie explicată altei persoane

1-15

-Se identifică cantitățile/ datele necunoscute

-Se identifică cantitățile date/ furnizate/ cunoscute

-Se identifică condițiile date

-Se identifică alte condiții impuse (restricții implicite)

-Se introduc/ imaginează notații potrivite pentru ceea ce nu se cunoaște

-Pot fi utile anumite *desene*, *hărți* sau *diagrame*

- <u>Conceperea unui plan</u> =

-Se stabilesc toate relațiile identificabile dintre cantitățile cunoscute și necunoscute (ideal, acestea vor fi expresii algebrice sau ecuații, ***sau***...)

1-16

-Dacă nu se găsesc asemenea relații evidente/ imediate, se pot încerca (simultan/ separat) următoarele:

a)Se împarte problema în subprobleme având rezolvări știute

b)Se compară problema dată cu alte probleme similare, rezolvate anterior

c)Pentru început, se încearcă rezolvarea unei versiuni mai simple a problemei date

d)Se ghicește pur și simplu o soluție pentru problemă și apoi se lucrează „înapoi”

e)Se introduc date suplimentare/ valori intermediare, care „par” naturale

f)Se caută anumite pattern-uri/ șabloane/ tipare

1-17

- Executarea planului =

-Se efectuează calculele necesare (dacă ...)

-Se rezolvă toate „ecuațiile”

-Se găsește o (presupusă) soluție

-***Se verifică fiecare pas***. Ideal, ar trebui ca fiecare pas al planului să fie ***demonstrat*** că ***este „corect”*** (înainte de execuție: *chirurgie robotizată* ...)

- „Privirea” înapoi =

-Se „examinează” soluția obținută

-Se verifică rezultatele pe enunțul original

-Se verifică dacă s-au folosit *toate* informațiile

-Răspusul obținut/ rezolvarea/ soluția *au sens* ?

Exemple de probleme (semi)rezolvate

(I)<u>Suma a doi întregi/ naturali consecutivi</u> =

Suma a doi întregi consecutivi (numere naturale) este 23. Aflați valoarea celor doi întregi.

1.<u>Înțelegerea problemei</u>:

-date cunoscute = suma celor doi întregi este 23

-necunoscute = *x* (*numele* primului întreg dintre cei doi „consecutivi”); *x + 1* (evident, *denotă* cel de-al doilea întreg)

2.<u>Planul</u>:

$x + (x + 1) = 23$

3.<u>Executarea planului</u>:

$x = 11$, $x + 1 = 12$

4.<u>Verificarea</u>:

$11 + 12 = 23$

- Lucrurile par foarte simple
- Să luăm un alt exemplu de problemă „reală”

(II)<u>Trei fetițe obraznice</u> =

Se știe că una și numai una dintre Dolly, Ellen și Frances este autoarea/ vinovata „poznei”. Mai mult, atunci când s-a petrecut fapta, „inculpata” era cu certitudine în casă. Ce declară însă cele trei ?

1-20

-Dolly: *„Nu eu am făcut-o, nu eram în casă; Ellen a făcut-o”;*

-Ellen: *„Nu am fost eu, și n-a fost nici Frances; dar dacă aș fi făcut-o eu cu adevărat, atunci ar fi participat și Dolly, sau ea ar fi fost în casă”;*

-Frances: *„Nu am făcut-o eu, Dolly era în casă; dacă Dolly era în casă și a făcut pozna, atunci a făcut-o și Ellen”.*

S-a descoperit că niciuna nu a spus adevărul.

Cine este vinovata ? (răspusul este *scopul* rezolvării)

1-21

- Să rezolvăm problema ca mai înainte, adică urmând cei „4 pași Polya”
- Observăm că „limbajul aritmeticii” nu ne este acum prea util
- Trebuie să procedăm altfel pentru a obține datele și notațiile pentru informațiile cunoscute/ necunoscute și relațiile dintre acestea pentru a putea concepe un plan (*algoritm*) de rezolvare

1. <u>Înțelegerea problemei</u>:

-date cunoscute = știm că autoarea „poznei” este una dintre cele trei; le putem privi drept ***constante*** și le vom nota cu *d* (Dolly), *e* (Ellen) și *f* (Francis)

1-22

-necunoscute =

a) nu știm cine este vinovata (să zicem că o notăm generic cu x), dar trebuie să existe așa ceva; putem nota cu V(x) afirmația/ ***predicatul*** „x este vinovata” (V(d) ar fi o ***propoziție elementară*** particulară, indivizibilă)

b) similar, nu știm cine este/ a fost în casă (în momentul producerii poznei), dar trebuie să fi fost cineva (să zicem, y); acest „cineva” este, în plus, și inculpata; notăm cu H(y) predicatul „y este în casă în momentul producerii poznei”

2.<u>Planul</u> (algoritmul + limbajul de programare ...):

Știm din enunț că este *adevărată* afirmația

$\mathbf{(}\forall x\mathbf{)(}\mathrm{V}(x) \rightarrow \mathrm{H}(x)\mathbf{)}$ (explicații)

1-23

Mai exact, în contextul dat, știm de la început, înainte de a le interoga pe fete (**c**...) :

(i) $V(d) \vee V(e) \vee V(f)$ (F1)

(ii1) $V(d) \rightarrow H(d)$ (F2)

(ii2) $V(e) \rightarrow H(e)$ (F3)

(ii2) $V(f) \rightarrow H(f)$ (F4)

Tot de la început, se știe că doar una dintre fete a făcut pozna, adică:

(iii1) $\neg\mathbf{(}V(d) \wedge V(e)\mathbf{)}$ (F5)

(iii2) $\neg\mathbf{(}V(d) \wedge V(f)\mathbf{)}$ (F6)

(iii3) $\neg\mathbf{(}V(f) \wedge V(e)\mathbf{)}$ (F7)

În sfârșit, după interogări și considerarea negațiilor celor afirmate de către fete, mai cunoaștem:

(iv) $V(d) \lor H(d) \lor \urcorner V(e)$ (F8)

(*negația a ce a spus Dolly*: $\urcorner V(d) \land \urcorner H(d) \land V(e)$)

(v) $V(e) \lor V(f) \lor \urcorner \mathbf{(}V(e) \to \mathbf{(}V(d) \lor H(d)\mathbf{))}$ (F9)

(Ellen: $\urcorner V(e) \land \urcorner V(f) \land \mathbf{(}V(e) \to \mathbf{(}V(d) \lor H(d)\mathbf{))}$)

(vi) $V(f) \lor \urcorner H(d) \lor \urcorner ((H(d) \land V(d)) \to V(e))$ (F10)

(Frances: $\urcorner V(f)) \land H(d) \land \mathbf{((}H(d) \land V(d)\mathbf{)} \to V(e)\mathbf{)}$)

- Ideea este că în loc de o mulțime de relații/ ecuații, am găsit o mulțime de afirmații/ ***formule*** (***compuse***) care sunt „adevărate”
- Le-am notat cu F1, ..., F10; dispunem deci de mulțimea de „adevăruri” F = {F1, ..., F10}
- De fapt, „dispunem” de adevărul unei afirmații, „mai” *compuse*, F = F1 $\land$ ... $\land$ F10

1-25

- Prin urmare, pentru a executa planul, nu vom proceda la „rezolvarea” unei mulțimi de „ecuații algebrice”, ci la ***demonstrarea*** „***adevărului”*** sau ***„falsității”*** unei ***formule logice***

<u>Legendă</u> (alte comentarii, parantezele ...):

- ($\forall x$) – ***cuantificator/ cuantor*** *universal* (partea doua a cursului; ca și: *variabile*, *predicate* etc.)
- $\rightarrow$ - ***implicația/*** *dacă ... atunci ...* (***conector*** sau ***conectiv logic***)
- $\wedge$ - ***conjuncția/*** *și, and* (conector)
- $\vee$ - ***disjuncția/*** *sau, or* (conector)
- $\rceil$ - ***negația/*** *non, not* (conector)

3. <u>Executarea planului</u>:

Ca modalitate generală, există acum două abordări (principial) diferite:

(A)O modalitate bazată pe ***semantică*** (***tabele de adevăr*** pentru anumite funcții booleene, „scrise” eventual în *format text* = ***forme normale***).

- În cazul nostru, se poate „construi tabelul” (și se „vede” V(*f*) ...); sau (mai „merge” și „stil” **PROLOG**, adică <u>sintactic, într-un mod specific</u>, combinat cu semantica): *bănuind* că Frances este vinovata, se arată (tot cu tabele) că formula $F \wedge \rceil V(f)$ este *falsă*

(B)O modalitate bazată pe ***sintaxă***; adică, mai mult sau mai puțin explicit, se folosește ***rezoluția*** (în curs, <u>mai târziu</u>); deși nu toate ***demonstratoarele automate*** (de fapt, inclusiv *compilatoarele/ interpreterele* **PROLOG**) se bazează (doar) pe rezoluție.

4.<u>Verificarea</u> (execuției corecte a) planului va fi tot specifică contextului precizat și o amânăm (ca, de altfel, și detaliile privind „execuția" sintactică și/ sau semantică) până după asimilarea unor necesare cunoștințe teoretice

- Să prezentăm și un exemplu de problemă rezolvată (sintactic) *folosind deducția naturală*

1-28

- Este necesar și asta pentru că:

A rezolva o problemă este un fel de „artă practică”, cum ar fi înnotul, schiatul sau cântatul la pian: învățarea acesteia se poate face doar prin imitare (cu șabloane/ tipare) ***și practică ... dacă vrei să înveți să înnoți, trebuie*** (măcar) ***să te bagi în apă, iar dacă vrei să devii un*** (bun) ***rezolvitor de probleme trebuie să rezolvi*** (multe și diversificate) ***probleme*** („reale”)**.**

G. Polya

- <u>Vezi alte exerciții (seminar, suplimente”, etc.)</u>

(III)<u>O întâlnire</u> =

Dacă trenul ajunge mai târziu și nu sunt taxiuri în stație, atunci John va întârzia la întâlnirea fixată. Se știe că John nu întârzie la întâlnire, deși trenul ajunge într-adevăr mai târziu. Prin urmare, erau taxiuri în stație.

1.<u>Înțelegerea problemei</u>:

-date cunoscute = putem nota cu niște nume concrete (sunt *constante …*) *afirmațiile elementare/ indivizibile*

p: „Trenul ajunge mai târziu”

q: „Sunt taxiuri în stație”

r: „John întârzie la întâlnirea fixată”

1-30

- Se observă că în cazul de față nu există practic elemente necunoscute; scopul, ce se cere a fi „rezolvat” este să se arate că ***afirmația compusă*** F = $(((p \wedge (\rceil q)) \rightarrow r) \wedge (\rceil r) \wedge p) \rightarrow q$ este „adevărată” (ca *formulă* în **LP**; **c** - implicație, tabele, ***adevăr***)

2. Planul:

- **Raționament = demonstrație** folosind ***regulile de inferență*** *ale deducției naturale* (legătura *sintaxă* vs *semantică* aici ...)
- $(p \wedge (\rceil q)) \rightarrow r, \rceil r, p \vdash q$ este o ***secvență***; în stânga sunt ***premize***; în dreapta este ***concluzia***

3. Execuția planului (se arată că „secvența este *validă*”):

1-31

- După cum am sugerat: „*Aplicând anumite reguli corecte premizelor, sperăm să obținem alte formule (rezultat), și, în continuare, aplicând mai multe reguli de tipul menționat acestor rezultate, să obținem, în cele din urmă, și concluzia (inițială)*” (<u>M. Huth, M. Ryan</u>: **H/ R**)
- Concret:

-să „pornim” cu premizele $(p \wedge (\rceil q)) \to r$, $\rceil r$, p (nu contează de fapt ordinea acestora; însă ele se presupun a fi „adevărate”); să arătăm că este adevărată q, în modul sugerat;

-să presupunem, prin *reducere la absurd*, că este adevărată $\rceil q$;

-știm acum că sunt adevărate *p* (premiză inițială) și $\rceil q$ (presupunere); atunci (folosind o *regulă corectă*, ca și R.A., de altfel) va fi adevărată $p \wedge \rceil q$;

-prin urmare, știm că sunt adevărate $p \wedge \rceil q$ (demonstrată mai înainte) și $(p \wedge (\rceil q)) \rightarrow r$ (premiză inițială); atunci (folosim *modus ponens*, ca *regulă corectă*) va rezulta că *r* este adevărată;

-știm acum că *r* este adevărată (am arătat mai sus), dar și că $\rceil r$ este adevărată (premiză inițială), ceea ce *este absurd*;

-rezultă că presupunerea făcută nu este adevărată, deci, de fapt, *q* este adevărată (formal, folosim sintactic *întregul raționament*)

4.<u>Verificarea</u> va putea fi și ea făcută în mod formal...

- Continuăm, așa cum am precizat inițial, cu

MULȚIMI DEFINITE CONSTRUCTIV/ STRUCTURAL

- Cum introducem o mulţime (știm deja ...):

-Prin enumerarea elementelor sale:

N = {0, 1, 2, ...}; **c**: finit, infinit numărabil, „...”, vezi…

-Prin specificarea unei proprietăţi caracteristice:

A = {$x \in$ **R** | $x^2 + 9x - 8 = 0$}, este mulţimea rădăcinilor reale a ... (infinit, „de orice tip”; nu trebuie **N**, ci orice element ...)

- Putem însă defini **N** folosind doar 4 simboluri, să le notăm **0**, **(**, **)**, și **s** (deocamdată, neavând semnificație/ interpretare „declarată explicit”)

- „Noua” mulțime va fi notată **N1** (definită „cu câte elemente este nevoie”, pornind cu alfabetul precizat, adică **L** = {**0**, **(**, **)**, **s**})

Definiția constructivă/ structurală a mulțimii numerelor naturale.

Baza. **0** $\in$ **N1** („zero” este număr natural).

Pas constructiv (**structural**). Dacă *n* $\in$ **N1**, atunci

s(*n***)** $\in$ **N1** (dacă *n* este număr natural, atunci „succesorul

său imediat”, de fapt **textul** „**s(***n***)**”, este număr natural).

Nimic altceva nu mai este număr natural (există exprimări „echivalente”: *închidere*, *inferențe* ...).

- „Traducerea” (semi)algoritmică (pseudocod...); alte ***notații*** (reguli, axiome ...); alte **c** ...

1-35

- Un *prim avantaj* al definiției este acela că se poate folosi aceeaşi metodă pentru a introduce alte definiţii care sunt legate de mulţimea respectivă (aici, **N1**), în totalitatea ei
- Putem da astfel o definiţie constructivă/ structurală (*recursivă, algoritmică, inductivă* ...) a *adunării* numerelor naturale (+; $\mathbf{s}(n) \triangleq n + 1$; folosim „toate” notațiile; calculați **2** + **3**; alte observații – *comutativitate*, ...)
- Diverse *variante de prezentare* a lor... (la seminar...)

1-36

- Un *al doilea avantaj*, cel mai important: folosirea în demonstraţii a ***principiului inducţiei structurale*** (*matematice*, în cazul lui **N1** și ... **N**)
- Dacă vrem să arătăm că o anumită proprietate/ afirmație, notată „P(n)”, este adevărată pentru fiecare $n \in$ **N1** (formal, F1 = $(\forall n)\mathbf{(}P(n)\mathbf{)}$ este *adevărată*), folosind principiul amintit vom arăta de fapt că este adevărată *formula*

  F2 = $P(0) \wedge \mathbf{(}(\forall n)\mathbf{(}P(n) \rightarrow P(n + 1)\mathbf{))}$; nu totdeauna sunt *echivalente* ... ***metalimbaj*** (<u>**c**</u>)
- Revăzând definiția constructivă a lui **N1**, cele de mai sus se pot generaliza pentru definirea altor mulțimi (<u>vezi seminar</u>, „<u>suplimente</u>”, ș.a.)

Sintaxa logicii propoziționale (LP); definiție constructivă/ structurală

- Fie o mulţime de *variabile propoziţionale* (sau: *formule elementare*, *formule atomice pozitive*, *atomi pozitivi*),

  A = {A_1, A_2, … } numită ***alfabet*** (numărabil) de „litere”/ nume generice/ ***constante*** (sau: **A** = {*p*, *q*, … })
- Notăm cu **C** = {$\rceil$, $\vee$, $\wedge$} mulţimea *conectorilor logici* (...)
- Fie **P** = {**(**, **)**} mulţimea *parantezelor rotunde*
- ***Formulele*** (elementele lui **LP**) vor fi *cuvinte* (cu „punctuație”), sau *expresii bine formate* (**w**ell **f***ormed* **f***ormulae*/ **wff**) peste *alfabetul extins* **L** = **A** $\cup$ **C** $\cup$ **P**
- Atunci **LP** va fi practic „*construită* pas cu pas, începând cu mulțimea vidă”, astfel:

Baza (*formulele elementare sunt formule*): **A** $\subseteq$ **LP**.

Pas constructiv (*obţinere formule noi din formule vechi*, *folosind conectorii*):

(i) Dacă F $\in$ **LP** atunci **(**$\rceil$F**)** $\in$ **LP**.

(ii) Dacă F_1, F_2 $\in$ **LP** atunci **(**$F_1 \vee F_2$**)** $\in$ **LP**.

(iii) Dacă F_1, F_2 $\in$ **LP** atunci **(**$F_1 \wedge F_2$**)** $\in$ **LP**.

(iv) Dacă F $\in$ **LP** atunci **(**F**)** $\in$ **LP**.

Nimic altceva nu mai este în **LP**.

- **Definiții** ulterioare imediate, bazate pe precedenta: *arb*(F), *subf*(F), *prop*(F)); **c**, **e** ...

1-39

- <u>Citiți</u> cu atenție/ concentrați ***toate*** slide-urile aferente <u>Cursului 1</u> (și accesați pagina web)
- (<u>Scrieți</u> *în jurnalul personal de învățare*)
- ***Important de reținut***: *realitate*, *rezolvarea problemelor*, *algoritmică*; limbaj „natural” vs limbaj „formal” sau/ și „de programare”; ***definiții structurale*** ale mulțimilor *cel mult numărabile* (*primare* = mulțimea însăși + definițiile *derivate*); ***principiul inducției structurale - <u>e</u>***; definiția structurală a ***sintaxei*** logicii propoziționale: limbajul formal/ ***logica***/ mulțimea de formule numit(ă) **LP** (*calculul* ...)

1-40

- Link-uri suplimentare utile: conform paginii web personale menționate (explicațiile conținuturilor fișierelor sunt în pagină; sau se „auto-explică”); în prealabil, se dă click pe Logic (Romanian) (L)

FINAL Curs 1

2-1 (42)

Continuare sintaxă LP (reluat exemple: **N1**, suma, inducție, *arb*(F), *subf*(F), *prop*(F))

- $((\rceil F) \vee G)$ se va nota cu $(F \rightarrow G)$ $(\triangleq)$
- Apoi avem $(((\rceil F) \vee G) \wedge ((\rceil G) \vee F)) \triangleq$ $(F \leftrightarrow G) \triangleq ((F \rightarrow G) \wedge (G \rightarrow F))$
- $\bigwedge_{i=1}^{n} F_i$ este o prescurtare pt. $F_1 \wedge F_2 \wedge \ldots \wedge F_n$
- $\bigvee_{i=1}^{n} F_i$ este o prescurtare pt. $F_1 \vee F_2 \vee \ldots \vee F_n$
- **c**: noi simboluri; multe/ puține ...; paranteze, „viitoare" asociativitate, comutativitate, ...
- **Literal**: o variabilă propoziţională sau negaţia sa
- $A \in$ **A** - **literal pozitiv**, iar orice element de forma $\rceil A$, $A \in$ **A** va fi un **literal negativ** (vom nota cu **Ā** mulțimea $\{\rceil A_1, \rceil A_2, \ldots\}$)

2-2 (43)

- Dacă L este un literal (adică $L \in \mathbf{A} \cup \bar{\mathbf{A}}$), atunci **complementarul** său, $\bar{L}$, va denota literalul $\rceil A$, dacă $L = A \in \mathbf{A}$ şi respectiv literalul A dacă $L = \rceil A$
- **Clauză**: orice disjuncţie (finită) de literali
- **Clauză Horn**: o clauză care are cel mult un literal pozitiv
- **Clauză pozitivă**: clauză care conţine doar literali pozitivi; o **clauză negativă** va conţine doar literali negativi
- O **clauză Horn pozitivă** va conţine exact un literal pozitiv (dar, posibil, şi literali negativi)

2-3 (44)

Semantica generală (0-1) a LP (n-am terminat însă complet cu sintaxa; vezi și Capitolul 2 ...)

- *Semantica (înţelesul)* unei formule propoziţionale este, conform principiilor logicii aristotelice, o valoare de adevăr ***a***devărat ($\triangleq$ **1**) sau ***f***als ($\triangleq$ **0**), obţinută în mod determinist și independentă de (orice alt) context
- De fapt, vom „lucra” cu *algebra booleană* $\mathcal{B} = \langle \mathbf{B}, \bullet, +, \overline{\ } \rangle$, $\mathbf{B} = \{\mathbf{0}, \mathbf{1}\}$ (vom reveni, formal, **Curs 5**: operații/ funcții booleene, tabele...)
- Noţiunea principală este cea de ***asignare*** **(*interpretare*, structură)**
- **Definiţie.** Orice funcţie **S**, **S** : **A** $\rightarrow$ **B** se va numi *asignare*.

2-4 (45)

- **Teoremă (de extensie)**. Pentru fiecare asignare **S** există o *unică extensie* a acesteia, **S'** : **LP** $\rightarrow$ **B** (numită în continuare *structură* sau *interpretare*), care satisface:

(i) **S'**(A) = **S**(A), pentru fiecare A $\in$ **A**.

(ii) **S'**((⌉ F)) = $\overline{\mathbf{S'}(F)}$, pentru fiecare F $\in$ **LP**.

(iii) **S'**((F$_1$ $\wedge$ F$_2$)) = **S'**(F$_1$) • **S'**(F$_2$), pentru fiecare F$_1$, F$_2$ $\in$ **LP**.

(iv) **S'**((F$_1$ $\vee$ F$_2$)) = **S'**(F$_1$) + **S'**(F$_2$), pentru fiecare F$_1$, F$_2$ $\in$ **LP**.

(v) **S'**((F)) = **S'**(F), pentru fiecare F $\in$ **LP**.

(<u>**d**</u>: ***inducție structurală***: în ***metalimbaj*** (!) arătăm <u>($\forall$F)(P(F))</u>; P(F): ($\forall$**S**)($\exists$!**S'**)(**S'** extinde **S și** ... vezi (i)-(v) de mai sus); de fapt, se arată doar unicitatea: P(A) **și** ...

2-5 (46)

- Vom folosi de acum **S** (în loc de **S’** etc.)
- **Definiţie**. O formulă F $\in$ **LP** se numeşte *satisfiabilă* dacă *există măcar* o structură **S** (***corectă*** pentru ...; prop(F) ...) pentru care formula este adevărată (**S**(F) = **1**); se mai spune în acest caz că **S** este model pentru F (simbolic, se scrie **S** $\models$ F). O formulă este *validă* (*tautologie*) dacă *orice* structură este model pentru ea. O formulă este *nesatisfiabilă* (*contradicţie*) dacă este falsă în *orice* structură (**S**(F) = **0**, pentru fiecare **S**; sau **S** $\not\models$ F).

2-6 (47)

- **Teoremă**. O formulă F $\in$ **LP** este validă dacă şi numai dacă ($\rceil$ F) este contradicţie. (**<u>d</u>**)
- Clasa tuturor formulelor propoziţionale **LP** este astfel partiţionată în trei mulţimi *nevide şi disjuncte*: **tautologii** (formule **valide**), formule **satisfiabile** (dar nevalide), **contradicţii** (formule nevalide); *desen „oglindă”*: F, (G, $\rceil$G), $\rceil$F… (**<u>e</u>**)
- *Problema* **SAT** *este rezolvabilă în timp exponențial* (<u>revenim</u> în cursul următor...)

2-7 (48)

- **Definiţie**. Două formule F_1, $F_2 \in$ **LP** se numesc *tare echivalente* dacă *pentru fiecare* structură **S** ele au aceeaşi valoare de adevăr, adică $\mathbf{S}(F_1) = \mathbf{S}(F_2)$ (simbolic, vom scrie $F_1 \equiv F_2$). F_1 şi F_2 se numesc *slab echivalente* dacă F_1 satisfiabilă *implică* F_2 satisfiabilă şi reciproc (vom scrie $F_1 \equiv_s F_2$), ceea ce înseamnă că *dacă* există $\mathbf{S_1}$ *astfel încât* $\mathbf{S_1}(F_1) = \mathbf{1}$, *atunci* există $\mathbf{S_2}$ *astfel încât* $\mathbf{S_2}(F_2) = \mathbf{1}$ şi reciproc). (**e**)

2-8 (49)

- **Definiție**. O formulă $F \in$ **LP** este *consecinţă semantică* dintr-o mulţime (nu neapărat finită) de formule $\mathbf{G} \subseteq$ **LP**, dacă: *pentru fiecare structură **corectă*** **S**, *dacă* ***S*** *satisface* ***G*** (adică avem **S**(G)= **1** pentru fiecare $G \in \mathbf{G}$) *atunci* **S** *satisface* F (simbolic, vom scrie $\mathbf{G} \models F$).
- **Teoremă**. Fie $G \in$ **LP** şi $\mathbf{G} = \{ G_1, G_2, \ldots, G_n \} \subseteq$ **LP**. Următoarele afirmaţii sunt echivalente:
- (i) G este consecinţă semantică din **G**.
- (ii) $(\bigwedge_{i=1}^{n} G_i) \rightarrow G$ este tautologie.
- (iii) $(\bigwedge_{i=1}^{n} G_i) \wedge \rceil G$ este contradicţie.

(idei pentru **<u>d</u>**: (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (i); meta...; nu ind...)

2-9 (50)

- **Teoremă (de echivalenţă)**. Sunt adevărate următoarele echivalenţe tari, pentru oricare F, G, H $\in$ **LP** (atenție: nu se știu proprietățile de *algebră booleeană* pentru „•, +, ¯ ”):

  (a) $F \wedge F \equiv F$.

  (a') $F \vee F \equiv F$. (*idempotenţă*)

  (b) $F \wedge G \equiv G \wedge F$.

  (b') $F \vee G \equiv G \vee F$. (*comutativitate*)

  (c) $(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$.

  (c') $(F \vee G) \vee H \equiv F \vee (G \vee H)$. (*asociativitate*)

  (d) $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$.

  (d') $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$. (*distributivitate*)

  (e) $F \wedge (F \vee G) \equiv F$.

  (e') $F \vee (F \wedge G) \equiv F$. (*absorbţie*)

2-10 (51)

(f) $\neg\neg F \equiv F$. (*legea dublei negaţii*)

(g) $\neg(F \wedge G) \equiv \neg F \vee \neg G$.

(g') $\neg(F \vee G) \equiv \neg F \wedge \neg G$. (*legile lui deMorgan*)

(h) $F \vee G \equiv F$.

(h') $F \wedge G \equiv G$. (*legile validităţii*; adevărate ***doar dacă*** F este tautologie)

(i) $F \wedge G \equiv F$.

(i') $F \vee G \equiv G$. (*legile contradicţiei*; adevărate **doar dacă** F este contradicţie)

(**d**: câteva)

- Generalizări pentru mai multe formule; ***mulțimi***: vezi *algebrele booleene* (și $\boldsymbol{M} = \langle M, \cap, \cup, C_M \rangle$ ...)

2-11 (52)

- **Teoremă (de substituţie)**. Fie H $\in$ **LP**, oarecare. Fie orice F, G $\in$ **LP** astfel încît F este o subformulă a lui H şi G este tare echivalentă cu F. Fie H' formula obţinută din H prin înlocuirea (unei apariţii fixate a) lui F cu G. Atunci H $\equiv$ H'.

(<u>**d**</u> prin ***inducție structurală în metalimbaj***: <u>$(\forall$H)(P(H))</u>; P(H): $(\forall$F)$(\forall$G)$(\forall$H') (**Dacă** F este subformulă a lui H **și** H' se obține din ... **și** F $\equiv$ G, **atunci** H $\equiv$ H'))

- Urmează alte câteva definiții și rezultate „combinate" (sintaxă + semantică) importante

Forme normale

- Vom studia **simultan** *formele normale conjunctive* şi *formele normale disjunctive* pentru formulele din **LP**, ca texte
- **Definiţie**. O formulă F $\in$ **LP** se află în **formă normală conjunctivă** (**FNC**, pe scurt; în stânga) dacă este o *conjuncţie de disjuncţii de literali*, adică o *conjuncţie de clauze*; respectiv, F $\in$ **LP** este în **formă normală disjunctivă** (**FND**, pe scurt; în dreapta), dacă este o *disjuncţie de conjuncţii de literali*:

$$F = \bigwedge_{i=1}^{m}\left(\bigvee_{j=1}^{n_i} L_{i,j}\right) \qquad\qquad F = \bigvee_{i=1}^{m}\left(\bigwedge_{j=1}^{n_i} L_{i,j}\right)$$

- În cele de mai sus $L_{i,\ j} \in$ **A** U **Ā**.

2-13 (54)

- **Teoremă (existență forme normale)**. Pentru fiecare formulă F $\in$ **LP** există cel puţin două formule

  F_1, F_2 $\in$ **LP**, F_1 aflată în **FNC** şi F_2 aflată în **FND**, astfel încât $F \equiv F_1$ şi $F \equiv F_2$ (se mai spune că F_1 şi F_2 sunt **o FNC**, și respectiv **o FND**, *pentru* F).

(**d**: $(\forall F)(P(F))$; P(F): $(\exists F_1)(\exists F_2)$($F_1$ este în **FNC și** F_2 este în **FND și** $F_1 \equiv F$ **și** $F_2 \equiv F$); F – o privim ca *arbore* ... **e**)

- Conform teoremei anterioare (***ind. struct. în meta***...), precum şi datorită comutativităţii şi idempotenţei disjuncţiei, comutativităţii şi idempotenţei conjuncţiei (repetarea unui element, fie el literal sau clauză, este nefolositoare aici), este justificată ***scrierea ca mulţimi a formulelor aflate în*** **FNC**

2-14 (55)

- Astfel, dacă F este în **FNC**, vom mai scrie

  $F = \{C_1, C_2, ..., C_m\}$ (virgula denotă ...)
- Fiecare clauză C_i poate fi la rândul ei reprezentată ca o mulţime, $C_i = \{L_{i,1}, L_{i,2},\ldots, L_{i,ki}\}$, $L_{i,j}$ fiind literali (virgula denotă acum ...)
- **Observație**. Dacă avem $F \in$ **LP** reprezentată ca mulţime (de clauze) sau ca mulţime de mulţimi (de literali), putem elimina clauzele C care conţin atât L cât şi complementarul său, $\overline{L}$, deoarece $L \vee \overline{L} \equiv \mathbf{1}$, $\mathbf{1} \vee C \equiv \mathbf{1}$ şi deci aceste clauze sunt tautologii (*notate generic* cu **1**; contradicțiile - cu **0**); tautologiile componente nu au nici o semnificaţie pentru stabilirea valorii semantice a unei formule F aflate în **FNC** ($\mathbf{1} \wedge C \equiv C$)

2-15 (56)

- Importanța formelor normale (și a echivalențelor semantice existente) rezultă imediat, gândindu-ne *întâi* la *standardizare*: este posibil să folosim structuri de date și ***algoritmi sintactici unici*** pentru întregul **LP**, deși formulele ar putea fi scrise în moduri foarte diverse
- Vom exemplifica acest lucru și pentru cazul rezolvării sintactice a problemei **SAT**, în cursul următor (<u>revenim</u> și după tratarea ***rezoluției*** cât și după studiul ***funcțiilor booleene***)

2-16 (57)

- ***Important de reținut***:

-Definițiile structurale pentru *arb*(F), *subf*(F), *prop*(F); din nou, *noțiunea de infinit* utilizată, importanța *inducției structurale* în *metalimbaj* ...

-Definiția unei *structuri* (și **Teorema de existență** a *extensiei unice*)

-Formule *satisfiabile*, *valide*, *contradicții* (**Teoremă**)

-*Echivalență* tare și slabă (**Teoremă**)

-*Consecință semantică* (**Teoremă**)

-**Teorema de substituție**

-Forme normale (*clauze*; **FNC**, **FND**; algoritmica din substrat; eventual – Horn, complexitate ...)

2-17 (58)

- <u>Link-uri suplimentare</u> utile: conform paginii web personale menționate (explicațiile conținuturilor fișierelor sunt în pagină; sau se „auto-explică”; sau „se citesc” fișierele ...); rezolvați singuri exercițiile (chiar cele suplimentare); „încercați” și aplicațiile propuse de Ș. Ciobâcă ...

FINAL Curs 2

3-1 (60)

- Prezentăm un prim algoritm sintactic „de satisfiabilitate” pentru rezolvarea problemei **SAT-LP**, *Algoritmul* **D***avis*-**P***utnam*-**L***ogemann*-**L***oveland*: **DP**(**LL**)
- Acesta poate fi generalizat la logici de *ordin superior*, de ex. **LP1** (**FNSC**, *ground clauses*, *teorema de compactitate*, etc.); sau chiar la logici *neclasice*, cum ar fi *logicile modale*, **LTL** ...
- **DP**(**LL**) este clasic și un exemplu edificator/ *o instanță* pentru o întreagă clasă de algoritmi ***nedeterminiști*** și ***concurenți*** (sau distribuiți), (**FG**), care pot fi descriși succinct prin fraza: „***Execută, atât timp cât este posibil, operațiile grupate într-o mulțime O***”
- Trebuie determinat ***O*** astfel încât **DP**(**LL**) să fie ***corect și complet pentru*** **SAT**, așa cum de fapt se dorește

3-2 (61)

- Facem o scurtă recapitulare a unor notații/ rezultate pe care le vom folosi; și câteva „noutăți”
- Concatenarea (prin „$\wedge$”) a 2 „formule” în **FNC** se va reprezenta (și) prin „+” (pentru 2 clauze, va fi vorba de „$\wedge$”; în reprezentarea cu mulțimi, vom folosi și „U” (similar, folosim „-” în loc de „\”)
- O formulă este validă ddacă negația sa este contradicție
- O clauză este satisfiabilă ddacă există (măcar) o structură în care (măcar) un literal component al clauzei este adevărat; se admite existența ***clauzei vide*** (notată □)
- O formulă în **FNC** este satisfiabilă ddacă există o structură în care toate clauzele componente sunt satisfiabile (prin convenție, □ este nesatisfiabilă)
- O clauză în care apar atât un literal L cât și complementarul său $\overline{L}$ este tautologie

3-3 (62)

- Fie $\boldsymbol{c} = \{C_1, C_2, \ldots, C_n\} \subseteq$ **LP** și $C \in$ **LP**; atunci $(\bigwedge_{i=1}^{n} C_i) \wedge (\neg C)$ <u>este contradicție ddacă</u> $\boldsymbol{c} \models C$
- Pentru a evita folosirea excesivă a acoladelor, vom scrie uneori o mulțime și ca o listă, având „cap” (A) și „coadă”: [A|BC] (sau chiar ABC) în loc de {A, B, C} („stil” **PROLOG** ...)
- Dacă X, Y sunt 2 clauze (mulțimi de literali) și $X \subseteq Y$, atunci se mai spune că X ***subsumează*** (pe) Y
- <u>Dacă X subsumează Y atunci $X \models Y$</u>
- <u>Mai târziu</u> (**Curs 4**) ne va folosi și: „Dacă $F \in$ **LP** este nesatisfiabilă și se află în **FNC**, atunci există o ***demonstrație prin rezoluție*** a lui □ pornind cu clauzele lui F, și reciproc” (F poate fi chiar o mulțime infinită/ numărabilă de clauze – conform *teoremei de compactitate*)

3-4 (63)

- Repetăm: elementele lui **LP** (formulele) pot fi reprezentate ca texte (inclusiv standard, prin forme normale); dar (vom vedea) și ca grafuri ((**O**)**BDD**-uri), ca închideri ale unor mulțimi de operatori etc. (definițiile pot fi toate ***constructive***)
- Să vedem mai îndetaliu care este problema **SAT** de care ne ocupăm ((re)vedeți și anumite referințe bibliografice ...)

Problemă (**SAT-LP**). Dată orice formulă din **LP** este ea satisfiabilă (/ validă/ contradicție – același) ? (pentru conceperea algoritmilor corespunzători, semantica formală depinde și de sintaxă …)

3-5 (64)

Teoremă (SAT). Problema satisfiabilității pentru clasa formulelor logicii propoziționale (**LP**) este ***rezolvabilă/ decidabilă*** (***în timp exponențial***…).

Observație. Ideea, pe scurt, este să se propună (din teorie) un ***algoritm*** (aici - **DP**(**LL**)) ***care rezolvă problema*** și apoi să se arate ***terminarea și corectitudinea/ soundness*** algoritmului respectiv, fie că este de natură (preferabil) *sintactică*, fie *semantică*; se mai folosește exprimarea: **DP**(**LL**) ***este corect și complet pentru* SAT** (explicație)

Presupunere. Considerăm că formulele din **LP** cu care lucrăm sunt în **FNC**, reprezentate ca texte (mulțimi de mulțimi de literali, notate ***c***) (problema rezolvată ar fi putea fi **CNFSAT**; se poate și **3CNFSAT** …)

3-6 (65)

- <u>**DP**(**LL**) este un algoritm sintactic</u>: pornind cu o formulă oarecare, F $\in$ **LP**, se execută succesiv anumite operații asupra reprezentării sale *curente* (***c***); ***c*** va avea mereu o „lungime” mai mică; se va asigura astfel ***terminarea*** (cu răspunsul „**DA**”/ „**NU**”)
- Ideal, operațiile vor „păstra”, ca ***proprietate invariantă***, satisfiabilitatea/ nesatisfiabilitatea formulei curente; prin asta se va asigura și ***corectitudinea***
- Prin adăugarea (eventuală a) unor intrări suplimentare și extinderea unor operații, la sfârșitul execuției algoritmului se va putea obține (în cazul răspunsului „**DA**”) și o structură corespunzătoare **S**, a.î. **S** $\models$ F
- Din mulțimea inițială, ***nevidă***, notată ***c***, vom elimina de la început (sintactic, „manual” …) tautologiile (clauzele care conțin atât un literal cât și complementarul său); în plus:

3-7 (66)

- Cum mulțimile (prin definiție) nu conțin „dubluri”, considerăm eliminate și dublurile (fie literali, fie clauze), înainte de începerea execuției algoritmului
- Ca o intrare suplimentară, putem considera mulțimea *atomilor peste care este construită* F, *prop*(F); aceasta nu este absolut necesară, dar poate simplifica exprimarea de moment (odată „prelucrat” prin operații, un atom A - și $\urcorner$A, de fapt – acesta poate dispare din ***c***-ul curent; A va dispare astfel și din *prop*(*F*)); în plus, ea poate facilita unele demonstrații prin referiri explicite (ex.: inducție după nr. atomilor din *prop*(F))
- „Diferențele” între { }/ {} , {Ø}, Ø, {□} …

3-8 (67)

- Astfel, în context, { } (sau {}) poate fi Ø, ca „fostă” mulțime de „elemente”, din care „s-a scos/ tăiat tot”; poate fi și □, dacă este vorba despre o clauză (din care s-au „tăiat” toți literalii), și atunci {{}} poate fi, de fapt, {□}
- Oricum, în condițiile **DP**(**LL**), se va putea distinge clar între o mulțime de clauze (mulțime de mulțimi de literali) de forma {} (reprezentând Ø și caracterizând în final satisfiabilitatea formulei inițiale; adică, din reprezentarea inițială s-au tăiat toate clauzele); și una de forma {□} (caracterizând nesatisfiabilitatea formulei inițiale), și care provine din {{A}, {$\rceil$A}, …} din care s-au șters cele 2 clauze unitare printr-o operație fixată (numită – mai târziu - *rezoluție*)

3-9 (68)

- Operațiile executate asupra „formulei” curente ***c*** (notate cu ***o*** și formând o mulțime ***O***), sunt descrise în continuare sub o formă funcțională ***o***(***c***) = ***c'***, deși într-un singur caz ***c'*** **nu** va fi *tot* o simplă (reprezentare de) formulă din **LP** (în **FNC**), ca (repetăm) mulțime de clauze (= mulțimi de literali)

Exemplul 3-1. Fie F $\in$ **LP** (F = …), aflată în **FNC**, și scrisă sub forma: ***c*** = $\{C_1, C_2, C_3, C_4, C_5\} = \{\{A, B\}, \{\rceil A, \rceil B\}, \{\rceil B, C\}, \{\rceil C, A\}, \{C, D\}\}$.

- Operațiile generale din ***O*** folosite în **DP**(**LL**), vor acționa asupra lui ***c*** și asupra „succesoarelor” ei; avem:
- ***O*** = {**SUBS**, **PURE**, **UNIT**, **SPLIT**}
- La momentul descrierii algoritmului, sau în demonstrația corectitudinii, descrierea operațiilor va fi mai detaliată (<u>vezi și</u> slide-urile 3-29 (88), 3-30 (89))

3-10 (69)

-**<u>SUBS</u>**: dacă C este o clauză din ***c subsumată*** de (include o) alta, atunci C se scoate din ***c*** ; mai exact, **SUBS**(***c***) = ***c'***, unde ***c'*** = ***c*** - C

-**<u>PURE</u>**: dacă L $\in$ C ($\in$ ***c***) este un literal ***pur*** (adică *complementarul său nu este prezent nicăieri în altă clauză* din ***c***), atunci *se scot din **c** toate* clauzele care conțin L; punem **PURE**(***c***) = ***c'*** (este simplu să se descrie ***c'*** „mai” formalizat)

-**<u>UNIT</u>**: Dacă C = {A} este ***un fapt*** (***pozitiv***) în ***c***, *se scot din **c** toate* clauzele care conțin A (inclusiv C), iar din restul clauzelor rămase se scoate $\rceil$A (acolo unde există, desigur); operația se poate efectua *similar* și asupra ***unui fapt*** (***negativ***) C = {$\rceil$A}, cu $\rceil$A și A având roluri inversate; avem **UNIT**(***c***) = ***c'***, iar ***c'*** va (putea) fi definit (și aici mai) formalizat

3-11 (70)

- Din motive tehnice, ***c'***, ca rezultat al aplicării **UNIT** (asupra lui ***c***), va fi notat cu $(\boldsymbol{c}')_1$, dacă provine din prelucrarea lui {A} și cu $(\boldsymbol{c}')_2$, dacă provine din prelucrarea lui {⏋A}

-**<u>SPLIT</u>**: ***c*** *poate să nu se regăsească în niciuna dintre situațiile anterioare*; astfel, se poate ca în ***c*** curent să existe măcar o clauză C *neunitară*, care conține măcar un atom/ literal pozitiv A neprelucrat încă prin alte operații (neșters din ***c*** și, eventual, din *prop*(*F*)); mai mult, pot exista în ***c*** și *alte* clauze, anume cele care conțin pe ⏋A (A și ⏋A nu pot fi în aceeași clauză și nici nu pot forma ambele clauze unitare în ***c***; <u>vom vedea</u>); construim acum *simultan* din ***c***, și separat, entitățile $(\boldsymbol{c}')_1$ și $(\boldsymbol{c}')_2$, așa cum sunt ele definite la operația **UNIT**; în sfârșit, punem: **SPLIT**(***c***) = <$(\boldsymbol{c}')_1$, $(\boldsymbol{c}')_2$> (rezultatul aplicării operației va fi constituit din 2 „formule”, nu din una, ca până acum)

3-12 (71)

- Să notăm că operația **SPLIT** este necesară atât pentru a „acoperi” toate situațiile în care s-ar putea afla ***c***-ul curent pe parcursul execuției algoritmului, cât și pentru a avea mereu un rezultat ***c’*** „mai scurt”
- Modalitatea de prezentare a „noului” ***c***, notat acum ***c’*** = <(***c’***)$_1$, (***c’***)$_2$> nu vrea să sugereze că rezultatul aplicării operației **SPLIT** asupra lui ***c*** este acest ***c’*** în ansamblu (e „mai lung” decât ***c*** !), dar nici (***c’***)$_1$ sau (***c’***)$_2$, în mod nedeterminist, la alegere; ci ***ambele***, ***simultan***
- Mai exact, vom ***relua***/ continua algoritmul **DP**(**LL**) ***atât cu*** noua intrare ***c*** = (***c’***)$_1$, pe de o parte, ***cât și cu*** intrarea ***c*** = (***c’***)$_2$, pe de altă parte (simultan)

Observație. Este posibil să se lucreze și cu „***c’*** total”, dar această variantă necesită calcule suplimentare (de ex. readucerea formulei intermediare la **FNC**, folosind distributivitatea; suplimentarea demonstrației privind terminarea algoritmului, etc.). Pe de altă parte, s-ar putea evita recursia ...

Exemplul 3-1 (**continuare**). Fie F (adică ***c***):

F = {{A, B}, {$\rceil$A, $\rceil$B}, {$\rceil$B, C}, {$\rceil$C, A}, {C, D}} = {C_1, C_2, C_3, C_4, C_5}.

Asupra lui ***c*** curent, executăm operații din ***O*** *atât timp cât este posibil*; „vizual", construim un ***arbore de execuție*** (se poate construi chiar un ***arbore complet*** ...):

1) D $\in$ C_5 este un literal pur; **PURE**(***c***) = ***c'***, cu:

c' = {{A, B}, {$\rceil$A, $\rceil$B}, {$\rceil$B, C}, {$\rceil$C, A}} = {C_1, C_2, C_3, C_4}.

2) Aplicăm **SPLIT** pentru L = B $\in$ C_1, și vom „merge" apoi simultan pe 2 căi; subarborele „stâng" al lui ***c'*** va avea rădăcina (***c'***)$_1$ = {{$\rceil$A}, {C}, {$\rceil$C, A}}; iar cel „drept", rădăcina (***c'***)$_2$ = {{A}, {$\rceil$C, A}}, astfel:

3-14 (73)

-pentru „stânga”, se șterg clauzele care conțineau pe B (adică C_1) și pe $\rceil B$ din cele în care acesta apărea (adică din C_2 și C_3); C_4 rămâne neschimbat;

-pentru „dreapta”, s-au șters cele două clauze în care apare $\rceil B$ (anume C_2 și C_3), iar B - din clauza în care apare (C_1); din nou, C_4 rămâne neschimbat.

3)În „stânga”, avem ***c*** = {{$\rceil$A}, {C}, C_4 = {$\rceil$C, A}} și putem continua cu **UNIT**, pentru clauza unitară {$\rceil$A}; obținem drept „formulă” curentă pe ***c'*** = {{C}, {$\rceil$C}} și, în sfârșit, găsim ***c'*** = {□} (tot prin **UNIT**; sau, prin rezoluție într-un pas); prin urmare, ne oprim (<u>vezi algoritmul</u>) cu „**NU**” (totuși, pentru a trage o concluzie finală, mai întâi va trebui să ne oprim și pe ramura „dreapta”).

4) În „dreapta” (revenim la **2)**) avem noul

c = {{A}, {$\rceil$ C, A}} (fostul (***c’***)$_2$); luând cele 2 clauze rămase, prima o subsumează pe a doua; aplicăm **SUBS**, găsind ***c’*** = {{A}}; în sfârșit, și pe această ramură terminăm, de data aceasta cu {} (adică cu Ø), deoarece în ultima mulțime de clauze, atomul/ literalul L = A este pur (și ultimul ***c*** curent, adică ***c’*** de mai înainte, conține o unică clauză; se aplică deci **PURE**); ne oprim și pe această ramură (nu se mai pot aplica operații); de această dată, cu „**DA**”.

- În această situație, F (va fi, totuși) ***este satisfiabilă*** (revenim după descrierea exactă a **DP**(**LL**))

Observație. În exemplul anterior puteam proceda și altfel („progresând” în construcția arborelui complet). Astfel, pentru **SPLIT**, în **2)**, putea fi selectat și L = C $\in$ C_3. Atunci, puteam continua (și) cu (arborele complet ...):

5)Aplicând **SPLIT** pentru C din C_3, găsim pentru „stânga" noua mulțime curentă (***c'***)$_1$ = {{A,B}, {$\rceil$A, $\rceil$B}, {A}}, și pentru „dreapta" pe (***c'***)$_2$ = {{A, B}, {$\rceil$A, $\rceil$B}, {$\rceil$ B}}. Apoi, în stânga aplicăm **UNIT** (pentru {A}) și avem {{$\rceil$ B}}; acum aplicăm **PURE**, găsim Ø și ne oprim (cu „**DA**"). În dreapta, vom aplica **UNIT** (pentru {$\rceil$ B}) și apoi **PURE**; în final obținem tot Ø (și *satisfiabilitate*).

- Revenind la **SPLIT**-ul din **2)**, am fi putut alege literalul L = A $\in$ C_1; *sau*, tot pe L = A, dar cel din C_4; <u>finalizați singuri întregul proces</u> (concluzii, arbore complet, satisfiabilitate F ...)

Observație. Am sugerat că indiferent de drumul (de la rădăcina F) pe care ne găsim, oprirea (într-o frunză/ nod curent ***c***) se face fie atunci când lui ***c*** nu i se mai poate aplica nicio operație din ***O***, fie când ***c* =** Ø, fie când ***c* =** {□} ... Iar concluzia ar fi ...

3-17 (76)

- Detaliind, să subliniem încă o dată faptul că alegerile pur nedeterministe pot fi reprezentate toate (noduri succesor - ***divizare***) în construcția arborelui complet, care nu este neapărat binar (deși acest lucru nu este necesar pentru a obține un răspuns final)
- ***În același*** arbore complet, se „pun” însă și nodurile rezultate din **SPLIT** (adică, să zicem, dintr-o ***branșare***)
- Pentru a diferenția „tipul ramificării”, este suficient să adoptăm notații diferite: e.g., arce din ***linii punctate*** pentru branșarea generată de **SPLIT**, și arce din ***linii continue*** pentru explicitarea nedeterminismului
- Alegerile nu vor schimba deloc „tipul drumului” care urmează, adică, pentru satisfiabilitate, va fi suficient ca măcar una dintre ramurile generate de un **SPLIT** să „conducă” la un „**DA**”

3-18 (77)

- Se va demonstra de fapt că F ***este satisfiabilă*** ddacă ***există (măcar) un drum (de la rădăcină la o frunză)*** în arborele de execuție menționat ***în care frunza este etichetată cu*** „**DA**” (adică ***c*** final este Ø)
- În acest caz, se poate genera și o structură **S**, care să fie model pentru F; inițial structura este „pusă pe zero” (**S**(A)= = **0**, pentru fiecare A $\in$ *prop*(F)) (***configurație***: ***p*** = <**S**, ***c*** >)
- Apoi, operațiile care implică alegerea unui literal, pot fi *completate*: de fiecare dată când este selectat un (nou) literal/ atom L, se „modifică” și **S**(L), nu doar ***c***
- Este practic imediat faptul că algoritmul sugerat ***se termină pentru orice intrare***: fiecare (nou) ***c*** curent va avea strict mai puține caractere (clauze/ literali) decât cel vechi, după orice execuție a unei operații, pe fiecare drum
- Continuăm cu ***enunțul exact/ formal al algoritmului recursiv*** (apoi, <u>demonstrația de corectitudine și alte comentarii utile: urmați link-urile de sfârșit</u>)

3-19 (78)

procedure DPLL(**S**, ***c***): boolean; % ***recursivă***

Pasul 1. **Dacă** (***c*** = Ø (= {}))

atunci % *ieșire*

Pasul 2. **return**(**1**) % *satisfiabilitate*

sf_Dacă

Pasul 3. **Dacă** (($\exists$A)((A $\in$ *prop*(F)) $\wedge$ ({A},{$\rceil$A}$\in$ ***c***))

atunci % *ieșire*

Pasul 4. ***c*** := {□} % *„forțat – mai* $\exists$…”

Pasul 5. *prop*(F) := Ø % *„forțat”*

Pasul 6. **return**(**0**) % *nesatisfiabilitate*

sf_Dacă

3-20 (79)

Pasul 7. **Dacă** $((\exists C)(\exists D)((C, D \in \boldsymbol{c}) \wedge (D \subseteq C))$

atunci

Pasul 8. *Alege un asemenea* C

Pasul 9. $\boldsymbol{c} := \boldsymbol{c} \setminus C$

Pasul 10. **return**(**DPLL**(**S**, ***c***)) % *apel recursiv* **SUBS**

sf_Dacă

Pasul 11. **Dacă** $((\exists L)(\exists C)((L \in prop(F)) \vee (\overline{L} \in prop(F)) \wedge (L \in C) \wedge$

$\wedge (C \in \boldsymbol{c}) \wedge$

$\wedge (\forall D)((D \in \boldsymbol{c}) \to (\overline{L} \notin D)))$

atunci

Pasul 12. *Alege asemenea* L *și* C

Pasul 13. **S** := **S'**, *unde* **S'** *coincide cu* **S**, *exceptând faptul că* **S'**(L) = **1**

Pasul 14. $\boldsymbol{c} := \boldsymbol{c} \setminus \{D \mid L \in D\}$

Pasul 15. $prop(F) := prop(F) \setminus \{L\}$

(*respectiv*: $prop(F) := prop(F) \setminus \{\overline{L}\}$)

Pasul 16. **return**(**DPLL**(**S**, ***c***)) % *apel recursiv* **PURE**

sf_Dacă

3-21 (80)

Pasul 17. **Dacă** $((\exists A)((A \in \mathit{prop}(F)) \wedge (\{A\} \in \boldsymbol{c}))$

atunci % *e măcar o clauză cu* $\rceil$A, *neunitară*

Pasul 18. ***c*** := ***c’***, *unde* ***c’*** *se obține din* ***c*** *prin scoaterea tuturor clauzelor care conțin* A; *de asemenea, din toate clauzele deja în* ***c’***, *se scoate literalul* $\rceil$A

Pasul 19. **S** := **S’**, *unde* **S’** *coincide cu* **S**, *exceptând faptul că* **S’**(A) = **1**

Pasul 20. *prop*(F) := *prop*(F) \ {A}

Pasul 21. **return**(**DPLL**(**S**, ***c***)) % *apel*

sf_Dacă % *recursiv* **UNIT** „*cu* A”

Pasul 22. **Dacă** $((\exists A)((A \in$ *prop*$(F)) \wedge (\{\rceil A\} \in$ ***c***$))$

atunci % *e măcar o clauză cu* A, *neunitară*

Pasul 23. ***c*** := ***c’’***, *unde* ***c’’*** *se obține din* ***c*** *prin scoaterea tuturor clauzelor care conțin* $\rceil$A; *de asemenea, din toate clauzele deja în* ***c’’***, *se scoate literalul* A

Pasul 24. **S** := **S’’**, *unde* **S’’** *coincide cu* **S**, *exceptând faptul că* **S’’**(A) = **0**

Pasul 25. *prop*(F) := *prop*(F) \ {A}

Pasul 26. **return**(**DPLL**(**S**, ***c***)) % *apel*

sf_Dacă % *recursiv* **UNIT** „*cu* $\rceil$A”

3-23 (62)

Pasul 27. *Alege* A, A $\in$ *prop*(F), A $\in$ C_1 (C_1 $\in$ ***c***, *neunitară*)

% *apel recursiv* **SPLIT**; *există și un* C_2 *„la fel, dar cu* $\rceil$ A”

Pasul 28. $\boldsymbol{c_1}$:= ***c’***, *unde* ***c’*** *se construiește ca în* **Pasul 18**

Pasul 29. $\mathbf{S_1}$:= **S’**, *unde* **S’** *se construiește ca în* **Pasul 19**

Pasul 30. $\boldsymbol{c_2}$:= ***c’’***, *unde* ***c’’*** *se construiește ca în* **Pasul 23**

Pasul 31. $\mathbf{S_2}$:= ***S’’***, *unde* ***S’’*** *se construiește ca în* **Pasul 24**

Pasul 32. *prop*(F) := *prop*(F) \ {A}

Pasul 33. **return**(**DPLL**($\mathbf{S_1}$, $\boldsymbol{c_1}$) $\vee$ **DPLL**($\mathbf{S_2}$, $\boldsymbol{c_2}$)) % *apel*

% *recursiv simultan (branșare)*; <u>**c**</u>: și *i* „$\vee$” *j*, *cu i*, *j* $\in$ **B**, *la final*

sf_procedure DPLL(**S**, ***c***).

- ***Apelul*** procedurii recursive anterioare se face desigur prin **DPLL**(**S**, ***c***); <u>este esențial că pașii se fac conform scrierii (**c**)</u>; pentru algoritmul **DP**(**LL**) în sine rămân de precizat doar intrările și ieșirile, evidente din cele discutate deja

3-24 (83)

Algoritmul DP(LL)

Intrare. Orice formulă F $\in$ **LP**, conținând măcar un simbol, aflată în **FNC** și reprezentată ca mulțime de clauze (clauză = mulțime de literali), notată ***c*** (*prop*(F) și **S** denotă lucruri cunoscute).

Ieșire. „**DA**”, dacă formula F este satisfiabilă + structura **S**, finală, care este model pentru F; „**NU**”, în cazul în care F este nesatisfiabilă.

Metodă.

Pasul 1. *Verifică apartenența lui* F (nevidă) *la* **LP**

Pasul 2. *Obține* ***c*** (din F)

Pasul 3. *Obține prop*(F) (din F)

Pasul 4. *Inițializează funcția* **S**, **S**(A) := **0**, *pentru fiecare* A $\in$ *prop*(F)

Pasul 5. DPLL(S, *c*)

Pasul 6. Dacă (DPLL(S, *c*) = 1)

atunci

Pasul 7. Scrie „DA” și S

sf_Dacă

Pasul 8. Dacă (DPLL(S, *c*) = 0)

atunci

Pasul 9. Scrie „NU”

sf_Dacă

3-26 (85)

- Comentarii ajutătoare privind **(I)** algoritmul **DP**(**LL**) și **(II)** procedura recursivă **DPLL** (nu uităm de <u>demonstrație ... detaliată în suplimente</u> ...)
- **Pentru (I)** - algoritmul:

-Primii 4 pași sunt necesari doar dacă nu au fost deja efectuați; oricum, înainte de apelul procedurii recursive (în **Pasul 5** din algoritm), presupunem că ***c*** și **S** sunt în forma cerută

- Tot înainte de **Pasul 5** eliminăm „dublurile” din fiecare clauză, precum și clauzele duble (din F); din ***c*** se elimină și toate clauzele care sunt tautologii (conțin măcar un literal și complementarul său); clauzele de forma {} se elimină și ele; în acest fel putem „ajunge” iar la ***c*** = Ø (și, implicit, la *prop*(F) = Ø), ceea ce, de fapt, este același lucru cu a spune că F este vidă și execuția algoritmului nostru **nu mai are rost**

3-27 (86)

-Execuția procedurii va returna în final, după epuizarea tuturor (auto)apelurilor, doar valorile **0** sau **1**

-Nu insistăm asupra algoritmilor „adiacenți” menționați (sau asupra limbajului particular de programare folosit la implementare și structurile de date concrete): *algoritm pentru testarea apartenenței* (*membership*) lui F (nevidă) la **LP**; construirea unei **FNC** pentru formula inițială; transformarea lui F în mulțime (nevidă) de mulțimi (nevide), cu eliminarea dublurilor; eliminarea tautologiilor; calculul lui *prop*(F) etc.

-Presupunem totuși că la apelul (inițial al) **DPLL**: ***c*** este o mulțime (nevidă) de mulțimi (nevide) de literali (atomii corespunzători regăsindu-se în *prop*(F), de asemenea nevidă) și că **S** este o funcție „pusă pe **0**” (în sensul precizat)

3-28 (87)

- **Pentru (II)** – procedura:

-**Pasul 1** identifică terminarea unui apel al procedurii (se returnează **1**) și execuția ulterioară a **Pasul**ui **7** din **(I)** (mai exact: „**DA**”, F este satisfiabilă și **S** $\models$ F)

-Similar, **Pasul 3** identifică terminarea unui apel al procedurii (se returnează **0**) și execuția ulterioară a **Pasul**ui **9** din **(I)** („**NU**”, F este nesatisfiabilă)

-În **Pasul 7** (din **(II)**) se execută o operație **SUBS** asupra „formulei” curente ***c***, o clauză C $\in$ ***c*** fiind aleasă și apoi eliminată deoarece este subsumată de o altă clauză D $\in$ ***c***; ***c*** se modifică corespunzător, **S** și *prop*(F) nu suferă modificări și, desigur, se reia procedura, recursiv, în mod „liniar”, adică *fără branșare*

3-29 (88)

-**Pasul 11** din **(II)** descrie execuția unei operații **PURE** asupra configurației curente ***p*** = <**S**, ***c*** >, prin alegerea unei clauze C $\in$ ***c*** și a unui literal L, pur (pozitiv sau negativ), L $\in$ C; se fac transformările cunoscute asupra lui **S** și ***c*** (precum și asupra lui *prop*(F)), rezultând noua configurație ***p'***, cu care se reia execuția fără branșare a lui **(II)**; atomul corespunzător lui L se elimină din mulțimea *prop*(F) curentă

-În **Pasul 17** se identifică în configurația curentă un fapt pozitiv C (= {A}, {A} $\in$ ***c***), se fac transformările indicate de o operație **UNIT** și se reia execuția **DPLL** (din nou, fără branșare); în noua configurație ***p'*** vom avea astfel (înainte de noul auto-apel al lui **(II)**) **S'**(A) = **1**, și, formal:

c ' := {C $\in$ ***c*** |A $\notin$ C și ($\rceil$A) $\notin$ C} $\cup$ {C \ {$\rceil$A}| C $\in$ ***c*** }; cum o clauză nu poate conține atât pe A cât și pe $\rceil$A (acestea fiind deja eliminate), în noua „formulă” nu se vor regăsi clauze care să-l conțină (doar) pe $\rceil$A (cazul {A}, {$\rceil$A} $\in$ ***c*** fiind deja exclus prin **Pasul 3** din **(II)**, iar clauzele care îl conțineau pe A au fost complet eliminate); în acest mod, A nu va mai putea fi selectat din nou (ulterior) și el se elimină și din *prop*(F); repetăm, acest ultim lucru nu este esențial pentru procedură/ algoritm, dar face „mai vizibilă" terminarea, simplificând și demonstrația corectitudinii

-În mod similar cu ceea ce am descris mai înainte, **Pasul 22** din **(II)** reprezintă tot o operație **UNIT**, efectuată de data aceasta asupra faptului negativ C = {$\rceil$A}; A și $\rceil$A au roluri „inversate" (deși, desigur, tot A se va scoate din *prop*(F)), în noua configurație având **S''**(A) = **0** și

c'' := {C $\in$ ***c*** | A $\notin$ C și ($\rceil$A) $\notin$ C} $\cup$ {C \ {A} | C $\in$ ***c***}

3-31 (90)

-Ultimii pași ai procedurii (**Pasul 27** - **Pasul 33**) descriu o operație **SPLIT** efectuată asupra unei clauze C $\in$ ***c*** (nevidă, neunitară) și asupra unui atom A $\in$ C; această operație va genera o branșare și, în consecință, o continuare a execuției algoritmului pe două ramuri diferite cu (*re*, sau *auto*)apelări recursive (diferite) ale procedurii **DPLL**

-Deoarece operațiile admise (**SUBS**, **PURE**, **UNIT**, **SPLIT**) <u>se execută</u> în **(II)** <u>în ordinea textuală a scrierii</u> procedurii (prioritatea fiind indicată chiar de lista precedentă), putem presupune că, în momentul execuției unui **SPLIT** ca mai sus, „formula” curentă ***c*** este nevidă, nu conține clauze vide sau care sunt „tautologii imediate”, nu conține literali puri și nu conține clauze unitare (nici pozitive, nici negative); în acest caz, în condițiile date (nu se poate aplica niciuna dintre **SUBS**, **PURE**, **UNIT**), <u>chiar există</u> (măcar) o clauză C, neunitară, care conține (măcar) un atom A (altfel, faptele/ clauze unitare s-ar elimina prin **UNIT**; iar dacă C ar conține doar literali negativi, atunci măcar unul dintre ei ar trebui să existe nenegat într-o altă clauză neunitară și neeliminată încă - în caz contrar, C însuși ar fi fost eliminată prin **PURE**); concluzia imediată este că **Pasul 27** este „valid" și se poate aplica o operație **SPLIT**

-Subliniem încă o dată că ***orice apel*** (recursiv) al **procedurii DPLL** (deci și ***orice execuție*** a **Algoritmului DP**(**LL**)) ***se termină***, indiferent dacă este vorba despre o ramură „normală" sau despre una rezultată prin branșare: după orice „transformare" (în urma execuției oricărei operații), „formula"/ configurația curentă are, strict (pe „ramura" respectivă), fie mai puține clauze, fie mai puțini literali (fie ambele); deci, „lungime" mai mică ...

3-33 (92)

-Mai mult (exceptând operația **SUBS**), odată cu fiecare aplicare a unei operații, ***exact un atom*** este „afectat” de o asignare prin **S** (ulterior, el nu va mai putea fi ales; oricum, va fi scos din *prop*(F))

- ***Important de reținut***:

-Convingeți-vă că ați înțeles exact algoritmii descriși (sau sugerați)

-Faceți singuri alte exemple (eventual, dintre cele propuse pentru Seminar)

-Reluați probema **(II)<u>Trei fetițe obraznice</u>**, din **Cursul 1**, și rezolvați-o după „schema” **DP**(**LL**)

3-34 (93)

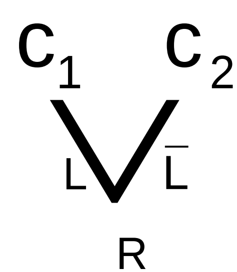
- Link-uri suplimentare utile: deja ar trebui să știți singuri de ceea ce aveți nevoie; totuși link-ul *Informații suplimentare pentru fiecare curs* (adică, de acolo, partea privind ...) ar fi cumva ***foarte necesar*** măcar ***de consultat***, deoarece conține ***demonstrația formală completă*** de corectitudine și completitudine a algoritmului principal (și vă ajută la o înțelegere mai profundă a structurii acestuia, dacă aveți răbdare ...)

FINAL Curs 3

4-1 (95)

În scopul prezentării altor algoritmi sintactici pentru SAT-LP, continuăm cu studiul *rezoluției* pentru LP

- Reamintim că lucrăm cu formule din **LP** aflate în **FNC**, reprezentate sub formă de mulţimi (finite) de clauze, iar clauzele ca mulţimi (finite) de literali
- **Definiţie (rezolvent)**. Fie clauzele C_1, C_2, R. Spunem că R **este rezolventul lui** C_1, C_2 (sau că C_1, C_2 **se rezolvă în** R, sau că R **se obţine prin rezoluţie *într-un pas* din** C_1, C_2), pe scurt, $R = Res_L(C_1, C_2)$, dacă şi numai dacă există un literal $L \in C_1$ astfel încât $\overline{L} \in C_2$ şi $R = (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\})$
- Vom putea reprezenta acest lucru (1-*rezoluția*) şi grafic, prin *arborele de rezoluţie*:



4-2 (96)

Observații (poate unele afirmații sunt deja cunoscute):

- Rezolventul a două clauze este este tot o clauză (mai mult, rezolventul a două clauze Horn este tot o clauză Horn)
- Clauza vidă (□ - ***nesatisfiabilă***) poate fi obţinută prin rezoluţie din două clauze de forma $C_1 = \{A\}$ şi $C_2 = \{\rceil A\}$
- Dacă în definiţia anterioară C_1 şi C_2 ar coincide, atunci $C_1 = C_2 = (C =) \ldots \vee L \vee \ldots \vee \overline{L} \vee \ldots \equiv \mathbf{1}$, adică acele clauze sunt tautologii, neinteresante d.p.d.v. al satisfiabilității și detectabile (chiar sintactic) *aprioric*
- Vom „rezolva” astfel doar clauzele în care literalul L cu acea proprietate este unic (<u>**e**</u> ...); și R poate fi validă …

4-3 (97)

- **Teoremă (lema rezoluţiei)**. Fie orice formulă $F \in$ **LP** (aflată în **FNC** şi reprezentată ca mulţime de clauze) şi R un rezolvent pentru C_1, $C_2 \in F$. Atunci $F \equiv F \cup \{R\}$.

(**<u>d</u>**: arătăm că pentru fiecare structură corectă **S**, dacă **S**(F) = **1**, atunci **S**($F \cup \{R\}$) = **1**; și reciproc)

- În teorema anterioară am fi putut considera în loc de F o mulţime oarecare de clauze, chiar infinită (numărabilă, cf. **Teoremei de compactitate**, care <u>va urma</u> ...)

4-4 (98)

- **Definiţie**. Fie F o mulţime oarecare de clauze din **LP** şi C o (altă) clauză. Spunem că lista C'_1, C'_2 , … , C'_m este o **demonstraţie prin rezoluţie (*în mai mulţi paşi*) a lui C pornind cu (bazată pe) F** dacă sunt satisfăcute condiţiile:

  (i) Pentru fiecare $i \in [m]$ (*explicație*), fie $C'_i \in F$, fie C'_i este obţinut prin rezoluţie într-un pas din C'_j, C'_k, cu j, k < i și (desigur) $j \neq k$.

  (ii) $C = C'_m$.

4-5 (99)

- În condiţiile definiţiei, se mai spune că **C** este ***demonstrabilă prin rezoluţie*** în mai mulți pași, *pornind cu/ bazată pe* F (sau, ***în ipotezele date de*** F); **<u>c</u>**: 1-rezoluția „=” *regulă de inferență* ...
- Pentru aceasta, este de fapt suficient ca F să poată fi *inserată* (prezentă) într-o demonstraţie şi nu să fie neapărat ultimul element al acesteia
- Intuitiv, o demonstraţie prin rezoluţie în mai mulţi paşi înseamnă o succesiune finită de rezoluţii într-un pas, care poate fi reprezentată şi grafic (printr-un arbore, sau chiar un graf oarecare ... *desen*; și *fracții „supraetajate”*; **<u>c</u>**: și după **N1**; revenim în legătură cu *demonstrații, raționamente* ...)

4-6 (100)

- Dacă C = □ (*French*: …), atunci demonstraţia respectivă se numeşte şi ***respingere*** (*English*: …)
- **Numărul de paşi** dintr-o demonstraţie bazată pe F este dat de *numărul de clauze obţinute prin rezoluţie într-un pas* (din clauze anterioare), la care se adaugă de obicei și numărul de clauze din F („inițiale”/ „axiome” ...) folosite în demonstrație
- Acesta poate fi considerat ca fiind o *măsură* a „mărimii” (*lungimii/ dimensiunii*) demonstraţiei
- Altă măsură (pentru o demonstraţie reprezentată *ca text*): lungimea „listei”, adică numărul total de clauze (sau numărul total de clauze distincte; ***câte sunt***, dacă … ?)
- Apoi, dacă reprezentăm o demonstraţie ca un arbore, putem folosi şi măsuri specifice, cum ar fi *adâncimea/ înălțimea* arborelui, *numărul de niveluri,* etc.

4-7 (101)

- **Definiţie (mulţimea rezolvenţilor unei mulţimi de clauze)**. Fie F o mulţime de clauze din **LP** (nu neapărat finită). Notăm succesiv:

  -Res(F) = F U {R | există C_1, $C_2 \in$ F astfel încât R = Res(C_1, C_2)}.

  -$\mathrm{Res}^{(n+1)}$(F) = Res($\mathrm{Res}^{(n)}$(F)), n $\in$ **N**.

  -$\mathrm{Res}^{(0)}$(F) este o altă notație pentru F şi atunci vom putea „pune” și

  $\mathrm{Res}^{(1)}$(F) = Res(F).

  Mai mult, considerăm mulțimea *tuturor rezolvenților*:

$$\mathrm{Res}^*(F) = \bigcup_{n \in N} \mathrm{Res}^{(n)}(F)$$

- $\mathrm{Res}^{(n)}$(F) este astfel ***mulţimea rezolvenţilor lui*** F ***obţinuţi în cel mult*** *n* ***paşi***, iar Res*(F) - ***mulţimea (tuturor) rezolvenţilor lui*** F
- Aceasta constituie definiţia *iterativă* a lui Res*(F)
- Va urma (imediat) și o *definiție structurală* (**c**: *iterativ* vs *recursiv*...)

4-8 (102)

- Direct din definiţie urmează că:

  $F = Res^{(0)}(F) \subseteq Res(F) = Res^{(1)}(F) \subseteq \ldots$

  $\ldots \subseteq Res^{(n)}(F) \subseteq \ldots \subseteq \ldots \subseteq Res^*(F)$
- Vom nota acum cu Resc(F) mulţimea definită prin:

  Baza. $F \subseteq Resc(F)$.

  Pas constructiv: Dacă $C_1, C_2 \in Resc(F)$ şi $R = Res(C_1, C_2)$, atunci $R \in Resc(F)$.

  (**Nimic altceva** ...)
- <u>**e**</u>: nu insistăm (la seminar; *Capitolul 2* ...)

4-9 (103)

- **Teoremă**. Pentru fiecare F $\in$ **LP**, avem Res*(F) = Resc(F).

(**d**: incluziunea Res*(F) $\subseteq$ Resc(F) se arată *prin inducție matematică*; invers, *prin inducție structurală*)

- Vom putea astfel folosi ambele notaţii (definiţii) pentru găsirea și/ sau manipularea mulţimii rezolvenţilor oricărei mulţimi (*cel mult numărabile* !) de clauze (deci și a unei formule oarecare F $\in$ **LP**, aflată în **FNC** și reprezentată ca o mulțime de clauze)

- **Teoremă**. Fie F o mulţime de clauze din **LP** (nu neapărat finită). O clauză C $\in$ **LP** se poate demonstra prin rezoluţie pornind cu clauzele lui F ddacă există k $\in$ **N**, asfel încât C $\in$ $Res^{(k)}$(F).

(**d**: „$\Leftarrow$”, e imediată; apoi … simplă: nr. finit de literali/ număr maxim de clauze …)

4-10 (104)

- **Teoremă**. Fie F $\in$ **LP**, aflată în **FNC** şi reprezentată ca mulţime (finită) de clauze. Atunci Res*(F) este finită.

(**d**: simplă)

- **Teoremă (de compactitate, pentru LP).** Fie **M** o mulţime infinită (doar *numărabilă*; *de ce* ?) de formule din **LP**. Atunci **M** este satisfiabilă dacă şi numai dacă fiecare submulţime **finită** a sa este satisfiabilă.

(fără **d**: este totuși în cartea tipărită; *interesantă* …)

Important. Putem **reformula** teorema de compactitate astfel: *„O mulțime infinită (numărabilă !) de formule este nesatisfiabilă dacă și numai dacă există o submulțime finită a sa care este nesatisfiabilă”* (și folosirea cuvântului „nesatisfiabilă” în teorema următoare devine „naturală”).

4-11 (105)

- **Teoremă (teorema rezoluţiei pentru LP)**. Fie F o mulţime oarecare de clauze din **LP**. Atunci F este nesatisfiabilă dacă şi numai dacă $\square \in$ Res*(F).

(**<u>d</u>** – *idei*: din teorema de compactitate reformulată, este suficient să considerăm că F este finită; implicația „$\Leftarrow$”, numită <u>*corectitudinea*</u> (vezi și …) *rezoluției*, se arată folosind teoremele enunțate anterior și aplicând de un număr finit de ori lema rezoluției; invers, „$\Rightarrow$”, adică <u>*completitudinea*</u>, rezultă demonstrând prin inducție matematică *metateorema*: ($\forall$*n* $\in$ **N**)(**dacă** |*prop*(F)| = *n* **și** F este nesatisfiabilă, **atunci** $\square \in$ Res*(F))) (*carte …*)

4-12 (106)

- Folosind rezoluția, deducem (mai jos) un alt algoritm sintactic pentru rezolvarea **SAT-LP**; reamintim că ea este, totuși, **NP-*completă*** (vezi **Cursul 8**)
- Exceptând anumite subclase „convenabile” ale **LP** (e.g., *subclasa alcătuită din formulele Horn* – suplimentele la **Cursul 2** …), am putea folosi anumite ***strategii*** pentru a ajunge *cât mai repede* la clauza vidă
- ***Restricțiile*** sunt și ele utile atunci când strategiile nu ne sunt de nici un folos/ nu se pot aplica
- ***Rafinările*** rezoluţiei (= strategii + restricții) sunt metode prin care se urmăreşte *obţinerea clauzei vide (dacă acest lucru este posibil) într-un număr cât mai mic de paşi de rezoluţie* (**PROLOG**; ceva *explicații – mai jos*…)

4-13 (107)

- ***Important de reținut***:

-Pornind cu „formula”, în **FNC**, $F = \{C_1, C_2, \dots , C_n\}$, unde clauzele sunt scrise ca mulţimi de literali, se poate *construi* mulţimea $Res^*(F)$, care este finită și poate fi reprezentată ca un graf (ne)orientat (chiar arbore, dacă repetăm aparițiile unor noduri având o aceeași etichetă)

-În graf, nodurile sunt rezolvenţii succesivi, inclusiv clauzele iniţiale, iar muchiile sunt introduse prin rezoluţiile aplicate într-un pas (*desenul*: cu $Res^{(i + 1)}(F) \setminus Res^{(i)}(F)$...)

-Acest graf *poate să cumuleze toate demonstraţiile posibile care pornesc cu clauzele lui* F (anumiţi rezolvenţi vor fi însă excluşi sintactic, deoarece reprezintă/ generează tautologii)

4-14 (108)

-Demonstrația **Teoremei rezoluţiei** sugerează astfel un *nou algoritm sintactic* (după **DP(LL)**) pentru rezolvarea **SAT-LP** (cumva deja descries pe scurt): se *construiește* întâi „*graful de rezoluţie total* ” descris mai sus şi apoi se *parcurge* acesta pentru a vedea dacă □ este (eticheta unui) nod în graf

-Mai exact, algoritmul (*rudimentar* …) menționat creează și „inspectează” graful complet/ total de rezoluție

-Evident că este mult mai „eficient” să găsim **direct** o respingere în loc de a crea şi parcurge întregul graf

-Altfel spus, putem restrânge de la bun început spațiul de căutare, construind „cât mai repede” acel subgraf (nici măcar el în întregime...) care ar putea să conțină □ (chiar dacă ***complexitatea formală*** rămâne la fel ...)

4-15 (109)

- Link-uri suplimentare utile: cum am mai spus, ar trebui să știți singuri de ceea ce aveți nevoie în acest moment; dar, ca mai înainte, link-ul *Informații suplimentare pentru fiecare curs* (partea „potrivită”) este util în ceea ce privește consolidarea unor cunoștințe predate în **Cursul 4** (prin studiul strategiilor și restricțiilor rezoluției); de menționat *Capitolul 2* (pentru studiul separat al clauzelor Horn) și *Capitolul 5* (prezentarea limbajelor de tip **PROLOG**)

FINAL Curs 4

5-1 (111)

- Revenim asupra semanticii **LP**, mai întâi asupra a ceea ce numim ***algebra funcțiilor booleene***
- $\mathbf{B} = \{\mathbf{0}, \mathbf{1}\}$
- $\mathbf{B}^n = \mathbf{B} \times \mathbf{B} \times \ldots \times \mathbf{B}$ (de $n \in \mathbf{N}^*$ ori)
- $\mathbf{FB}^{(n)} = \{f \mid f : \mathbf{B}^n \to \mathbf{B}\}$
- Vom pune și: $\begin{cases} \mathbf{FB} = \bigcup\limits_{n \geq 0} \mathbf{FB}^{(n)} \\ \mathbf{FB}^{(0)} = \mathbf{B} \end{cases}$
- Orice funcţie booleană n-ară, ca element al lui $\mathbf{FB}^{(n)}$, deci al lui **FB**, poate fi reprezentată printr-o *tabelă de adevăr* (**c**: $|C|^{|D|}$; 2^n, 2 *la puterea* 2^n; *legături* F $\in$ **LP** $\rightsquigarrow$ f $\in$ **FB**: desenul pt. **LP** …)
- Funcții necesare din $\mathbf{FB}^{(n)}$ ($n = 0, 1, 2, \ldots$)

5-2 (112)

- 4-uplul $\mathcal{B} = \langle \mathbf{B}, \bullet, +, \overline{\ } \rangle$ (**B** este o mulțime *suport*, „•”, „+” și „$\overline{\ }$” sunt, respectiv, *produsul*, *suma* și *opusul „în baza 2”*) este o ***algebră Boole***, adică sunt satisfăcute „legile” (operații cu mulțimi …):

1) $x \bullet y = y \bullet x$ *comutativitatea* lui „•”

2) $(x \bullet y) \bullet z = x \bullet (y \bullet z)$ *asociativitatea* lui „•”

3) $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

distributivitatea lui „•” faţă de „+”

4) $(x \bullet y) + y = y$ *absorbţia*

5) $(x \bullet \overline{x}) + y = y$ *legea contradicţiei*

5-3 (113)

1') $x + y = y + x$ *comutativitatea* lui „+"

2') $(x + y) + z = x + (y + z)$ *asociativitatea* lui „+"

3') $x + (y \bullet z) = (x + y) \bullet (x + z)$

distributivitatea lui „+" faţă de „•"

4') $(x + y) \bullet y = y$ *absorbţia*

5') $(x + \overline{x}) \bullet y = y$ *legea tautologiei*

- Simbolul „=" reprezintă aici egalitatea de funcții
- *Principiul dualității* și alte considerente algebrice (**e**: *latici*, mulțimi *parțial ordonate*; *Capitolul 1* …)
- **e** (la Seminar...): alte legi (vezi **Tabelul 1.1**, pag.30, din carte sau **slide 85** din *Suplimente* …)

5-4 (114)

- Pentru partea de hard (descrierea structurii fizice reale a unui computer), partea teoretică similară algebrelor booleene e numită ***teoria circuitelor***
- Funcțiile booleene se pot reprezenta și ca *text/ expresii*
- Notaţii (1 și 0, ca indici „sus” nu sunt elemente din **B**):
  $x^1 \triangleq x$ şi $x^0 \triangleq \overline{x}$ (dar …)
- Acești indici nu se supun astfel principiului dualităţii (de exemplu, nu este adevărat că: „($x^1 = x$) *coincide cu* ($x^0 = x$)”
- Dacă $x, \alpha, x_i, \alpha_i \in$ „**B**” atunci, direct din notaţiile de mai sus, rezultă că: $(x^0)^\alpha = (x^\alpha)^0$; şi: $x^\alpha =$ **1** ddacă $x = \alpha$ (prin simplă verificare și **Tabelul 1.1**)

5-5 (115)

- **Teoremă** *(de descompunere, în sumă de „termeni”)*. Pentru fiecare $n \in$ **N***, $f \in$ **FB**$^{(n)}$ şi fiecare $k \in [n]$, avem:

$$f(x_1, x_2, \ldots, x_n) = \sum_{\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathrm{B}} x_1^{\alpha_1} \bullet x_2^{\alpha_2} \bullet \ldots \bullet x_k^{\alpha_k} \bullet f(\alpha_1, \alpha_2, \ldots, \alpha_k, x_{k+1}, \ldots, x_n)$$

oricare ar fi x_1, x_2, ... , $x_n \in$ **B**.

(**d**)

- Voi: Enunţaţi ***teorema duală*** („α cu bară”, *doar unde nu este* ...)
- **Observație**. Nu confundați „formulă” cu „funcție”, etc.

5-6 (116)

- **Definiţie**. Fie $n \in$ **N*** şi $x_1, x_2, \ldots, x_n \in$ **B** variabile/ nume („booleene” …) distincte; notăm mulţimea (de fapt, lista, sau …) acestora cu $X = \{x_1, x_2, \ldots, x_n\}$. Se numeşte ***termen (n-ar, peste*** X***)*** orice produs

$$t = x_{i_1}^{\alpha_1} \bullet x_{i_2}^{\alpha_2} \bullet \ldots \bullet x_{i_k}^{\alpha_k}$$

unde $0 \leq k \leq n$, $\alpha_1, \alpha_2, \ldots, \alpha_k \in$ **B** şi

$1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

5-7 (117)

- Termenul generat pentru $k = 0$ este „**1**” (prin convenţie; privit ca element din $\mathbf{FB}^{(0)}$); pentru $k = n$ obţinem aşa-numiţii *termeni maximali* (***maxtermeni***), adică *acei* termeni în care fiecare dintre variabilele considerate apare o dată şi numai o dată (barată sau nebarată), în ordinea precizată (adică $x_1, x_2, \ldots, x_n$)
- Numărul de termeni şi maxtermeni *n*-ari distincţi (și maxtermenii sunt termeni): 3^n și 2^n (de ce ? *calculați* voi…)
- Dual: ***factori*** şi ***maxfactori***

5-8 (118)

- **Definiţie** (din nou, atenție: evitați confuziile).

-Se numeşte ***formă normală disjunctivă*** (*n*-***ară***, $n \in$ **N***), pe scurt (*n*-)**FND**, orice sumă finită de termeni *n*-ari distincţi.

-Se numeşte ***formă normală disjunctivă perfectă*** (*n*-***ară***), (*n*-)**FNDP**, orice sumă finită de maxtermeni *n*-ari distincţi.

- Facând abstracţie de ordinea/ comut. (max)termenilor dintr-o sumă, vom considera că *oricare două sume care diferă doar prin ordinea termenilor sunt identice*
- Atunci vor exista combinări de 3^n luate câte *k* forme normale disjunctive *n*-are având *k* termeni, $0 \leq k \leq 3^n$ (prin convenţie, pentru $k = 0$, unica formă care este acceptată, *fiind şi perfectă*, va fi „**0**” – celălalt element din $\mathbf{FB}^{(0)}$)

  -Care va fi numărul total al (n -)**FND** – urilor? Dar cel al (n-)**FNDP**–urilor ?

5-9 (119)

- **Teoremă**. Orice funcţie booleană f se poate „reprezenta” în mod **unic** ca o **FNDP**:

$$f(x_1, x_2, ..., x_n) = \sum_{\alpha_1, \alpha_2, ..., \alpha_n \in \mathrm{B}} x_1^{\alpha_1} \bullet x_2^{\alpha_2} \bullet ... \bullet x_n^{\alpha_n} \bullet f(\alpha_1, \alpha_2, ..., \alpha_n)$$

(<u>**d**</u> – descompunerea...; apoi, *f*(…) **= 1**, care poate fi „pus jos, la sumă”); se deduce și un algoritm pentru „tabel versus text”, la reprezentarea funcțiilor; poate, alte <u>**c**</u>: +/ • vs ∨/ ∧, evitare confuzii …

- Mai spunem că expresia din membrul drept al reprezentării este <u>**o**</u> **FND(P)** pentru *f*

5-10 (120)

- Prin dualizare, folosind noţiunile de (*n*-)factor peste X şi maxfactor (*n*-ar, peste X), putem defini noţiunea de *formă normală conjunctivă* (*n*-ară) ((*n*-)**FNC**: orice produs de factori dictincţi) şi respectiv *formă normală conjunctivă* (*n*-ară) perfectă ((*n*-)**FNCP**: orice produs de maxfactori distincţi)
- **Convenţie**: doi factori nu vor fi considerați a fi distincți dacă diferă doar prin ordinea componentelor
- Enunţaţi duala teoremei anterioare, pentru **FNCP**
- Numărul total al (*n*-)**FNC** – urilor și cel al (*n*-)**FNCP**–urilor : 2 *la* 3^n respectiv 2 *la* 2^n (de ce ? e altfel, numeric, față de **FND(P)** ?)

5-11 (121)

- Există și alte modalități de a reprezenta/ genera clase întregi de funcții booleene (inclusiv **FB**)

Se poate vorbi astfel despre:

- *Forme normale disjunctive/ conjunctive minimale* (corespunzător: *algoritmul lui Quine*)
- *Clasa funcțiilor booleene elementare*, *superpoziții*, M-*șiruri*, *închideri* și *mulțimi închise de funcții booleene* (Ø, **E**, **FB**, $\mathbf{T_0}$, $\mathbf{T_1}$, **Aut**, **Mon**, **Lin**); *mulţimi precomplete*, *complete*, *baze*, etc. (vezi link-ul *Capitolul 1*...)

5-14 (122)

- O ultimă modalitate importantă de reprezentare a funcțiilor booleene este cea folosind *diagramele de decizie binare (ordonate)*: (**O***rdered*) **B***inary* **D***ecision* **D***iagrams*, pe scurt **(O)BDD**
- Ştim ce înseamnă *funcţie booleană* şi reprezentarea acestor funcții cu ajutorul tabelelor de adevăr/ (adică) matrici sau cu expresii/ texte (uzuale sau **FN**-uri)
- Alegerea celei mai „convenabile” reprezentări (ca *structură de date*) depinde însă de context (totuși, în principal, de problema de rezolvat)
- În anumite cazuri o (**O**)**BDD** poate fi mai „compactă” și mai „vizibilă” decât o tabelă de adevăr/ text pentru o aceeaşi funcţie (este un graf și anumite redundanţe pot fi exploatate mai „simplu/ vizibil”)

5-15 (123)

- **Definiţie.** Se numeşte *diagramă de decizie binară peste* <u>lista</u> X = {x_1, x_2, … , x_n} un graf *orientat*, *aciclic*, *etichetat* (pe noduri şi pe arce) în care:

  -există o unică *rădăcină* (nod în care nu intră nici un arc);

  -*frunzele* (nodurile din care nu iese nici un arc) sunt etichetate cu **0** sau **1**, iar celelalte noduri (inclusiv rădăcina) sunt etichetate cu elemente din X (aici se permit etichetări multiple, adică noduri diferite pot avea aceeaşi etichetă); ideea este şi ca fiecare x_i să fie folosit *măcar o dată*; cerinţa nu este însă obligatorie;

  -fiecare nod care nu este frunză are exact doi succesori imediaţi, arcele care îi leagă fiind și ele etichetate: cu **0** (*stânga*) respectiv **1** (*dreapta*)

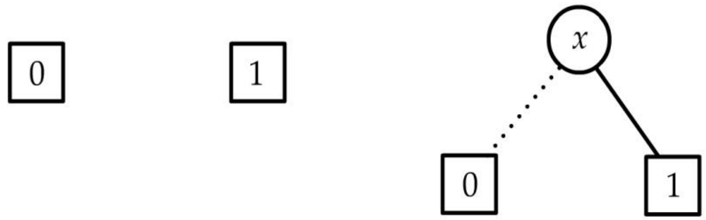
- O **subBDD** (într-o **BDD** dată) este un *subgraf generat* de un nod fixat împreună cu *toţi* succesorii săi (inclusiv arcele care le leagă)

5-16 (124)

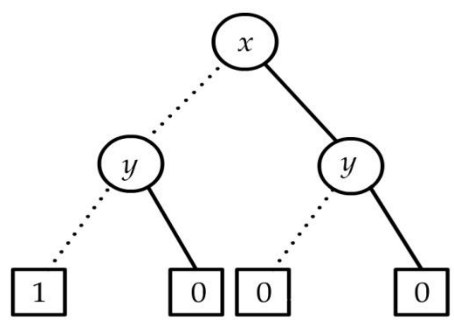
- De obicei, într-un „desen” care reprezintă o (**O**)**BDD**, frunzele pot fi identificate (şi) prin pătrate (nu cercuri, ca restul nodurilor), orientarea arcelor este implicită („de sus în jos”), arcele etichetate **0** sunt reprezentate prin linii punctate (stânga), iar cele etichetate **1** sunt linii continue (dreapta); formal, este evident altceva (structura de date …)
- În primele exemple (care urmează), grafurile sunt chiar arbori

5-17 (125)

- (I) **D0**, **D1** (peste ∅) şi **Dx** (peste X = {x}):



- (II) O **BDD** peste X = {x, y}



5-18 (126)

- **Observaţie.** Orice **BDD** peste $X = \{x_1, x_2, \ldots, x_n\}$ defineşte/ reprezintă/ <u>calculează</u> o **unică** funcţie booleană $f \in \mathbf{FB}^{(n)}$
- Astfel, pentru $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbf{B}$ (considerate ca fiind valori „de asignat variabilelor booleene” din X) se „porneşte” cu rădăcina (unică) şi se „parcurge” un drum (unic) în graf „până” la o frunză (să spunem că aceasta este etichetată cu $\beta \in \mathbf{B}$)
- La fiecare pas, plecând din nodul curent, se alege acel arc (prin urmare şi noul nod curent) care are ataşată valoarea **0** sau **1** conform valorii α deja atribuite ex-nodului curent x (în asignarea aleasă)
- Valoarea β este chiar $f(\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle)$

5-19 (127)

- În acest mod, **BDD**-ul din ultimul exemplu anterior reprezintă funcţia

$$f(x, y) = \overline{x} \cdot \overline{y}$$

- Este clar că putem proceda şi invers, adică pornind cu orice funcţie f $\in$ **FB**$^{(n)}$, dată printr-un tabel de adevăr, putem construi (<u>măcar</u>) un arbore (**BDD**) binar, complet şi având *n* + 1 *niveluri*, notate 0 - *rădăcina*, 1, … , *n* – 1; și *n* – pe care sunt frunzele (alternativ, arborele are *adâncimea n*) care „calculează” *f*, în modul următor:

5-20 (128)

- Se *(re)ordonează* mulţimea de variabile cu ajutorul căreia este exprimată funcţia, să zicem X = $\{x_1, x_2, \ldots, x_n\}$, sub forma $\langle x_{k,1}, x_{k,2}, \ldots, x_{k,n}\rangle$, $\langle k,1; k,2; \ldots; k,n\rangle$ fiind o permutare pentru $\langle 1, 2, \ldots, n\rangle$ (dar permutarea poate fi ea însăși)
- Nodurile interioare (care nu sunt frunze) situate pe nivelul i -1 sunt etichetate (**toate**) cu $x_{k,i}$ ($i \in [n]$); rădăcina este etichetată cu $x_{k,1}$ ea fiind (singurul nod de) pe nivelul 0
- Cele două arce care ies din fiecare nod sunt etichetate (normal) cu **0** şi respectiv **1**
- Frunzele sunt etichetate cu **0** sau **1** conform tabelei de adevăr pentru *f* (drumul de la rădăcină la frunza corespunzătoare furnizează exact linia care trebuie aleasă din tabelă: eticheta fiecărui arc de pe drum reprezintă valoarea atribuită variabilei care este eticheta nodului din care iese arcul)

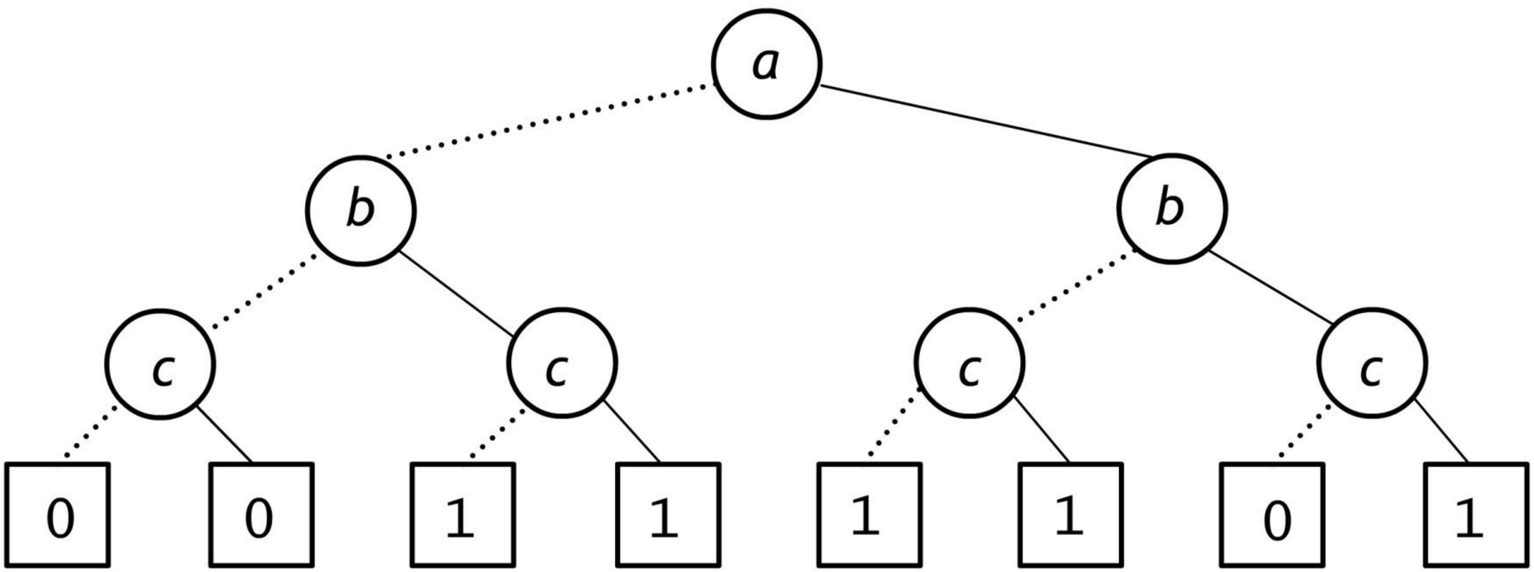
5-21 (129)

- **Exemplu.** Fie *f* $\in$ **FB**$^{(3)}$ dată prin

$$f(a,b,c) = (a+b)\cdot(\overline{a}+\overline{b}+c)$$

  deci exprimată cu ajutorul lui X = {a, b, c},
  $\langle$a, b, c$\rangle$ fiind ordinea impusă asupra variabilelor;
  tabela sa de adevăr este… (voi)
- **BDD**-ul care calculează *f* după algoritmul sugerat anterior este... (urmează pe alt slide)
- **Observaţie**. În exprimările care urmează, câteodată nu vom face o distincţie explicită între o funcţie booleană şi o formulă din **LP** (deși „este de evitat” …)

5-22 (130)



5-23 (131)

- Construirea unei **BDD** care calculează o funcţie dată nu este un proces cu rezultat unic (spre deosebire de procesul „invers”): e destul să schimbăm ordinea variabilelor ca să găsim altceva (dar…)
- <u>Impunerea</u> unei ordini asupra etichetelor nodurilor este un prim pas spre găsirea unor *forme normale* (și) pentru **BDD**-uri
- Un alt aspect care trebuie avut în vedere pentru atingerea acestui scop este acela că reprezentarea ca arbore a unei **BDD** nu este deloc mai eficientă/ compactă decât o tabelă de adevăr (nici decât, de exemplu, o **FNC(P)**): dacă $f \in$ **FB**$^{(n)}$, atunci tabela sa de adevăr va avea 2^n linii iar în reprezentarea **BDD** sugerată de noi (ca arbore, în care fiecare nivel este „destinat” unei variabile şi pe fiecare drum de la rădăcină la o frunză apar toate variabilele exact o dată) vor fi exact $2^{n+1} – 1$ noduri
- Putem <u>*compacta*</u> o **BDD** dacă îi aplicăm următoarele <u>*procedee de reducere/ optimizare*</u> (în cele de mai jos, când ne referim la nodul n, m, etc. nu ne referim la eticheta din X; sunt doar niște nume/ etichete noi):

5-24 (132)

C1 (înlăturarea frunzelor duplicat). Dacă o **BDD** conţine mai mult de o frunză etichetată cu **0**, atunci:

-păstrăm una dintre ele;

-ştergem celelalte frunze etichetate cu **0**, împreună cu arcele aferente, care de fapt se „redirecţionează" spre singura **0**-frunză rămasă (fiecare păstrându-şi nodul sursă);

-se procedează în mod similar cu **1**-frunzele; admitem şi înlăturarea unei frunze dacă, <u>în final</u>, ea nu are nici un arc incident cu ea.

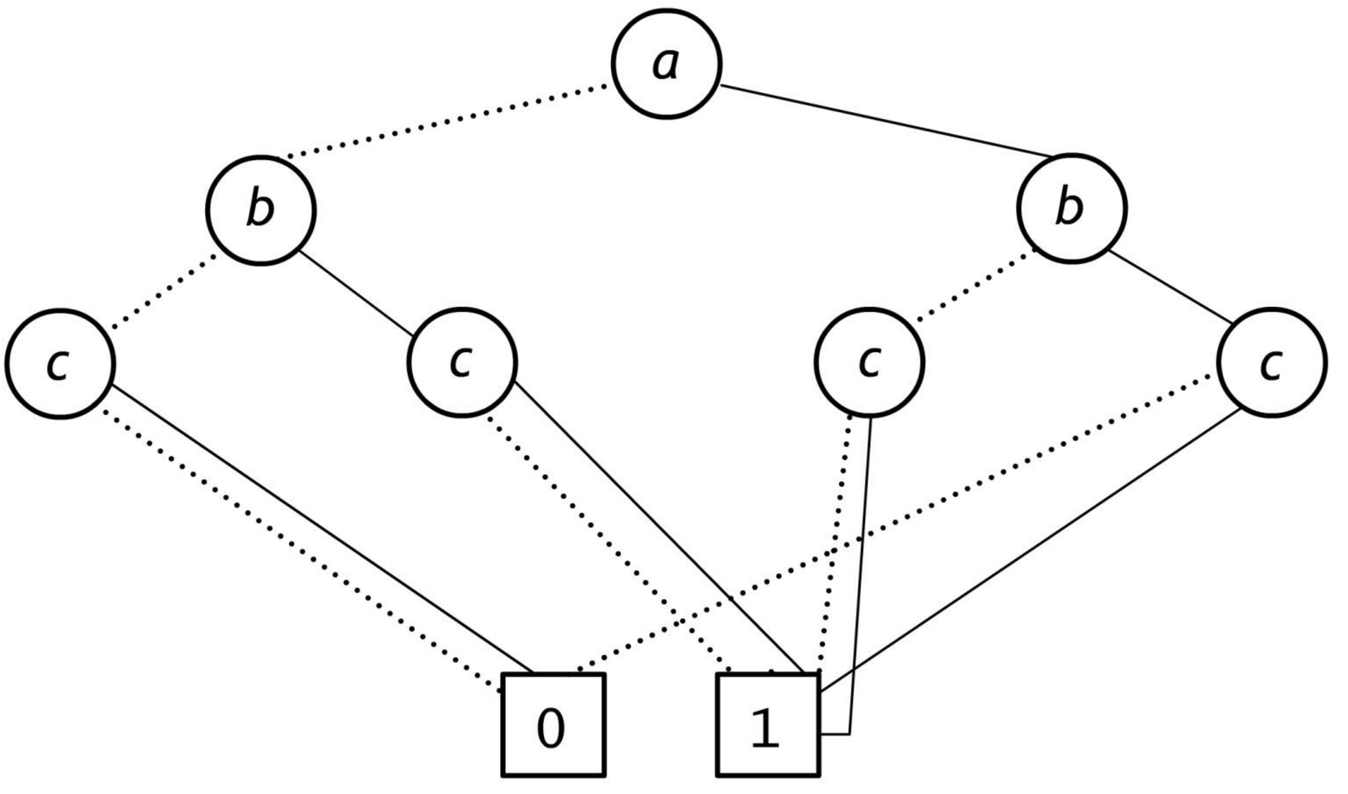
C2 (eliminarea „testelor" redundante). Dacă în **BDD** există un nod (interior) n pentru care atât **0**-arcul cât şi **1**-arcul au ca destinaţie acelaşi nod m (lucru care se poate întâmpla doar dacă s-a efectuat măcar un pas de tip **C1**), atunci nodul n se elimină (împreună cu arcele sale care „punctează" spre m), iar arcele care înainte punctau spre n sunt „redirecţionate" (derect) spre m.

C3 (eliminarea nodurilor duplicat care nu sunt frunze). Dacă în **BDD** există două noduri interioare distincte, să zicem n şi m, care sunt rădăcinile a două **subBDD**-uri identice (fiind identice, n şi m sunt şi pe acelaşi nivel, să zicem că m este „plasat mai în dreapta"), atunci se elimină unul dintre ele, să zicem m (împreună cu arcele care-l au ca sursă), iar arcele care punctau spre m se „redirecţionează" spre n.

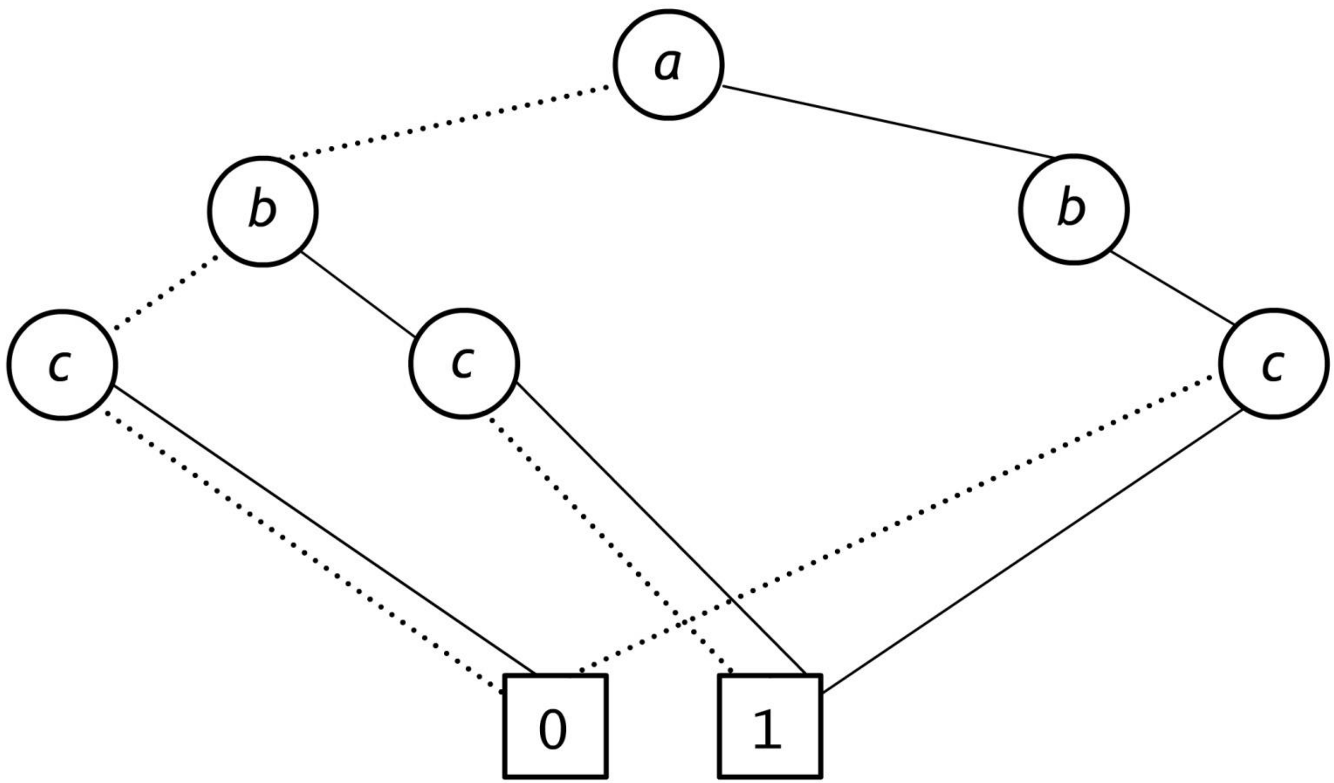
5-25 (133)

- **Exemplu.** Plecând de la **BDD**-ul din ultimul exemplu construit, obţinem succesiv (mai întâi am aplicat toate transformările posibile de tip **C1**, apoi toate cele de tip **C3** şi, în sfârşit, toate cele de tip **C2**):

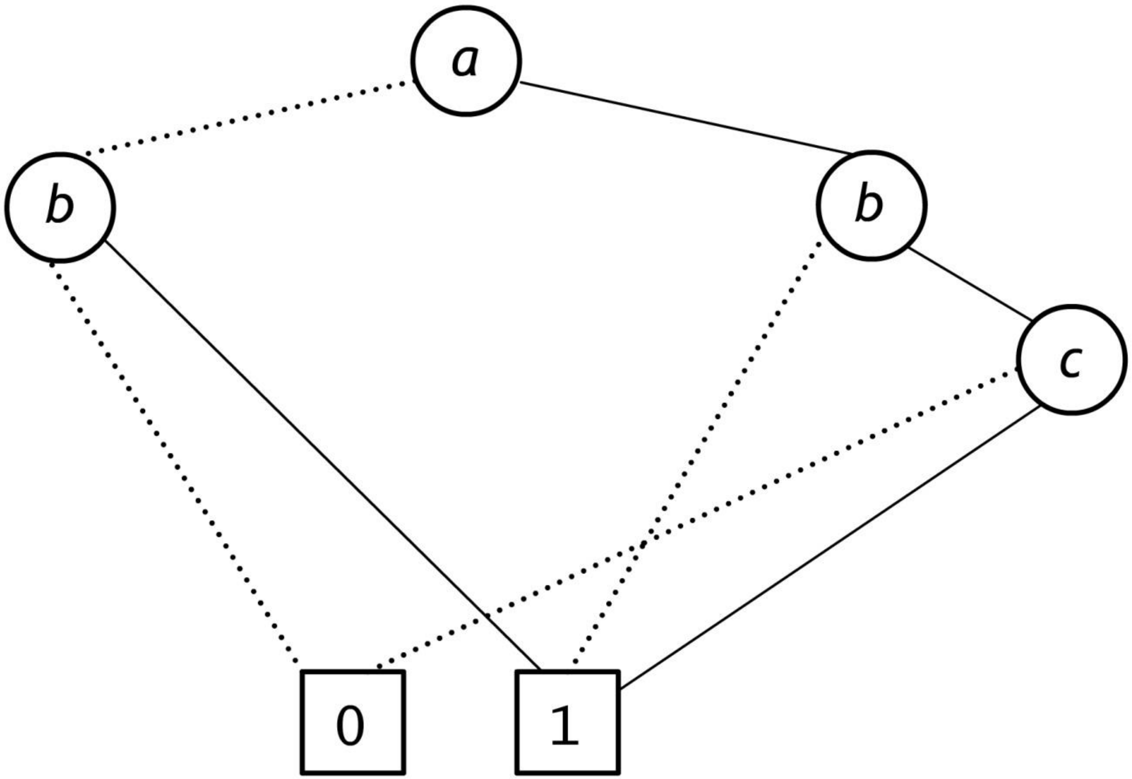
5-26 (134)



5-27 (135)



5-28 (136)



5-29 (137)

- Ultima **BDD** este ***maximal redusă*** (nu se mai pot face alte transformări de tipurile precizate)
- Desigur că ordinea în care se efectuează eliminările de tipul **C1**-**C3** <u>poate influenţa</u> aspectul structural al unei **BDD** şi <u>pot exista</u> mai multe transformări distincte care să fie simultan admise pentru o aceeaşi structuri
- În schimb, <u>nicio transformare **nu modifică** funcţia booleană calculată</u>
- De fapt, ar fi bine să *vă puneți singuri anumite întrebări privind relația exactă dintre conceptele introduse* …

5-30 (138)

- ***Important de reținut***:

-**BDD**-urile pot fi uneori „convenabil de compacte”

-Putem reformula anumite definiţii ale funcţiilor booleene pentru noua reprezentare, căpătând, de exemplu, idei noi de algoritmi pentru rezolvarea problemei **SAT**

-Dacă celelalte reprezentări ale funcțiilor booleene sunt utile pentru „zona software” (programare ...), **(O)BDD**-urile sunt utile în special pentru „zona hardware” (circuite logice)

-Proiectarea sistemelor multiagent, verificarea automată folosind teoria modelelor și anumite logici specializate utilizează și ele intens aceste diagrame

5-31 (139)

-Ideea de bază (în ceea ce privește revenirea la semantica **LP** într-o formă mai „aprofundată”) este legată de introducerea unei modalitatăți de a „lega” formulele propoziționale de funcțiile booleene

-**LP** $\rightsquigarrow$ clase „$\equiv$” (și structuri) $\rightsquigarrow$ **FB** /(**O**)**BDD**, și reciproc (nu prea a fost timp de <u>(**O**)</u> – *Suplimente*)

-De reținut noile modalități de a furniza/ reprezenta constructiv întreaga clasă **FB** (ca și „**LP**”)

-Pentru aprofundarea unor concepte cum ar fi *decidabilitate* și *complexitate*, *probleme* și *algoritmi*, *paradigme*, **P** vs. **NP**, *reduceri*, *automate*, consultați suportul suplimentar de informație (<u>Cursul 8 ...</u>)

5-32 (140)

-Problema de a decide, pentru o funcție booleană, dacă „ia” doar valoarea **0**, doar valoarea **1**, sau atât valoarea **0** cât și valoarea **1** (**B**oolean **S**atisfiability **P**roblem - **BSP**) este de importanță majoră în teoria complexității

-Reținem că <u>din aceasta derivă (istoric) de fapt varianta</u> pentru **LP**/ **FB** (adesea *identificată* cu precedenta) și numită **P**ropositional **S**atisfiability **P**roblem (**PSP**) sau **Sat**isfiability problem (pe scurt: **SAT**; cu „versiuni” deja posibil ade a fi intuite – **(3)CNFSAT** etc.)

5-33 (141)

- Câteva remarci despre *gramatici/* **BNF**-uri, *automate*...
- Link-uri suplimentare utile: *Capitolul 1* și *Capitolul 2* (pentru studiul funcțiilor booleene, a clasei **FB**, în general); **H/ R** și *Informații suplimentare pentru fiecare curs* (aici, suplimentele la **Cursul 5**), în special în legătură cu diagramele de decizie binare ***ordonate***, pe care nu le-am studiat deloc; aveți în vedere mereu *exercițiile propuse* …

FINAL Curs 5

6-1 (143)

- Începem ***tratarea globală a claselor de formule*** (**<u>e</u>**: **LP**); vorbim despre *sisteme deductive* (**SD** - noțiune **sintactică**) și *teorii logice* (**TE** - noțiune **semantică**)
- O ***teorie logică*** este o mulţime de formule *închisă la consecinţă semantică* (**<u>e</u>**: *clasa formulelor valide* dintr-o logică dată)
- Un ***sistem deductiv*** este o mulțime compusă din ***axiome*** (+ ***ipoteze*** *suplimentare …*) și ***reguli de inferenţă*** (+ ***reguli*** *derivate …*)
- O ***teoremă de corectitudine şi completitudine*** certifică legătura dintre „adevăr (semantic)” și „demonstrație (sintactică)”: *tot ceea ce este „demonstrabil” este „adevărat”* (*corectitudine*) și, reciproc, *tot ceea ce este „adevărat” este „demonstrabil”* (*completitudine*)

6-2 (144)

- **Definiție**. O mulţime **TE** de formule este teorie logică dacă pentru fiecare submulţime M $\subseteq$ **TE** şi fiecare (altă) formulă G care este consecinţă semantică din M, avem G $\in$ **TE**.
- Notaţii pentru „consecinţă semantică”:

  $\mathcal{I} \models$ F; $\models$ F sau Ø $\models$ F (pentru „F – validă” ...

  „*adevărat prin lipsă/ true by default*”)
- Cu alte cuvinte, în momentul de față ne va interesa mulțimea

$\mathcal{Val}$(**LP**) = {F $\in$ **LP** | F e validă} $\triangleq$ {F $\in$ **LP** | $\models$ F}

6-3 (145)

- **Definiție**. Se numeşte sistem deductiv în **LP** un cuplu **SD** = $\langle \mathcal{A}, \mathcal{R} \rangle$ unde
  - $\mathcal{A} \subseteq$ **LP** este o mulţime de *axiome* iar
  - $\mathcal{R} \subseteq$ **LP**$^+ \times \mathcal{C}$ este o mulţime de *reguli de inferenţă* (*de deducţie*, *de demonstraţie*)
- **LP**$^+$ denotă mulţimea relaţiilor de oricâte argumente (cel puţin unul) peste **LP**, iar $\mathcal{C}$ reprezintă o mulţime de *condiţii de aplicabilitate*
- Fiecare regulă de inferenţă r $\in \mathcal{R}$, are astfel aspectul r = $\langle\langle G_1, G_2, \ldots, G_n, G\rangle, c\rangle$, unde n $\in$ **N**, iar $G_1, G_2, \ldots, G_n, G \in$ **LP** și $c \in \mathcal{C}$

6-4 (146)

- G_1, G_2, … , G_n sunt *ipotezele (premizele) regulii*, G reprezintă *concluzia (consecinţa)* iar *c* desemnează *cazurile (modalităţile) în care regula poate fi aplicată*
- Putem scrie și r = $\langle\langle\{G_1, G_2, \ldots, G_n\}, G\rangle, c\rangle$ deoarece *ordinea ipotezelor nu este esenţială*
- Mulţimea $\mathcal{C}$ nu a fost specificată formal, din cauza generalităţii ei; elementele sale se mai numesc (meta)*predicate* (condiții „de satisfăcut" petru formule, demonstrații, etc.)
- Regulile vor fi folosite pentru a construi demonstraţii în paşi succesivi, la un pas aplicându-se o regulă (**e**: am făcut deja despre *demonstrații prin rezoluție într-un pas*, demonstrații prin rezoluție în mai mulți pași, etc. ...)

6-5 (147)

- O regulă r = $\langle\langle\{G_1, G_2, \ldots , G_n\}, G\rangle, c\rangle$ va fi scrisă şi ca (arbore, grafic ...):

$$\frac{G_1, G_2, \ldots, G_n}{G}, c$$

- Câteodată, alături de *c*, sunt explicitate *separat* şi *restricţiile sintactice locale* asupra (formei) (meta)formulelor
- În cazul în care n = 0 şi *c* lipseşte, r poate fi identificată ca fiind o axiomă, după cum rezultă din definiţia care va urma
- Regulile/ axiomele sunt, de fapt, nişte *scheme generale*

6-6 (148)

- De fapt, există posibilitatea, prin *c*, ca în afara restricţiilor sintactice „locale”, date de sintaxa formulelor implicate să se interzică, e.g., aplicarea regulii (schemei) pe considerente „globale” (forma demonstraţiei, apariţia în demonstraţie a unei formule nedorite la acel pas, păstrarea completitudinii unei teorii logice, etc.)
- Dacă *c* este ataşată unei reguli r (*c* poate lipsi; mai exact, ea poate fi „condiţia permanent adevărată indiferent de context”), înseamnă că în *orice* demonstraţie, r va putea fi aplicată la un moment dat *doar dacă c* este adevărată la momentul respectiv

6-7 (149)

- **Definiţie**. Fie un sistem deductiv **SD** = $\langle \mathcal{A}, \mathcal{R} \rangle$ în **LP**. Se numeşte *demonstraţie* (pentru F_k, pornind cu $\mathcal{A}$) în **SD** o listă de formule, ($\mathcal{D}$), F_1, F_2, … , F_k astfel încât pentru fiecare i $\in$ [*k*], fie $F_i \in \mathcal{A}$, fie F_i este obţinut din F_{j1}, F_{j2}, … , F_{jm} folosind o regulă r = $\langle\langle\{F_{j1}, F_{j2}, \dots, F_{jm}\}, F_i\rangle, c\rangle$ din $\mathcal{R}$,, unde $j_1, j_2, \dots, j_m < i$.
- Fiecare element al listei ($\mathcal{D}$) este fie o axiomă, fie este concluzia unei reguli de inferenţă ale cărei ipoteze sunt elemente anterioare din listă (toate acestea se numesc și ***teoreme***)

6-8 (150)

- O demonstrație (***raționament formal***), se poate reprezenta și ca un graf, sau chiar ca un arbore...
- Un sistem de demonstrație poate conține, pe lângă axiome (de regulă - formule *valide*, „știute” ca fiind așa printr-o *altă metodă credibilă*), și niște *ipoteze/ axiome suplimentare*; de obicei, acestea sunt *măcar* formule *satisfiabile*)
- Vorbim atunci despre **SD’** = $\langle \mathcal{A}', \mathcal{R} \rangle$, $\mathcal{A}' = \mathcal{A} \cup \mathcal{I}$, $\mathcal{I}$ reprezentând axiomele suplimentare
- Notăm cu $\mathcal{Th}$(**SD**) = {F $\in$ **LP** | există măcar o demonstraţie ($\mathcal{D}$) pentru F în **SD**} (<u>dați voi</u> o <u>definiţie constructivă echivalentă</u> pentru $\mathcal{Th}$(**SD**))

6-9 (151)

- În legătură cu sistemele deductive putem discuta despre: *tipuri* de sisteme deductive (*boolean complete*, *finit axiomatizabile* etc.), sau despre *clasificarea generală* a acestora (Hilbert, Gentzen, etc.); nu insistăm ...
- Exemplele tratate (în *Suplimente* …) sunt toate *corecte și complete pentru clasa formulelor valide* ($\mathcal{Val}$(X)) din orice logică dată (la noi, X $\triangleq$ **LP**) și *echivalente* între ele
- **Definiţie.** Două sisteme **SD'** şi **SD''** sunt ***echivalente*** dacă pentru fiecare mulţime de formule $\mathcal{I}$ (poate fi chiar vidă) şi fiecare formulă F avem:

  $\mathcal{I} \vdash_{\mathbf{SD'}}$ F dacă şi numai dacă $\mathcal{I} \vdash_{\mathbf{SD''}}$ F.

6-10 (152)

- Dacă un sistem are „mai multe” reguli de inferență decât axiome, el se numește de tip Gentzen(-Jaskowski)
- Un asemenea sistem va fi **SD0**, sau **deducţia naturală**, care nu are nicio axiomă (!); revenim mai jos …
- În cazul în care balanţa este inversată (există „mai multe” axiome decât reguli de inferenţă), sistemele sunt cunoscute sub numele de sisteme (de tip) Hilbert

- **Definiţie (*regulă de inferenţă derivată*).** Considerând orice prefix al oricărei demonstraţii (privită textual) ($\mathcal{D}$) dintr-un sistem **SD**, acesta poate fi considerat ca o nouă regulă de inferenţă („derivată” din cele iniţiale): concluzia noii reguli este ultima formulă din demonstraţia respectivă, iar ipotezele sunt reprezentate de restul formulelor care apar.

6-12 (154)

- Prezentăm **SD0** (***deducția naturală***), cf. **H/ R**
- Revenim la problema din primul curs, **(III)<u>O întâlnire</u>** =

Dacă trenul ajunge mai târziu și nu sunt taxiuri în stație, atunci John va întârzia la întâlnirea fixată. Se știe că John nu întârzie la întâlnire, deși trenul ajunge într-adevăr mai târziu. Prin urmare, erau taxiuri în stație; notasem:

p: *Trenul ajunge mai târziu*

q: *Sunt taxiuri în stație*

r: *John întârzie la întâlnirea fixată*

- Ajunsesem la ideea că pentru a rezolva problema trebuie să arătăm că *secvența* $(p \wedge (\neg q)) \rightarrow r, \neg r, p \vdash q$ este *validă/ corectă/* ***sound*** (în sens *sintactic*: „dacă … și … și … atunci q”)

6-13 (155)

- Formal, o ***secvență*** este un text **(s)** $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ (formulele $\varphi_1, \varphi_2, \ldots, \varphi_n, \psi \in$ **LP**: păstrăm notațiiile și „stilul” din **H/ R**)
- Demonstrațiile în **SD0** (definiția generală nu se schimbă: axiome + reguli ...) pot „lucra” chiar asupra lor însăși: implicit sau explicit (vezi conceptul de „box”), pot conține alte demonstrații, exprimate eventual prin alte reguli de inferență (*reguli derivate*, în sensul anterior)
- Semantica **0**/ **1** (cunoscută de noi) o vom utiliza doar la nivel informal/ intuitiv (**c**: semantica nici nu e, încă, definită formal; deci n-avem nici proprietăți sintactice …)
- În acest cadru, o secvență **(s)** este ***validă*** dacă, pornind cu $\varphi_1, \varphi_2, \ldots, \varphi_n$ ca ipoteze suplimentare, se „ajunge” la ψ drept concluzie finală prin utilizarea succesivă a regulilor

6-14 (156)

- Sintaxa **LP** (în **H/ R**) este puțin diferită de cea cunoscută de noi (formulele atomice se notează cu p, q, ...; conectorii logici „apar” toți; **<u>c</u>**)
- Mai precis, în *forma Backus-Naur* (**BNF**) avem (**<u>c</u>**):

φ ::= p | ($\neg\,\varphi$) | ($\varphi \wedge \varphi$) | ($\varphi \vee \varphi$) | ($\varphi \rightarrow \varphi$)

- **Observație**. Pentru o scriere „textuală” mai simplă, în „**Tabel**ul tuturor celor 16 reguli” (care <u>va urma</u>) vom denota un *dreptunghi/ cutie/ box* (vezi și exemplele care vor urma) prin **box**(σ, τ), unde σ este *presupunerea*, iar τ este „concluzia demonstrației înscrise în cutie”
- **SD0** <u>nu conține decât (scheme de/ șabloane/ *pattern-uri*) reguli de inferență și nicio axiomă</u>

6-15 (157)

- Cele 16 reguli de inferență (**Tabel** din **H/ R**, adică Fig. 1.2, p.27), le vom și „numi”, respectiv: ($\wedge$**i**), ($\wedge$$\mathbf{e_1}$), ($\wedge$$\mathbf{e_2}$), ($\vee$$\mathbf{i_1}$), ($\vee$$\mathbf{i_2}$), ($\vee$**e**), ($\rightarrow$**i**), ($\rightarrow$**e**), ($\neg$**i**), ($\neg$**e**), ($\bot$**e**), ($\neg\neg$**e**)
- Ultimele 4 sunt reguli derivate, numite respectiv (**MT**), ($\neg\neg$**i**), (**PBC**) și (**LEM**); adică: **M***odus* **T***ollens* (*modul negativ*), the *Law of Double Negation* (*legea dublei negații*), **P***roof* **B***y* **C***ontradiction* (*reducerea la absurd* - *reductio ad absurdum*), și the **L***aw of the* **E***xcluded* **M***iddle* (*terțiul/ al treilea este exclus* - *tertium non datur*)
- **i** = **i***ntroducere*; **e** = **e***liminare* (în tabel: stânga/ dreapta)
- Într-o „demonstrație de validitate”, la fiecare pas, va fi o problemă ***alegerea*** regulii potrivite/ *matching* (<u>reveniți voi</u> la **(III)** = **Exemplul 1** din **Curs 1**; necesită „box” !)

6-16 (158)

Tabel

$$\frac{\varphi, \quad \psi}{\varphi \wedge \psi}\ (\wedge\mathbf{i}) \qquad \frac{\varphi \wedge \psi}{\varphi}\ (\wedge\mathbf{e_1}) \qquad \frac{\varphi \wedge \psi}{\psi}\ (\wedge\mathbf{e_2})$$

$$\frac{\varphi}{\varphi \vee \psi}\ (\vee\mathbf{i_1}) \qquad \frac{\psi}{\varphi \vee \psi}\ (\vee\mathbf{i_2}) \qquad \frac{\varphi \vee \psi,\ \mathbf{box}(\varphi, \theta),\ \mathbf{box}(\psi, \theta)}{\theta}\ (\vee\mathbf{e})$$

$$\frac{\mathbf{box}(\varphi, \psi)}{\varphi \rightarrow \psi}\ (\rightarrow\mathbf{i}) \qquad \frac{\varphi, \quad \varphi \rightarrow \psi}{\psi}\ (\rightarrow\mathbf{e/\ MP})$$

6-17 (159)

$$\frac{\mathbf{box}(\varphi, \perp)}{\neg\varphi}\ (\neg\mathbf{i}) \qquad \frac{\varphi, \quad \neg\varphi}{\perp}\ (\neg\mathbf{e})$$

$$\frac{\perp}{\varphi}\ (\perp\mathbf{e}) \text{ (de pus în dreapta) } \frac{\neg\neg\varphi}{\varphi}\ (\neg\neg\mathbf{e})$$

- Și liniile (5) anterioare sunt etichetate respectiv cu: ($\wedge$), ($\vee$), ($\rightarrow$), ($\neg$/ $\neg$), ($\perp$), ($\neg\neg$/ $\neg\neg$)
- În plus, cele 4 reguli derivate amintite sunt:

6-18 (160)

$$\frac{\varphi \to \psi, \quad \neg\psi}{\neg\varphi}\ (\mathbf{MT}) \qquad\qquad \frac{\varphi}{\neg\neg\varphi}\ (\neg\neg\mathbf{i})$$

$$\frac{\mathbf{box}(\neg\varphi, \bot)}{\varphi}\ (\mathbf{PBC}) \qquad\qquad \frac{}{\varphi \vee \neg\varphi}\ (\mathbf{LEM})$$

- Înafară de exemplificări, vom prezenta și ***intuiția*** *procedurală/* imperativă din „spatele” regulilor: ***cum*** se aplică o regulă și *când*
- Rezolvând exemplele, va deveni transparentă și intuiția de tip *declarativ*: ***de ce*** regula are forma considerată …)

6-19 (161)

- ($\wedge$**i**): pentru a demonstra $\varphi \wedge \psi$, trebuie mai întâi să demonstrăm separat φ și ψ (apoi – folosirea efectivă)
- ($\wedge$**e**$_1$): pentru a demonstra φ, se încearcă a se demonstra $\varphi \wedge \psi$ (abia apoi – utilizarea lui φ); acesta nu pare o idee prea bună: probabil ar fi mai greu de demonstrat $\varphi \wedge \psi$ decât de a se demonstra direct doar φ; există însă și posibilitatea ca $\varphi \wedge \psi$ să fi fost deja (anterior) demonstrată ...
- ($\vee$**i**$_1$): pentru a arăta $\varphi \vee \psi$, se încearcă întâi să se demonstreze φ; ca posibilitate, ar putea fi însă mai greu să se demonstreze φ (doar dacă nu cumva ... este deja) decât să se demonstreze $\varphi \vee \psi$; astfel, dacă vrem să arătăm $q \vdash p \vee q$, nu vom putea folosi *doar* pe ($\vee$**i**$_1$); dar probabil va „merge” dacă utilizăm și ($\vee$**i**$_2$) …

6-20 (162)

- ($\vee$**e**): dacă avem demonstrată $\varphi \vee \psi$ și dorim să „arătăm o (altă) afirmație” θ, e nevoie să să arătăm θ <u>atât „din”</u> φ, <u>cât și „din”</u> ψ (folosind, <u>eventual, și alte *premise* sau *presupuneri* avute</u> deja la dispoziție; la *box*, revenim…); asta deoarece nu știm care dintre ele (φ/ ψ) este adevărată
- ($\rightarrow$**i**): similar; dacă vrem să demonstrăm $\varphi \rightarrow \psi$, pare mai util să încercăm să demonstrăm pe ψ, pornind cu *presupunerea* că φ (mai „simplă” …) este adevărată (sau, poate chiar e deja demonstrată); revenim (<u>box</u> – explicată formal)
- ($\rceil$**i**): pentru a arăta $\rceil\varphi$, este (poate mai) bine să demonstrăm $\perp$ / ***false*** „din” *presupusa* φ (*box* …)

6-21 (163)

- Regulile ($\neg$**e**) și ($\perp$**e**) sunt, intuitiv, simplu de explicat: *dacă atât* φ, *cât și* $\neg\varphi$ *sunt adevărate, atunci nu putem demonstra decât* ***false***; respective: *din* ***false*** *deducem orice*
- În context, $\perp$ (clauza/ formula vidă, notată de noi $\square$) denotă *orice* formulă ***false***, adică de forma $\varphi \wedge \neg\varphi$ (sau $\neg\varphi \wedge \varphi$; știm că n-avem încă semantică/ proprietăți sintactice …); dar, **e**:
- **Exemplul 2**. Arătați că *secvența* $p \wedge q, r \vdash q \wedge r$ este validă.
- Prim mod de (tran)scriere a unei secvențe: $p \wedge q$

  r

  spații (interlinii)

  $q \wedge r$
- A *construi o demonstrație* înseamnă a completa spațiile aflate între premize și concluzie (prin aplicarea unei secvențe de reguli „potrivite”)

6-22 (164)

- Obținem

1 $p \wedge q$	*premiză*	
2 r	*premiză*	
3 q	$(\wedge \mathbf{e_2})$	1
4 $q \wedge r$	$(\wedge \mathbf{i})$	3, 2

- Desigur că φ și ψ din **Tabel**, pot fi oricând *instanțiate* nu doar cu formule atomice (sunt „scheme” ...; substituții ...)
- Demonstrația precedentă poate fi prezentată și printr-un arbore etichetat (desen …), având rădăcina (etichetată cu) $q \wedge r$ și frunzele $p \wedge q$, și r:

6-23 (165)

$$\dfrac{p \wedge q}{q}\ (\wedge \mathbf{e_2})\ (\text{arc} \ldots)$$

$$\dfrac{q \qquad\qquad\qquad\qquad r}{q \wedge r}\ (\wedge \mathbf{i})\ (\text{arce} \ldots)$$

- Astfel, pentru a demonstra o concluzie/ construi o demonstrație/ găsi o secvență validă/ elabora un raționament (corect/ sound): liste „sus-jos/ arbori (utilizând și ordini diferite de aplicare a regulilor, reguli diferite etc.)
- **Important**: orice demonstrație este de fapt **verificată** că este **corectă**, în sensul aplicării „sound” a tuturor regulilor, la fiecare pas), prin „controlul” fiecărei linii în parte („începând de sus”, în reprezentarea textuală liniară/ listă): este din **Tabel**, este instanțiată prin substituție formală etc.

6-24 (166)

- Nici explicarea intuitivă a regulilor referitoare la dubla negație (i.e. ($\neg\neg$**e**) și derivata ($\neg\neg$**i**)) nu este complicată
- Nu există astfel vreo diferență între formulele φ și $\neg\neg\varphi$: *The sentence* “It is **not** true that it does **not** rain” *is just a more sophisticated way of saying* “It rains”
- **Exemplul 3**. Demonstrați singuri (înainte de ... a vă uita la ceea ce urmează, ca și la exemplul anterior) validitatea secvenței:

  $p, \neg\neg(q \wedge r) \vdash \neg\neg p \wedge r.$

6-25 (167)

- Iată o demonstrație pentru **Exemplul 3**:

1	p	*premiză*
2	$\neg\neg(q \wedge r)$	*premiză*
3	$\neg\neg p$	($\neg\neg$**i**) 1
4	$q \wedge r$	($\neg\neg$**e**) 2
5	r	($\wedge$**e**$_2$) 4
6	$\neg\neg p \wedge r$	($\wedge$**i**) 3, 5

- **Exemplul 4**. Demonstrați că (<u>preferabil singuri</u>; apoi comparați ceea ce ați făcut voi cu soluția ce urmează): $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$.

6-26 (168)

1	$(p \wedge q) \wedge r$	*premiză*
2	$s \wedge t$	*premiză*
3	$p \wedge q$	**($\wedge$e$_1$)** 1
4	q	**($\wedge$e$_2$)** 3
5	s	**($\wedge$e$_1$)** 2
6	$q \wedge s$	**($\wedge$i)** 4, 5

- **Ce este cu însă cu box-urile ?**
- Vom explica mai pe larg (**$\rightarrow$i**), deja amintită, deoarece construirea unei implicații corecte este provocatoare (mai provocatoare decât „distrugerea”/ eliminarea uneia: cazul lui (**MP**)/ (**$\rightarrow$e**))
- În general, deschidem un box „când nu știm ce regulă să mai aplicăm”; și-l închidem „când se poate”

6-27 (169)

- Comentariile pentru celelalte reguli nedetaliate aici (($\vee$**e**) și ($\neg$**i**)) sunt similare
- Gândindu-ne acum la (**MT**) (este o regulă derivată și poate fi demonstrată ca atare …), putem afirma datorită ei (intuitiv) că secvența $p \rightarrow q, \neg q \vdash \neg p$ este validă: *If Abraham Lincoln was Ethiopean, then he was African. Abraham Lincoln was not African; therefore he was not Ethiopean*

6-28 (170)

- În consecință, similar cu (**MT**), pare la fel de plauzibil că și secvența $p \rightarrow q \vdash \rceil q \rightarrow \rceil p$ este validă, cele două „spunând *cam* același lucru”
- Mai în amănunt, plecând cu $p \rightarrow q$ ca premiză și ***presupunând temporar*** că $\rceil q$ este „adevărată” (ar fi trebuit să folosim o altă premiză, dar începem cu aceasta, care „pare” și ea „naturală”), putem chiar demonstra formal afirmația anterioară, în ceea ce urmează (**Exemplul 5**):

6-29 (171)

1	$p \rightarrow q$	*premiză*
2	$\neg q$	***presupunere*** *(=premiză temporară)*
3	$\neg p$	(**MT**) 1, 2
4	$\neg q \rightarrow \neg p$	($\rightarrow$**i**) 2-3

- În cele de mai sus, prin subliniere s-a „marcat” faptul că $\neg q$ și $\neg p$ *constituie* (pe verticală, în această ordine) dreptunghiul/ box care reprezintă ipoteza regulii ($\rightarrow$**i**), iar 2-3 specifică prima și ultima formulă din dreptunghi
- Dreptunghiul specifică de fapt domeniul sintactic al presupunerii temporare $\neg q$ (similare conceptual: *subprogram*, *variabilă locală*)

6-30 (172)

- În concluzie, am procedat astfel: am făcut presupunerea $\neg q$, „deschizând” un dreptunghi și „punând $\neg q$ deasupra/ la început”; am continuat să aplicăm reguli în mod normal (ca în construcția oricărei demonstrații, dar „în interiorul dreptunghiului”)
- Ultima regulă *care a depins de* $\neg q$ (aici - și singura) a fost (**MT**), prin care s-a creat $\neg p$; $\neg p$ „va merge” și ea în interiorul dreptunghiului
- Putem „ieși” acum din dreptunghi: nici regula aplicată, ($\rightarrow$**i**), nici concluzia $\neg q \rightarrow \neg p$, nu mai depind de presupunerea temporară $\neg q$

6-31 (174)

- Într-un alt context, ne putem gândi la $p \rightarrow q$ ca denotând ***tipul*** unei proceduri într-un limbaj de programare real; astfel, *propoziția* p *ar putea exprima faptul că procedura considerată așteaptă să primească la intrare o valoare intreagă* x, iar q *faptul că, la ieșire, aceeași procedură va returna o valoare booleană* y
- Atunci, „validitatea" lui $p \rightarrow q$ (a întregii secvențe) apare ca un „adevăr" al unei aserțiuni de forma *presupune-garantează*:

6-32 (174)

Dacă intrarea procedurii este un număr întreg, atunci ieșirea va fi o valoare booleană

- Acest lucru nu înseamnă că aceeași procedură nu poate face anumite „lucruri trăznite”, sau că ea nu se poate „bloca” dacă intrarea a nu va fi un număr întreg
- Revenind, regula (→**i**) are ca ipoteză un dreptunghi (care *începe cu formula indicată prin* φ *și se termină cu formula indicată prin* ψ, *între ele putând însă exista alte demonstrații*), iar drept concluzie pe $\varphi \to \psi$

6-33 (175)

- Deci, *pentru a ni se permite să introducem o implicație* ($\varphi \rightarrow \psi$) *într-un raționament*, *facem presupunerea* (*temporară*) *asupra lui* φ (că ea este adevărată) și *demonstrăm* (adevărul lui) ψ (dacă reușim, renunțăm apoi la φ ...)
- Partea cu „se demonstrează” este conținută în acele <u>„puncte, puncte”</u> din interiorul cutiei și trebuie respectate niște <u>reguli sintactice concrete, suplimentare</u>: <u>„acolo”</u>, se poate folosi φ și orice alte formule „valide”, inclusiv premize sau concluzii provizorii făcute/ obținute până la momentul/ punctul respectiv al demonstrației

6-34 (176)

- În continuare, <u>ca reguli generale</u>, o formulă θ se poate folosi la un punct din/ linie de demonstrație, doar dacă acea formulă a „apărut” (măcar ca premiză) în demonstrație înainte de punctul de folosire și dacă niciun dreptunghi în interiorul căruia se găsește (acea apariție a lui) θ nu a fost deja închis
- În cursul unei demonstrații oarecare, pot exista astfel doar dreptunghiuri imbricate sau disjuncte, și nu care se intersectează (*deschide un nou dreptunghi de-abia după ce s-a închis precedentul*, sau: *deschide-l și închide-l, dacă precedentul deschis nu a fost încă închis*)

6-35 (177)

- De asemenea, linia care urmează imediat celei prin care se „închide” un dreptunghi, trebuie să respecte *forma/ pattern-ul concluziei* (sintactic, ca formulă) regulii de inferență care utilizează acel dreptunghi ca ipoteză
- Pentru regula r = ($\rightarrow$**i**), aceasta înseamnă că imediat după un dreptunghi închis (care începe cu φ și se termină cu ψ), și care este ipoteza unei instanțe a lui r, trebuie să continuăm numai cu $\varphi \rightarrow \psi$

6-36 (178)

- Pentru regulile derivate (**PBC**) și (**LEM**), lucrurile sunt tot relativ simple, dacă ne gândim la intuiție
- ***Important de reținut***:

-Atenția „cade” asupra rezolvării **SAT** cu algoritmi sintactici, de dorit a fi „performanți” (existând diverse **ATP**-uri …)

-Și știm că există legături fundamentale între sintaxă și semantică: ***teoreme de corectitudine și completitudine***, concretizate prin proiectarea și implementarea unor ***algoritmi corecți și compleți față de* SAT**

6-37 (179)

-Din punctul de vedere al **Logicii**, rezolvarea oricărei probleme se reduce la rezolvarea **SAT** pentru o ***formulă*** (= „codificare" ***problemă reală***)

-Este nevoie „doar" de abilitatea de a selecta „logica" potrivită pentru a exprima „exact" cerințele problemei date printr-o formulă (din logica aleasă)

-Nu toți algoritmii *corecți și compleți* (necesitate !) *pentru rezolvarea* **SAT** sunt și *eficienți* (majoritatea – nu sunt; performanți – lucrează „pe subclase")

6-38 (180)

-Algoritmii ***sintactici*** sunt de preferat celor semantici, dar aceștia trebuie să folosească în substrat un ***sistem deductiv*** „dotat” cu o ***teoremă de corectitudine și completitudine*** vs o ***teorie logică*** ($\mathcal{Val}$ (**LP**) = $\mathcal{Th}$ (**SD0**))

-<u>De fapt avem: secvența $G_1, G_2, \ldots, G_n \vdash G$ este validă (sintactic !) ddacă</u>

<u>$F = G_1 \rightarrow (G_2 \rightarrow (\ldots \rightarrow (G_n \rightarrow G)) \ldots)$ este validă (semantic !)</u>

-Azi există suficiente demonstratoare automate comerciale (**Clam, Otter, Boyer-Moore, Setheo, PTTP, UT, Vampire, Isabelle, HOL, Spass**; anumite variante de **PROLOG**)

-**Notație**: $G_1 \dashv\vdash G_2$ denotă „$G_1 \vdash G_2$ și $G_2 \vdash G_1$”, pentru oricare $G_1, G_2 \in$ **LP**.

6-39 (181)

- Link-uri suplimentare utile: *Capitolul 1, Capitolul 2* și *Capitolul 5* ; *Informații suplimentare pentru fiecare curs*; aveți în vedere și *exercițiile propuse* …; desigur, și … **H/ R** (partea cu *Natural Deduction for Propositional Logic*)

FINAL Curs 6

7-1 (183)

- Acest **Curs** (**7**) conține o trecere în revistă a definițiilor (**D.**), teoremelor (**T.**) și algoritmilor (**A.**) din primele 6 cursuri (+ seminarii) și care trebuie cunoscute/ cunoscuți pentru *prima Lucrare de evaluare* (din săptămâna a 8-a: 20-26 noiembrie; pe 22, de fapt)
- În plus trebuie cunoscute și alte concepte, rezultate, rezolvări (de probleme), care nu au atașate (în mod explicit) denumirile amintite mai sus (le reamintim și pe acestea, curs cu curs)
- La test pot fi date și demonstrații (scurte) de teoreme (sub formă de exerciții), sau prezentări generale ale unor algoritmi

7-2 (184)

Cursul 1

- Conceptul de *Problem solving*; *definiții structurale*; *principiul inducției structurale* (pentru demonstrații în metalimbaj)
- *Sintaxa* **LP** (**D.**); atomi (variabile propoziționale positive și negative, …), conectori/ operatori logici (conective logice), alfabet (logic), formule; priorități pentru operatori și „eliminarea” parantezelor
- Metode și metodologii generale pentru rezolvarea problemelor reale (*idei*)

7-3 (185)

Cursul 2

- Definițiile constructive pentru *arb*(F) (inclusiv arborele sintactic), *subf*(F), *prop*(F), unde F $\in$ **LP**
- Literali (pozitivi și negativi; literali complementari); clauze (unitare, pozitive, negative, ~~Horn~~; clauza vidă: □)
- *Structură* (/ asignare/ interpretare) (**D.**)
- Formule *valide/ tautologii*, *contradicții/ nesatisfiabile*, *satisfiabile* dar *nevalide* (**D.**)
- *Semantica* formulelor din **LP**, conform **Teoremei de extensie** (**și** clasificării anterioare)
- Formule *tare echivalente* și formule *slab echivalente* (**D.**)

7-4 (186)

- *Consecință semantică*, a unei formule dintr-o mulțime de formule (în **LP**): $\mathcal{G} \vDash F$ (**D.**)
- *Formă normală conjunctivă* (**FNC**) și *formă normală disjunctivă* (**FND**) pentru formulele din **LP**
- Reprezentarea ca mulțimi de clauze și, în final, ca mulțime de mulțimi de literali a formulelor aflate în **FNC**
- **T. de extensie** (a fiecărei structuri, „corecte” pentru fiecare formulă, **S** : **A** $\to$ **B**, la **S’** : **LP** $\to$ **B**); legătura dintre F și $\rceil$F (**T.**); legătura dintre consecințe semantice, tautologii și contradicții (**T.**); **T. de echivalențe** (tari); **T. de substituție**; **T. de existență a formelor normale (TEFN)**

7-5 (187)

- Problema **SAT** pentru **LP** (**SAT-LP**): enunț, *idei* de rezolvare și complexitate (rezolvarea **SAT** cu ajutorul „tabelelor de adevăr” = **TA**)
- **A.** pentru aflarea simultană a (unei) **FNC și** a (unei) **FND**, algoritm dedus din demonstrația (prin inducție structurală) a **TEFN** (și care poate fi aplicat atât unei formule F $\in$ **LP**, cât și arborelui sintactic *arb*(F))

7-6 (188)

Cursul 3

- *Algoritmi corecți și compleți* pentru o problemă (concept general; de exemplu – pentru **SAT-LP**)
- Operații asupra formulelor (= mulțimilor de clauze) din **LP** (o clauză fiind reprezentată, la rândul ei, ca o mulțime de literali): **SUBS**, **PURE**, **UNIT**, **SPLIT**
- Algoritmul și procedura recursivă datorate lui **Davis-Putnam-Logemann-Loveland** (notația pe scurt, pe care o adoptăm acum: **DPLL**)
- Câteva *idei despre terminarea și corectitudinea* **DPLL** (intrări, ieșiri, structură, pași importanți …)

7-7 (189)

- Reprezentarea rezolvării **SAT-LP** (cu **DPLL**, pentru o formulă dată), prin *grafuri* (*parțiale* sau *complete*) „de execuție” (*arbori sintactici* pentru un compilator); noduri cu succesori datorați nedeterminismului, versus noduri „cu branșare” (create în urma aplicării unei operații **SPLIT**)
- Păstrarea „caracterului” unei formule *de-a lungul unui drum*; legătura cu „versiunea” semantică (ce utilizează **TA**)

7-8 (190)

Cursul 4

- *Rezolvent/ rezoluție* într-un pas/ *arbore de rezoluție* (într-un pas) (**D.**)
- *Demonstrație prin rezoluție* și *rezoluție în mai mulți pași*; demonstrația unei clauze C pornind cu (sau, bazată pe) mulțimea de clauze (sau formula) F (**D.**)
- *Mulțimea tuturor rezolvenților* unei mulțimi de clauze/ formule F $\in$ **LP** ($\mathrm{Res}^{(n)}(F)$, Res*(F)); proprietăți generale; *graful/ arborele* (*complet*) *de rezoluție* pentru F $\in$ **LP** (*reprezentat pe niveluri*)
- O altă definiție (<u>constructivă</u>; notație: Resc(F)) pentru Res*(F)

7-9 (191)

- Câteva *idei* legate de *rafinări/ strategii și restricții ale rezoluției* (*rezoluția unitară* …)
- **Lema rezoluției** (**T.**); legătura dintre Res*(F) și Resc(F) (**T.**); legătura dintre o demonstrație prin rezoluție în *k* pași a lui C bazată pe F și $Res^{(k)}$(F) (**T.**); finitudinea lui Res*(F) dacă F este o formulă din **LP** (și nu o mulțime oarecare, *numărabilă*, de clauze) (**T.**); **Teorema de compactitate pentru LP**: **<u>TC</u>**; **Teorema rezoluției pentru LP**: **<u>TR</u>**
- **A.** pentru rezolvarea **SAT-LP** sugerat de demonstrația teoremei rezoluției; *idei* de terminare și corectitudine; legătura cu ceilalți algoritmi cunoscuți (bazați pe **TA** = pe semantică; sau, **DPLL** = pe sintaxă)

7-10 (192)

Cursul 5

- **B**, $\mathbf{FB}^{(n)}$, **FB**, adică *funcții booleene*; funcții booleene *importante de* 1 *și* 2 *argumente*; *cardinalități*; reprezentarea (*tuturor*) funcțiilor booleene prin matrici (sau … **TA**; *idei*)
- *Algebre booleene* (**D.** generală este aici dată direct, adică pentru $\mathcal{B} = \langle \mathbf{B}, \bullet, +, \overline{\ } \rangle$ …)
- *Principiul dualității* într-o algebra booleeană; *legi derivate* (**Tabelul 1.1** – vezi cursul …); modalități de demonstrare a „adevărului” acestora
- Notația x^{α}; proprietăți ale notației și utilizarea lor
- *Termeni n*-ari peste o mulțime/ listă ordonată

$X = \{x_1, x_2, \ldots, x_n\}$; *maxtermeni*; *cardinalități*

7-11 (193)

- Dual: *factori* și *maxfactori*
- **D.**: *forme normale disjunctive perfecte* (**FNDP**) și *forme normale conjunctive perfecte* (**FNCP**) pentru funcțiile booleene; *cardinalități*
- <u>*Idei*</u> despre *alte* (una – mai sus) *modalități* de a reprezenta *orice* funcție booleeană prin câteva funcții fixate (*mulțimi complete*, *baze*, **SUP**, E)
- 4-uplul $\mathcal{B} = \langle \mathbf{B}, \bullet, +, \overline{} \rangle$ este o algebră booleeană (**T.**)
- **T. de descompunere a unei funcții booleene în sumă de termeni** și, dual, **în produs de factori**

7-12 (194)

- **T. de reprezentare** (**unică !**) **a unei funcții** booleene **printr-o FNDP** și, dual, printr-o **FNCP**
- **A.** de aflare (separată) a (unei) **FNCP/ FNC** și a (unei) **FNDP/ FND** pentru funcțiile booleene
- Legătura dintre algoritmii amintiți mai sus și algoritmii, cunoscuți, de aflare a **FNC** și/ sau **FND** pentru formulele din **LP**, reprezentate (tot) ca text/ expresii; mai general, legătura „bidirecțională” dintre clasa formulelor **LP** reprezentate peste (exact) *n* atomi și $\mathbf{FB}^{(n)}$
- **D.**: *diagrame de decizie binare* (**BDD**) peste mulțimea/ lista X = $\{x_1, x_2, \ldots, x_n\}$; exemple
- Funcția calculată de o ~~(**O**)~~**BDD** și ~~(**O**)~~**BDD**-urile „atașate” unei funcții booleene date „tabelar” (**TA**)

7-13 (195)

- **BDD**-uri și **subBDD**-uri, reprezentări grafice
- **A.** de „trecere” a unei funcții și/ sau formule dintr-o reprezentare în alta: text (sau „reprezentant” = formă normală = **FN**), matrice = tabel = **TA**, graf/ ~~(**O**)~~**BDD**
- *Procedeele de optimizare/ reducere* (**C1**, **C2**, **C3**) aplicabile unei **BDD** și obținerea unei **BDD** *maximal reduse*
- Rezultat important: aplicarea procedeelor amintite anterior unei/ unor (succesiv) **BDD** oarecare *nu modifică funcția booleeană calculată de acele diagrame*
- Se poate astfel deduce un **A.** de găsire a unei **BDD** maximal reduse „echivalente” (calculează aceeași funcție) cu o **BDD** „inițială (*cât timp este posibil, aplică un procedeu* …)

Cursul 6

- Noțiunea de *teorie logică*, **TE** (**D.**); $\mathcal{val}$ (**LP**)
- Noțiunea de *sistem deductiv/ de demonstrație*, **SD**; **SD0**/ *deducția naturală* (**D.**)
- Reprezentarea sub diverse forme a *regulilor de inferență*
- *Demonstrație* și *teoreme* într-un sistem deductiv (**D.**); $\mathcal{Th}$ (**SD**)
- Sisteme deductive *echivalente* (**D.**)
- *Reguli de inferență derivate* într-un sistem deductiv (**D.**)
- <u>Deducția naturală</u>: *secvențe*, *secvențe valide* (sintactic); *box*-uri; reguli/ *scheme* și reguli derivate

7-15 (197)

- *Presupuneri*; „intrarea”, „ieșirea” și formarea box-urilor (câteva, *alte*, *reguli interne*)
- *Construirea* (completarea spațiilor goale) unei/ (dintr-o) demonstrații(e) (uitându-ne *top-down* **și** *bottom-up*); diverse modalități de reprezentare/ scriere a demonstrațiilor și box-urilor
- Interpretarea *declarativă* și *imperativă/ procedurală* a regulilor; <u>și</u> a demonstrațiilor folosind (și) box-uri
- Notația „$\dashv\vdash$”; câteva idei despre legătura sintaxă-semantică („transferul $\rightarrow$ în dreapta lui $\vdash$”) în **SD0**; și, în general, despre **teoreme de corectitudine și completitudine** ($\mathcal{val}$(**LP**) = $\mathcal{Th}$(**SD**))

7-16 (198)

- *Generalități*, iar *reveniri* asupra **A. sintactici și semantici** (**corecți și compleți**) pentru rezolvarea **SAT-LP**

FINAL Curs 7

SFÂRȘIT CURS