UNIVERSITATEA TEHNICĂ A MOLDOVEI

Victor Beşliu

CICLU DE PRELEGERI

la disciplina "Matematica discretă"

Chişinău 2002

În lucrare sunt prezentate compartimentele principale ale matematicii discrete, utilizate în informatică: mulţimi, relaţii şi algebre, inclusiv algebra relaţională cu aplicaţii în bazele de date, elemente de logică matematică - logica booleană, logica fuzzy, logica de ordinul unu, grafuri şi algoritmi pe grafuri, modele algoritmice şi maşinile Turing. Este propusă o gamă variată de exemple, exerciţii şi probleme.

Pentru studenţii specialităţilor “Tehnologii Informaţionale”, „Automatizări”, „Calculatoare”, „Microelectronică şi dispozitive cu semiconductori”, grupele de masterat şi doctoranzi.

Aprobat prin Hotărârea Consiliului Metodic al Facultăţii „Calculatoare, Informatică şi Microelectronică” a Universităţii Tehnice a Moldovei protocol Nr. 1 din „ 15 ” ianuarie 2002

Recenzenţi: Doctor habilitat în ştiinţe tehnice Anatol Popescu

Doctor în ştiinţe fizico-matematice Vasile Moraru

Redactor: Doctor în ştiinţe fizico-matematice Alexandru Moloşniuc

CUPRINSUL

PREFAŢĂ

Lucrarea prezintă o introducere în metodele matematice utilizate în informatică şi este o încercare de a oferi, în primul rând studenţilor, un material util, dar şi cu gândul la nespecialist şi la viitorul specialist. Sunt tratate doar câteva din compartimentele matematicii discrete, care studiază organizarea şi ordonarea structurilor, legate de mulţimi finite: corespondenţe şi relaţii, algebre şi logici, grafuri şi modele de algoritmi. Lipsesc unele capitole, cum ar fi combinatorica şi probabiltăţi discrete, automate şi limbaje formale, etc, care sunt studiate conform planului de studiu în cadrul altor discipline.

Am încercat să combin strictețea demonstraţiilor cu exemple relaxante, tinzând să prezint în mod unitar unele concepte care, de obicei, sunt obiectul unor lucrări separate de electronică, calculatoare sau chiar inteligenţă artificială.

După o succintă introducere, în capitolul 2 (de început), sunt tratate noţiuni fundamentale, cum ar fi mulţimi, corespondenţe, funcţii, operaţii, relaţii, algebre, modele, sisteme algebrice. Am acordat o atenţie deosebită algebrei relaţionale, care este fundamentul bazelor de date relaţionale.

Capitolul 3 este consacrat prezentării bazelor logicii matematice, începând cu logica booleană şi până la logica fuzzy. Noţiunile de funcţii ale algebrei logicii, forme canonice, circuite logice sau minimizarea funcţiilor booleene, extrem de importante pentru un viitor calculatorist, au migrat din domeniul tehnic în cel fundamental matematic. Acelaşi lucru s-a produs şi cu logica enunţurilor, logica de ordinul unu sau logica fuzzy - fundamentul matematic al dispozitivelor tehnice, bazate pe inteligenţa artificială.

În capitolul 4 sunt expuse bazele teoriei grafurilor – de la noţiuni generale şi metode de reprezentare pe calculator până la algoritmi clasici, propuşi de matematicieni cu mult înainte de apariţia informaticii, dar fără de care tehnologiile informaţiei nu pot exista.

Încercarea de a preciza noţiunea de algoritm este făcută în ultimul capitol. Dintre toate modelele algoritmice cunoscute sunt prezentate doar maşinile Turing în speranţa că acestea vor trezi cititorului un interes cât de mic pentru a face cunoştinţă cu celelalte modele.

Ciclul de prelegeri a fost testat pe mai multe generaţii de studenţi, pe care îi mulţumesc şi pe această cale, corespunde programei analitice a cursului „Matematica discretă" pentru facultatea „Calculatoare, Informatică şi Microelectronică" de la Universitatea Tehnică a Moldovei, dar, sper, poate fi util tuturor celor interesaţi de domeniu.

1. INTRODUCERE

Prezenta lucrare este dedicată metodelor matematicii discrete, aplicate în informatică şi are ca destinatari pe studenţii şi specialiştii în ştiinţa calculatoarelor şi tehnologia informaţiei, la fel ca şi pe toţi doritorii să facă cunoştinţă cu acest domeniu matematic.

Matematica tradiţională, inclusiv matematica aplicată, care formează fundamentul ştiinţelor tehnice clasice, s-a conformat pe parcursul anilor cerinţelor acestora. Există monografii şi manuale, lucrări metodice şi îndrumare consacrate problemelor legate de ştiinţele tehnice clasice şi metodele matematice respective. Nimeni nu mai consideră necesar să explice noţiunile legate de calculul diferenţial sau integral în cadrul electrotehnicii sau termodinamicii. Nu este aceeaşi situaţie şi în cazul disciplinelor din ştiinţa calculatoarelor, informatica teoretică şi aplicată, inclusiv, sisteme expert sau inteligenţa artificială. Foarte mulţi autori sunt convinşi de necesitatea de a începe de la "zero", explicând ce este o funcţie logică sau un tabel de adevăr, un circuit sau un ciclu într-un graf. Lucrarea de faţă este o încercare de a permite viitorilor autori să se concentreze totalmente problemelor tehnice tratate.

Un alt obiectiv este legat de faptul că majoritatea inginerilor consideră matematica un fel de dicţionar mare. Este suficient să poţi doar localiza informaţia necesară, deschizându-l la pagina respectivă. Inginerii sunt obligaţi să "iubească" formulele, dar nu şi teoremele (ca să nu vorbim de demonstrarea lor). Însă domeniile de vârf ale ştiinţei contemporane cer cu insistenţă să fie schimbată această mentalitate, or problemele tehnico-ştiinţifice principale sunt legate nu atât de lucrul cu modele cunoscute, cât de elaborarea unor modele noi. Pentru aceasta este necesar ca matematica să fie nu numai o metodă de calcul, ci şi una de gândire.

2. SISTEME ALGEBRICE

2.1. Mulţimi şi submulţimi

2.1.1. Noţiuni generale

Teoria mulţimilor studiază conceptele de mulţime şi de infinit şi legile de operare cu mulţimile. Noţiunea de ***mulţime*** este una din noţiunile primare ale matematicii şi, ca şi majoritatea noţiunilor primare nu are o definiţie unanim acceptată. George Cantor (1845-1918) a definit astfel această noţiune: "*înţeleg prin mulţime, în general, tot ceea ce este mult, dar care poate fi conceput ca o entitate, adică orice colecţie de anumite obiecte putând fi închegate într-un*

întreg cu ajutorul unei legi oarecare". Obiectele care formează mulţimile se numesc ***elemente***. De obicei, elementele se notează cu litere mici, iar mulţimile cu litere mari cu sau fără indici. Elementele se iau în acolade şi se separă prin virgule.

Vom nota apartenenţa elementului a mulţimii C în felul următor: $a \in C$, iar dacă g nu aparţine mulţimii C se va scrie $g \notin C$. Mulţimile pot fi finite sau infinite.

Expresia $x \in S$ semnifică, deci, faptul că elementul x este un element al mulţimii S. Dacă $x_1, x_2,..., x_n$ sunt toate elemente ale mulţimii S, atunci poate fi scris $S=\{x_1, x_2,..., x_n\}$. Fiecare x trebuie să fie distinct; nu putem repeta scrierea unui element într-o mulţime. Ordinea în care elementele sunt scrise în cadrul mulţimii este arbitrară - mulţimile nu posedă o structură internă.

Numărul elementelor unei mulţimi finite se numeşte cardinalul acestei mulţimi.

Cardinalul mulţimii A se notează prin $|A|$. Despre cardinalul mulţimilor infinite se va discuta în paragrafele următoare. Un loc aparte îi este rezervat **mulţimii de cardinal zero** (fără elemente, $\{\ \}$). Această mulţime se numeşte **mulţime vidă** şi se notează prin $\varnothing$.

Exemplul 2.1. Fie $A = \{1, 3, 6\}$; altfel spus, A este mulţimea care are ca elemente valorile întregi 1, 3 şi 6, alte elemente nu există. Putem scrie $1 \in A$, $3 \in A$ şi $6 \in A$. Din contra, afirmaţia $2 \in A$ este falsă ca şi oricare alta, care afirmă că altceva poate fi element din A.
Mulţimile pot avea ca elemente alte mulţimi. De exemplu, fie $B = \{\{1, 2, 3\}, 3, \varnothing\}$. B are aici trei elemente. Primul element este mulţimea $\{1, 2, 3\}$, al doilea este numărul întreg 3, iar al treilea este mulţimea vidă. Următoarele afirmaţii sunt juste: $\{1, 2, 3\} \in B$, $3 \in B$, şi $\varnothing \in B$. Afirmaţia $1 \in B$ est falsă. Adică, din faptul că 1 este element al unuia dintre elementele lui B nu rezultă că 1 este şi element al lui B.◄

Mulţimea A se va numi **submulţime** (sau **parte**) a mulţimii B (se va nota $\boldsymbol{A \subseteq B}$, simbolul $\subseteq$ se numeşte **simbol de incluziune**), dacă fiecare element al mulţimii A este şi element al mulţimii B. Se va mai spune că B acoperă A. Două mulţimi A şi B se vor considera egale dacă conţin aceleaşi elemente, altfel, dacă $A \subseteq B$ şi $B \subseteq A$, atunci $A = B$. În cazul în care $A \subseteq B$ şi $A \neq B$ se va scrie $A \subset B$, iar A se va numi submulţime proprie sau strictă a lui B.

Exemplul 2.2. Mulţimea A, formată din numerele întregi, care se împart fără rest la 6, este o submulţime B, formată din numerele întregi pare. ◄

N.B. Mulţimea vidă este submulţime a oricărei mulţimi.

Exemplul 2.3. Toate relaţiile de mai jos sunt adevărate:

1. $\{1, 2, 3\} \subseteq \{1, 2, 3, 5\}$
2. $\{1, 2, 3\} \subset \{1, 2, 3, 5\}$
3. $\{1, 2, 3\} \subseteq \{1, 2, 3\}$

Remarcăm, că o mulţime este întotdeauna şi submulţime a sa (p.3), dar niciodată submulţime proprie, adică afirmaţia $\{1, 2, 3\} \subset \{1, 2, 3\}$ este falsă. ◄

Pentru fiecare mulţime A există o mulţime $\boldsymbol{\rho}(A)$ care conţine toate părţile lui A (inclusiv mulţimea vidă şi A). Această mulţime $\boldsymbol{\rho}(A)$ se va numi **booleanul** lui A.

Exemplul 2.4. Pentru mulţimea $F = \{a, b, c\}$ vom avea $\boldsymbol{\rho}(F) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$. Adică, $\boldsymbol{\rho}(F)$ este o mulţime cu opt elemente, fiecare element fiind la rândul lui o mulţime. Un alt exemplu, $\boldsymbol{\rho}(\varnothing) = \{\varnothing\}$ deoarece $\varnothing \subseteq S$. Notăm că $\{\varnothing\}$, mulţimea care conţine mulţimea vidă nu este aceeaşi ca şi mulţimea vidă. Prima are un element - $\varnothing$, iar mulţimea vidă nu are nici unul. ◄

Mulţimea tuturor obiectelor posibile în cadrul unei cercetări concrete vom numi **mulţime universală** sau **universum** (fără pretenţii de stricteţe) şi se notează prin E sau U.

2.1.2. Metode de definire a mulţimilor

Mulţimile pot fi definite prin simpla enumerare a elementelor lor, dar această metodă este valabilă doar pentru mulţimile finite cu un număr mic de elemente.

O altă metodă propune să se pornească de la o mulţime S şi o proprietate P a elementelor, definind o mulţime ca toate elementele lui S care verifică proprietatea P. Notaţia acestei operaţii, numită *abstracţie* este

$\{x \mid x \in S \text{ şi } P(x)\}$

sau *toate elementele x din S, care verifică proprietatea P.* Variabila x din ultima expresie este locală, adică putem scrie cu acelaşi succes $\{y \mid y \in S \text{ şi } P(y)\}$ pentru a descrie aceeaşi mulţime.

Exemplul 2.5. Fie A, mulţimea $\{1, 3, 6\}$ din exemplul 2.1 şi $P(x)$ proprietatea "*x este impar*". Atunci, $\{x \mid x \in A \text{ şi } x \text{ este impar}\}$ este o altă modalitate de a defini mulţimea $\{1, 3\}$. Altfel, noi acceptăm elementele 1 şi 3 din A pentru că ele sunt impare, dar refuzăm elementul 6, pentru că el nu este impar.

Considerăm mulţimea $B = \{\{1, 2, 3\}, 3, \varnothing\}$ din exemplul 2.1. Atunci, $\{A \mid A \in B \text{ şi } A \text{ este o mulţime }\}$ defineşte mulţimea $\{\{1, 2, 3\}, \varnothing\}$.

Alt exemplu: mulţimea numerelor întregi nenegative, sau naturale, este adesea notată prin $\boldsymbol{N}$. Fie $P(x)$ proprietatea *"x* este număr primar" (adică $x > 1$ şi nu are alţi divizori decât 1 şi el însuşi). Mulţimea numerelor

primare va fi notată în acest caz $\{x \mid x \in N$ şi $P(x)\}$. Această expresie defineşte mulţimea infinită $\{2, 3, 5, 7, 11, ...\}$.◄

Este tentant să presupunem, că mulţimile sunt *finite* sau că există un întreg n şi mulţimea considerată are exact n elemente. De exemplu, mulţimea {1, 3, 6} are trei elemente. Însă, în foarte multe cazuri mulţimile sunt infinite, adică nu există un întreg care ar limita numărul elementelor mulţimii. Iată câteva exemple:

- $\boldsymbol{N}$ - mulţimea numerelor naturale;
- $\boldsymbol{Z}$ - mulţimea numerelor întregi;
- $\boldsymbol{R}$ – mulţimea numerelor reale;
- $\boldsymbol{C}$ - mulţimea numerelor complexe.

Plecând de la aceste mulţimi pot fi create prin abstracţie alte mulţimi infinite.

Exemplul 2.6. Mulţimea $\{x \mid x \in Z$ şi $x < 3\}$ este mulţimea tuturor numerelor întregi negative la care se adaugă 0, 1 şi 2. Mulţimea $\{x \mid x \in Z$ şi $\sqrt{x} \in Z\}$ reprezintă mulţimea numerelor întregi care sunt pătrate perfecte, {0, 1, 4, 9, 16, ...}.◄

A treia metodă constă în definirea mulţimilor prin intermediul unei **proceduri generatoare**, care va descrie modalitatea de obţinere a elementelor mulţimii din elemente iniţiale şi/sau elemente, care au fost obţinute anterior. Se vor considera elemente ale mulţimii toate obiectele care pot fi construite cu ajutorul acestei proceduri. De pildă, mulţimea $M = \{1, 2, 3, 5, 8, 13,...\}$ poate fi definită cu ajutorul următoarei proceduri:

$$1)\ x_1 = 1,\ x_2 = 2;$$
$$2)\ x_{i+2} = x_{i+1} + x_i,\ i = 1, 2, 3,....$$

O mulţime poate fi definită cu ajutorul **funcţiei caracteristice**, care are domeniul de definiţie mulţimea universală U, iar domeniul de valori mulţimea $\{0, 1\}$:

şi $$\mu(x) = \begin{cases} 1, & \text{dacă } x \in U \text{ aparţine lui } M \\ 0, & \text{în caz contrar.} \end{cases}$$

2.2. Mulţimi vagi

Definirea unei mulţimi cu ajutorul funcţiei caracteristice presupune că elementul $x \in U$ aparţine sau nu aparţine mulţimii M, o a treia posibilitate este exclusă. Însă în realitate, pentru o mare diversitate de obiecte nu există criterii exacte de apartenenţă: funcţia caracteristică doar pentru unele elemente este zero (se cunoaşte precis neapartenenţa elementului la mulţimea dată) sau unu (elementul aparţine sigur mulţimii considerate). Pentru restul elementelor funcţia

$\mu(x)$ ar trebui să ia valori între *0* şi *1*. Aceste elemente formează obiectul teoriei mulţimilor **vagi (fuzzy)** [1]**.**

Exemplul 2.7. Mulţimea $A = \{x \mid x>>1\}$ este definită ca setul de numere mult mai mari ca unu. În sensul obişnuit *A* nu poate fi considerată o mulţime. Se poate spune precis, că numerele mai mici decât *1* nu aparţin lui *A*. Numerele mai mari ca *1* pot fi considerate elemente din *A* cu un anumit grad de subiectivism: cu cât numărul considerat este mai mare cu atât pare mai corect să admitem, că acesta aparţine lui *A*. Ultima afirmaţie poate fi descrisă, de exemplu, cu ajutorul funcţiei caracteristice de mai jos

$$\mu_A(x) = \begin{cases} 0, & \text{dacă } x \le 1 \\ \dfrac{1}{1+\dfrac{1}{x-1}}, & \text{dacă } x > 1. \end{cases} \blacktriangleleft$$

O mulţime fuzzy *A* se va defini utilizând aplicaţia mulţimii universale *U* în segmentul *[0, 1]*, adică $\mu_A(x):U\to[0,\ 1]$, care determină gradul de apartenenţă a fiecărui $x\in U$ mulţimii fuzzy *A*. În acest caz funcţia $\mu_A(x)$ se numeşte **funcţie de apartenenţă**. Altfel spus, ***o mulţime fuzzy A poate fi definită ca o mulţime de perechi*** $(x, \mu_A(x))$***, în care*** $x\in U$***, iar*** $\mu_A(x)$ ***este funcţia de apartenenţă.***

Exemplul 2.8. Mulţimea fuzzy a numerelor naturale "*nu prea mari*" poate fi definită ca mulţimea *A* a perechilor *{(1,1), (2,1), (3,1), (4,0.9), (5,0.8), (6,0.7), (7,0.6), (8,0.5), (9,0.3), (10,0.1), (11,0),(12,0),...}.*
Mulţimea universală în acest caz este mulţimea numerelor naturale, iar funcţia de apartenenţă este definită, evident, subiectiv. ◄

Mulţimea fuzzy vidă are funcţia de apartenenţă egală cu *0*: $\forall x\in U\ \mu_{\varnothing}(x) = 0$. Mulţimea universală are funcţia de apartenenţă identic *1:* $\forall x\in U\ \mu_U(x) = 1$.

Noţiunea de egalitate a două mulţimi fuzzy se introduce de asemenea cu ajutorul funcţiei de apartenenţă: **două mulţimi fuzzy *A* şi *B* se numesc egale dacă** $\forall x\in U$ **avem** $\mu_A(x) = \mu_B(x)$.

La fel şi relaţia de incluziune: vom spune, că **mulţimea fuzzy *A* este submulţime a mulţimii fuzzy *B* (este inclusă în *B*,** $A\subseteq B$**), dacă** $\forall x\in U$ **are loc inegalitatea** $\mu_A(x)\le\mu_B(x)$**.** Mulţimea fuzzy vidă este submulţime a oricărei mulţimi.

2.3. Operaţii cu mulţimi

În acest paragraf vom defini trei operaţii cu mulţimi: **reuniunea, intersecţia** şi **complementara**, operaţii de bază, şi două operaţii suplimentare: **diferenţa şi diferenţa simetrică**.

Reuniunea a două mulţimi A şi B se va numi mulţimea C ($C = A \cup B$), care conţine toate elementele lui A şi toate elementele mulţimii B, alte elemente nu are.

De exemplu, pentru $A = \{a, b, c, d\}$ şi $B = \{c, d, e, f\}$ reuniunea lor va fi mulţimea $C = \{a, b, c, d, e, f\}$. Cu alte cuvinte, $C = A \cup B = \{x \mid x \in A$ sau $x \in B\}$.

Reuniunea este o operaţie

a) *comutativă:* $A \cup B = B \cup A$;
b) *asociativă:* $(A \cup B) \cup C = A \cup (B \cup C) = A \cup B \cup C$
c) *Ø este elementul său neutru:* $A \cup \emptyset = A$
d) *idempotentă:* $A \cup A = A$

Remarcăm, că $A \subset B$ atunci şi numai atunci, când $A \cup B = B$.

Intersecţia a două mulţimi A şi B se va numi mulţimea $C = A \cap B$, care conţine toate elementele comune ale acestor două mulţimi: $C = \{x \mid x \in A$ şi $x \in B\}$. Pentru aceleaşi A şi B din exemplul precedent $A \cap B = \{c, d\}$.

Intersecţia este o operaţie

e) *comutativă:* $A \cap B = B \cap A$;
f) *asociativă:* $(A \cap B) \cap C = A \cap (B \cap C) = A \cap B \cap C$
g) *Ø este elementul său absorbant:* $A \cap \emptyset = \emptyset$
h) *idempotentă:* $A \cap A = A$

Remarcăm, că $A \subset B$ atunci şi numai atunci, când $A \cap B = A$.

Operaţiile de reuniune şi intersecţie pot fi definite pentru o familie de mulţimi A_i, $i = 1,...,n$:

$$\cup A_i = \{x \mid x \in A_1 \text{ sau } x \in A_2 \text{ sau } \ldots x \in A_n\},$$
$$\cap A_i = \{x \mid x \in A_1 \text{ şi } x \in A_2 \text{ şi } \ldots x \in A_n\}.$$

Diferenţa a două mulţimi A şi B, notată $A - B$ (sau $A \setminus B$) se numeşte mulţimea $C = \{x \mid x \in A$ şi $x \notin B\}$. Pentru aceleaşi A şi B vom avea $A - B = \{a, b\}$, iar $B - A = \{e, f\}$. Observăm necomutativitatea acestei operaţii.

Diferenţa simetrică se defineşte ca $C = A \Delta B = \{x \mid x \in A$ şi $x \notin B$ sau $x \notin A$ şi $x \in B\}$. Pentru exemplul precedent vom avea $A \Delta B = B \Delta A = \{a, b, e, f\}$.

Exemplul 2.9. Fie S, mulţimea $\{1, 2, 3\}$ şi T mulţimea $\{3, 4, 5\}$. Atunci $S \cup T = \{1, 2, 3, 4, 5\}$, $S \cap T = \{3\}$, şi $S - T = \{1,2\}$. Adică $S \cup T$ conţine toate elementele care apar fie în S fie în T. Chiar dacă 3 apare o dată în S şi încă o dată în T, el va fi regăsit o singură dată în $S \cup T$, deoarece elementele nu pot să se repete, analogic pentru mulţimea $S \cap T$. Mulţimea $S - T$ conţine elementele

1 şi 2, deoarece ele sunt în *S*, dar nu se află în *T*. Elementul 3 nu este în mulţimea *S* - *T* deoarece, deşi este în *S*, el aparţine şi lui *T*. ◄

Complementara mulţimii *A* în *U* ($A \subseteq U$) notată cu $C_U A$ se va numi mulţimea care conţine toate elementele mulţimii *U* ce nu aparţin mulţimii *A:* $C_U A = \{x \mid x \in U \text{ şi } x \notin A\}$. Atunci când este evident care este mulţimea *U*, indicele poate fi omis. Complementara lui *A* se va mai nota prin $\bar{A}$ (*A* barat).

De exemplu, dacă $U = \{a, b, c, d, e, f, g\}$, atunci pentru aceleaşi mulţimi *A* şi *B* vom avea $C_U A = \bar{A}_U = \{e, f, g\}$, iar $C_U B = \{a, b, g\}$.

Numim **partiţie** a mulţimii *A* orice set de părţi $X_1, X_2, X_3, ..., X_n$ ale lui *A*, care verifică condiţiile:

$$\begin{aligned} &1.\ X_i \neq \varnothing, && i = 1, 2, ..., n; \\ &2.\ X_i \cap X_j, = \varnothing, && i \neq j; \\ &3.\ \cup X_i = A, && i = 1, 2, ..., n \end{aligned} \qquad (2.1)$$

Utilizând funcţia de apartenenţă, operaţiile de determinare a complementarei unei mulţimi fuzzy *A*, a reuniunii şi a intersecţiei a două mulţimi fuzzy *A* şi *B* sunt definite astfel:

Complementara unei mulţimi fuzzy *A* se numeşte mulţimea fuzzy $\bar{A}$ cu funcţia de apartenenţă

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x); \qquad (2.2)$$

Reuniunea a două mulţimi fuzzy *A* şi *B* numim mulţimea fuzzy $A \cup B$ cu funcţia de apartenenţă

$$\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)]; \qquad (2.3)$$

Intersecţia a două mulţimi fuzzy *A* şi *B* numim mulţimea fuzzy $A \cap B$ cu funcţia de apartenenţa

$$\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)]; \qquad (2.4)$$

Există şi alte definiţii ale ultimelor două operaţii.

2.4. Demonstrarea echivalenţei cu ajutorul incluziunilor

Două mulţimi *S* şi *T* sunt egale dacă şi numai dacă $S \subseteq T$ şi $T \subseteq S$; adică fiecare este simultan o submulţime a celeilalte. Aceasta este analogic regulii aritmetice care afirmă că $a = b$ atunci şi numai atunci când simultan $a \leq b$ şi $b \leq a$ sunt adevărate. Putem demonstra echivalenţa a două expresii *E* şi *F* arătând că fiecare este inclusă în cealaltă. Altfel

1. se va lua un element arbitrar *x* din *E* şi se va demonstra că el aparţine de asemenea şi lui *F*,

2. se va lua un element arbitrar x din F şi se va demonstra că el aparţine de asemenea şi lui E.

Exemplul 2.10. Să demonstrăm asociativitatea reuniunii şi diferenţei, $(S - (T \cup R)) \equiv ((S - T) - R)$. Începem cu presupunerea, că x aparţine expresiei din stângă. Secvenţa etapelor este arătată în tab.2.1.

Tabelul 2.1. Prima jumătate a demonstraţiei.

	Etapa	**Justificarea**
1)	x este din $S - (T \cup R)$	dat
2)	x este din S	definiţia operaţiei "-" şi (1)
3)	x nu este în $T \cup R$	definiţia operaţiei "-" şi (1)
4)	x nu este în T	definiţia operaţiei "$\cup$" şi (1) (3)
5)	x nu aparţine lui R	definiţia operaţiei "$\cup$" şi (3)
6)	x aparţine lui $S - T$	definiţia operaţiei "-" cu (2) şi (4)
7)	x este în $(S - T) - R$	definiţia operaţiei "-" cu (6) şi (5)

Am ajuns la concluzia, că x aparţine şi părţii drepte. Concluzie: deoarece x a fost luat arbitrar, partea stângă este submulţime a părţii drepte. Dar asta nu-i totul. Mai trebuie să demonstrăm, că şi partea dreaptă este submulţime a părţii stânga, adică dacă un x arbitrar este din $(S-T)–R$, atunci el se conţine şi în $S - (T \cup R)$.

Tabelul 2.2. A doua jumătate a demonstraţiei.

	Etapa	**Justificarea**
1)	x este din $S - (T - R)$	dat
2)	x este din $S-T$	definiţia operaţiei "-" şi (1)
3)	x nu este în R	definiţia operaţiei "-" şi (1)
4)	x este în S	definiţia operaţiei "-" şi (2)
5)	x nu aparţine lui T	definiţia operaţiei "-" şi (2)
6)	x nu aparţine lui $R \cup T$	definiţia operaţiei "$\cup$" cu (3) şi (5)
7)	x este în $S – (T \cup R)$	definiţia operaţiei "-" cu (6) şi (4)

Am arătat acest lucru în tabelul 2.2. ◄

Exemplul 2.11. Demonstrăm, că dacă $S \subseteq T$, atunci $S \cup T \equiv T$. Admitem că $x \in S \cup T$. Conform definiţiei reuniunii

1. x aparţine sau lui S,
2. sau lui T.

În primul caz, deoarece noi am presupus $S \subseteq T$, concluzionăm că $x \in T$. În cazul (2), x este imediat în T. Deci, în ambele cazuri x este element de T, şi am terminat prima jumătate a demonstraţiei.

Fie acum $x \in T$. În acest caz $x \in S \cup T$ conform definiţiei de reuniune. Deci, $T \subseteq (S \cup T)$, ceea ce constituie partea a doua a demonstraţiei. Concluzia: $(S \cup T) \equiv T$. ◄

2.5. Vectori şi produs cartezian

2.5.1. Definiţii

Un set ordonat de elemente se va numi **vector** sau **cortej**. Aceasta nu este definiţia noţiunii de vector, care la fel ca şi noţiunea de mulţime nu se defineşte. Elementele, care formează vectorul se numesc coordonate sau componente şi se numerotează de la stânga spre dreapta. În acest sens se va înţelege noţiunea **de set ordonat**. Vectorii se scriu în paranteze rotunde, componentele se vor separa, la necesitate, prin virgulă. Numărul de componente se numeşte **lungimea** sau **dimensiunea** vectorului.

De exemplu, $v = (2,3,0,3)$ este un vector de lungime *4* şi diferă de $b = (3,2,0,3)$. Observăm, că este admisă coincidenţa coordonatelor. Vectorul de lungime 2 se mai numeşte pereche sau cuplu, iar de lungime n - ***n-uplu***.

Doi vectori sunt **egali** dacă au aceeaşi lungime, iar componentele respective coincid.

Se numeşte **produs cartezian** a două mulţimi A şi B (se notează AxB) mulţimea **tuturor** perechilor (a,b) pentru care $a \in A$, $b \in B$. Pentru $A = B$ vom avea Ax$A = A^2$. Prin analogie, A_1xA_2x...xA_n se va numi mulţimea tuturor vectorilor de lungime n $(a_1,a_2,...,a_n)$ pentru care $a_1 \in A_1$, $a_2 \in A_2$,..., $a_n \in A_n$. Dacă $A_1=A_2=...=A_n=A$ produsul cartezian A_1xA_2x...xA_n se va nota A^n.

Exemplul 2.12. Mulţimea R x $R = R^2$ este mulţimea tuturor perechilor (a,b) pentru care a, $b \in R$ şi reprezintă coordonatele punctelor unui plan. Această reprezentare a punctului unui plan a fost propusă de către matematicianul şi filosoful francez Réné Descartes (1596-1650) din care cauză produsul a două mulţimi îi poartă numele.
Fie A o mulţime finită de simboluri (litere, cifre, semne de operaţii şi ortografice, etc.), care, de obicei, se numeşte alfabet. Elementele mulţimii A^n se numesc cuvinte de lungime n în alfabetul A. Mulţimea $A = A^1 \cup A^2 \cup A^3 \cup ...$ defineşte toate cuvintele alfabetului A. ◄

Exemplul 2.13. Dacă $A = \{c, d\}$ şi $B = N_3$, elementele produsului AxB sunt 6 cupluri $(c, 1)$, $(c, 2)$, $(c, 3)$, $(d, 1)$, $(d, 2)$, $(d, 3)$, iar elementele mulţimii BxA sunt: $(1,c)$, $(2, c)$, $(3, c)$, $(1, d)$, $(2, d)$, $(3, d)$. ◄

Ultimul exemplu arată cum poate fi formată lista elementelor produsului AxB, când A şi B sunt definite prin enumerarea elementelor. Metoda este generală: produsul a două mulţimi, definite prin enumerare, poate totdeauna fi evidenţiat prin enumerare. O altă metodă de inventariere a cuplelor unui produs AxB sunt **diagramele carteziene**. O diagramă cartezianã este un dreptunghi împărţit în celule (casete), fiecare casetă corespunzând unui cuplu (fig. 2.1 – 2.4). Fiecare

linie a dreptunghiului corespunde unui element al mulţimii *A*, vom avea aici toate cuplurile, care au prima componentă din *A*. Fiecare coloană corespunde unui element din *B* şi în acest mod vom depista în acest dreptunghi toate cuplurile, care au acest element în calitate de componenta a doua.

Exemplul 2.14. Pentru mulţimile din exemplu 2.13, figura 2.1 reprezintă diagrama carteziană a produsului *AxB*, iar figura 2.2 pe cea a produsului *BxA*. Ordinea de aranjare a elementelor în *A* şi *B* determină poziţia cuplurilor în diagramă; dacă va fi modificată această ordine diagrama nu va mai fi aceeaşi.

(*c*, *1*)	(*c*, *2*)	(*c*, *3*)
(*d*, *1*)	(*d*, *2*)	(*d*, *3*)

Fig.2.1. Diagrama produsului *AxB*

(*1*, *c*)	(*1*, *d*)
(*1*, *c*)	(*1*, *d*)
(*1*, *c*)	(*1*, *d*)

Fig.2.2. Diagrama produsului *BxA*

	1	*2*	*3*
c			
d			

Fig.2.3. Diagrama produsului *AxB*

	c	*a*
1		
2		
3		

Fig.2.4. Diagrama produsului *BxA*

Adesea elementele mulţimilor *A* şi *B* sunt indicate în partea stângă şi, respectiv, superioară a dreptunghiului, elementele lui *A* corespunzând liniilor, iar cele ale lui *B* – coloanelor diagramei (fig.2.3 şi 2.4). ◄

La concret, a construi un *n*-uplu înseamnă să alegem un prim obiect (componentă) din prima mulţime, un al doilea obiect dintre elementele mulţimii a doua şi tot aşa până la componenta cu numărul *n* din mulţimea cu acelaşi număr. Putem întâlni foarte multe *n*-upluri în viaţa de toate zilele.

Exemplul 2.15. Figura de mai jos reprezintă lista bucatelor, propuse într-o oarecare zi la cantina studenţească. Vom considera, că a compune un meniu înseamnă să se aleagă un aperitiv, un fel unu, un fel doi şi un desert. Dacă vom nota prin *A*

Lista bucatelor la cantină pentru "___"___2002

Aperitive
1. Salată de roşii
2. Salată de varză
3. Salată de castraveţi
4. Julien de ciuperci

Felul doi
1. Cotlet
2. Antrecot
3. Ciulama de puişor
4. Tocăniţă cu mămăliguţă

Felul unu
1. Ciorbă de burtă
2. Ciorbă cu perişoare
3. Zeamă de puişor

Desert
1. Suc de mere
2. Compot de ananas
3. Suc mango fresh

mulţimea aperitivelor, prin F_1, F_2 – mulţimile felurilor unu şi doi, respectiv, iar prin D – deserturile, fiecare meniu, de exemplu: (*salată de roşii, ciorbă de burtă, tocăniţă cu mămăliguţă, suc mango fresh)*, este un element al produsului $A \times F_1 \times F_2 \times D$. ◄

2.5.2. Cardinalul produsului cartezian

Teorema 2.1. Dacă $A_1, A_2, ..., A_n$ sunt mulţimi finite cu $|A_1|=m_1$, $|A_2|=m_2, ...,$ $|A_n|=m_n$, atunci $|A_1 \times A_2 \times ... \times A_n| = m_1 m_2 ... m_n$.

Demonstraţie. Vom apela la metoda inducţiei matematice. Pentru $n = 1$ teorema este evident corectă. Presupunem că teorema are loc pentru $n = k$ şi vom demonstra justeţea ei pentru $n = k+1$.

Conform presupunerii $|A_1 \times A_2 \times ... \times A_k| = m_1 m_2 \cdots m_k$. Vom considera un vector arbitrar $(a_1, a_2, ..., a_k) \in A_1 \times A_2 \times ... \times A_k$ şi vom adăuga pe locul $k+1$ componenta $a_{k+1} \in A_{k+1}$. Vom obţine m_{k+1} vectori diferiţi din $A_1 \times ... \times A_{k+1}$. Cu alte cuvinte, adăugând la $m_1 m_2 \cdots m_k$ vectori de lungime k componenta cu numărul $k+1$ din A_{k+1} se vor obţine $m_1 m_2 \cdots m_k m_{k+1}$ vectori diferiţi din $A_1 \times A_2 \times ... \times A_k \times A_{k+1}$. Alţi vectori în acest produs cartezian nu există. Teorema este adevărată pentru $n=k+1$ şi, deci, este adevărată pentru n arbitrar.

Consecinţă. $|A^n| = |A|^n$.

Proiecţia vectorului v pe axa i se va numi componenta cu numărul i a acestui vector:

dacă $v=(a_1, a_2, ..., a_i, ..., a_n)$, atunci $pr_i v = a_i$.

Proiecţia lui v pe axele $i_1, i_2, ..., i_k$ se numeşte vectorul de lungime k: $pr_{i1,i2,...,ik} v = (a_{i1}, ..., a_{ik})$.

Pentru o mulţime V de vectori de aceeaşi lungime se introduce noţiunea de proiecţie a lui V pe axa i şi pe axele $i_1, i_2, ..., i_k$: $pr_i V = \{pr_i v \mid v \in V\}$ şi $pr_{i1,i2,...,ik} V = \{pr_{i1,i2,...,ik} v \mid v \in V\}$.

2.6. Corespondenţe şi funcţii

Vom numi corespondenţă între mulţimile A şi B submulţimea $\boldsymbol{G \subseteq A \times B}$. Dacă $(a, b) \in G$ se va spune că b corespunde lui a în corespondenţa G.

Mulţimea $\boldsymbol{pr_1 G}$ se va numi **domeniu de definiţie** (sau **mulţimea sursă**), iar $\boldsymbol{pr_2 G}$ - **domeniu de valori** (sau **imaginea** lui A) ale corespondenţei G. Dacă $\boldsymbol{pr_1 G = A}$ corespondenţa se va numi **total definită** (în caz contrar - parţial definită). Corespondenţa pentru care $\boldsymbol{pr_2 G = B}$ se numeşte **surjectivă**. Corespondenţa G se va numi **funcţională,** dacă fiecărui element din $pr_1 G$ îi va corespunde un singur

element din $pr_2G = B$ şi **injectivă** dacă fiecare element din domeniul de valori corespunde unui singur element din domeniul de definiţie.

O corespondenţă se numeşte biunivocă (corespondenţă 1:1) dacă ea este

- total definită,
- surjectivă,
- funcţională,
- injectivă.

Exemplul 2.16. Reprezentarea lunilor anului prin numerele lor este o corespondenţă biunivocă între mulţimea lunilor şi mulţimea $\boldsymbol{N}_{12}$ a numerelor întregi de la 1 până la 12. ◄

Teorema 2.2. Dacă între două mulţimi A şi B există o corespondenţă biunivocă, atunci

$$|\boldsymbol{A}|=|\boldsymbol{B}|.$$

Demonstraţia este banală şi poate fi realizată prin metoda reducerii la absurd. Într-adevăr, dacă teorema nu este justă, pot avea loc două cazuri:

1. $|A| > |B|$
2. $|A| < |B|$

În primul caz, deoarece corespondenţa este total definită, în A pot fi determinate două elemente cărora le va corespunde unul şi acelaşi element $b \in B$ - nu se respectă injectivitatea. Cazul $|A| > |B|$ trebuie exclus.

În al doilea caz, corespondenţa fiind surjectivă în B vor fi cel puţin două elemente care corespund unuia şi aceluiaşi $a \in A$ - nu se respectă funcţionalitatea.

Şi într-un caz şi în altul am ajuns la contradicţie, deci, nu ne rămâne decât să fim de acord cu afirmaţia teoremei.

Prima afirmaţie din teorema precedentă, care mai poate fi formulată "dacă $|A| > |B|$, atunci nu poate exista o injecţie de la A la B", este o formulare a celebrei proprietăţi, cunoscute sub numele ***principiul sertarelor*** *(celulelor)* sau ***principiul lui Dirichlet***: *„Dacă trebuie să repartizăm o mulţime de obiecte în sertare şi avem mai multe obiecte decât sertare, există întotdeauna cel puţin un sertar, care conţine mai mult de un obiect".*

Teorema 2.2 permite:

- stabilirea egalităţii cardinalelor a două mulţimi fără calcule şi
- determinarea cardinalului unei mulţimi prin stabilirea unei corespondenţe biunivoce a acestei mulţimi cu o altă mulţime, cardinalul căreia este cunoscut sau uşor de calculat.

2.6.1. Cardinalul booleanului unei mulţimi

Teorema 2.3. Dacă A este o mulţime finită şi $|A|= n$, atunci numărul tuturor submulţimilor mulţimii A este 2^n $(|\rho(A)|=2^{|A|}=2^n)$.

Demonstraţie. Vom numerota elementele mulţimii A cu numere de la 1 până la n: $A = \{a_1, a_2,..., a_n\}$ şi vom considera mulţimea B^n de vectori binari de lungime n. Fiecărei submulţimi $A^* \subseteq A$ îi vom pune în corespondenţă un vector $v = (v_1, v_2,...,v_n) \in B^n$ în aşa mod încât $v_i=0$ dacă $a_i \notin A^*$ şi $v_i=1$ dacă $a_i \in A^*$. Astfel, mulţimii vide îi va corespunde vectorul $(00...0)$, iar lui A – vectorul $(11...1)$. Evident corespondenţa stabilită între mulţimea părţilor lui A şi mulţimea vectorilor binari de lungime n este biunivocă şi de aceea $|\rho(A)| = |B^n|$. Dar $B^n = \underbrace{B \text{x} B \text{x} ... \text{x} B}_{n}$ şi $B = \{0, 1\}$. Conform consecinţei teoremei 2.1, $|B^n| = |B|^n = 2^n$, ceea ce trebuia demonstrat.

2.6.2. Mulţimi numărabile şi continuale

Două mulţimi au acelaşi cardinal dacă între ele există o corespondenţă biunivocă. Pentru mulţimile finite această afirmaţie a fost demonstrată, iar pentru cele infinite serveşte drept definiţie a acestui concept.

Definiţie. Mulţimile de acelaşi cardinal cu $\boldsymbol{N}$ (mulţimea numerelor naturale) se numesc **numărabile**.

Teorema 2.4. Mulţimea numerelor reale din segmentul $[0;1]$ nu este numărabilă.

Demonstraţia a fost propusă de Georg Cantor şi poartă denumirea de metoda diagonală Cantor. Presupunem că această mulţime este numărabilă şi, deci, există o numerotare a elementelor ei. Să reprezentăm toate numerele, care, în caz general au forma unor fracţii zecimale infinite, conform acestei numerotări:

$$0,a_{11}a_{12}a_{13}a_{14}...$$
$$0,a_{21}a_{22}a_{23}a_{24}...$$
$$0,a_{31}a_{32}a_{33}a_{34}...$$
$$.....$$

Vom considera o fracţie zecimală oarecare $0,b_1b_2b_3b_4...$ pentru care $b_1 \neq a_{11}$, $b_2 \neq a_{22}$, $b_3 \neq a_{33}$ s.a.m.d. Această fracţie nu se conţine în secvenţa de mai sus deoarece se deosebeşte de primul număr prin prima cifră după virgulă, de al doilea - prin a doua, etc. Deci, toate numerele segmentului $[0;1]$ nu pot fi numerotate şi mulţimea numerelor reale ale acestui segment este nenumărabilă. Cardinalul mulţimilor de acest tip se numeşte **continuum**, iar mulţimile - **continuale**.

2.6.3. Funcţii. Compoziţia şi superpoziţia funcţiilor

Se numeşte **funcţie** o corespondenţă funcţională. Dacă funcţia f stabileşte o corespondenţă între mulţimile A şi B se va spune că f are tipul de la A la B şi se va nota $f: A \to B$. Două funcţii f şi g se numesc **egale**, dacă $f(x)=g(x)$ pentru orice valoare a lui x (se mai notează $f=g$). Funcţia total definită $f: A \to B$ se numeşte **aplicaţie (funcţie totală)** a lui A în B. Dacă corespondenţa f în acest caz este surjectivă vom spune că are loc aplicaţia lui A pe B. Aplicaţia de tipul $A \to A$ se numeşte **transformarea** (transformata) lui A.

Funcţia $f: A_1xA_2x...xA_n \to B$ se numeşte funcţie n-ară (funcţie de n argumente). Pentru $n = 1$ vom vorbi de funcţii unare, pentru $n = 2$ – funcţii binare, etc.

Exemplul 2.17. a) Într-o arhivă dosarele sunt plasate în căsuţe speciale pentru păstrare şi accesare rapidă. Asociind fiecărui dosar căsuţa, care îl conţine, se va defini o funcţie de tipul "mulţimea dosarelor" în "mulţimea căsuţelor". Imaginea acestei funcţii este mulţimea căsuţelor ocupate (nevide). Zicând că aceasta este o aplicaţie spunem că toate dosarele sunt plasate în căsuţe. Aplicaţia dată este injectivă, dacă fiecare căsuţă conţine cel mult un dosar. Dacă nu există căsuţe libere aplicaţia este surjectivă.

b) Numărul de înmatriculare a unui automobil este format dintr-o succesiune de litere şi cifre în felul următor: 2, 3 sau 4 litere urmate de 2, 3 sau 4 cifre, de exemplu, CES 919. În termeni matematici înmatricularea automobilelor defineşte o funcţie, care pune în corespondenţă mulţimii automobilelor A mulţimea B, elementele căreia sunt obţinute în modul susnumit.

c) Dacă $A=\{Andrei, Elena, Mihai, Ana, Victor, Valentina, Eugen, Corina, Christian\}$ este o mulţime de persoane concrete şi B o mulţimea formată din zilele anului, poate fi definită o aplicaţie $f : A \to B$ asociind fiecărui element din A ziua sa de naştere. Aplicaţia dată poate fi reprezentată prin tabelul său de valori. Este uşor să se ajungă la concluzia că această aplicaţie poate să nu fie injectivă. ◄

Exemplul 2.18. Fie A o submulţime a unei mulţimi B. Aplicaţia $f: A \to B$ definită prin $f(x) = x$ pentru orice $x \in A$ se numeşte ***injecţie canonică a lui A în B.*** După cum reiese şi din denumire această aplicaţie este injectivă. ◄

Fie $G \subseteq AxB$. **Dacă corespondenţa $H \subseteq BxA$ are loc atunci când perechea (b,a) aparţine lui H dacă şi numai dacă $(a,b) \in G$, H se va numi inversa lui G şi se va nota prin G^{-1}**. Dacă corespondenţa inversă funcţiei $f:A \to B$ este funcţională ea se va numi funcţie inversă funcţiei f şi se va nota prin f^{-1}.

Fie $f:A\to B$ şi $g:B\to C$. Funcţia $h:A\to C$ se va numi **compunerea** funcţiilor f şi g (vom nota fog), dacă are loc egalitatea $h(x)=g(f(x))$, în care $x\in A$. Se mai spune că h a fost obţinută prin substituirea lui f în g.

Menţionăm fără demonstraţie [2]:

Teorema 2.5. Dacă f şi g sunt injective, la fel va fi şi gof.

Teorema 2.6. Dacă f şi g sunt surjective, la fel va fi şi gof.

Teorema 2.7. Dacă f şi g sunt bijective, la fel va fi şi gof şi $f^{-1}og^{-1}$.

Pentru funcţii de mai multe argumente $f:A^m\to B$, $g:B^n\to C$ sunt posibile mai multe variante de substituţie, care conduc la funcţii de diferite tipuri. Un interes aparte prezintă cazul când avem funcţii de tipul $f_1:A^{m1}\to A$,..., $f_n:A^{mn}\to A$. În acest caz sunt posibile orice substituţii de funcţii şi redenumiri de argumente. Funcţia obţinută din f_1,..., f_n printr-o substituire oarecare şi redenumirea argumentelor se numeşte **superpoziţia** f_1,..., f_n.

2.7. Relaţii şi proprietăţile lor

2.7.1. Noţiuni introductive

Se numeşte **relaţie** n-ară submulţimea R a produsului cartezian $A_1xA_2\ldots An$: $\boldsymbol{R\subseteq A_1xA_2\ldots An}$. Vom zice că a_1,..., a_n, sunt în relaţia R dacă $(a_1,\ldots, a_n)\in R$. Cardinalul mulţimii vectorilor, care formează relaţia se numeşte cardinalul relaţiei.

O relaţie unară este o parte a mulţimii M şi determină o proprietate a elementelor unei submulţimi a mulţimii M din care cauză pentru $n = 1$ denumirea de relaţie practic nu este utilizată. Un interes mai mare prezintă cazul când $n = 2$ - relaţiile binare. Dacă a şi b se află în relaţia R, aceasta se va scrie aRb.

Exemplul 2.19. Pentru mulţimea N: relaţia "<" are loc în cazul perechii *(3,9)* şi nu are loc pentru perechea *(6,4)*. Relaţia *"a fi divizor"* are loc pentru perechea *(7,35)* şi nu are loc pentru *(18,2)* sau *(4,9)*. Pentru o mulţime de oameni pot fi relaţii de tipul *"a fi prieteni", "a locui in acelaşi oraş", "a fi fiu"*, etc. ◄

Restricţia lui R pe $M_1\subseteq M$ este $R^1 = R\cap M_1^2$.

Pentru definirea unei relaţii pot fi utilizate oricare din metodele de definire a mulţimilor. O posibilitate suplimentară este matricea de relaţie. Pentru mulţimea $M = \{a_1, a_2,\ldots, a_m\}$ aceasta este o matrice mxm în care

$$a(i,j) = \begin{cases} 1 \text{ dacă } a_iRa_j, \\ 0 \text{ în caz contrar.} \end{cases}$$

Deoarece relaţia este în ultimă instanţă o mulţime, pot fi executate aceleaşi operaţii (reuniune, intersecţie, etc.) şi cu relaţiile.

Noţiunea *relaţie* formează fundamentul bazelor de date de tip relaţional. Relaţiile sunt analogul matematic al tabelelor. Chiar termenul "reprezentarea relaţională a datelor", introdusă de către Edgar Codd [3], provine de la termenul *relation*. Două momente trebuie subliniate în mod deosebit.

Primul moment: toate elementele unei relaţii sunt vectori (cortejuri) de acelaşi tip. Din această cauză cortejurile sunt analogul liniilor într-un simplu tabel, adică un tabel în care fiecare linie conţine acelaşi număr de câmpuri, iar în câmpurile respective sunt păstrate date de acelaşi tip. De exemplu, relaţia, care conţine trei cortejuri {(1, Petrescu Diana, TI-981), (3, Popescu Adrian, TI-982), (5, Ivanov Vadim, TI-983)} poate fi considerată tabel cu date despre trei studenţi (numărul de ordine, numele şi prenumele şi grupa academică). Acest tabel are trei linii şi trei coloane, în fiecare coloană conţinându-se date de acelaşi tip.

Din contra, mulţimea {(1), (1, 2), (1, 2, 3)}, care conţine cortejuri numerice de tipuri diferite, nu este relaţie nici în R, nici în R^2, nici în R^3. Din cortejurile, care intră în această mulţime nu poate fi construit un tabel simplu.

Al doilea moment: relaţia poate să nu cuprindă toate cortejurile posibile ale produsului cartezian (cu excepţia cazului extrem, când relaţia este chiar produsul cartezian). Aceasta înseamnă, că fiecărei relaţii îi corespunde un *criteriu*, care permite să se determine care dintre cortejuri aparţin relaţiei date şi care nu. Acest criteriu determină *sensul* (*semantica*) relaţiei.

O relaţie se numeşte **inversa** relaţiei R (se va nota R^{-1}) dacă $a_i R a_j$ are loc atunci şi numai atunci când are loc $a_j R^{-1} a_i$.

2.7.2. Proprietăţile relaţiilor

Relaţia poate poseda o serie de proprietăţi dintre care vom menţiona **reflexivitatea, simetria** şi **tranzitivitatea**. Dacă pentru $\forall a \in M$ are loc aRa relaţia R se numeşte **reflexivă**. Diagonala principală a matricei relaţiei R conţine numai unităţi. Relaţia R se numeşte **antireflexivă** dacă nu există $a \in M$ pentru care ar avea loc aRa. Diagonala principală a matricei unei astfel de relaţii conţine numai zerouri. Relaţiile *"≥", "a avea un divizor comun"* sunt reflexive. Relaţiile *"a fi fiu", ">"* - sunt antireflexive.

Dacă pentru o pereche $(a,b) \in M^2$ din aRb rezultă bRa (relaţia are loc în ambele părţi sau nu are loc de fel), relaţia R se numeşte **simetrică**. Pentru astfel de relaţii $c(i,j) = c(j,i)$: matricea este simetrică faţă de diagonala principală.

Relaţia se va numi **antisimetrică**[*]**,** dacă din a_iRa_j şi a_jRa_i rezultă că $a_i=a_j$. Relaţia "≤" este antisimetrică, iar *"a locui în acelaşi oraş"* - simetrică.

Relaţia R se numeşte **tranzitivă** dacă pentru oricare a, b şi c din aRb şi bRc rezultă aRc. Relaţiile *"a locui în acelaşi oraş", "egal", "<"* sunt tranzitive, iar *"a fi fiu"* nu este tranzitivă.

O relaţie binară se numeşte ***relaţie funcţională***, dacă atunci când $(x, y) \in R$ şi $(y, z) \in R$ vom avea în mod obligator $x=z$ (univocitatea funcţiei).

Pentru oricare relaţie R poate fi definită noţiunea de închidere tranzitivă R^*: aR^*b (a se află în relaţia R^* cu b), dacă în M există o secvenţă de n elemente $a=a_1,..., a_{n-1}, a_n = b$ în care pentru elementele vecine are loc R: $aRa_2, a_2Ra_3,..., a_{n-1}Rb$. Dacă R este tranzitivă, atunci $R^*=R$. Pentru relaţia *"a fi fiu"* relaţia *"a fi descendent direct"* este închidere tranzitivă (este reuniunea relaţiilor *"a fi fiu", "a fi nepot", "a fi strănepot"* s.a.m.d.).

O relaţie care posedă proprietăţile reflexivitate, simetrie şi tranzitivitate se numeşte relaţie de **echivalenţă**.

Se numeşte **relaţie de ordine** oricare relaţie care posedă proprietăţile reflexivitate, antisimetrie şi tranzitivitate. O relaţie antireflexivă, antisimetrică şi tranzitivă se numeşte **relaţie de ordine strictă**. Două elemente a şi b se numesc **comparabile** conform relaţiei de ordine R dacă are loc aRb sau bRa. O mulţime M cu o relaţie de ordine definită pe M se numeşte **total ordonată** dacă oricare două elemente din M sunt comparabile şi **parţial ordonată,** în caz contrar.

Exemplul 2.20. Relaţiile *"≤", "≥"* pentru o mulţime de numere sunt relaţii de ordine, iar *"<", ">"* - de ordine strictă. Relaţia de subordonare în cadrul unei întreprinderi defineşte o ordine strictă (dar parţială - nu pot fi comparaţi colaboratorii diferitor departamente).
În alfabetul latin literele sunt aranjate într-o ordine binecunoscută: se află în relaţia de precedare a literelor. Conform acestei relaţii poate fi stabilită relaţia de precedare a cuvintelor - ordinea lexicografică a cuvintelor (utilizată de exemplu în dicţionare). Relaţia de ordine lexicografică poate fi definită şi pentru informaţii numerice. De exemplu, în calculator data şi anul sunt memorizate sub forma *"anul, luna, ziua"* pentru ca ordinea de creştere a datei totale să coincidă cu ordinea lexicografică. ◄

Exemplul 2.21. B^* este mulţimea tuturor cuvintelor binare de orice lungime şi ε desemnează unicul cuvânt binar de lungime *0*. Dacă m este un cuvânt

[*] Termenul *antisimetric* nu a fost ales prea norocos, deoarece se poate crede că o relaţie, care nu este simetrică este totdeauna antisimetrică, ceea ce nu este corect.

binar de lungime p, vom nota biţii acestuia $m_1, m_2,..., m_p$. Relaţia de ordine $\mathfrak{R}$, definită peste B^*, se numeşte de ordine lexicografică, dacă:

a) $\varepsilon \mathfrak{R} m, \forall m$

b) pentru m şi n de lungime p şi q cu proprietatea $0 < p \le q$

$m \mathfrak{R} n$, dacă $m_1=n_1, m_2=n_2,..., m_p=n_p$,

$m \mathfrak{R} n$, dacă $m_1=n_1, m_2=n_2,..., m_{s-1}=n_{s-1}, m_s < n_s$, pentru $1 \le s \le p$,

$n \mathfrak{R} m$, dacă $m_1=n_1,..., m_{s-1}=n_{s-1}, m_s > n_s$, pentru $1 \le s \le p$. ◄

2.7.3. Alte metode de descriere a relaţiilor

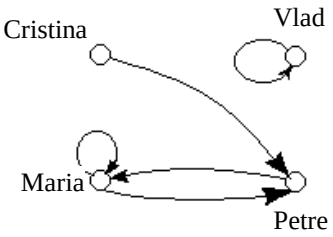
Fie mulţimea tinerilor {*Vlad, Petre, Maria, Elena*}despre care se cunoaşte că:

1. *Vlad* îl iubeşte pe *Vlad* (egoism☺),
2. *Petre* o iubeşte pe *Maria* (reciproc☺),
3. *Maria* îl iubeşte pe *Petre* (reciproc☺),
4. *Maria* o iubeşte pe *Maria* (egoism feminin☺),
5. *Cristina* îl iubeşte pe *Petre* (dragoste nefericită☹).

Informaţia despre relaţiile dintre aceşti tineri poate fi descrisă cu ajutorul relaţiei binare "*a iubi*", definite peste mulţimea iniţială. Relaţia dată poate fi descrisă prin câteva metode.

Metoda 1. Enumerarea faptelor sub formă de text de formă arbitrară (vezi mai sus).

Metoda 2. Grafic (graful relaţiei):



Metoda 3. Cu ajutorul matricei de relaţie:

	Vlad	Petre	Maria	Cristina
Vlad	1	0	0	0
Petre	0	0	1	0
Maria	0	1	1	0
Cristina	0	1	0	0

Metoda 4. Cu ajutorul unui tabel al faptelor:

Cine iubeşte	Pe cine iubeşte
Vlad	Vlad
Petre	Maria
Maria	Petre
Maria	Maria
Cristina	Petre

Din punctul de vedere al bazelor de date de tip relaţional mai convenabilă este ultima metodă, deoarece permite păstrarea şi manipularea cea mai simplă, eficientă şi comodă a datelor. Într-adevăr, enumerarea faptelor în formă textuală este binevenită în cazul unei opere literare, dar este dificil de algoritmizat. Forma grafică este cea mai elocventă şi poate fi utilizată pentru reprezentarea finală a informaţiilor, dar păstrarea datelor în formă grafică presupune eforturi suplimentare. Matricea relaţiei este mai aproape de cerinţele unui sistem informaţional, fiind comodă pentru manipulare, deşi adesea puternic rarefiată. Însă modificări neesenţiale (de exemplu a apărut un oarecare Vasile care s-a îndrăgostit de nefericita Elena) pot conduce la modificarea întregii matrice (vor apare şi linii şi coloane noi). Tabelul faptelor este liber de toate acestea.

Ultima relaţie nu este nici tranzitivă, nici simetrică sau antisimetrică, nici reflexivă. Din această cauză ea nu este nici relaţie de echivalenţă, nici relaţie de ordine, nici o oarecare altă relaţie care ar conduce la ceva raţional. Cea mai mare parte a literaturii universale prezintă interes anume din cauza, că relaţia binară "*a iubi*" nu este o relaţie de echivalenţă. În particular, asta este cauza, că omenirea nu se împarte în clase de echivalenţă de oameni, care se iubesc reciproc. De cercetarea caracteristicilor acestei relaţii s-au ocupat (şi continuă să se ocupe) foarte mulţi experţi, dintre care amintim pe M. Eminescu, L.Tolstoi, W. Shakespeare, etc.

2.8. Operaţii şi algebre. Proprietăţile operaţiilor

Funcţia de tipul $\varphi: M^n \to M$ se va numi **operaţie** *n*-ară pe *M*. Setul $A = <M, \Omega>$, în care Ω este o mulţime de operaţii definite pe *M*, se numeşte **algebră**. Mulţimea *M* se va numi mulţime de bază sau suportul, iar $\Omega = \{\varphi_1, \varphi_2, \ldots, \varphi_m, \ldots\}$ - signatura algebrei *A*. Vectorul, componentele căruia sunt arităţile operaţiilor φ_1, φ_2,... se numeşte tipul algebrei *A*.

Operaţia φ se numeşte:

a) comutativă, dacă $a\varphi b = b\varphi a$,
b) asociativă, dacă pentru oricare *a, b, c* are loc $(a\varphi b)\varphi c = a\varphi(b\varphi c)$,
c) idempotentă, dacă $a\varphi a = a$,

d) distributivă stânga faţă de operaţia g, dacă pentru oricare a, b, c are loc relaţia

e) $a\varphi(bgc) = (a\varphi b)g(a\varphi c)$, şi distributivă dreapta dacă $(agb)\varphi c = (a\varphi c)g(b\varphi c)$.

Dacă există un element e pentru care are loc $a\varphi e = e\varphi a = a$, atunci acest element se numeşte neutru (sau unitate).

Exemplul 2.22. a) Pentru o mulţime arbitrară U şi mulţimea tuturor părţilor $\boldsymbol{B(U)}$, algebra $A = \{\boldsymbol{B(U)}, \cup, \cap, \overline{\ } \}$ se numeşte algebră booleana a mulţimilor. Tipul ei este $(2,2,1)$.

b) Algebra $A = \{R,+,x\}$ se numeşte câmp al numerelor reale. Ambele operaţii sunt binare, deci tipul este $(2,2)$.◄

Algebrele $L = \{M,\cup,\cap\}$ (cu două operaţii binare - reuniunea şi intersecţia) se numesc ***latice***, dacă au loc axiomele:

P1: $a\cup b = b\cup a$, $a\cap b = b\cap a$ - **comutativitate**,

P2: $a\cup(b\cup c) = (a\cup b)\cup c$, $a\cap(b\cap c) = (a\cap b)\cap c$ - **asociativitate**,

P3: $a\cup(b\cap a) = a$, $a\cap(b\cup a) = a$ - **absorbţie**,

pentru oricare $a, b, c \in M$.

Se poate observa că în acest sistem de axiome se pot schimba între ele simbolurile $\cup$ şi $\cap$, proprietate cunoscută sub denumirea de principiul dualităţii pentru latici. De asemenea, plecând de la axiomele de mai sus se poate demonstra proprietatea numită **idempotenţă** $a\cup a = a$, $a\cap a = a$ pentru oricare $a \in M$. Prin definiţie, o latice finită (mărginită) are un element care este cea mai mică margine superioară, numit prim element al laticii, notat prin 1, astfel încât $a\cup 1 = 1$, $a\cap 1 = a$, $a \in M$ şi un element care este cea mai mare margine inferioară, numit ultim element al laticii, notat prin 0, astfel încât $a\cup 0 = 0\cup a = a$, $a\cap 0 = 0\cap a = 0$, $a \in M$.

Fie $L = \{M, \cup, \cap, 0, 1\}$ o latice finită şi $a \in M$. Un element complementar sau pe scurt un complement al elementului a este elementul $\bar{a}$ (non a), astfel încât

$a\cup\bar{a} = 1$ - principiul terţului exclus,

$a\cap\bar{a} = 0$ - principiul contradicţiei.

Evident, nu oricare element dintr-o latice finită are un complement, iar dacă acesta există, nu este în mod necesar unic. Subliniem aici, că elementele 0 şi 1 au fiecare un complement unic, respectiv 1 şi 0: $\bar{0} = 1$, $\bar{1} = 0$.

Dacă într-o latice finită orice element a are un complement $\bar{a}$, această latice se numeşte complementară.

O latice L este distributivă dacă şi numai dacă
$(a \cup b) \cap c = (a \cup c) \cap (b \cup c)$,
$(a \cap b) \cup c = (a \cap c) \cup (b \cap c)$, $a, b, c \in M$.

Pentru două algebre $A = \{C, \varphi_1, \varphi_2, ..., \varphi_n\}$ şi $B = \{D, g_1, g_2, ..., g_n\}$ de acelaşi tip se numeşte **omomorfismul** algebrei A în algebra B aplicaţia $\Gamma: C \to D$, care verifică condiţia

$$\Gamma(\varphi_i(c_{j1}, ..., c_{jl(i)})) = g_i(\Gamma(c_{j1}), ..., \Gamma(c_{jl(i)})) \qquad (2.5)$$

pentru toţi $i = 1, ..., n$ şi toate elementele $c_{jr} \in C$.

Un omomorfism biunivoc se numeşte **izomorfism** al algebrei A în algebra B. Dacă există izomorfismul lui A în B, atunci există şi izomorfismul lui B în A - algebrele A şi B se vor spune **izomorfe**.

Pentru cazul $A = B$ izomorfismul se va numi **automorfism**.

2.9. Modele si sisteme algebrice. Algebra relaţiilor

Noţiunea de **model** este una din noţiunile de bază în matematica discretă. Se va numi **model** M setul care constă din mulţimea D - suportul modelului, şi o mulţime de relaţii S definite pe D:

$$M = <D, S>, \qquad (2.6)$$

În (2.6) $S = \{R_{11}, R_{12}, ..., R_{1,n1}, R_{21}, R_{22}, ..., R_{2,n2}, ..., R_{m1}, R_{m2}, ..., R_{m,nm}\}$ este **signatura** modelului, $R_{ij} \in M^i$. Exponenta suportului determină aritatea relaţiei. Două relaţii R_i şi R_j care au aceeaşi aritate se numesc **compatibile.**

Setul care conţine mulţimea D, operaţiile şi relaţiile definite pe D

$$A = <D, F, S> \qquad (2.7)$$

se numeşte **sistem algebric**.

Modelul este un caz particular al sistemului algebric, când mulţimea F este vidă, iar pentru o algebră mulţimea S este vidă.

Un alt caz particular al sistemelor algebrice îl constituie **algebra relaţiilor** şi extensia acesteia - **algebra relaţională**. Pentru o algebră a relaţiilor drept suport serveşte mulţimea relaţiilor considerate, iar signatura o formează operaţiile de reuniune, intersecţie, diferenţă şi produsul cartezian extins al relaţiilor. Să facem cunoştinţă cu aceste operaţii.

Reuniunea $R_i \cup R_j$ a două relaţii compatibile R_i şi R_j este mulţimea tuturor cortejurilor, fiecare dintre care aparţine cel puţin uneia din relaţii.

Intersecţia $R_i \cap R_j$ a două relaţii compatibile R_i şi R_j se va numi mulţimea tuturor cortejurilor care aparţin simultan ambelor relaţii.

Diferenţa $R_i \setminus R_j$ a două relaţii compatibile R_i şi R_j se numeşte mulţimea tuturor cortejurilor care aparţin lui R_i şi nu aparţin lui R_j .

Produs cartezian extins $R_i x R_j$ a două relaţii R_i şi R_j se va numi mulţimea tuturor cortejurilor formate prin concatenarea lui $a \in R_i$ şi a lui $b \in R_j$.

De exemplu, dacă R_i*={(a,b),(a,c),(a,e)}*, iar R_j *= {(a,b,c),(c,d,e)}*, atunci $R_i x R_j$ *= {(a,b,a,b,c), (a,b,c,d,e), (a,c,a,b,c), (a,c,c,d,e), (a,e,a,b,c), (a,e,c,d,e)}.*

Algebra relaţiilor şi modelele sunt utilizate pentru formalizarea unor obiecte reale. Vom exemplifica prin folosirea algebrei relaţiilor în cazul bazelor relaţionale de date.

O **bază de date** de tip relaţional este un tablou bidimensional în care coloanele determină aşa numitele *domene* (atribute), iar liniile sunt cortejuri de valori concrete ale atributelor, care se află în relaţia *R*.

Exemplul 2.23. Relaţia R_5 - "examene" (v.tab. 2.3). Relaţia R_5 este o submulţime a produsului cartezian $D_1 x D_2 x D_3 x D_4 x D_5$. Elemente ale domenului D_i sunt valorile atributelor:

D_1 *= {3-101, 3-501, 3-502, 3-310}* - numerele auditoriilor unde au loc examenele;
D_2 *= {Matematica discretă în inginerie, Microelectronica, Fizica, Circuite integrate, Electrotehnica}* - denumirea disciplinelor;
D_3 *= {conf.V.Popescu, conf. V.Negură, conf.A.Diligul, conf.V.Şontea, prof.I.Samusi}* - examinatorii;
D_4 *= {3 iunie, 4 iunie, 8 iunie, 13 iunie} - data examenului;*
D_5 *= {TI-961, TI-962, TI-963, C-941, C-942, C-951, C-952}* - codul grupei.

Tabelul 2.3. Relaţia "examene".

R_5	D_1	D_2	D_3	D_4	D_5
1	3-101	Matematica discretă	conf. V.Popescu	3 iunie	TI-961
2	3-202	Microelectronica	prof. V.Şontea	4 iunie	C-951
3	3-310	Fizica	prof. I.Samusi	3 iunie	TI-962
4	3-101	Circuite integrate	conf. V. Negură	4 iunie	C-941
5	3-104	Electrotehnica	conf. A. Diligul	3 iunie	TI-951
6	3-101	Matematica discretă	conf. V. Popescu	8 iunie	TI-962
7	3-101	Matematica discretă	conf. V. Popescu	13 iunie	TI-963
8	3-202	Microelectronica	prof. V. Şontea	8iunie	C-952
9	3-310	Fizica	prof. I. Samusi	8iunie	TI-961
10	3-101	Circuite integrate	conf. V. Negură	8iunie	C-942

Numerele *1, 2,..., 10* din prima coloană identifică elemente ale relaţiei R_5. ◄

2.10. Algebra relaţională

Algebra relaţională este o extensie a algebrei relaţiilor în sens că signatura *S* în afară de cele 4 operaţii descrise anterior mai conţine câteva operaţii speciale, cum ar fi ***proiecţia, selecţia, joncţiunea***. Mai pot fi incluse operaţia de atribuire, care permite să se păstreze în baza de date rezultatele calculelor unor expresii algebrice, operaţia de redenumire a atributelor, care dă posibilitatea formării corecte a schemei relaţiei rezultante, etc. Ideea principală a algebrei relaţionale constă în faptul că, deoarece relaţiile sunt mulţimi, mijloacele de manipulare a relaţiilor se pot baza pe operaţiile tradiţionale cu mulţimi, completate cu câteva operaţii suplimentare, specifice bazelor de date.

Există mai multe abordări în definirea algebrei relaţionale, care diferă prin setul de operaţii şi interpretarea lor, abordări în ultimă instanţă echivalente. Vom descrie în continuare o extensie a variantei iniţiale, propuse de Codd [3]. Aici signatura algebrei relaţionale conţine şapte operaţii, patru legate de mulţimi şi trei speciale:

- reuniunea relaţiilor;
- intersecţia relaţiilor;
- diferenţa relaţiilor;
- produsul cartezian extins;
- selecţia;
- joncţiunea,
- proiecţia.

Operaţiile enumerate mai sus pot fi interpretate după cum urmează.

- Din reuniunea a două relaţii rezultă relaţia, care include toate cortejurile din cadrul relaţiilor-operanzi.
- Din intersecţia a două relaţii rezultă relaţia, care include doar cortejurile prezente în ambele relaţii-operanzi.
- Relaţia diferenţa a două relaţii conţine toate cortejurile primului operand, care nu sunt prezente şi în cel de-al doilea operand.
- În rezultatul calculării produsului cartezian extins obţinem o nouă relaţie, cortejurile căreia sunt concatenarea (în sensul descris mai sus) cortejurilor din prima şi a doua relaţie.

Operaţia **selecţie** permite evidenţierea unei submulţimi de cortejuri care posedă o proprietate dată. De exemplu, operaţia selecţie permite evidenţierea relaţiei *orarul conf. V. Popescu* - liniile în care valoarea domenului D_3 este conf. V. Popescu:

Tabelul 2.4. Rezultatul operaţiei *selecţie* pentru valoarea "conf. V. Popescu"

R_5	D_1	D_2	D_3	D_4	D_5
1	3-101	Matematica discretă	conf. V.Popescu	3 iunie	TI-961
6	3-101	Matematica discretă	conf. V.Popescu	8 iunie	TI-962
7	3-101	Matematica discretă	conf. V.Popescu	13 iunie	TI-963

Operaţia **proiecţie** se defineşte introducând pentru suportul D al algebrei relaţionale o partiţie de n submulţimi (n este aritatea relaţiei) $R_n \in D^n$. Proiecţia relaţiei binare $R_2 \in AxB$ pe A (PrR_2/A) se numeşte mulţimea $\{a_i \mid (a_i,b_i) \in R_2\}$. Proiecţia $PrR_n/A_{i1},...,A_{im}$ a relaţiei n-are $R_n \in A_1xA_2x...xA_n$, $m \le n$, *pe* $A_{i1}, A_{i2},..., A_{im}$ se numeşte mulţimea cortejurilor $(a_{i1}, a_{i2},..., a_{im})$, în care $a_{i1} \in A_{i1}$, $a_{i2} \in A_{i2},...,$ $a_{im} \in A_{im}$ şi fiecare cortej este parte a unui element al relaţiei n-are R_n. Cu alte cuvinte, operaţia *proiecţie* permite construirea unei submulţimi *verticale* a relaţiei (a unei mulţimi de submulţimi de atribute care se obţine prin alegerea unor domene concrete). De exemplu, $Pr(R_5/D_2,D_3)$ determină denumirea examenelor şi numele examinatorilor (liniile care coincid se scriu o singură dată, v.tab. 2.5).

Tabelul 2.5. Rezultatul operaţiei "*proiecţie*".

D_2	D_3
Matematica discretă	conf. V.Popescu
Microelectronica	prof. V.Şontea
Fizica	prof. I.Samusi
Circuite integrate	conf. V.Negură
Electrotehnica	conf. A.Diligul

Operaţia **joncţiune** (join) a două tabele care au un domen comun permite construirea unui tabel nou în care fiecare linie se va obţine din unirea a două linii din tabelele iniţiale. Aceste linii corespund aceluiaşi atribut din domenul comun. Domenul comun se va scrie o singură dată. De exemplu, pentru tabele 2.6 şi 2.7 domenul comun este D_5, rezultatul operaţiei de joncţiune este prezentat în tabelul 2.8.

Tabelul 2.6.

D_1	D_2	D_3	D_4	D_5
3-202	Microelectronica	prof. V.Şontea	4 iunie	C-951
3-310	Fizica	prof. I.Samusi	3 iunie	TI-962
3-104	Electrotehnica	conf. A.Diligul	3 iunie	TI-951

Tabelul 2.7.

D_1	D_2	D_3	D_4	D_5
3-104	Electrotehnica	conf. A.Diligul	13 iunie	C-951
3-310	Matematica	conf. L.Dogotaru	13 iunie	TI-962
3-202	Microelectronica	prof. V.Şontea	14 iunie	TI-951

Tabelul 2.8. Rezultatul operaţiei "*join*".

D_1	D_2	D_3	D_4	D_{11}	D_{21}	D_{31}	D_{41}	D_5
3-202	Microelectronica	conf. V.Şontea	4 iunie	3-104	Electrotehnica	conf. A.Diligul	13 iunie	C-951
3-310	Fizica	prof. I.Samusi	3 iunie	3-310	Matematica	conf. L.Dogotaru	13 iunie	TI-962
3-104	Electrotehnica	conf. A.Diligul	3 iunie	3-202	Microelectronica	prof. V.Şontea	14 iunie	TI-951

Operaţia *join* este definită nu numai pentru condiţia de *egalitate* a două domene, ci pot fi şi alte condiţii de comparare, de exemplu, $>$, $\geq$, $<$, $\leq$, etc.

2.11. Exerciţii şi probleme

2.1. Care sunt elementele mulţimii $\{\{a, b, c\}, \{a\}, \{b, c\}\}$?

2.2. Demonstraţi, că $\rho(A) \subset \rho(B)$, dacă $A \subset B$.

2.3. Este oare justă relaţia $\{a\} \in \{a, b, c\}$? Formaţi lista părţilor mulţimii $A=\{a, b, c\}$.

2.4. Pentru cazurile de mai jos determinaţi dacă mulţimile A şi B sunt egale.

a) $A=\{x \in \mathbf{R} \mid x>0\}$ $B=\{x \in \mathbf{R} \mid x \geq |x|\}$;

b) $A=\{x \in \mathbf{R} \mid x>0\}$ $B=\{x \in \mathbf{R} \mid x \leq |x|\}$;

c) $A=\mathbf{Z}$ $B=\{x \in \mathbf{Z} \mid x^2 - x$ este număr par$\}$;

d) $A=\{x \in \mathbf{N_{20}} \mid x$ - impar şi nu se împarte la 3$\}$ $B=\{x \in \mathbf{N_{20}} \mid x^2-1$ este divizibil prin 24$\}$.

2.5. Pentru $\mathbf{B}=\{0, 1\}$ determinaţi:

a) Este oare justă relaţia $\mathbf{B} \in \mathbf{B}$?

b) Care sunt elementele lui $\rho(\mathbf{B})$?

c) Care sunt elementele lui $\rho(\rho(\mathbf{B}))$?

2.6. Definiţi mulţimile:

a) Mulţimea numerelor întregi mai mari ca 100.

b) Mulţimea numerelor întregi pare.

2.7. Propuneţi câte o procedură generatoare pentru mulţimile de mai jos.

a) $A=\{1, 2, 4, 8, 16, 32, 64,...\}$;

b) $B=\{1, 2, 7, 14,...\}$;

c) $C=\{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20\}$.

2.8. Definiţi prin enumerare următoarele mulţimi:

a) $A=\{x \in R \mid x(x+5)=14\}$;

b) $B=\{x \in N \mid x(2x+3)=14\}$;

c) $C=\{x \in N_{25} \mid x$ este suma pătratelor a două numere naturale; $N_n=\{1, 2,..., n\}\}$.

d) $D=\{x \in N_{12} \mid x$ este număr perfect$\}$.

e) $E=\{x \in N_{10} \mid x^4 - 1$ se împarte la 5$\}$.

2.9.Demonstraţi echivalenţele:

a) $(S \cup (T \cap R)) \equiv ((S \cup T) \cap (S \cup R))$

b) $((S \cup T) - R) \equiv ((S - R) \cup (T - R))$

c) $(S - (T \cup R)) \equiv ((S - T) - R)$

2.10. Fie $S \subseteq T$, demonstraţi că:

a) $(S \cap T) = S$

b) $(S - T) = \varnothing$

c) $\rho(S) \subseteq \rho(T)$.

2.11. Arătaţi că

a) $\rho(S) \cup \rho(T) \subseteq \rho(S \cup T)$

b) $\rho(S \cap T) \subseteq \rho(S) \cap \rho(T)$

Pot *(a)* sau *(b)* fi adevărate dacă incluziunea este înlocuită prin echivalenţă ? Ce reprezintă $\rho(\rho(\rho(\varnothing)))$?

2.12. Determinaţi reuniunea mulţimilor *A* şi *B*:

a) $A = \{x \in N \mid x \text{ este impar}\}$ $B = \{x \in N \mid x \text{ nu se împarte la } 5\}$

b) $A = \{x \in R \mid 0 < x < 4\}$ $B = \{x \in R \mid 1 < x < 3\}$

c) $A = \{(x, y) \in R^2 \mid x + y < 2\}$ $B = \{x \in R \mid 3 < 4x - y\}$

2.13. Pentru trei mulţimi se cunoaşte că $A \cap B \cap C = \varnothing$. Sunt ele oare în mod obligator disjuncte două câte două? Exemplificaţi.

2.14. Asociem fiecărui număr real $a \in R$ mulţimea $E_a = \{x \in R \mid (1-a^2) \le x \le (1-a^2)\}$, ceea ce generează o familie de mulţimi indexate cu elemente din *R*. Determinaţi reuniunea şi intersecţia mulţimilor acestei familii.

2.15. Pentru fiecare dintre cazurile care urmează să se determine dacă mulţimile *A* şi *B* sunt egale, sau dacă una se conţine în cealaltă, sau dacă există o injecţie a uneia în alta.

a) $A = \rho(E \times F)$ $B = \rho(E) \times \rho(F)$

b) $A = \rho(E_1 \times E_2 \times ... \times E_i)$ $B = \rho(E_1) \times \rho(E_2) \times ... \times \rho(E_i)$, $i \in I$

c) $A = \rho(E \cup F)$ $B = \rho(E) \cup \rho(F)$

d) $A = \rho(E \cap F)$ $B = \rho(E) \cap \rho(F)$

2.15. Reprezentaţi grafic A^2, B^2, C^2, $A \times B$, $A \times C$, $B \times C$, dacă $A = [-3; -1] \cup [1; 3]$, $B = \{-3, -1\} \cup [1; 3]$, $C = \{-3, -1\} \cup \{1, 3\}$.

2.16. Sunt date mulţimile *E, F* şi *G*. Pentru fiecare dintre cazurile care urmează să se determine dacă mulţimile *A* şi *B* sunt egale sau dacă una se conţine în cealaltă.

a) $A = E \times (F \cup G)$ $B = (E \times F) \cup (E \times G)$

b) $A = E \times (F \cap G)$ $B = (E \times F) \cap (E \times G)$

c) $A = E \cup (F \cap G)$ $B = (E \cup F) \cap (E \cup G)$

d) $A = E \cap (F \cup G)$ $B = (E \cap F) \cup (E \cap G)$

e) $A = E \cup (F \times G)$ $B = (E \cup F) \times (E \cup G)$

f) $A = E \cap (F \times G)$ $B = (E \cap F) \times (E \cap G)$

2.17. Definiţi două mulţimi *A* şi *B* şi o corespondenţă (o aplicaţie $f: A \to B$) care ar permite interpretarea fiecărei dintre situaţiile de mai jos

a) registrul unui hotel cu 100 camere;

b) o carte de telefoane;

c) rezultatele unui tiraj "Superloto";

d) cuprinsul unei cărţi.

Ce puteţi spune despre proprietăţile acestor corespondenţe (aplicaţii)?

2.18. Pentru fiecare dintre cazurile de mai jos să se stabilească dacă corespondenţa dintre A şi B (aplicaţia $f:A\to B$) este surjectivă, injectivă sau bijectivă. Determinaţi inversa, atunci când f este bijectivă.

a) $A=\mathbf{R}$ $B=\mathbf{R}$, $f(x)=x+7$

b) $A=\mathbf{R}$ $B=\mathbf{R}$, $f(x)=2x^2+12x+16$

c) $A=\{x\in\mathbf{R} \mid |x|\le 3\}$ $B=\{x\in\mathbf{R} \mid 20\le|x|\le 100\}$ $f(x)=x^2+6x+8$

d) $A=\mathbf{R}$ $B=\mathbf{R}$, $f(x)=5x-4|x|$

e) $A=\mathbf{R}$ $B=\mathbf{R}$, $f(x)=e^x+2$

f) $A=\mathbf{N}$ $B=\mathbf{N}$, $f(x)=x^2+x$.

2.19. Fie aplicaţiile f şi g: $\mathbf{N_{10}}\to\mathbf{N_{10}}$ definite cu ajutorul tabelelor de mai jos

x	1	2	3	4	5	6	7	8	9	10
$f(x)$	5	4	6	3	2	8	1	9	10	7

x	1	2	3	4	5	6	7	8	9	10
$g(x)$	1	2	2	8	5	6	7	4	9	10

a) Reprezentaţi în acelaşi mod aplicaţiile: *fof, fog, gof, gog;*

b) Stabiliţi care din aplicaţiile iniţiale (f sau g) este bijectivă şi stabiliţi inversa ei.

2.20. Aduceţi exemple de produs cartezian, analogice exemplului

2.21. Demonstraţi că dacă x şi y sunt numere naturale arbitrare numărul $((x+y)(x+y+1))/2$ este la fel un număr natural. Fie A mulţimea punctelor din plan cu coordonatele x şi y, numere naturale. Demonstraţi că aplicaţia $f: A\to N$ definită de $f(x, y) = x + ((x+y)(x+y+1))/2$ este bijectivă.

2.22. Demonstraţi că mulţimea cuvintelor binare este numărabilă.

2.23. Demonstraţi că N^n este numărabilă oricare ar fi n.

2.24. Demonstraţi că mulţimea submulţimilor finite ale mulţimii N este numărabilă.

2.25. Demonstraţi că produsul şi reuniunea a două mulţimi numărabile sunt numărabile.

2.26. Asociem fiecărui cuplu $(x, y) \in N^2$ numărul natural $u = 2^y(2x+1)-1$.

a) Reprezentaţi în baza 2 x şi u, dacă $x = 5$, $y = 3$.

b) Cum poate fi exprimată reprezentarea în baza doi a lui u în caz general?

c) Demonstraţi că aplicaţia care pune în corespondenţă u cuplului (x, y) este o bijecţie între N şi N^2.

2.27. Fie x şi $y \in Z$, iar relaţiile R_1 şi R_2 definite după cum urmează:

$x\ R_1\ y$ dacă $x+y$ este un număr par

$x\ R_2\ y$ dacă $x-y$ este un număr par

Sunt oare aceste două relaţii egale?

2.28. Ce puteţi spune despre proprietăţile relaţiilor de mai jos?

a) $A = \mathbf{R}$ şi $x \mathcal{R} y$ dacă $|x| = |y|$
b) $A = \mathbf{R}$ şi $x \mathcal{R} y$ dacă $\sin^2 x + \cos^2 y = 1$
c) $A = \mathbf{R}$ şi $x \mathcal{R} y$ dacă există p şi q întregi astfel ca $y = px^q$.
d) A este mulţimea punctelor unui plan şi $x \mathcal{R} y$ dacă distanţa dintre x şi y este mai mică de 50 km.

2.29. Definim o relaţie R peste cuvintele limbii române astfel: "cuvântul A este legat de cuvântul B, dacă B este o anagramă a lui A". Demonstraţi că R este o relaţie de echivalenţă. Care este clasa de echivalenţă a cuvânului *rasă*?

2.30. Peste mulţimea numerelor întregi definim relaţia S: "$x \; S \; y$, dacă $x+y$ este par". Demonstraţi, că S este o relaţie de echivalenţă. Descrieţi clasele sale de echivalenţă.

2.31. Peste mulţimea numerelor cuvintelor binare de lungime 7 definim relaţia $\mathcal{R}$: "$m \; \mathcal{R} \; n$ când cuvintele m şi n nu diferă mai mult decât în 5 biţi". Este oare această relaţie o relaţie de echivalenţă?

2.32. Noţiunea de ordine lexicografică peste $\boldsymbol{B}^*$ a fost definită în exemplul 2.20.

a) Demonstraţi că ordinea lexicografică este o relaţie de ordine.
b) Cuvântul *111* are oare un succesor imediat? Este el oare succesor imediat şi al altui cuvânt?
c) Care cuvinte se află între cuvintele *111* şi *1111*?
d) Generalizaţi noţiunea de ordine lexicografică înlocuind $\boldsymbol{B}$ cu o mulţime finită oarecare total ordonată (de exemplu, cele 26 litere ale alfabetului latin).

3. ELEMENTE DE LOGICĂ MATEMATICĂ

Logica matematică reuneşte teoria mulţimilor, teoria algoritmilor, teoria modelelor şi teoria demonstraţiilor. Vom face cunoştinţă cu unele momente legate de **teoria algoritmilor**, care studiază modelele matematice ale operaţiilor mecanice executate de oameni sau dispozitive speciale atunci când rezolvă probleme de masă de acelaşi tip, în capitolul 5. **Teoria modelelor** a apărut prin aplicarea metodelor logicii matematice la algebră, transformându-se într-o disciplină de-sine-stătătoare, care studiază modelele matematice ale teoriilor ştiinţifice, punând în evidenţă legile comune tuturor teoriilor ce se exprimă printr-un limbaj formalizat dat. **Teoria demonstraţiilor**, care reprezintă partea principală a logicii matematice, studiază modelele matematice ale procesului gândirii, structura gândirii, ale raţionamentelor utilizate în matematică. Orice proces de gândire, printre care şi cel utilizat în matematică, este legat de următoarele patru obiecte:

1. Un limbaj L în care se exprimă premisele iniţiale (datele) ale gândirii, diferite momente ale gândirii, rezultatele obţinute prin raţionament. De obicei, L este limba unui popor, îmbogăţită cu termeni şi concepte caracteristice teoriei obiectelor studiate.
2. O clasă K a obiectelor studiate N.
3. Un concept de adevăr al enunţului a în limbajul L privind obiectul studiat $N \in K$.
4. Un proces de elaborare a enunţurilor folosite în raţionament, constând în trecerea de la unele enunţuri, numite **premise**, la un enunţ nou, numit **consecinţă** a enunţurilor iniţiale.

Modelul matematic al procesului gândirii constă din următoarele modele:

- Limbajul formalizat L_f - modelul matematic al limbajului L;
- Modelul matematic N_m al obiectului studiat N;
- Definiţia exactă a conceptului de adevăr a enunţului α din limbajul L_f în modelul N_m;
- Calculul logic sau modelul matematic al trecerii de la premise la consecinţe.

Dintre principalele orientări ale logicii matematice pot fi menţionate logica clasică, logica intuiţionistă, logica polivalentă, logica hibridă, logica fuzzy, etc.

Definirea riguroasă a problemelor legate de ştiinţa calculatoarelor, informatica teoretică şi aplicată este bazată pe principiile logicii matematice. Problemele tehnice privind circuitele logice şi comenzile secvenţiale, majoritatea modelelor matematice utilizate în inteligenţa artificială nu pot fi concepute în afara acestui domeniu.

3.1. Funcţiile algebrei logicii

Vom evidenţia în mod deosebit mulţimea *B* care conţine două elemente, notate de obicei prin *0* şi *1* $B=\{0,1\}$. Aceste elemente nu vor fi considerate numere în sensul obişnuit. Ne vom opri la interpretarea logică: *0* în sens de *nu* sau *fals* şi *1* în sens de *da* sau *adevărat*; (vă amintiţi de programare - *FALS* şi *TRUE*). Într-un caz mai general această interpretare nu este obligatorie - adesea elementele mulţimii *B* pot fi considerate simboluri formale care nu au un sens aritmetic.

Algebra $A = <B, F>$, în care *F* este mulţimea operaţiilor $f: B^n \to B$, $n = 1, 2,...,$ *m* se numeşte **algebra logicii** sau **booleană**, după numele matematicianului şi logicianului englez George Boole (1815 - 1864). Operaţiile $f: B^n \to B$ se numesc **funcţii ale algebrei logicii** sau **funcţii logice**, **funcţii booleene (FB).** Cu alte cuvinte, o funcţie booleană de *n* argumente $f(x_1, x_2,..., x_n)$ are domeniul de valori şi domeniul de definiţie mulţimea $B=\{0,1\}$. Mulţimea tuturor funcţiilor logice de *n* argumente notăm prin $P_2(n)$.

Luând în locul lui *B* o mulţime finită *M* de cardinal *k* de simboluri formale împreună cu toate operaţiile definite pe *M* vom ajunge la **logica polivalentă**. Mulţimea tuturor funcţiilor logice în logica polivalentă se va nota prin $P_k(n)$.

O **algebră booleană** mai poate fi definită ca o **latice complementară şi distributivă** cu axiomele şi proprietăţile respective (v.p. 2.8).

O funcţie logică $f(x_1, x_2,..., x_n)$ poate fi definită cu ajutorul unui tabel care conţine $n+1$ coloane. Primele *n* coloane vor enumera toate combinaţiile posibile ale argumentelor, iar ultima determină valorile posibile ale funcţiei.

Exemplul 3.1. Pentru o funcţie de trei argumente $f(x_1, x_2, x_3)$ putem avea următorul tabel:

Tabelul 3.1. Tabel de adevăr

x_1	x_2	x_3	$f(x_1,x_2,x_3)$	x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	**1**	1	0	0	**1**
0	0	1	**0**	1	0	1	**0**
0	1	0	**0**	1	1	0	**0**
0	1	1	**0**	1	1	1	**1**

Se va mai spune că combinaţia *(000)* este aplicată în *1*, *(001)* - în *0* ş.a.m.d. Combinaţiile au fost enumerate în ordine lexicografică, începând de la *0*, reprezentat în binar prin *(000)* până la *7* - *(111)*. Avem 8 combinaţii posibile în cazul unei funcţii de trei argumente. ◄

Pentru cazul unei funcţii de *n* argumente observăm că combinaţiile posibile ale argumentelor sunt vectori binari de lungime *n*. Este uşor de demonstrat, că

mulţimea tuturor combinaţiilor posibile este $k = 2^n$ (v. teorema 2.1). Din aceleaşi considerente pot exista $2^k = 2^{2^n}$ funcţii booleene distincte. Întradevăr, mulţimea tuturor vectorilor binari de lungime n prin definiţie reprezintă produsul cartezian B^n. Cardinalul acestui produs este $|B| = 2^n$. Analogic, deoarece oricare funcţie booleană de n argumente este un vector binar de lungime $k = 2^n$, există 2^{2^n} funcţii booleene.

Tabelele de tipul lui 3.1 se numesc **tabele de adevăr**.

Există situaţii când pentru unele combinaţii ale valorilor argumentelor valoarea *FB* nu este determinată. **Funcţiile Booleene nedeterminate** pentru una sau mai multe combinaţii ale valorilor argumentelor se numesc **incomplet definite**. În tabelul de adevăr valorile nedefinite le vom indica cu asterisc (*).

Exemplul 3.2. Fie $f(x_1, x_2, x_3)$ o FB dată prin tabelul de adevăr 3.2.

Tabelul 3.2. Exemplu de FB

x_1	x_2	x_3	$f(x_1,x_2,x_3)$	x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	**1**	1	0	0	*
0	0	1	*	1	0	1	**0**
0	1	0	**0**	1	1	0	**0**
0	1	1	**0**	1	1	1	*

Funcţia este nedeterminată pentru combinaţiile *(001)*, *(100)* şi *(111)* ale valorilor argumentelor, ele putând fi aplicate uneori arbitrar în *0* sau *1*. Funcţiile incomplet definite se întâlnesc frecvent în practica comenzilor secvenţiale, evidenţierea situaţiilor nedefinite şi atribuirea unor valori la necesitate fiind extrem de importantă pentru simplificarea lor. ◄

Funcţii booleene de un singur argument pot fi patru:

Tabelul 3.3. FB de un argument

x	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	1	1
1	0	1	0	1

Funcţiile f_0 şi f_3 se numesc constanta *0* si, respectiv, constanta *1*. Valorile lor nu depind de valoarea argumentului. În acest caz se va spune că argumentul este nesemnificativ, redundant sau fictiv. Funcţia f_1 repetă valoarea argumentului x:

vom scrie $f_1(x) = x$. Funcţia $f_2(x)$ se numeşte negaţia lui x (*NONx*) şi se notează $\bar{x}$ (x *barat*) sau $\neg x$.

Exista 16 *FB* de două argumente:

Tabelul 3.4. FB de două argumente

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1
x_1	x_2	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

FB $f_0(x_1,x_2)$ şi $f_{15}(x_1,x_2)$ sunt aceleaşi constante *0* şi *1*, respectiv, numai că în acest caz ambele argumente sunt fictive, iar funcţiile mai pot fi numite degenerate.

Funcţia $f_1(x_1,x_2)$, care aplică combinaţiile celor două argumente în *1* atunci şi numai atunci, când ambele argumente au valoarea *1*, se numeşte **conjuncţie, produs logic** sau **funcţia logică ŞI, (AND)**. Se notează $x_1 \& x_2$, $x_1 x_2$ sau $x_1 \wedge x_2$.

Funcţia $f_7(x_1,x_2)$ are valoarea *1* dacă cel puţin unul din argumente este *1* şi se numeşte **disjuncţie, sumă logică, funcţia logică SAU (OR).** Se notează $x_1 \vee x_2$ sau $x_1 + x_2$.

FB $f_6(x_1,x_2)$ se numeşte **suma modulo 2**, funcţia *SAU-EXCLUSIV* sau *neechivalenţă* şi se notează $f_6(x_1,x_2) = x_1 \oplus x_2$.

Funcţiile $f_3(x_1,x_2)$ şi $f_5(x_1,x_2)$ corespund valorilor argumentelor: $f_3(x_1,x_2) = x_1$ şi $f_5(x_1,x_2) = x_2$ şi se numesc funcţii identitate, iar $f_{10}(x_1,x_2)$ şi $f_{12}(x_1,x_2)$ corespund funcţiilor $f_3(x_1,x_2)$ şi $f_5(x_1,x_2)$ negate.

Funcţia $f_8(x_1,x_2) = \overline{f_7(x_1, x_2)}$ poartă denumirea de **funcţia lui Pierce** şi se notează: $f_8(x_1,x_2) = x_1 \uparrow x_2$. Se mai numeşte această funcţie *NICI* sau *NOR* din cauza că coincide cu $\overline{x_1 + x_2}$.

FB $f_9(x_1,x_2)$ care are valoarea 1 atunci şi numai atunci când valorile argumentelor coincid se numeşte funcţia de **echivalenţă** şi se notează prin $f_9(x_1,x_2) = x_1 \leftrightarrow x_2$ sau $x_1 \sim x_2$.

Funcţia $f_{13}(x_1,x_2)$ are valoarea *0* numai în cazul când x_1 este 1, iar $x_2 = 0$. Ea se numeşte **implicaţie**. Se va mai spune că x_1 **implică** x_2 sau **dacă** x_1 **atunci** x_2 şi vom nota $x_1 \to x_2$. Analogic, funcţia $f_{11}(x_1,x_2)$ este implicaţia lui x_2, *în* x_1: $x_2 \to x_1$.

FB $f_{14}(x_1,x_2)$ este *0* dacă şi numai dacă ambele argumente sunt *1* şi se numeşte funcţia lui **Sheffer** sau *ŞI-NU* (în engleză *NAND*). Se notează $f_{14}(x_1,x_2) = x_1|x_2$ sau $\overline{x_1 x_2}$.

Observăm, că în cazul funcţiilor de un argument jumătate din FB sunt degenerate (cu argumente fictive). În cazul a două argumente din 16 funcţii booleene şase sunt cu argumente fictive. Odată cu creşterea lui *n* numărul funcţiilor degenerate descreşte, tinzând către *0*, când *n* tinde spre infinit.

3.2. Transformări echivalente şi decompoziţia FB

3.2.1. Formule echivalente

În 2.6 am numit superpoziţie a funcţiilor $f_1, f_2,...,, f_n$ funcţia f obţinută prin substituiri şi redenumiri de argumente în aceste funcţii, iar prin formulă înţelegem expresia care descrie această superpoziţie. Vom concretiza noţiunea de formulă pentru funcţiile logice, introducând noţiunea de formulă peste $\Omega = \{f_1, f_2,...,, f_m,...\}$, Ω fiind o mulţime de funcţii logice date.

Considerăm *formulă* peste Ω toate expresiile care conţin numai simboluri de variabile, simboluri de funcţii şi paranteze.

Exemplul 3.3. Poate fi considerată formulă expresia: $f=((x_1 \oplus x_3)(x_2 \to x_4))$. Valoarea funcţiei dată de o formulă poate fi calculată, cunoscând valorile argumentelor şi tabelele de adevăr ale funcţiilor logice elementare (v. tab.3.4). Dacă $x_1=1$, $x_2=1$, $x_3=0$ şi $x_4=0$, avem $x_1 \oplus x_3=1$, $x_2 \to x_4=0$ şi $f=0$. ◄

Deci, o formulă, pune în corespondenţă fiecărui set de valori ale argumentelor o anumită *valoare de adevăr* şi poate servi, împreună cu tabelele de adevăr, drept metodă de definire şi calculare a valorilor funcţiilor logice. Se mai spune că o formulă reprezintă sau **realizează** o funcţie logică. Calculând valorile *FB* pentru toate 2^n combinaţii ale argumentelor restabilim tabelul de adevăr al acestei funcţii. Însă, spre deosebire de tabelele de adevăr, o funcţie logică poate fi realizată prin mai multe formule. Cu alte cuvinte, dacă între mulţimea tabelelor de adevăr şi mulţimea funcţiilor logice există o corespondenţă biunivocă, alta este situaţia cu formulele: o FB poate fi prezentată printr-o infinitate de formule. De exemplu, funcţia Pierce $f_8(x_1,x_2) = x_1 \uparrow x_2$ mai poate fi realizată prin formulele

$\overline{x_1 + x_2}$ sau $\overline{x_1}\,\overline{x_2}$, iar funcţia Sheffer $f_{14}(x_1,x_2) = x_1|x_2$ prin $\overline{x_1} + \overline{x_2}$ sau $\overline{x_1 x_2}$.

Formulele, care realizează aceeaşi funcţie logică se numesc ***echivalente***. Echivalenţa a două formule se va nota prin simbolul = sau ≡. Metoda generală de stabilire a echivalenţei a două formule constă în construirea tabelelor lor de adevăr şi compararea acestora. Altă metodă, denumită metoda transformărilor echivalente, presupune transformarea uneia dintre formule (sau a ambelor) până se ajunge la o formă evident comună. Un interes aparte îl prezintă procedeele de reprezentare a funcţiilor booleene prin aşa numitele forme normale sau canonice şi forme minimale.

3.2.2. Decompoziţia funcţiilor booleene

Notăm $x^0 = \overline{x}$, $x^1 = x$. Este simplu să ne convingem (de exemplu prin construirea tabelului de adevăr), că pentru un parametru α egal cu *0* sau cu *1*, reieşind din notaţia introdusă, avem

$$x^\alpha = 1 \text{ când } x = \alpha$$
şi
$$x^\alpha = 0, \text{ dacă } x \neq \alpha \qquad (3.1)$$

Teorema 3.1. Orice funcţie booleană $f(x_1, x_2,..., x_n)$ poate fi pusă sub forma:

$$f(x_1,..., x_m, x_{m+1},..., x_n) = \bigvee_{\alpha_1 \ldots \alpha_m} x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_m^{\alpha_m} f(\alpha_1,..., \alpha_m, x_{m+1},..., x_n) \qquad (3.2)$$

în care $m \leq n$, iar disjuncţiile se vor lua pentru toate 2^m seturi, formate din variabilele $x_1, x_2,..., x_m$.

Relaţia (3.2) se numeşte ***decompoziţia*** funcţiilor booleene pentru variabilele $x_1, x_2,..., x_m$.

Demonstrăm prin introducerea unui set arbitrar de valori $(\sigma_1,...,\sigma_m,\sigma_{m+1},...,\sigma_n)$ ale argumentelor în ambele părţi ale relaţiei (3.2). Deoarece $x^\alpha = 1$ numai în cazul în care $x = \alpha$, dintre toate cele 2^m conjuncţii ale părţii drepte a relaţiei (3.2) numai una este egală cu *1* (restul sunt *0*), cea pentru care $\alpha_1 = \sigma_1$, $\alpha_2 = \sigma_2$,..., $\alpha_m = \sigma_m$. Obţinem

$f(\sigma_1,\sigma_2,...,\sigma_n) = \sigma_1^{\alpha_1} \sigma_2^{\alpha_2} \ldots \sigma_m^{\alpha_m} f(\sigma_1,\sigma_2,...,\sigma_m,\sigma_{m+1},...,\sigma_n) = f(\sigma_1,\sigma_2,...,\sigma_n)$,

şi, deoarece setul de valori a fost luat arbitrar, teorema este demonstrată.

Pentru $m = 1$ avem decompoziţia FB pentru o singură variabilă (de exemplu x_1):

$f(x_1, x_2,..., x_n) = \overline{x}_1 f(0, x_2,..., x_n) \vee x_1 f(1, x_2,..., x_n)$.

Prezintă interes deosebit cazul $m = n$ - decompoziţia FB pentru toate variabilele $x_1, x_2,..., x_n$. Avem:

$$f(x_1, x_2,..., x_n) = \bigvee_{\alpha_1 \dots \alpha_n} x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n} \; f(\alpha_1,..., \alpha_n) = \bigvee_{f(\alpha_1 \dots \alpha_n)=1} x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}, \qquad (3.3)$$

în care disjuncţiile se iau pentru combinaţiile argumentelor aplicate de f în *1*, iar conjuncţiile respective se numesc *elementare, termeni canonici conjunctivi* sau *termeni minimali (mintermi).*

Definiţie. Numim ***formulă booleană*** orice formulă care în afară de variabile şi paranteze conţine doar funcţiile disjuncţie, conjuncţie şi negaţie.

Teorema 3.2. Orice funcţie logică poate fi prezentată printr-o formulă booleană.

Demonstraţia este imediată şi reiese din teorema 3.1, relaţia (3.3) şi ultima definiţie. Constanta *0* poate fi reprezentată prin formula $x\bar{x}$.

3.2.3. Algebra booleană. Proprietăţile operaţiilor booleene

Algebra $A = \langle P_2, \{\vee, \wedge, \bar{\ }\}\rangle$, suportul căreia este mulţimea tuturor funcţiilor logice, iar în calitate de operaţii sunt luate disjuncţia, conjuncţia şi negaţia, se numeşte algebra booleană a funcţiilor logice. Operaţiile acestei algebre se mai numesc operaţii booleene.

Pentru operaţiile booleene s-a convenit a se considera următoarea **ordine de îndeplinire**: mai întâi vor fi îndeplinite toate operaţiile de negaţie (cea mai înaltă prioritate), apoi conjuncţiile, iar cea mai mică prioritate o are disjuncţia. La necesitate pot fi utilizate parantezele.

Este simplu de stabilit (prin construirea tabelelor de adevăr, de exemplu), că operaţiile booleene posedă proprietăţile:

- asociativitate
  $x_1(x_2x_3) = (x_1x_2)x_3;$
  $x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3;$ (3.4)
- comutativitate
  $x_1x_2 = x_2x_1;$
  $x_1 \vee x_2 = x_2 \vee x_1;$ (3.5)
- distributivitate
  $x_1(x_2 \vee x_3) = x_1x_2 \vee x_1x_3;$
  $x_1 \vee (x_2x_3) = (x_1 \vee x_2)(x_1 \vee x_3);$ (3.6)
- idempotenţă
  $xx = x;$
  $x \vee x = x;$ (3.7)
- principiul involuţiei
  $\bar{\bar{x}} = x;$ (3.8)
- operaţii cu constante
  $x\&1=x;\ x\&0=0;$
  $x \vee 1=1;\ x \vee 0=x;\ \bar{0}=1;\quad \bar{1}=0;$ (3.9)

- legile lui de Morgan $\overline{x_1 x_2} = \overline{x_1} + \overline{x_2}$;
  $\overline{x_1 + x_2} = \overline{x_1}.\overline{x_2}$ (3.10)
- contradicţie $x\overline{x} = 0$; (3.11)
- legea terţului exclus $x + \overline{x} = 1$. (3.12)

Operaţiile booleene posedă proprietăţile (3.4) - (3.12) chiar dacă argumentele x_i sunt la rândul lor funcţii sau vor fi obţinute în rezultatul unor calcule. Ca rezultat, aceste proprietăţi se mai numesc relaţii de echivalenţă, iar transformările care pot fi operate cu ajutorul lor se numesc transformări echivalente. Relaţiile de echivalenţă (3.4) - (3.12), împreună cu unele relaţii suplimentare, obţinute din primele, prezintă un interes deosebit în tratarea problemei de echivalenţă a formulelor, datorită eficacităţii mai mari în comparaţie cu tabelele de adevăr.

Mai aducem câteva relaţii de echivalenţă suplimentare (demonstraţiile se bazează pe proprietăţile de mai sus şi se propun ca exerciţiu):

- absorbţie $x \vee xy = x$; $x(x \vee y) = x$;
- alipire $xy \vee x\overline{y} = x$;
- alipire generalizată $xz \vee y\overline{z} \vee xy = xz \vee y\overline{z}$

3.3. Forme canonice

3.3.1. Forma canonică disjunctivă

Decompoziţia (3.3) se numeşte ***formă canonică disjunctivă (FCD)*** sau ***formă normal disjunctivă perfectă (FNDP)*** a funcţiei $f(x_1, x_2,..., x_n)$. *FCD* conţine tot atâtea conjuncţii elementare câte unităţi sunt în tabelul de adevăr al funcţiei date: fiecărui set de valori ale argumentelor pentru care $f = 1$ îi corespunde o conjuncţie în care x_i este negat, dacă $\alpha=0$ şi fără negaţie, dacă $\alpha=1$. Cu alte cuvinte, există o corespondenţă biunivocă între tabelul de adevăr al funcţiei $f(x_1, x_2,..., x_n)$ şi *FCD* a acestei funcţii. Unica funcţie booleană care nu posedă *FCD* este constanta *0*.

Relaţia (3.3) conduce la un algoritm simplu de elaborare a FCD pentru o FB arbitrară:

1º. Se construieşte tabelul de adevăr al funcţiei;

2º. Pentru fiecare combinaţie de valori ale argumentelor aplicate în 1 se scriu termenii canonici conjunctivi în care argumentul x_i este luat ca atare sau negat după cum valoarea lui în combinaţia respectivă este 1 sau 0;

3º. Se reunesc toţi termenii canonici conjunctivi, obţinuţi la pasul precedent, cu operaţia disjuncţie.

Dacă pentru un termen canonic conjunctiv fiecare variabilă va fi înlocuită prin *1* sau *0*, după cum este prezentă în acest TCC (fără negaţie sau negată, respectiv) vom avea reprezentarea binară a unei conjuncţii elementare. De exemplu, reprezentarea binară a conjuncţiei elementare $\overline{x_1}x_2\overline{x_3}\overline{x_4}$ este 0100. Reprezentarea binară poate fi transformată în zecimală, adică conjuncţia elementară de mai sus poate fi considerată *4* zecimal. Prin utilizarea acestor notaţii FCD este mult mai compactă. Notând prin Σ disjuncţia unui set de TCC, forma canonică disjunctivă a unei funcţii booleene oarecare poate avea forma $f(x_1,x_2,...,x_n) = \Sigma(0,2,4,...)$.

3.3.1. Forma canonică conjunctivă

Reprezentarea unei *FB* se poate face şi sub o altă formă, numită *forma canonică* ***conjunctivă***.

Pentru aceasta numim *numărul combinaţiei* numărul *i* ataşat unei combinaţii de valori $(\sigma_1,\sigma_2,...,\sigma_n)$ conform relaţiei:

$$i = \sigma_1 2^{n-1} + \sigma_2 2^{n-2} + ... + \sigma_n 2^0$$

şi introducem funcţia $S_i(x_1, x_2,..., x_n)$ definită astfel:

$$S_i = \begin{cases} 0, \text{ dacă numărul combinaţiei este } i \\ 1, \text{ în caz contrar.} \end{cases} \qquad (3.13)$$

Această funcţie se numeşte *funcţia caracteristică a lui zero* sau *constituentul lui zero.*

Poate fi demonstrată

Teorema 3.3. Orice *FB* poate fi scrisă sub următoarea formă

$$f(x_1, x_2,..., x_n) = \&_{i \in M_0} S_i \qquad (3.14)$$

unde M_0 este mulţimea combinaţiilor valorilor argumentelor pentru care funcţia $f(x_1, x_2,..., x_n)$ ia valoarea *0*.

Demonstraţie. Se consideră un set arbitrar de valori ale argumentelor $(\sigma_1,\sigma_2,...,\sigma_n)$. Sunt posibile două cazuri distincte: funcţia să aplice acest set de valori a) în *0* şi b) în *1*.

Dacă valoarea funcţiei este 0, atunci în partea dreaptă a relaţiei (3.14) se află funcţia S_{ik} al cărei indice i_k corespunde numărului setului considerat. În acest caz, conform cu (3.13), pentru setul respectiv de valori ale argumentelor S_{ik} va fi egală cu *0*. Conform proprietăţii $x\&0 = 0$ partea dreaptă a relaţiei (3.14) va fi egală cu 0. Dacă însă pentru setul considerat valoarea funcţiei este 1, atunci conform formulării teoremei, printre funcţiile S_{ik} din (3.14) nu va fi nici una pentru care indicele să coincidă cu numărul combinaţiei. În acest caz, conform cu (3.13), toţi membrii conjuncţiei din partea dreaptă a relaţiei (3.14) vor fi egali cu 1, deci partea dreaptă va fi egală cu 1. Deoarece s-a demonstrat că ambele părţi ale relaţiei (3.14) sunt identice pentru un set **arbitrar** de valori ale argumentelor rezultă că este adevărată pentru orice alt set de valori. Teorema este demonstrată.

Pentru a stabili expresiile funcţiilor S_{ik} se consideră expresia booleană

$$x_1^{\alpha_1} \vee x_2^{\alpha_2} \vee \ldots \vee x_n^{\alpha_n}. \qquad (3.15)$$

Conform relaţiei (3.1) funcţia (3.15) este *0* atunci şi numai atunci, când $x_1 \neq \alpha_1$, $x_2 \neq \alpha_2$,..., $x_n \neq \alpha_n$, fiind *1* pentru toate celelalte cazuri. Aplicând (3.13) pentru funcţia constituentul lui *0*, rezultă:

$$S_i(x_1, x_2, \ldots, x_n) = x_1^{\overline{\alpha_1}} \vee x_2^{\overline{\alpha_2}} \vee \ldots \vee x_n^{\overline{\alpha_n}}$$

cu condiţia ca $i = \alpha_1 2^{n-1} + \alpha_2 2^{n-2} + \ldots + \alpha_n 2^0$.

Deci, orice *FB* poate fi descrisă printr-o expresie de forma

$$f(x_1, x_2, \ldots, x_n) = \&_0 \left(x_1^{\overline{\alpha_1}} \vee x_2^{\overline{\alpha_2}} \vee \ldots \vee x_n^{\overline{\alpha_n}} \right) \qquad (3.16)$$

unde prin $\&_0$ s-a notat faptul că se consideră conjuncţia termenilor disjunctivi (3.15) pentru care funcţia ***f*** ia valoarea ***0***.

Reprezentarea *FB* sub forma (3.16) se numeşte ***forma canonică conjunctivă (FCC)***, iar termenii (3.15) cu condiţia $x_i \neq \alpha_i$ ***termeni canonici disjunctivi (TCD)***, ***termeni maximali*** sau ***maxtermi.***

Relaţia (3.16) permite stabilirea algoritmului realizării FCC dacă se cunoaşte tabelul de adevăr al funcţiei, care conţine următorii paşi:

1. *Din tabelul de adevăr al funcţiei se consideră toate combinaţiile pe care funcţia le aplică în 0;*
2. *Se scriu TCD, care corespund acestor combinaţii. În expresia TCD argumentul x_i intră direct sau negat după cum în combinaţia considerată are valoarea 0 sau 1;*
3. *TCD obţinuţi la pasul 2 se reunesc prin semnul conjuncţiei.*

Formele canonice sunt unice pentru o funcţie logică complet definită. Este recomandată FCD în cazul în care tabelul de adevăr al funcţiei conţine un număr mai mic de valori 1, decât de valori 0, şi FCC, în caz contrar.

3.4. Alte forme de reprezentare a funcţiilor booleene

În afară de reprezentarea FB cu ajutorul tabelelor de adevăr şi a formelor canonice mai sunt cunoscute alte forme cum ar fi diagramele Karnaugh, schemele logice, diagramele de timp, etc.

3.4.1. Diagrame Karnaugh

Diagramele Karnaugh au fost concepute pentru simplificarea FB şi reprezintă un tablou bidimensional, care pentru o funcţie de n argumente conţine 2^p linii şi 2^q coloane, iar $p+q = n$. Dacă n este par, atunci $p = q$ şi $p = q+1$, în caz contrar. De obicei, pot fi utilizate cu succes pentru $n = 4, 5$, mai dificil pentru $n = 6$ şi mai mare. Într-un tabel bidimensional Karnaugh titlurile coloanelor şi liniilor sunt formate din combinaţiile de valori posibile dispuse în cod Gray (binar reflectat). Acest cod fiind continuu şi ciclic asigură relaţia de adiacenţă între câmpurile diagramei (numim adiacente două câmpuri dacă titlurile lor diferă printr-un singur rang).

Exemplul 3.4. Pentru FB de 4 argumente cu tabelul de adevăr 3.5 diagrama Karnaugh este prezentată în figura 3.1.◄

Tabelul 3.5. Tabelul de adevăr al unei FB

x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$	x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	1	0	1	1	1
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	1	1	1
10	1	1	1	1

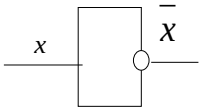
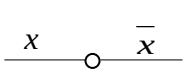
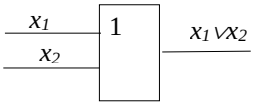
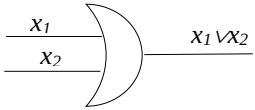
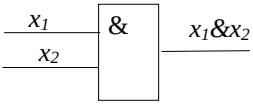
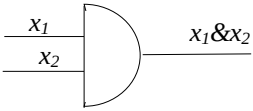
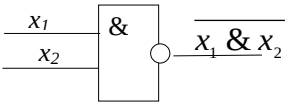
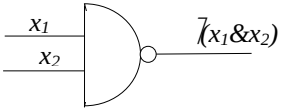
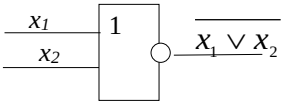
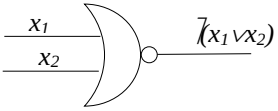
Fig. 3.1. Diagramă Karnaugh

Combinaţiile valorilor argumentelor x_1 şi x_2 sunt dispuse în partea superioară a diagramei, iar cele ale argumentelor x_3 şi x_4 vertical în partea stângă. La intersecţia unei coloane şi a unei linii este câmpul diagramei în care se trece *0* sau *1*, conform valorii funcţiei în tabelul de adevăr.

3.4.2. Circuite logice

Circuitul logic (sau schema logică) este o reprezentare grafică a FB, obţinută prin adoptarea unor semne convenţionale pentru

Tab.3.6. Reprezentarea FB prin scheme logice

Denumirea funcţiei	Reprezentarea grafică	
	Standardele fostei URSS	Standarde internaţionale
Negaţia $f(x) = \overline{x}$		
Disjuncţia $f(x_1,x_2) = x_1 \vee x_2$		
Conjuncţia $f(x_1,x_2) = x_1 \wedge x_2$		
Pierce $f(x_1,x_2) = x_1 \uparrow x_2$		
Sheffer $f(x_1,x_2) = x_1/x_2$		

operaţiile logice. Simbolurile grafice adoptate constituie o reprezentare a circuitelor logice, care materializează funcţiile logice elementare. **Prin**

implementarea unei funcţii logice se înţelege realizarea ei cu ajutorul circuitelor de bază. Unele dintre cele mai des utilizate semne grafice pentru FB elementare sunt prezentate în tab. 3.6.

Exemplul 3.5. Să se reprezinte prin schemă logică funcţia $f(x_1, x_2, x_3) = \overline{\bar{x}_1 x_2 \bar{x}_3 \vee \overline{x_1 x_2 x_3}}$.

Utilizând figurile grafice din tab.3.6 schema logică este prezentată în fig. 3.2.

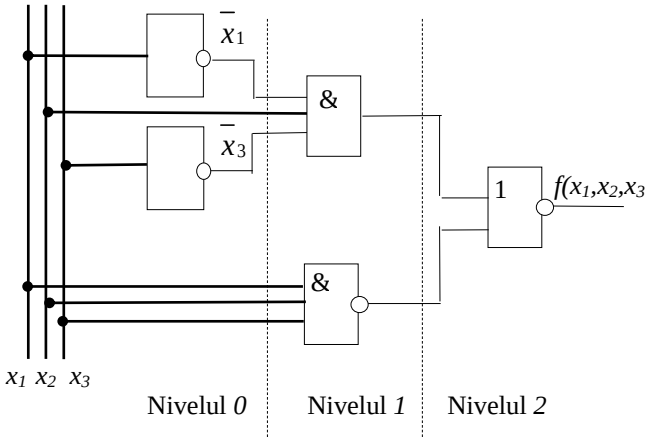


Fig.3.2. Schemă logică

Pot fi indicate şi nivelele logice - mulţimea elementelor fizice care operează simultan. Aici avem nivelele logice 0, 1 şi 2. ◄

3.4.3. Diagrame temporale

Dacă vom reprezenta grafic argumentele x_i ca funcţii de timp, ataşând valorilor *0* şi *1* valori distincte (de exemplu, valorii *0* un nivel coborât, iar valorii *1* un nivel ridicat), astfel ca să existe o diferenţiere evidentă a acestor nivele, şi vom face acelaşi lucru şi cu valorile funcţiei vom obţine *reprezentarea unei FB prin diagrame în timp (diagrame temporale)*, reprezentare extrem de utilă în studiul sistemelor secvenţiale în evoluţia cărora intervine timpul.

Exemplul 3.6. Diagrama temporală a funcţiei $f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_3 \vee x_1 \bar{x}_2$ are forma prezentată în fig. 3.3.

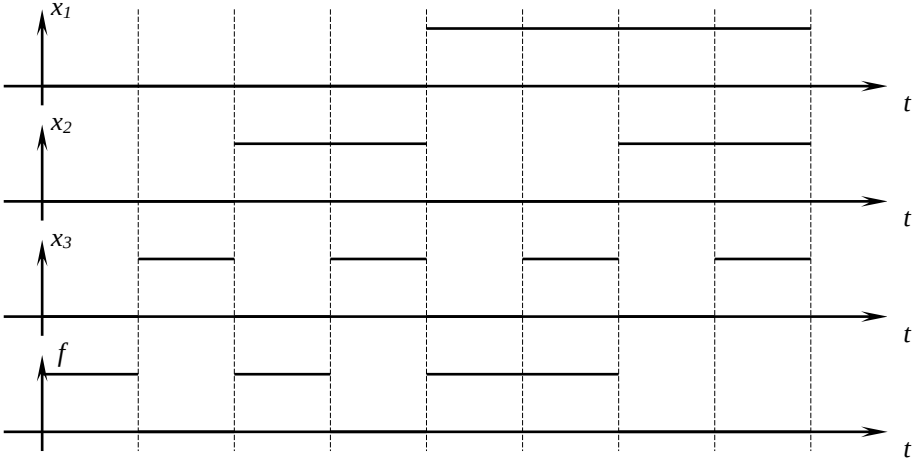


Fig.3.3. Exemplu de diagramă temporală ◄

3.5. Sisteme complete de funcţii booleene

Definiţie. Numim *sistem complet* de funcţii booleene (bază) în clasa $\mathcal{R}$ sistemul $S=(f_1, f_2,..., f_k)$, dacă orice funcţie $\varphi \in \mathcal{R}$ poate fi reprezentată prin superpoziţia funcţiilor din acest sistem.

În calitate de $\mathcal{R}$ poate fi luată mulţimea $P_2(n)$. În această clasă există un sistem complet şi anume toate cele 2^{2^n} funcţii de n argumente. Conform teoremei 3.2, orice funcţie logică de n argumente de asemenea poate fi reprezentată utilizând doar funcţiile negaţie, disjuncţie şi conjuncţie. Deci, în aceeaşi clasă pot exista mai multe sisteme complete cu un număr diferit de funcţii. Un interes deosebit prezintă problema alegerii bazei, care conţine un număr minim de funcţii.

Numim **bază minimală** (sistem complet minimal) un sistem complet arbitrar de funcţii booleene $(f_1, f_2,..., f_k)$, care odată cu eliminarea oricărei funcţii aparţinând sistemului devine incomplet.

Pentru a stabili completitudinea unui sistem oarecare de FB este suficient să se arate că funcţiile sistemului considerat pot reprezenta funcţiile sistemului ($\vee$, $\wedge$, $\bar{}$). Pot fi demonstrate teoremele:

Teorema 3.4. Sistemul ($\vee$, $\bar{}$) este un sistem complet în clasa $P_2(n)$.

Teorema 3.5. Sistemul ($\wedge$, $\bar{}$) este un sistem complet în clasa $P_2(n)$.

Teorema 3.6. Funcţia lui Pierce ($\uparrow$) formează în clasa $P_2(n)$ un sistem complet.

Teorema 3.7. Funcţia lui Sheffer ($/$) formează în clasa $P_2(n)$ un sistem complet.

Pentru a demonstra, de exemplu, teorema 3.5 este suficient să se arate că funcţia disjuncţie poate să fie reprezentată prin funcţiile negaţie şi conjuncţie. Utilizând principiul involuţiei şi una din relaţiile lui De Morgan, avem

$$f(x_1, x_2,..., x_n) = x_1 \vee x_2 \vee ... \vee x_n = \overline{\overline{x_1 \vee x_2 \vee ... \vee x_n}} = \overline{\overline{x_1}\,\overline{x_2}...\overline{x_n}}$$

ceea ce trebuia demonstrat.

Analogic se demonstrează teorema 3.4. Am stabilit că sistemul ($\vee, \wedge, \bar{}$) este redundant. Una din funcţii (disjuncţia sau conjuncţia) poate fi eliminată, sistemul rămânând complet.

Pentru a demonstra teorema 3.6 vom arăta că funcţia lui Pierce poate reprezenta sistemul ($\vee, \bar{}$). Negaţia se poate scrie astfel:

$$\bar{x} = \overline{x \vee x} = x \uparrow x.$$

Funcţia conjuncţie poate fi exprimată în modul următor:

$$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{x_1 \uparrow x_2} = (x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2).$$

Analogic demonstrăm şi teorema 3.7.

Teoremele 3.6 şi 3.7 prezintă un interes deosebit datorită numărului minim posibil de elemente care formează baza**: putem utiliza un singur tip de circuit pentru materializarea oricărei funcţii booleene**. În acest context este importantă trecerea de la FCD sau FCC la forme cu funcţii Pierce (SAU-NU) sau Sheffer (ŞI-NU), trecere denumită *implementare*.

3.6. Minimizarea funcţiilor booleene

Problema reprezentării funcţiilor booleene prin sisteme complete care conţin un număr minim de funcţii elementare vizează posibilitatea folosirii unui număr cât mai redus de tipuri de circuite logice pentru materializarea FB considerate. Există şi un alt aspect al problemei - cel care priveşte utilizarea unui număr cât mai mic de circuite standard. Teoretic această problemă se reflectă în simplitatea funcţiilor booleene. Este evident că formele canonice sunt departe de a fi cele mai simple. Obţinerea unor forme mai simple poate fi realizată prin metoda transformărilor echivalente utilizând relaţiile p.3.2. Însă simplitatea finală depinde de măiestria şi experienţa cercetătorului, mai mult - nu există siguranţa că forma obţinută este cea mai simplă. Din această cauză au fost căutate metode sistematice pentru obţinerea expresiilor minimale a FB.

3.6.1. Metoda lui Quine

Numim *termen normal conjunctiv (TNC)* conjuncţia $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_k^{\alpha_k}$ ($k \le n$), în care fiecare variabilă se întâlneşte numai o singură dată. Numărul literelor unui termen normal conjunctiv numim *rangul* termenului, iar disjuncţia *TNC* - formă normal disjunctivă (FND). Reieşind din aceste definiţii putem spune că *FCD* a unei *FB* de *n* argumente este *FND* la care toţi termenii sunt de rang *n* (forma normală cea mai complexă).

Definiţie. Forma normal disjunctivă care conţine cel mai mic număr de litere $x_i^{\alpha_i}$ în comparaţie cu toate celelalte *FND* ale unei *FB* date numim *formă disjunctivă minimă (FDM)*.

Definiţie. Numim *implicanţi primi* ai unei FB de *n* argumente termenii conjunctivi de forma $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_k^{\alpha_k}$ ($k \le n$) care implică funcţia fără a se putea elimina vre-o variabilă (*TNC* de rang minim care implică funcţia). De exemplu, dacă pentru $f(x_1,x_2,x_3,x_4)$ avem

$x_1 \overline{x_2}\, x_3 \overline{x_4} \to f(x_1,x_2,x_3,x_4)$	$x_1 \overline{x_2}\, x_3 \overline{x_4}$ este implicant al $f(x_1,x_2,x_3,x_4)$
$x_1 x_3 \overline{x_4} \to f(x_1,x_2,x_3,x_4)$	$x_1 x_3 \overline{x_4}$ este implicantul funcţiei $f(x_1,x_2,x_3,x_4)$
$x_1 x_3 \to f(x_1,x_2,x_3,x_4)$	$x_1 x_3$ este implicantul funcţiei $f(x_1,x_2,x_3,x_4)$
$x_1 \to f(x_1,x_2,x_3,x_4)$	x_1 nu este implicantul funcţiei $f(x_1,x_2,x_3,x_4)$
$x_3 \to f(x_1,x_2,x_3,x_4)$	x_3 nu este implicantul funcţiei $f(x_1,x_2,x_3,x_4)$

atunci x_1x_3 este un implicant prim al funcţiei $f(x_1,x_2,x_3,x_4)$.

Implicanţii primi pot fi determinaţi plecând de la *FCD* prin aplicarea sistematică la câte doi termeni conjunctivi care se deosebesc printr-un singur rang (sunt adiacenţi) proprietatea de alipire parţială (v.p. 3.2)

$$A x_i \vee A \overline{x_i} = A \qquad (3.17)$$

Adesea, în rezultatul efectuării operaţiei de alipire parţială, pot apărea termeni normal disjunctivi în repetare sau asupra cărora poate fi executată operaţia absorbţie. Primii, conform proprietăţii idempotenţă se vor scrie o singură dată, pentru cei de categoria a doua se va opera absorbţia.

Teorema 3.8. Executând asupra *FCD* a unei *FB* toate operaţiile posibile de alipire parţială şi de absorbţie obţinem disjuncţia implicanţilor primi.

Demonstraţie. Conform teoremei are loc relaţia

$$f(x_1,x_2,\ldots,x_n) = \bigvee_k \varphi_k, \qquad (3.18)$$

în care φ_k este setul respectiv de implicanţi primi. Această relaţie trebuie să fie adevărată şi atunci când $f(x_1,x_2,\ldots,x_n)=0$ şi pentru cazul în care $f(x_1,x_2,\ldots,x_n) = 1$. În primul caz toţi implicanţii primi sunt egali cu *0*, altfel am avea valoarea *0*

pentru funcţia $\varphi_i \to f(x_1,x_2,...,x_n)$, unde $\varphi_i \neq 0$ (deci φ_i nu ar fi implicantul funcţiei $f(x_1,x_2,...,x_n)$, v.p.3.2). În cel de-al doilea caz va exista cel puţin un implicant prim $\varphi_j = 1$, drept rezultat partea dreapta a relaţiei (3.18) va fi ca şi partea stângă egală cu *1*.

Expresia (3.18) se numeşte *formă disjunctivă prescurtată (FDP)*. În *FDP* există în caz general implicanţi primi de prisos (redundanţi, care implică suplimentar funcţia), deci FDP nu este minimă. Implicanţii primi strict necesari (obţinuţi după eliminarea implicanţilor redundanţi) se numesc *implicanţi esenţiali.* Disjuncţia implicanţilor esenţiali conduce la *FDM*. În concluzie, putem afirma că minimizarea unei *FB* pornind de la *FCD* presupune următorii doi paşi:

1º. determinarea formei disjunctive prescurtate (3.18);
2º. alegerea implicanţilor esenţiali.

Implicanţii esenţiali pot fi aleşi construind un tabel special, numit *tabelul implicanţilor primi* sau *tabel de acoperire.* Fiecare linie în acest tabel corespunde unui implicant prim, iar fiecare coloană unui *TCC* din *FCD* iniţială. Vom spune că un implicant prim se află în relaţia de acoperire cu un *TCC*, dacă el se conţine în acesta. Se va construi matricea acestei relaţii binare (la intersecţia liniei i cu coloana j se va pune *1*, dacă implicantul prim cu numărul i se află în relaţia de acoperire cu *TCC* cu numărul j, şi *0* în caz contrar.) Vom alege cel mai mic număr de implicanţi primi strict necesari (esenţiali) pentru ca să fie acoperiţi toţi *TCC*.

Fie funcţia $f(x_1,x_2,x_3,x_4)$ definită prin *FCD*:

$$f(x_1,x_2,x_3,x_4) = \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\,\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\,\overline{x_3}\,x_4 \vee x_1\overline{x_2}\,\overline{x_3}\,x_4 \vee x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}.$$

Să se determine forma disjunctivă minimă.

1º. Determinăm *FDP* evidenţiind toţi implicanţii primi:

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\,\overline{x_3}\,\overline{x_4} = \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} = \overline{x_2}\,\overline{x_3}\,\overline{x_4},$$
$$\overline{x_1}\,x_2\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\overline{x_3}\,x_4 = \overline{x_1}\,x_2\overline{x_3},$$
$$\overline{x_1}\,x_2\overline{x_3}\,x_4 \vee x_1x_2\overline{x_3}\,x_4 = x_2\overline{x_3}\,x_4,$$
$$x_1x_2\overline{x_3}\,x_4 \vee x_1\overline{x_2}\,\overline{x_3}\,x_4 = x_1\overline{x_3}\,x_4,$$
$$x_1\overline{x_2}\,\overline{x_3}\,x_4 \vee x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} = x_1\overline{x_2}\,\overline{x_3}.$$

Deoarece alipiri parţiale pentru termenii normali de rang 3 nu se pot opera, avem următoarea formă disjunctivă prescurtată:

$$f(x_1,x_2,x_3,x_4)= \overline{x_1}\,\overline{x_3}\,\overline{x_4} \vee \overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\,\overline{x_3} \vee x_2\,\overline{x_3}\,x_4 \vee x_1\,\overline{x_3}\,x_4 \vee x_1\,\overline{x_2}\,\overline{x_3}$$

2º. Construim tabelul de acoperire:

Tabelul 3.7. Tabel de acoperire

Implicanţii primi	**Termenii canonici conjunctivi** (în binar)					
	0000	*0100*	*0101*	*1101*	*1001*	*1000*
$\overline{x_1}\,\overline{x_3}\,\overline{x_4}$	*1*	*1*	*0*	*0*	*0*	*0*
$\overline{x_2}\,\overline{x_3}\,\overline{x_4}$	*1*	*0*	*0*	*0*	*0*	*1*
$\overline{x_1}\,x_2\,\overline{x_3}$	*0*	*1*	*1*	*0*	*0*	*0*
$x_2\,\overline{x_3}\,x_4$	*0*	*0*	*1*	*1*	*0*	*0*
$x_1\,\overline{x_3}\,x_4$	*0*	*0*	*0*	*1*	*1*	*0*
$x_1\,\overline{x_2}\,\overline{x_3}$	*0*	*0*	*0*	*0*	*1*	*1*

Avem două posibilităţi de alegere:

$$f(x_1,x_2,x_3,x_4) = \overline{x_1}\,\overline{x_3}\,\overline{x_4} \vee x_2\,\overline{x_3}\,x_4 \vee x_1\,\overline{x_2}\,\overline{x_3}\ ,$$

$$f(x_1,x_2,x_3,x_4) = \overline{x_1}\,\overline{x_3}\,\overline{x_4} \vee \overline{x_1}\,x_2\,\overline{x_3} \vee x_1\,\overline{x_3}\,x_4,$$

alegerea făcându-se la opţiune. Concluzionăm, că o *FB* poate avea mai multe forme minime.

3.6.2. Metoda Quine – McCluskey

Metoda prezentată mai sus poartă numele lui Quine, care a propus-o, şi are un neajuns evident, datorat necesităţii comparării la primul pas a tuturor perechilor de termeni (complexitatea creşte în mod factorial). Dar aceasta nu este necesar, deoarece operaţia de alipire parţială poate fi executată doar dacă doi termeni se deosebesc într-un singur bit. McCluskey a propus să se transcrie în binar *TCC* şi să se împartă pe grupe după numărul de biţi 1. Vom avea grupa *0*, grupa *1*, etc. Alipirile parţiale pot avea loc numai pentru elementele grupelor vecine, deoarece

aceste grupe diferă între ele cu un singur bit *1*. În locul variabilelor eliminate la alipire se trece o liniuţă.

Metoda Quine-McCluskey presupune îndeplinirea paşi lor:

1. Ordonarea echivalenţilor binari ai TCC, corespunzători valorilor 1 ale FB, pe nivele începând cu nivelul 0;
2. Determinarea implicanţilor primi prin comparaţii succesive ale echivalenţilor binari, aparţinând nivelelor adiacente;
3. Determinarea tabelului de acoperire al funcţiei;
4. Calculul formal de determinare a tuturor soluţiilor funcţiei.

Exemplul 3.7. [4] Fie funcţia $f(x_1,x_2,x_3,x_4) = \Sigma(0,1,2,3,4,7,8,11,12,13,15)$.

1. Ordonarea pe nivele

Nivelele	Echivalentul binar	Echivalentul zecimal
0	0 0 0 0	0
1	0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0	1 2 4 8
2	0 0 1 1 1 1 0 0	3 12
3	0 1 1 1 1 0 1 1 1 1 0 1	7 11 13
4	1 1 1 1	15

Fig. 3.4. Nivelele funcţiei booleene

2. Determinarea implicanţilor primi.

Vom nota prin A, B,... implicanţii primi, adică acei termeni ce nu se mai pot reduce. Putem cupla mai departe conjuncţii vecine, cu simbolul „-" în acelaşi rang şi pentru care echivalenţii binari diferă într-un singur rang. Conjuncţia, care nu se mai poate cupla cu nici o altă conjuncţie din tabel, va fi un implicant prim al funcţiei date.

Primul element de comparaţie se notează cu ($\vee$), iar al doilea cu (*). Prin cuplarea conjuncţiei cu echivalentul binar (0 0 0 -) cu conjuncţia cu (0 0 1 -) rezultă conjuncţia cu echivalentul binar (0 0 - -), etc. Ţinând cont de cele de mai sus, rezultă primul tabel de comparare (alipire), prezentat în figura 3.5.
Avem implicanţii primi

$$A=x_1x_2\overline{x_3},\ B=x_1x_2x_4,\ C=\overline{x_1}\,\overline{x_2},\ D=\overline{x_3}\,\overline{x_4} \text{ şi } E=x_3x_4.$$

	0 0 0 - 0 0 – 0 0 – 0 0 - 0 0 0	∨ ∨ ∨ ∨
	0 0 – 1 0 0 1 – - 1 0 0 1 – 0 0	* * * *
A	0 – 1 1 - 0 1 1 1 1 0 -	∨ ∨
B	- 1 1 1 1 – 1 1 1 1 – 1	* *

Fig. 3.5. Prima alipire

În figura 3.6 este prezentat al doilea tabel de comparare.

C	0 0 - -
D	- - 0 0
E	- - 1 1

Fig.3.6. A doua alipire

3. Tabelul de acoperire

Implicantul prim	Echivalentul zecimal al TCC iniţial										
	0	1	2	3	4	7	8	11	12	13	15
A									1	1	
B										1	1
C	1	1	1	1							
D	1				1		1		1		
E				1		1		1			1

Funcţia poate avea două expresii

1) $f(x_1,x_2,x_3,x_4) = A+C+D+E = x_1x_2\overline{x_3} \vee \overline{x_1}\,\overline{x_2} \vee \overline{x_3}\,\overline{x_4} \vee x_3x_4$ sau

2) $f(x_1,x_2,x_3,x_4) = B+C+D+E = x_1x_2x_4 \vee \overline{x_1}\,\overline{x_2} \vee \overline{x_3}\,\overline{x_4} \vee x_3x_4$. ◄

3.7. Logica enunţurilor

3.7.1. Conectori logici şi formule

În logica enunţurilor [5] prezintă interes propoziţiile afirmative, care pot fi adevărate sau false, dar nici într-un caz şi una şi alta în acelaşi timp. Propoziţiile afirmative de acest tip se numesc **enunţuri**. Mai strict, **numim enunţ o propoziţie afirmativă, care este sau adevărată sau falsă, dar nu şi una şi alta**

instantaneu. Exemple de enunţuri: "Soarele este fierbinte", "Zăpada este albă", "UTM este instituţie superioară de învăţământ", "Logica este o ştiinţă". Valoarea "**adevărat**", sau "**fals**" atribuită unui enunţ se numeşte **valoare de adevăr.** De obicei, valoarea de adevăr **adevărat** este notată cu ***T*** sau cu ***1***, iar valoarea **fals** – cu ***F*** sau ***0***. Convenim să folosim litere mari sau lanţuri de litere mari pentru a nota enunţurile. De exemplu, putem nota enunţurile astfel:

P − Soarele este fierbinte,
Q − Zăpada este albă,
R − Logica este o ştiinţă.

Simbolurile P, Q, R, etc., care sunt utilizate pentru notarea enunţurilor, se numesc **formule atomare** sau **atomi.**

Din enunţuri putem construi enunţuri compuse, folosind **conectorii logici.** De exemplu, "Cerul este albastru şi zăpada este albă", "Soarele înfierbântă puternic şi vara este toridă", "Dacă pisica nu-i acasă, atunci şoarecii urcă pe masă". Conectori logici în aceste enunţuri compuse sunt "**şi**", "**dacă..., atunci**". În logica enunţurilor sunt utilizaţi cinci conectori logici: "$\neg$" (*nu)*, "$\wedge$" (*şi*), "$\vee$" (*sau*), "$\rightarrow$" (*dacă..., atunci*), "$\leftrightarrow$" (*atunci şi numai atunci*). Aceşti conectori sunt folosiţi pentru a construi enunţuri compuse, din enunţuri compuse enunţuri şi mai complexe, etc. De exemplu, notăm "*Umiditatea este mare*" prin P, "*Temperatura este înaltă*" prin Q şi "*Ne simţim bine*" prin E. Atunci propoziţia "*Dacă umiditatea este mare şi temperatura înaltă, atunci nu ne simţim bine*" poate fi scrisă astfel: $((P \wedge Q) \rightarrow (\bar{E}))$. Observăm, că un enunţ compus poate exprima o idee relativ complexă. În logica enunţurilor expresia, care reprezintă un enunţ, de exemplu, P sau un enunţ compus $((P \wedge Q) \rightarrow (\bar{E}))$ se numeşte **formulă construită corect** (formulă corectă, formulă).

Definiţie. Numim **formulă corectă** sau **formulă propoziţională**:

1. Un atom este formulă.
2. Dacă E este formulă, atunci şi $(\bar{E})$ este formulă.
3. Dacă G şi H sunt formule, atunci $(G \wedge H)$, $(G \vee H)$, $(G \rightarrow H)$ şi $(G \leftrightarrow H)$ sunt formule.
4. Nu există alte formule, decât cele generate de regulile 1 – 3.

Nu este dificil să observăm, că expresiile $(G \wedge)$, $(G \vee)$, $(\rightarrow H)$ nu sunt formule. În unele cazuri, când nu pot să apară neînţelegeri, unele perechi de paranteze pot fi omise. Pentru aceasta menţionăm, că există o ordine a priorităţilor conectorilor logici. Prioritatea cea mai mare o are conectorul "*nu*", urmează "*şi*", "*sau*", "*dacă..., atunci*", "*atunci şi numai atunci*". Deci, aceasta este ordinea de descreştere a priorităţilor şi atunci când parantezele sunt lipsă, calculele se vor face conform acestei convenţii. Dacă există mai multe operaţii cu aceeaşi prioritate, calcule se vor face de la stânga la dreapta în ordinea în care sunt scrise

în expresie. Astfel, $P \wedge Q \to \bar{E}$ va însemna $((P \wedge Q) \to (\bar{E}))$, iar $P \to Q \wedge \bar{E} \vee S$ respectiv $(P \to ((Q \wedge (\bar{E})) \vee S))$.

Valorile de adevăr ale formulelor $(\bar{E})$, $(G \wedge H)$, $(G \vee H)$, $(G \to H)$ şi $(G \leftrightarrow H)$ sunt date de tabele de adevăr ale funcţiilor booleene "*NON*" (negaţie), "*ŞI*" (produs logic), "*SAU*" (sumă logică), "*IMPLICAŢIE*", şi "*ECHIVALENŢĂ*" (v. 3.1).

3.7.2. Interpretarea formulelor în logica enunţurilor

Valoarea de adevăr a unei formule poate fi calculată plecând de la valorile de adevăr ale atomilor şi folosind tabelele de adevăr ale celor cinci formule de bază.

Definiţie. Fie G o formulă propoziţională dată, iar $A_1, A_2, ..., A_n$, atomii acestei formule. Numim **interpretare** a formulei G atribuirea valorilor de adevăr atomilor $A_1, A_2, ..., A_n$, astfel încât fiecărui atom A_i i se atribuie sau ***T,*** sau ***F*** (dar nu ambele instantaneu).

Definiţie. Vom spune, că formula G este **adevărată într-o interpretare** oarecare atunci şi numai atunci, când G ia valoarea ***T*** în această interpretare. În caz contrar vom spune, că G este falsă în această interpretare.

Dacă o formulă are n atomi diferiţi, atunci există 2^n interpretări diferite ale acestei formule. Pot fi construite tabele de adevăr analogic celor din 3.1. Adesea, dacă $A_1, A_2, ..., A_n$ sunt toţi atomii unei formule este mai comod să prezentăm o interpretare prin vectorul $(m_1, m_2, ..., m_n)$, unde m_i este A_i sau $\bar{A}_i$. De exemplu, vectorul $(P, \bar{A}, \bar{E})$ reprezintă interpretarea în care atomilor P, A, E le-au fost atribuite valorile de adevăr ***T, F, F***, respectiv. Altfel, dacă în calitate de componentă a vectorului în cauză intră atomul A ca atare, atunci acestui atom i se atribuie valoarea de adevăr ***T***, iar dacă intră negaţia lui, atunci acestui atom i se atribuie valoare ***F***.

În logica enunţurilor prezintă un interes aparte formulele care au valoarea de adevăr ***T*** şi formulele care au valoarea de adevăr ***F*** în toate interpretările posibile.

Fie formula $G = ((P \to Q) \wedge P) \to Q$. Atomii acestei formule sunt P şi Q, formula având $2^2=4$ interpretări. Valorile de adevăr sunt prezentate în tabelul 3.8. Observăm, că formula G este adevărată în toate interpretările. Formulele de acest tip se numesc **identic adevărate** sau **tautologii**.

Tabelul 3.8. Exemplu de funcţie identic adevărată

P	Q	$(P \to Q)$	$(P \to Q) \wedge P$	$((P \to Q) \wedge P) \to Q$
F	***F***	***T***	***F***	***T***
F	***T***	***T***	***F***	***T***
T	***F***	***F***	***F***	***T***
T	***T***	***T***	***T***	***T***

Fie acuma formula $G = (P \to E) \wedge (P \to \bar{E})$. Tabelul de adevăr al acestei formule este prezentat în tabelul 3.9. Observăm, că G este falsă în toate interpretările posibile. Formulele de acest tip se numesc **formule contradictorii** sau simplu **contradicţii**.

Tabelul 3.9. Exemplu de contradicţie

P	E	$\bar{E}$	$(P \to E)$	$(P \wedge \bar{E})$	$(P \to E) \wedge (P \wedge \bar{E})$
F	F	T	T	F	F
F	T	F	T	F	F
T	F	T	F	T	F
T	T	F	T	F	F

Definiţie. Numim o formulă **identic adevărată** dacă şi numai dacă ea este adevărată în toate interpretările posibile.

Definiţie. Numim o formulă **contradictorie** (sau nerealizabilă) dacă şi numai dacă ea este falsă în toate interpretările posibile. Numim o formulă **necontradictorie** (realizabilă) atunci şi numai atunci când ea nu este contradictorie.

În baza definiţiilor de mai sus sunt evidente următoarele observaţii:

1. O formulă este identic adevărată atunci şi numai atunci când negaţia ei este nerealizabilă.
2. O formulă este nerealizabilă atunci şi numai atunci când negaţia ei este tautologie.
3. O formulă nu este identic adevărată atunci şi numai atunci când există cel puţin o interpretare în care această formulă este falsă.
4. O formulă este realizabilă atunci şi numai atunci când există cel puţin o interpretare în care această formulă este adevărată.
5. Dacă o formulă este identic adevărată, atunci ea este necontradictorie, dar nu şi invers.
6. Dacă o formulă este nerealizabilă, atunci ea nu este identic adevărată, dar nu şi invers.

Construind tabelele de adevăr putem stabili că:

a) $(E \wedge \bar{E})$ este contradictorie.
b) $(E \vee \bar{E})$ este identic adevărată, deci nu este contradictorie.
c) $(E \to \bar{E})$ nu este identic adevărată, dar nici contradictorie.

Dacă o formulă F este adevărată în interpretarea I, se spune că I **satisface** F sau F este **realizabilă** în interpretarea I. Pe de altă parte, dacă o formulă F este falsă în interpretarea I, se spune că I **răstoarnă** F sau F este **răsturnată** în interpretarea I. De exemplu, formula $(P \wedge (\bar{E}))$ este realizabilă în interpretarea $(P, \bar{E})$ şi răsturnată în interpretarea (P, E). Dacă interpretarea I satisface formula F, I se mai numeşte **modelul** lui F.

Vom vedea mai jos, că demonstrarea că o formulă este identic adevărată sau contradicţie este de o importanţă foarte mare. În logica enunţurilor, dacă numărul interpretărilor este finit, putem rezolva această problema prin metoda enumerării.

3.7.3. Forme normale şi consecinţe logice

Observăm, că noţiunea de atom din logica enunţurilor coincide cu cea de argument din logica booleană. Mai mult, dacă luăm în consideraţie ultima observaţie, cei cinci conectorii logici şi funcţiile booleene respective diferă doar la nivel frazeologic. Evident, toate rezultatele din 3.1 – 3.6 pot fi extinse şi pentru logica enunţurilor, inclusiv şi cele legate de echivalenţa a două formule, transformările echivalente, proprietăţile 3.4 – 3.12, formele canonice şi algoritmii de obţinere a formelor canonice, etc. Utilizând rezultatele paragrafelor precedente putem afirma, că o formulă poate fi reprezentată în forma sa canonică disjunctivă sau conjunctivă (forma disjunctivă normal perfectă (FDNP), forma conjunctivă normal perfectă (FCNP)) sau, pentru cazul în care unii atomi pot fi lipsă în cadrul mintermilor (maxtermilor), cu ajutorul formelor normale disjunctive sau conjunctive. Aceste forme normale dau posibilitatea să se stabilească, dacă două formule sunt echivalente, adică dacă reprezintă acelaşi enunţ compus. Pot fi utilizate metodele de minimizare pentru a ajunge la o reprezentare cât mai simplă a unui enunţ, reprezentare care prezintă diferite avantaje în cercetare şi implementare.

De exemplu, formula $(A \vee \bar{E}) \rightarrow R$ conform rezultatelor precedente, poate fi pusă sub forma sa normală disjunctivă $(\bar{A} \wedge E) \vee R$, iar formula $(A \wedge (Q \rightarrow E)) \rightarrow S$ sub forma normală conjunctivă $(S \vee \bar{A} \vee Q) \wedge (S \vee \bar{A} \vee \bar{E})$.

Adesea este necesar să se stabilească, dacă o afirmaţie dată reiese din câteva afirmaţii iniţiale. Fie următorul exemplu [5]:

Presupunem, că cursul acţiunilor scade, dacă preţul pentru vânzarea lor creşte. Presupunem, de asemenea, că majoritatea oamenilor sunt nefericiţi, dacă cursul acţiunilor cade.

Fie că preţul de vânzare al acţiunilor creşte. Putem concluziona, că majoritatea oamenilor sunt nefericiţi.

Pentru a verifica această concluzie, notăm afirmaţiile astfel:

P − Preţul de vânzare creşte
S − Cursul acţiunilor scade
U − Majoritatea oamenilor sunt nefericiţi

În exemplul nostru avem patru afirmaţii.

(1) Dacă preţul de vânzare al acţiunilor creşte, atunci cursul acţiunilor scade.
(2) Dacă cursul acţiunilor scade, atunci majoritatea oamenilor sunt nefericiţi.

(3) Preţul de vânzare al acţiunilor creşte.
(4) Majoritatea oamenilor sunt nefericiţi.

Reprezentarea simbolică a acestor afirmaţii este următoarea:

(1^1) $P \to S$,
(2^1) $S \to U$,
(3^1) P,
(4^1) U.

Trebuie să demonstrăm, că (4^1) are valoare de adevăr $\boldsymbol{T}$, dacă (1^1)$\wedge$(2^1)$\wedge$(3^1) este adevărată.

Aducerea la forma normală a $((P \to S) \wedge (S \to U) \wedge P)$, care reprezintă ($1^1$)$\wedge$($2^1$)$\wedge$($3^1$) şi care este lăsată ca exerciţiu, ne conduce la $P \wedge S \wedge U$.

Deci, dacă $((P \to S) \wedge (S \to U) \wedge P)$ are valoare de adevăr $\boldsymbol{T}$, atunci şi $(P \wedge S \wedge U)$ este adevărată. Deoarece $(P \wedge S \wedge U)$ este $\boldsymbol{T}$ numai dacă P, S şi U sunt adevărate, concluzionăm, că U este adevărată. Deoarece U este adevărată numai dacă $(P \to S)$, $(S \to U)$ şi P sunt adevărate, U se numeşte **consecinţă logică** a $(P \to S)$, $(S \to U)$ şi P.

Definiţie. Fie date formulele $F_1, F_2, \ldots, F_n$ şi formula G. Numim G **consecinţă logică** a formulelor $F_1, F_2, \ldots, F_n$ (sau G reiese logic din $F_1, F_2, \ldots, F_n$) atunci şi numai atunci, când pentru orice interpretare I, în care $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ este adevărată, G la fel este adevărată. $F_1, F_2, \ldots, F_n$ se numesc **axiome** (sau **postulate**, sau **premise**) ale formule G.

Teorema 3.9. Dacă sunt date formulele $F_1, F_2, \ldots, F_n$ şi formula G, atunci G este consecinţă logică ale $F_1, F_2, \ldots, F_n$ dacă şi numai dacă formula $((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to G$ este identic adevărată.

Demonstraţie. ($\Rightarrow$) Presupunem, că U este consecinţă logică a formulelor $F_1, F_2, \ldots, F_n$. Fie I o interpretare arbitrară. Dacă $F_1, F_2, \ldots, F_n$ sunt adevărate în I, atunci conform definiţiei noţiunii de consecinţă logică, U este adevărată în I. Drept rezultat $((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to U)$ este adevărată în I. Pe de altă parte, dacă nu toate formulele $F_1, F_2, \ldots, F_n$ sunt adevărate în I, atunci $((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to U)$ tot una este adevărată în I. Am demonstrat că $((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to U)$ este adevărată în orice interpretare, adică $((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to U)$ este o formulă identic adevărată.

($\Leftarrow$) Presupunem, că $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \to U$ este o formulă identic adevărată. Pentru orice interpretare I, dacă $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$ este adevărată în I, atunci U trebuie să fie adevărată în I (conform tabelului de adevăr). Deci, U este consecinţă logică a formulelor $F_1, F_2, \ldots, F_n$, c.c.t.d.

Teorema 3.10. Dacă sunt date formulele $F_1, F_2, \ldots, F_n$ şi formula U, atunci U este consecinţă logică ale $F_1, F_2, \ldots, F_n$ dacă şi numai dacă formula $((F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge \bar{U})$ este identic contradicţie.

Demonstraţie. Conform teoremei precedente U este consecinţă logică a formulelor $F_1, F_2,..., F_n$ dacă şi numai dacă formula $((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow U)$ este identic adevărată. Deci, U este consecinţă logică a formulelor $F_1, F_2,..., F_n$ dacă şi numai dacă negaţia formulei $((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow U)$ este contradicţie. Deoarece negaţia acestei formule este egală cu $((F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \bar{U})$, concluzionăm, că teorema 2.2 este demonstrată.

Aceste două teoreme sunt foarte importante. Conform lor putem afirma, că demonstrarea faptului, că o formulă oarecare este consecinţă logică a unei mulţimi finite de formule este echivalentă cu demonstrarea faptului că o formulă legată de formulele iniţiale este tautologie sau contradicţie.

Dacă U este consecinţă logică a formulelor $F_1, F_2,..., F_n$, atunci formula $((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow U)$ se numeşte **teoremă**, iar U se numeşte **concluzia** teoremei. În viaţa de toate zilele, ca şi în matematică, multe probleme pot fi formulate sub forma unor teoreme şi demonstrarea lor. Mai jos este adus un exemplu simplu, care arată cum pot fi folosite teoremele 3.9 şi 3.10.

Exemplul 3.8. Fie formulele $F_1 \rightleftharpoons (E \rightarrow U)$, $F_2 \rightleftharpoons \bar{U}$, $G \rightleftharpoons \bar{E}$. Vom arăta, că G este consecinţă logică a formulelor F_1 şi F_2.

Metoda 1. Putem construi tabelele de adevăr pentru a arăta, că formula G este adevărată în fiecare dintre modelele formulei $(E \rightarrow U) \wedge \bar{U}$.

Tab. 3.10. Tabelul de adevăr pentru $(E \rightarrow U) \wedge \bar{U}$ şi $\bar{E}$

E	U	$E \rightarrow U$	$\bar{U}$	$(E \rightarrow U) \wedge \bar{U}$	$\bar{E}$
F	F	T	T	T	T
F	T	T	F	F	T
T	F	F	T	F	F
T	T	T	F	F	F

Observăm, că există doar un model pentru $(E \rightarrow U) \wedge \bar{U}$, anume $(\bar{E}, \bar{U})$. Formula $\bar{E}$ este adevărată în acest model şi, în baza definiţiei noţiunii consecinţă logică, conchidem că G este consecinţă logică a formulelor $(E \rightarrow U)$ şi $\bar{U}$.

Metoda 2. Putem folosi teorema 3.9, extinzând tabelul precedent de adevăr.

Tab. 3.11. Tabelul de adevăr pentru $((E \rightarrow U) \wedge \bar{U}) \rightarrow \bar{E}$

E	U	$E \rightarrow U$	$\bar{U}$	$(E \rightarrow U) \wedge \bar{U}$	$\bar{E}$	$((E \rightarrow U) \wedge \bar{U}) \rightarrow \bar{E}$
F	F	T	T	T	T	T
F	T	T	F	F	T	T
T	F	F	T	F	F	T
T	T	T	F	F	F	T

Tabelul 3.11 arată, că formula $((E \to U) \wedge \bar{U}) \to \bar{E}$ este adevărată în orice interpretare, adică tautologie şi, conform teoremei 3.9, G este consecinţă logică a formulelor $(E \to U)$ şi $\bar{U}$.

Metoda 3. Putem folosi teorema 3.10, arătând că formula $((E \to U) \wedge \bar{U}) \wedge E$ este contradicţie.

Tab. 3.12. Tabelul de adevăr pentru $((E \to U) \wedge \bar{U}) \wedge E$

E	U	$E \to U$	$\bar{U}$	$(E \to U) \wedge \bar{U}$	$\bar{E}$	$((E \to U) \wedge \bar{U}) \wedge E$
F	F	T	T	T	T	F
F	T	T	F	F	T	F
T	F	F	T	F	F	F
T	T	T	F	F	F	F

Tabelul 3.12 arată, că formula $((E \to U) \wedge \bar{U}) \wedge E$ este falsă în orice interpretare, adică este contradicţie şi, conform teoremei 3.10, G este consecinţă logică a formulelor $(E \to U)$ şi $\bar{U}$. ◄

3.7.4. Aplicaţii ale logicii enunţurilor

După ce în paragrafele precedente au fost discutate diferite noţiuni de bază, să facem acum cunoştinţă cu unele aplicaţii ale logicii enunţurilor. Aceasta poate fi ilustrat prin exemple.

Exemplul 3.9. Presupunem, că are loc o grevă. Greva nu se va termina, dacă parlamentul refuză să adopte legi noi, cerute de grevişti, cu condiţia că greva nu durează mai mult de un an şi Preşedintele nu demisionează. Se va termina oare greva, dacă parlamentul refuză să adopte legi noi, iar greva numai ce a început?

Mai întâi vom transforma afirmaţiile în simboluri:

P : Parlamentul refuză să acţioneze,
Q : Greva se termină,
R : Preşedintele demisionează
S : Greva durează mai mult de un an.

Datele iniţiale pot fi atunci reprezentate prin următoarele formulele:

$F_1 : (P \vee \bar{R} \wedge \bar{S}) \to \bar{Q}$ – Dacă Parlamentul refuză să adopte legi noi sau Preşedintele nu demisionează şi greva nu durează mai mult de un an, atunci greva nu se va termina,

$F_2 : P$ – Parlamentul refuză să adopte legi noi,

$F_3 : \bar{S}$ – Greva a început recent.

Putem oare din F_1, F_2 şi F_3 să concluzionăm, că greva nu se va termina, adică putem arăta, că $\bar{Q}$ este consecinţă logică a formulelor F_1, F_2 şi F_3? Conform teoremei 3.9 asta este echivalent cu afirmaţia că este identic adevărată formula $(((P \vee \bar{R} \wedge \bar{S}) \to \bar{Q}) \wedge P \wedge \bar{S}) \to \bar{Q}$.

Notând $\overline{R} \wedge \overline{S}$ prin *1*, $(P \vee \overline{R} \wedge \overline{S})$ prin *2*, $((P \vee \overline{R} \wedge \overline{S}) \to \overline{Q})$ prin *3*, $(((P \vee \overline{R} \wedge \overline{S}) \to \overline{Q}) \wedge P$ prin *4*, $((((P \vee \overline{R} \wedge \overline{S}) \to \overline{Q}) \wedge P \wedge \overline{S})$ prin *5*, iar $((((P \vee \overline{R} \wedge \overline{S}) \to \overline{Q}) \wedge P \wedge \overline{S}) \to \overline{Q}$ prin *F*, tabelul de adevăr al formulei *F* are forma prezentată în tabelul 3.13.

Tab. 3.13. Tabelul de adevăr al formulei *F*

P	*Q*	*R*	*S*	$\overline{S}$	$\overline{R}$	*1*	*2*	$\overline{Q}$	*3*	*4*	*5*	*F*
F	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*T*	*T*	*F*	*F*	*T*
F	*F*	*F*	*T*	*F*	*T*	*F*	*F*	*T*	*T*	*F*	*F*	*T*
F	*F*	*T*	*F*	*T*	*F*	*F*	*F*	*T*	*T*	*F*	*F*	*T*
F	*F*	*T*	*T*	*F*	*F*	*F*	*F*	*T*	*T*	*F*	*F*	*T*
F	*T*	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*F*	*F*	*F*	*T*
F	*T*	*F*	*T*	*F*	*T*	*F*	*F*	*F*	*T*	*F*	*F*	*T*
F	*T*	*T*	*F*	*T*	*F*	*F*	*F*	*F*	*T*	*F*	*F*	*T*
F	*T*	*T*	*T*	*F*	*F*	*F*	*F*	*F*	*T*	*F*	*F*	*T*
T	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*
T	*F*	*F*	*T*	*F*	*T*	*F*	*T*	*T*	*T*	*T*	*F*	*T*
T	*F*	*T*	*F*	*T*	*F*	*F*	*T*	*T*	*T*	*T*	*T*	*T*
T	*F*	*T*	*T*	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*T*
T	*T*	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*F*	*F*	*F*	*T*
T	*T*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*F*	*F*	*F*	*T*
T	*T*	*T*	*F*	*T*	*F*	*F*	*T*	*F*	*F*	*F*	*F*	*T*
T	*T*	*T*	*T*	*F*	*F*	*F*	*T*	*F*	*F*	*F*	*F*	*T*

Acest tabel demonstrează, că formula $(((P \vee \overline{R} \wedge \overline{S}) \to \overline{Q}) \wedge P \wedge \overline{S}) \to \overline{Q}$ este identic adevărată, iar concluzia este, că greva nu se va termina. ◄

Exemplul 3.10. Sunt cunoscute următoarele reacţii chimice:

$$Mg+H_2 \to Mg+H_2O,$$
$$C+O_2 \to CO_2,$$
$$CO_2+H_2O \to H_2CO_3.$$

Presupunem că avem la dispoziţie o oarecare cantitate de MgO, H_2, O_2 şi C. Trebuie să arătăm, că putem obţine H_2CO_3.

Pentru aceasta vom considera MgO, H_2, O_2 şi C formule atomare. Atunci formulele de mai sus pot fi reprezentate cu ajutorul formulelor următoare:

$$(Mg \wedge H_2) \to (Mg \wedge H_2O) \quad : A_1,$$
$$(C \wedge O_2) \to CO_2 \quad : A_2,$$
$$(CO_2 \wedge H_2O) \to H_2CO_3 \quad : A_3.$$

Afirmaţia, că avem la dispoziţie o oarecare cantitate de MgO, H_2, O_2 şi C poate fi prezentată prin formulele:

MgO : A_4,
H_2 : A_5,
O_2 : A_6,
C : A_7.

Trebuie să demonstrăm, că H_2CO_3 este consecinţă logică a formulelor $A_1, A_2,..., A_7$. Conform teoremei 3.10 aceasta este corect, dacă formula $(A_1, A_2,..., A_7) \wedge \overline{H_2CO_3}$ este contradicţie.

Aceasta poate fi demonstrat, construind tabelul de adevăr sau operând transformări echivalente (lăsăm ca exerciţiu). ◄

Am arătat, că logica enunţurilor poate fi aplicată pentru rezolvarea unor probleme foarte diferite. Însă, metoda transformărilor echivalente sau a tabelelor de adevăr poate fi utilizată doar pentru cazuri relativ simple. În caz general, avem nevoie de programe eficiente de calculator, construite în baza unor metode matematice speciale şi algoritmi sofisticaţi.

3.8. Logica de ordinul unu

Elementele iniţiale în logica enunţurilor sunt atomii. Din atomi construim formule, iar aceste formule sunt utilizate pentru a exprima idei complexe. Atomul este considerat un tot întreg. Structura şi componenţa sa nu sunt analizate. Însă există idei, care nu pot fi exprimate într-o manieră atât de simplă. Cu titlu de exemplu următorul raţionament:

Fiecare om este muritor. Deoarece Confucius este om, el este muritor.

Acest raţionament este corect. Însă, dacă vom introduce notaţiile

P: Fiecare om este muritor,
Q: Confucius este om,
R: Confucius este muritor,

putem constata, că *R* nu este consecinţă logică a formulelor *P* şi *Q* în cadrul logicii enunţurilor. Aceasta are loc din cauza că în logica enunţurilor structura enunţurilor *P*, *Q* şi *R* nu este folosită. Vom introduce în paragrafele care urmează logica de ordinul unu, care spre deosebire de logica enunţurilor, mai conţine trei noţiuni logice, numite *termi*, *predicate* şi *cuantificatori*. Partea cea mai mare a unei limbi naturale, ca şi a limbajului matematic, poate fi formalizată în cadrul logicii de ordinul unu.

3.8.1. Noţiuni de bază

La fel ca şi pentru logica enunţurilor, vom defini mai întâi atomii logicii de ordinul unu. Vom începe cu câteva exemple. Fie că vrem să reprezentăm afirmaţia "*x este egal cu 5*". Definim mai întâi predicatul *EGAL(x,y)*, care

înseamnă* "*x este egal cu y"*. Atunci expresia "*x este egal cu 5*" poate fi prezentată prin expresia *EGAL*(*x*,*5*). Analogic, în loc de "*A iubeşte pe B"* scriem *IUBEŞTE*(*A*,*B*).

În logica de ordinul unu mai pot fi folosite simbolurile funcţionale. De exemplu, putem scrie *plus(x,y)* pentru a nota "*x+y*", *mama*(*x*), care ar însemna "*mama persoanei x*". În acest caz, propoziţia "*mama lui Eugen îl iubeşte pe Eugen*" poate fi reprezentată simbolic *IUBEŞTE*(*mama*(*Eugen*),*Eugen*), iar expresia "*x+y este egal cu z*" se va scrie *EGAL*(*plus*(*x*,*y*),*z*).

În exemplele de mai sus expresiile *EGAL*(*x*,*5*), *IUBEŞTE*(*A*,*B*), *EGAL*(*plus*(*x*,*y*),*z*) şi *IUBEŞTE*(*mama*(*Eugen*),*Eugen*) sunt atomi ai logicii de ordinul unu în care *EGAL* şi *IUBEŞTE* sunt simboluri predicative, *x*, *y*, *z*, *A*, *B* sunt variabile, *Eugen* şi *5* sunt simboluri individualizatoare sau constante, iar *mama* şi *plus* – simboluri funcţionale.

În caz general, pentru construirea atomilor sunt permise următoarele patru tipuri de simboluri:

1. Simboluri individualizatoare sau constante (nume de obiecte sau valori numerice),
2. Simboluri de variabile (litere mici *x, y, z*,..., este posibilă utilizarea indicilor),
3. Simboluri funcţionale (litere mici *f, g, h*,... sau cuvinte cu sens din litere mici, cum ar fi *mama, plus*),
4. Simboluri predicative (de obicei litere mari *P, Q, R*,...sau cuvinte din litere mari *EGAL, IUBEŞTE*).

O funcţie sau un predicat pot fi unare, binare, ternare, etc. De exemplu, *EGAL* şi *IUBEŞTE* sunt predicate binare, *mama* este simbol funcţional unar, iar *plus* – binar.

O funcţie aplică lista argumentelor într-o constantă. De exemplu, *mama* este o funcţie, care pentru valoarea dată a argumentului (*Eugen*), stabileşte persoana, care este mama lui Eugen, chiar dacă numele acestei persoane nu este concretizat. Expresia de tipul *mama*(*Eugen*) se numeşte în logica de ordinul unu *term*.

Iată definiţia formală a noţiuni **term**:

1. O constantă este term.
2. O variabilă este term.
3. Dacă *f* este un simbol funcţional *n*-ar şi $t_1, t_2,..., t_n$ sunt termi, atunci $f(t_1, t_2,..., t_n)$ este term.
4. Alţi termi, decât cei generaţi cu ajutorul regulilor 1 – 3 nu există.

Exemplul 3.11. Deoarece *x* şi *7* sunt termi, iar *plus* este un simbol funcţional binar, expresia *plus*(*x*,*7*), conform definiţiei de mai sus este term. Este uşor să ne convingem, că expresiile *plus(plus(x,5),x)* şi

* Am putea spune că un predicat este o relaţie, ceea ce nu-i tocmai corect. Predicatul este o funcţie, care ia valoarea *T* pentru seturile, care verififcă relaţia dată şi *F* îb caz contrar.

mama(mama(Eugen)) sunt termi cu valorile respective $x+5+x$ şi *bunica lui Eugen*. ◄

Predicatul este o aplicaţie, care aplică o listă de constante (argumentele predicatului) în *T* sau *F*. De exemplu, predicatul *EGAL* pentru perechea (*5*,*9*) are valoarea *F*, iar pentru (*7*,*7*) – *T*. Cunoscând ce sunt termii în cadrul logicii de ordinul unu, putem defini noţiunea de **atom**.

Definiţie. Dacă *P* este un simbol predicativ n – ar şi $t_1, t_2,..., t_n$ termi, atunci $P(t_1, t_2,..., t_n)$ este atom.

Pentru construirea formulelor, în afara celor cinci conectori logici, introduşi în paragraful precedent, vor mai fi utilizate două simboluri speciale $\forall$ şi $\exists$, care sunt chemate să caracterizeze variabilele. Aceste simboluri se numesc ***cuantificatori***: $\forall$ este cuantificatorul generalizării, iar $\exists$ de existenţă. Dacă *x* este o variabilă, atunci ($\forall x$) se citeşte "*pentru orice x*", "*pentru fiecare x*" sau "*pentru toţi x*", iar ($\exists x$) poate fi citit "*există x*", "*pentru unele valori ale lui x*" sau "*cel puţin pentru un x*". ($\forall x$) şi ($\exists x$) mai sunt numite complexe cuantificatoriale.

Exemplul 3.12. Vom prezenta printr-o formulă afirmaţiile:

(a) Orice număr raţional este număr real.
(b) Există un număr, care este număr prim.
(c) Pentru orice număr *x* există un număr *y*, astfel încât $x<y$.

Notăm afirmaţia "*x este număr prim*" prin *P(x)*, "*x este număr raţional*" prin *Q(x)*, "*x este număr real*" prin *R(x)* şi "*x este mai mic ca y*" prin *MAI_MIC(x, y)*. Expresiile de mai sus pot fi scrise după cum urmează:

(a^1) $(\forall x)(Q(x)\rightarrow R(x))$,
(b^1) $(\exists x)P(x)$,
(c^1) $(\forall x)(\exists y)$ $MAI\_MIC(x, y)$.

Fiecare din expresiile (a^1), (b^1) şi (c^1) se numeşte formulă. Înainte de a defini formal noţiunea de formulă, introducem noţiunile de **variabile legate** şi **variabile libere**. Pentru aceasta definim mai întâi **domeniul de acţiune** a unui cuantificator, care intră în componenţa unei formule – este acea formulă în care acest cuantificator este aplicat. De exemplu, domeniul de acţiune a cuantificatorului de existenţă în formula $(\forall x)(\exists y)MAI\_MIC(x, y)$ este formula $MAI\_MIC(x,y)$, iar a cuantificatorului generalizării – formula $(\exists y)$ $MAI\_MIC(x, y)$. Domeniul de acţiune al cuantificatorului generalizării în formula $(\forall x)(Q(x)\rightarrow R(x))$ este formula $(Q(x)\rightarrow R(x))$. ◄

Definiţie. Intrarea unei variabile într-o formulă se numeşte legată atunci şi numai atunci, când ea (intrarea) sau coincide cu intrarea în componenţa unui complex cuantificator (($\forall x$) sau ($\exists x$)), sau se află în domeniul de

acţiune a unui atare complex. Intrarea unei variabile într-o formulă este **liberă** atunci şi numai atunci când ea nu este legată.

Definiţie. O variabilă este **liberă** într-o formulă, dacă cel puţin o intrare a acestei variabile în formula dată este liberă. O variabilă este **legată** într-o formulă, dacă cel puţin o intrare a acestei variabile în formula dată este legată.

În formula $(\forall x)P(x,y)$ variabila x este legată, deoarece ambele intrări ale lui x sunt legate. Variabila y este liberă, deoarece unica intrare a acesteia este liberă. O variabilă poate fi în cadrul unei formule şi liberă şi legată în acelaşi timp. De exemplu, în cadrul formulei $(\forall x)P(x, y) \wedge (\forall y)Q(y)$ variabila y este şi liberă şi legată.

Utilizând noţiunile atom, conectori logici şi cuantificatori, putem defini formal conceptul de formulă.

Definiţie. Formulele construite corect (sau simplu **formule**) în logica de ordinul unu pot fi definite recursiv astfel:

1. Un atom (o formulă atomară) este formulă.
2. Dacă E şi F sunt formule, atunci şi $(\bar{E})$ $(E \wedge F)$, $(E \vee F)$, $(E \to F)$ şi $(E \leftrightarrow F)$ sunt formule.
3. Dacă F este formulă, iar x este o variabilă liberă în F, atunci $(\forall x)F$ şi $(\exists x)F$ sunt formule.
4. Nu există alte formule, decât cele generate de regulile 1 – 3.

Exemplul 3.13. Să exprimăm prin formulă exemplul de la începutul acestui paragraf. Notăm "*x este om*" prin $OM(x)$, "*x este muritor*" prin $MURITOR(x)$.
Atunci afirmaţia "fiecare om este muritor" poate fi reprezentată cu ajutorul formulei $(\forall x)(OM(x) \to MURITOR(x))$, afirmaţia "Confucius este om" prin formula $OM(Confucius)$, iar ultima afirmaţie "Confucius este muritor" – $MURITOR(Confucius)$.
Afirmaţia finală în întregime poate fi exprimată cu ajutorul formulei:

$$(\forall x)(OM(x) \to MURITOR(x)) \wedge OM(Confucius) \to MURITOR(Confucius).$$ ◄

Exemplul 3.14. Să exprimăm prin formule axiomele de bază ale numerelor naturale:

- A_1: Pentru orice număr există un număr, şi numai unul singur, care urmează imediat după numărul dat (succesorul lui x).
- A_2: Nu există un număr după care urmează imediat 0.
- A_3: Pentru orice număr, diferit de 0, există un număr, şi numai unul singur, care se află imediat înaintea numărului dat (predecesorul lui x).

Fie $f(x)$ şi $g(x)$ succesorul şi, respectiv, predecesorul lui x, iar afirmaţia "*x este egal cu y*" este notată prin $E(x, y)$. Atunci axiomele de mai sus pot fi reprezentate prin formulele:

A^1_1: $(\forall x)(\exists y)(E(y, f(x)) \wedge (\forall z)(E(z, f(x)) \rightarrow E(y, z)))$,

A^1_2: $\overline{((\exists x)E(0, f(x)))}$,

A^1_3: $(\forall x)(\overline{E(x,0)} \rightarrow ((\exists y)(E(y, g(x)) \wedge (\forall z)(E(z, g(x)) \rightarrow E(y, z)))))$. ◄

3.8.2. Interpretarea formulelor în logica de ordinul unu

În logica enunţurilor prin interpretarea unei formule am considerat atribuirea unor valori de adevăr atomilor, care intră în componenţa formulei. În logica de ordinul unu trebuie să facem mai multe, deoarece suntem obligaţi să luăm în consideraţie variabilele. Pentru a defini interpretarea unei formule trebuie să indicăm domeniul obiectiv (domeniul de valori al variabilelor obiective) şi valorile constantelor, simbolurilor funcţionale şi predicative, care sunt în cadrul formulei.

Definiţie. Interpretarea formulei F în logica de ordinul unu constă dintr-un domeniu (obiectiv) nevid D şi indicaţia "estimărilor" (valorilor) constantelor, simbolurilor funcţionale şi predicative, care formează F:

1. Fiecărei constante îi punem în corespondenţă un element oarecare din D.
2. Fiecărui simbol funcţional n-ar punem în corespondenţă aplicaţia D^n în D.
3. Fiecărui simbol predicativ punem în corespondenţă aplicaţia D^n în $\{T, F\}$.

Pentru a sublinia, că este vorba despre domeniul D, vom mai spune că **interpretăm** formula F în $\boldsymbol{D}$. Atunci când "estimăm", adică determinăm valoarea de adevăr a formulei în interpretările din domeniul D, complexul $(\forall x)$ va fi interpretat ca "pentru toţi x din D", iar $(\exists x)$ – "există un element x din D".

Pentru orice interpretare a formulei F în D, formulei i se vor atribui valori de adevăr în conformitate cu următoarele reguli:

1. Dacă sunt date valorile de adevăr ale formulelor G şi H, atunci valorile de adevăr ale formulelor $\overline{G}$, $G \wedge H$, $G \vee H$, $G \rightarrow H$, $G \leftrightarrow H$ sunt calculate reieşind din tabelele de adevăr ale acestor conectori logici.
2. $(\forall x)G$ ia valoare de adevăr T, dacă G are valoarea T pentru orice x din D; în caz contrar valoarea F.
3. $(\exists x)G$ ia valoare de adevăr T, dacă G are valoarea T pentru cel puţin un x din D; în caz contrar valoarea F.

Notăm, că formula, care conţine variabile libere, nu poate avea valoare de adevăr. Variabilele libere sunt, de obicei, considerate constante.

Exemplul 3.15. Fie formulele $(\forall x)P(x)$ şi $(\exists x)\overline{P(x)}$ şi următoarea interpretare:

Domeniul: $D = \{1, 2\}$.
Estimarea formulei P:

$P(1)$	$P(2)$
T	F

Este uşor să ne convingem, că $(\forall x)P(x)$ are valoarea F în această interpretare, deoarece $P(x)$ nu este T şi pentru $x=1$, şi pentru $x=2$. Pe de altă parte, deoarece $\overline{P(2)}$ este T în această interpretare, $(\exists x)\,\overline{P(x)}$ este la fel T. ◄

Exemplul 3.16. Considerăm formula $(\forall x)\,(\exists y)P(x, y)$.
Definim interpretarea după cum urmează:
Domeniul: $D = \{1, 2\}$,

$P(1, 1)$	$P(1, 2)$	$P(2, 1)$	$P(2, 2)$
T	F	F	T

Când $x=1$, putem observa, că există o valoare a lui y (anume $y=1$), pentru care $P(1, y)=T$. Dacă $x=2$, de asemenea există un y pentru care $P(2, y)=T$.
Deci, în interpretarea dată pentru orice x din D există o astfel de valoare a lui y, încât $P(x, y)$ este T, adică $(\forall x)\,(\exists y)P(x, y)$ are valoarea T în această interpretare. ◄

Exemplul 3.17. Fie formula $G \rightleftharpoons (\forall x)(P(x)\rightarrow Q(f(x), a))$.
În G avem o constantă a, un simbol funcţional unar f, un simbol predicativ unar P şi un simbol predicativ binar Q. Fie următoarea interpretare I a formulei G:
Domeniul: $D = \{1, 2\}$,
Estimarea pentru a:

a
1

Estimarea pentru f:

$f(1)$	$f(2)$
2	1

Estimările pentru P şi Q:

$P(1)$	$P(2)$	$Q(1, 1)$	$Q(1, 2)$	$Q(2, 1)$	$Q(2, 2)$
F	T	T	T	F	T

Dacă $x=1$, atunci $P(x)\rightarrow Q(f(x), a) = P(1)\rightarrow Q(f(1), a) = P(1)\rightarrow Q(2, 1) = F \rightarrow F = T$.
Analogic, dacă $x=2$, atunci $P(x)\rightarrow Q(f(x), a) = P(2)\rightarrow Q(f(2), a) = P(2)\rightarrow Q(1, 1) = T \rightarrow T = T$.
Deoarece $P(x)\rightarrow Q(f(x), a)$ are valoare de adevăr T pentru toate elementele $x \in D$, formula $G \rightleftharpoons (\forall x)(P(x)\rightarrow Q(f(x), a))$ este adevărată în interpretarea I. ◄

Exemplul 3.18. Să estimăm în interpretarea exemplului precedent formulele următoare:

(a) $(\exists x)(P(f(x)) \wedge Q(x, f(a)))$
(b) $(\exists x)(P(x) \wedge Q(x, a))$
(c) $(\forall x)(\exists y)(P(x) \wedge Q(x, y))$

Pentru *(a)*: dacă $x=1$, atunci $P(f(x)) \wedge Q(x, f(a)) = P(f(1)) \wedge Q(1, f(a)) = P(2) \wedge Q(1, f(1)) = P(2) \wedge Q(1, 2) = T \wedge T = T$, iar dacă $x=2$ - $P(f(x)) \wedge Q(x, f(a)) = P(f(2)) \wedge Q(2, f(a)) = P(1) \wedge Q(2, f(1)) = P(1) \wedge Q(2, 2) = F \wedge T = F$.
Deoarece în domeniul D există un element, anume $x = 1$, petnru care $P(f(x)) \wedge Q(x, f(a))$ are valloarea T, formula $(\exists x)(P(f(x)) \wedge Q(x, f(a)))$ este adevărată în interpretarea I.

Pentru *(b)*: dacă $x = 1$, atunci $P(x) \wedge Q(x, a) = P(1) \wedge Q(1, 1) = F \wedge T = F$;
dacă $x = 2$, atunci $P(x) \wedge Q(x, a) = P(2) \wedge Q(2, 1) = T \wedge F = F$.
Deci, în D nu este măcar un element pentru care $P(x) \wedge Q(x, a)$ ar fi adevărată, formula $(\exists x)(P(x) \wedge Q(x, a))$ este falsă în interpretarea I.

Pentru *(c)*: dacă $x = 1$, atunci $P(x) = P(1) = F$, iar $P(x) \wedge Q(x, y) = F$ şi pentru $y = 1$, şi pentru $y = 2$. Deoarece există un element x (anume $x = 1$) pentru care $(\exists y)(P(x) \wedge Q(x, y))$ este falsă, formula $(\forall x)(\exists x)(P(x) \wedge Q(x, y))$ este falsă în interpretarea I. ◄

Noţiunlie introduse în 3.7 (tautologie, contradicţie, consecinţe logice) pot fi definite şi pentru logcia de ordinul unu, inclusiv pot fi demonstrate teoreme analogice despre consecinţele logice (lăsăm ca exerciţiu pentru cititor).

3.8.3. Forma normală Prenex

Au fost introduse formele normale disjunctive şi conjunctive pentru logica enunţurilor. Şi în logica de gradul 1 există o formă normală, numită **formă normală Prenex**. Scopul introducerii acestei noţiuni este simplificarea demonstrării teoremelor.

Definiţie. Formula F în logica de gradul 1 se spune că este în forma normală Prenex, dacă şi numai dacă formula F este de forma

$$(Q_1x_1)...(Q_nx_n)(M),$$

unde fiecare (Q_ix_i), $i = 1, 2,..., n$, reprezintă sau $(\forall x_i)$, sau $(\exists x_i)$, iar (M) este formulă care nu conţine cuantificatori. $(Q_1x_1)...(Q_nx_n)$ se numeşte **prefix**, iar M – matricea formulei F.

Iată câteva exemple de formule în forma normală Prenex:

$(\forall x)(\forall x)(P(x,y) \wedge Q(y))$,
$(\forall x)(\forall x)(\overline{P(x, y)} \rightarrow Q(y))$,
$(\forall x)(\forall x)(\exists z)(Q(x,y) \rightarrow R(z))$.

Să facem cunoştinţă cu o metodă de transformare a unei formule în forma normală Prenex. Pentru această trebuie să cunoaştem câteva formule de bază, echivalente în logica de gradul 1. În afară de formulele echivalente din logica booleană, care sunt adevărate şi în logica de gradul 1, există şi alte perechi de formule, care conţin cuatificatori. Fie F o formulă, care conţine variabila liberă x. Pentru a sublinia, că x intră în F şi este liberă vom scrie $F[x]$. Fie G o formulă, care nu conţine variabila x. Avem în acest caz următoarele perechi de formule echivalente (aici Q este sau $\forall$ sau $\exists$):

$$(Qx)F[x] \vee G = (Qx)(F[x] \vee G); \qquad (3.19a)$$

$$(Qx)F[x] \wedge G = (Qx)(F[x] \wedge G); \qquad (3.19b)$$

$$\overline{((\forall x)F[x])} = (\exists x)(\overline{(F[x])}); \qquad (3.20a)$$

$$\overline{((\exists x)F[x])} = (\forall x)(\overline{(F[x])}). \qquad (3.20b)$$

$$(\forall x)F[x] \wedge (\forall x)H[x] = (\forall x)(F[x] \wedge H[x]); \qquad (3.21a)$$

$$(\exists x)F[x] \vee (\exists x)H[x] = (\exists x)(F[x] \vee H[x]); \qquad (3.21b)$$

Demonstrarea relaţiilor (3.19a) – (3.21b) este dată ca exerciţiu.

Subliniem că repartizarea cuantificatorului $\forall$ poate fi făcută doar pentru conectorul $\wedge$, ca şi repartizarea cuantificatorului $\exists$ poate fi făcută doar pentru conectorul $\vee$, adică trebuie să avem în vedere că:

$$(\forall x)F[x] \vee (\forall x)H[x] \neq (\forall x)(F[x] \vee H[x]);$$
$$(\exists x)F[x] \wedge (\exists x)H[x] \neq (\exists x)(F[x] \wedge H[x]).$$

În cazuri asemănătoare vom proceda în mod special. Deoarece fiecare variabilă legată într-o formulă poate fi considerată doar ca locul în care poate fi introdusă oricare altă variabilă, orice variabilă legată x poate fi redenumită în z şi formula $(\forall x)H[x]$ va trece în $(\forall z)H[z]$, adică $(\forall x)H[x] = (\forall z)H[z]$. Presupunem, că alegem variabila z, care nu este în $F[x]$. Atunci avem

$$(\forall x)F[x] \vee (\forall x)H[x] = (\forall x)F[x] \vee (\forall z)H[z] = (\forall x)(\forall z)(F[x] \vee H[z]).$$

În mod analogic putem scrie

$$(\exists x)F[x] \wedge (\exists x)H[x] = (\exists x)F[x] \wedge (\exists z)H[z] = (\exists x)(\exists z)(F[x] \wedge H[z]).$$

În caz general:

$$(Q_1x)F[x] \vee (Q_2x)H[x] = (Q_1x)(Q_2z)(F[x] \vee H[z]), \qquad (3.22a)$$

$$(Q_3x)F[x] \wedge (Q_4x)H[x] = (Q_3x)(Q_4z)(F[x] \wedge H[z]) \qquad (3.22b)$$

unde Q_1, Q_2, Q_3, şi Q_4 sunt $\forall$ sau $\exists$, iar z nu intră în $F(x)$. Evident, dacă $Q_1 = Q_2 = \exists$ şi $Q_3 = Q_4 = \forall$, variabila x nu va fi schimbată în z, ci vor fi utilizate direct relaţiile (3.21a) şi (3.21b).

Folosind relaţiile (3.4 – 3.12) şi (3.19 – 3.22), orice formulă poate fi adusă la forma normală Prenex confrom algoritmului de mai jos:

1^0. Eliminăm conectorii $\leftrightarrow$ şi $\rightarrow$ folosind relaţiile

$$F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F),$$

$$(F \rightarrow G) = \overline{F} \vee G,$$

2^0. Folosim principiul involuţiei, legile De Morgan şi

$$\overline{((\forall x)F[x])} = (\exists x)\ \overline{(F[x])},$$

$$\overline{((\exists x)F[x])} = (\forall x)\ \overline{(F[x])},$$

pentru a introduce semnul negaţiei în interiorul formulei.

3^0. Redenumim variabilele legate la necesitate.

4^0. Folosim legile (3.19 – 3.22) pentru a scoate cuantificatorii la începutul formulei.

Exemplul 3.19. Să se determine forma normală Prenex a formulei

$$(\forall x)P(x) \rightarrow (\exists x)Q(x).$$

Rezolvare.

$(\forall x)P(x) \rightarrow (\exists x)Q(x) = \overline{((\forall x)P(x))} \vee (\exists x)Q(x)$ conform relaţiei $(F \rightarrow G) = \overline{F} \vee G$,

$= (\exists x)\ \overline{(P[x])} \vee (\exists x)Q(x)$ conform relaţiei $\overline{((\forall x)F[x])} = (\exists x)\ \overline{(F[x])}$,

$= (\exists x)(\overline{(P[x])} \vee Q(x))$ conform relaţiei (3.21*b*). ◄

Exemplul 3.20. Să se determine forma normală Prenex a formulei $(\forall x)(\forall y)(((\exists z)P(x,z) \wedge P(y,z)) \rightarrow (\exists u)Q(x,y,u))$.

Rezolvare.

$(\forall x)(\forall y)(((\exists z)P(x,z) \wedge P(y,z)) \rightarrow (\exists u)Q(x,y,u)) =$

$(\forall x)(\forall y)(\overline{((\exists z)P(x,z) \wedge P(y,z))} \vee (\exists u)Q(x,y,u)) =$

$= (\forall x)(\forall y)((\forall z)(\overline{P(x,z)} \vee \overline{P(y,z)}) \vee (\exists u)Q(x,y,u)) =$

$(\forall x)(\forall y)(\forall z)(\exists u)(\overline{P(x,z)} \vee \overline{P(y,z)} \vee Q(x,y,u))$. ◄

3.8.4. Aplicaţii ale logicii de ordinul unu

În acest paragraf sunt aduse câteva exemple [5] care ilustrează posibilităţile logicii de ordinul unu în rezolvarea unor probleme simple. Ca şi în cazul logicii enunţurilor, va trebui să scriem mai întâi problemele prin formule, apoi să demonstrăm, că formula este identic adevărată sau contradictorie.

Exemplul 3.21. Fie exemplul de la începutul lui 3.8. Sunt date două axiome

A_1: $(\forall x)(OM(x) \rightarrow MURITOR(x))$,

A_2: *(OM(Confucius))*.

Să se demonstreze că *MURITOR(Confucius)* este consecinţă logică a premiselor A_1 şi A_2.

Rezolvare. Avem $A_1 \wedge A_2$: $(\forall x)(OM(x) \rightarrow MURITOR(x)) \wedge (OM(Confucius))$. Dacă $A_1 \wedge A_2$ are valoarea de adevăr *T* într-o interpretare oarecare *I*, atunci ambele formule sunt adevărate în *I*. Deoarece $(OM(x) \rightarrow MURITOR(x))$ este adevărată pentru orice *x*, atunci când înlocuim *x* prin *Confucius* formula $(OM(Confucius) \rightarrow MURITOR(Confucius))$ este la fel adevărată în *I*, adică $\overline{OM(Confucius)} \vee MURITOR(Confucius)$ este adevărată în *I*. Reiese că *MURITOR(Confucius)* trebuie să fie adevărată în *I*, ceea ce conform definiţiei înseamnă că *MURITOR(Confucius)* este consecinţă a premiselor iniţiale $A_1 \wedge A_2$. ◄

Exemplul 3.22. Nici un vânzător de automobile de ocazie nu cumpără astfel de maşini pentru familia sa. Unele persoane, care cumpără maşini de ocazie pentru familiile lor sunt absolut necinstiţi. Putem considera că unii oameni absolut necinstiţi nu sunt vânzători de maşini de ocazie?

Rezolvare. Notăm prin *U(x)* – persoana *x* este vânzător de automobile de ocazie,
B(x) – persoana *x* a cumpărat automobil de ocazie pentru familia sa,
D(x) - persoana *x* este absolut necinstită.
Avem în acest caz:
$A_1 : (\forall x)(U(x) \rightarrow \overline{B(x)})$,
$A_2 : (\exists x)(B(x) \wedge D(x))$.
Trebuie să demonstrăm, că
$A_3 : (\exists x)(D(x) \wedge \overline{U(x)})$
este consecinţă logică a premiselor A_1 şi A_2.

Presupunem, că A_1 şi A_2 sunt adevărate în interpretarea *I* cu domeniul *E*. Deoarece A_2 este adevărată în *I*, există un element *x* din *E*, pe care îl notăm prin *a*, astfel încât $B(a) \wedge D(a)$ este adevărată în *I*. Reiese că *B(a)* este adevărată în *I*, adică $\overline{B(a)}$ este falsă în *I*. A_1 poate fi transcrisă astfel:

$$A_1 : (\forall x)(\overline{U(x)} \vee \overline{B(x)}).$$

Deoarece A_1 este adevărată în *I*, iar $\overline{B(a)}$ este falsă în *I*, $\overline{U(a)}$ trebuie să fie adevărată în *I*. Însă, deoarece $B(a) \wedge D(a)$ este adevărată în *I*, adevărată în *I* este şi *D(a)*. Deci, $D(a) \wedge \overline{U(a)}$ este adevărată în *I*. Astfel

A_3, adică A_3 : $(\exists x)(D(x) \wedge \overline{U(x)})$, este adevărată în I sau A_3 este consecinţă logică a formulelor A_1 şi A_2. ◄

Exemplul 3.23. Unii pacienţi îşi iubesc medicii. Nici un pacient nu-i iubeşte pe vrăciuitori. Putem oare concluziona, că nici un medic nu este vrăciuitor?

Rezolvare. Notăm: $P(x)$: x este pacient,
$D(x)$: x este doctor,
$Q(x)$: x este vrăcuitor,
$L(x,y)$: x îl iubeşte pe y.

Datele iniţiale şi concluzia pot fi reprezentate simbolic după cum urmează:

$$F_1 : (\exists x)(P(x) \wedge (\forall y)(D(y) \rightarrow L(x,y))),$$
$$F_2 : (\forall x)(P(x) \rightarrow (\forall y)(Q(y) \rightarrow \overline{L(x,y)})),$$
$$G : (\forall x)(D(x) \rightarrow \overline{Q(x)}).$$

Trebuie să arătăm, că G este consecinţă logică a F_1 şi F_2.

Fie I o interpretare arbitrară cu domeniul obiectiv D. Presupunem că F_1 şi F_2 sunt adevărate în I. Deoarece F_1, adică $(\exists x)(P(x) \wedge (\forall y)(D(y) \rightarrow L(x,y)))$, este adevărată în I, în D există un element, notat aici prin e, pentru care $(P(e) \wedge (\forall y)(D(y) \rightarrow L(e,y)))$ este adevărată în I. Asta înseamnă, că atât $(P(e)$, cât şi $(\forall y)(D(y) \rightarrow L(e,y)))$ sunt adevărate în I. Pe de altă parte, deoarece $(P(x) \rightarrow (\forall y)(Q(y) \rightarrow \overline{L(x,y)}))$ este adevărată în I pentru toate elementele x din D, evident şi $(P(e) \rightarrow (\forall y)(Q(y) \rightarrow \overline{L(e,y)}))$ este adevărată în I. Deoarece $P(e)$ este adevărată în I, şi $(P(e) \rightarrow (\forall y)(Q(y) \rightarrow \overline{L(e,y)}))$ trebuie să fie adevărată în I. Drept rezultat avem, că pentru orice element y din D, atât $(D(y) \rightarrow L(x,y))$, cât şi $(Q(y) \rightarrow \overline{L(e,y)}))$ sunt adevărate în I. Dacă $D(y)$ este falsă în I, atunci $(D(y) \rightarrow \overline{Q(y)})$ este adevărată în I. Dacă $D(y)$ este adevărată în I, atunci $L(e,y)$ trebuie să fie adevărată în I, deoarece $(D(y) \rightarrow L(e,y))$ este adevărată în I. În rezultat, $Q(y)$ trebuie să fie falsă în I, deoarece $(Q(y) \rightarrow L(e,y))$ este adevărată în I. Deci, $(D(y) \rightarrow \overline{Q(y)})$ este adevărată în I. Din această cauză $(D(y) \rightarrow \overline{Q(y)})$ este adevărată pentru orice y din D, adică $(\forall y)(D(y) \rightarrow \overline{Q(y)})$ este adevărată în I. Am demonstrat în aşa mod, că dacă F_1 şi F_2 sunt adevărate în I, atunci $(\forall y)(D(y) \rightarrow \overline{Q(y)})$ este adevărată în I, adică G este consecinţă logică a F_1 şi F_2. ◄

În exemplele precedente am arătat cum consecinţele pot reieşi din fapte concrete. Argumentarea faptului, că o consecinţă reiese din axiome (premise) se

numeşte *demonstrare*. Procesul de determinare a demonstraţiei se numeşte *procedura demonstraţiei*.

3.9. Logica fuzzy

Logica fuzzy a devenit astăzi o tehnologie extrem de utilă. Aceasta se datorează în mare parte asemănării foarte mari cu procesul de luare a deciziilor de către factorii umani, care pot adopta decizii exacte în baza unor informaţii aproximative.

Spre deosebire de metodele obişnuite, care implică utilizarea unor relaţii matematice exacte pentru descrierea modelelor comportamentului real al sistemelor, metodele logicii fuzzy oferă posibilitatea construirii unei punţi între limbajului natural, care poate conţine multe ambiguităţi şi logica matematică. Logica fuzzy pune la dispoziţie o metodă intuitivă pentru specificarea sistemelor în termeni obişnuiţi omeneşti permiţând automatizarea transformării acestor specificaţii în modele eficiente.

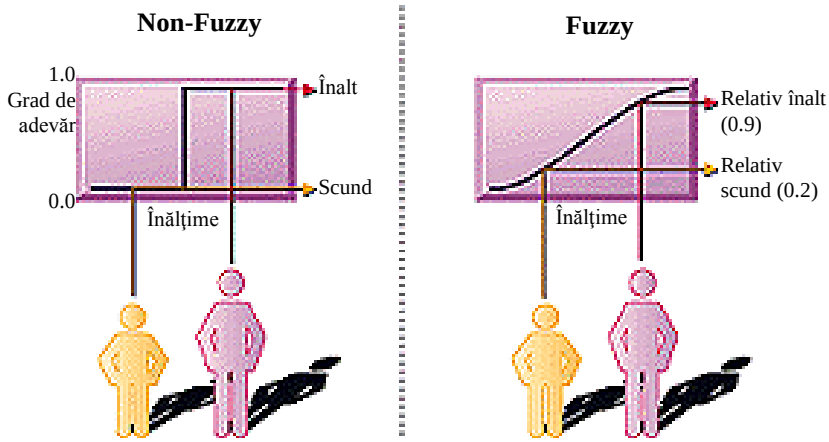


Fig.3.4. Relaţia Fuzzy - Non-Fuzzy

Primele aplicaţii ale logicii fuzzy au fost în industrie. În 1987 în oraşul japonez Sendai a fost deschis primul metrou comandat de logica fuzzy: controlerele, construite în baza logicii fuzzy sporesc confortul călătoriilor în metrou, deoarece pornirea şi oprirea trenurilor este mult mai lină. Dar cel mai important este că toată comanda unui tren se reduce la apăsarea butonului „Start"! Logica fuzzy este pe larg utilizată în domeniul lifturilor, micşorând timpul de aşteptare, în domeniul maşinilor de spălat sau a cuptoarelor cu microunde, sporind satisfacţia gospodinelor.

Bazele logicii fuzzy au fost puse în 1965 de Lotfi A. Zadeh, profesor de informatică la Universitatea Berkeley din California, prin introducerea noţiunii de mulţimi fuzzy (v. capitolul 2). L.Zadeh a motivat următoarele caracteristici obligatorii ale logicii fuzzy:

- în logica fuzzy raţionamentele exacte sunt considerate caz particular al raţionamentelor aproximative;
- în logica fuzzy totul depinde de gradul de apartenenţă;
- orice sistem logic poate fi *fuzzy*-ficat;
- în logica fuzzy cunoştinţele sunt interpretate ca un set de restricţii flexibile (vagi) peste o mulţime de variabile;
- procesul de elaborare a unei concluzii logice este considerat un proces de extindere a restricţiilor flexibile.

Logica booleană este în acest caz o submulţime a logicii fuzzy. În afara celor două valori de adevăr *FALS* şi *ADEVĂRAT* sunt introduse valori intermediare (fuzzy), cum ar fi *PROBABIL FALS*, *POSIBIL*, *PROBABIL ADEVĂRAT*. Operaţia de trecere de la valori concrete numerice din domeniul de cercetare la aceste valori fuzzy se numeşte fuzzyficare, iar operaţia inversă – defuzzyficare. Valorile de adevăr ale unei funcţii fuzzy, ca şi valorile argumentelor, sunt de tipul *FALS*, *PROBABIL FALS*, *POSIBIL*, *PROBABIL ADEVĂRAT* şi *ADEVĂRAT*. De la *FALS* la *ADEVĂRAT* pot fi şi alte valori intermediare. Calcularea valorii de adevăr a funcţiei fuzzy se produce conform unor reguli, legate de domeniul de cercetare, reguli generate de legităţile domeniului, cunoştinţele unor experţi, intuiţie sau anumiţi factori obiectivi şi chiar subiectivi.

Exemplul care urmează [6] descrie posibilitatea utilizării logicii fuzzy în proiectarea controlerului unui semafor inteligent. Controlerul are ca funcţie modificarea valorii curente a duratei ciclului de comutare a culorilor în dependenţă de intensitatea traficului (fluxului de automobile).

Într-un semafor standard durata ciclului de comutare a unei culori este constantă, ceea ce nu poate fi optimal. Ideal ar fi ca durata ciclului de comutare să varieze în aşa mod, încât într-o unitate de timp „pe verde" să poată trece cât mai multe automobile, iar "pe roşu" să aştepte cât mai puţine. Elaborarea matematică a modelului unui atare sistem în cadrul logicii obişnuite este o problemă foarte dificilă. Logica fuzzy este de un ajutor esenţial în acest caz.

Schema unei intersecţii este reprezentată in fig.3.5. La cele 4 semafoare tradiţionale sunt adăugate opt contoare *c1 – c8*, repartizate conform schemei.

Contoarele plasate mai aproape de semafor (contoarele cu numerele pare) duc evidenţa automobilelor, care au trecut intersecţia pe perioada unui ciclu, iar contoarele cu numerele impare calculează numărul de automobile, care se

îndreaptă spre semafor. Numărul de automobile, care se află înaintea semaforului este determinat de diferenţa celor două contoare. De exemplu, numărul de automobile din faţa semaforului străzii „Sud" este egal cu *c3-c4*. Distanţa D dintre două contoare permite calcularea densităţii critice (densitatea maximală, permisă pentru aşteptare în caz de situaţie apropiată de saturaţie). În acest scop se va calcula numărul de unităţi de transport pentru ambele sensuri (Nord-Sud sau Vest-Est) de deplasare, iar suma se va împărţi la *2D*. Pentru direcţia Est-Vest vom avea [(*c1-c2*)+(*c5-c6*)]/(*2D*).

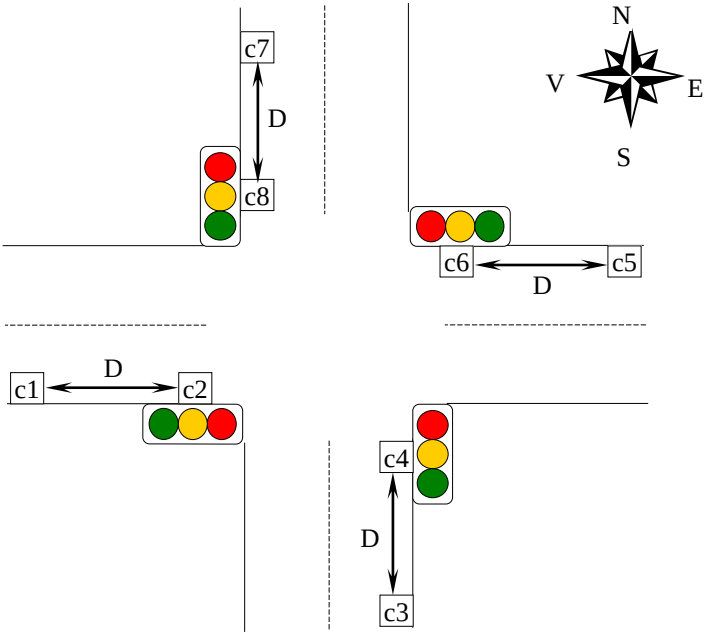


Fig.3.5. Schema unei intersecţii

Procesul de decizie fuzzy presupune executarea următorilor trei paşi:

1º. ***Fuzzyficarea.*** Trebuie să determinăm intrările şi ieşirile proiectului. Fie că culoarea roşie este pentru străzile Nord şi Sud, distanţa *D* este constantă, iar intrările modelului sunt

1) Durata ciclului de comutare a culorii,
2) Numărul de automobile în faţa semafoarelor pe roşu,
3) Numărul de automobile în faţa semafoarelor pe verde.

Prin numărul de automobile în faţa semafoarelor înţelegem numărul maxim de automobile în ambele direcţii.

Prin ***ieşirea sistemului*** vom înţelege probabilitatea schimbării valorii curente a duratei ciclului de comutare – p_c.

Vom exprima în termeni matematici situaţiile posibile, adică vom pune în corespondenţă diapazoanelor posibile ale duratei ciclului şi ale numărului de automobile valori fuzzy - nivele. Pentru intrările 1 şi 2 prin valoare nulă (durată nulă a ciclului sau număr nul de automobile) vom considera cazul, când valoarea duratei ciclului este între 0 şi 2 unităţi de timp sau există de la **0** până la două unităţi de transport, adică **nul** – **(0, 2).** Analogic **jos** – **(1, 7)**, **mediu** – **(4, 11)**, **înalt** – **(7, 18)** şi **haos** – **(14, 20)**. De la **nul** până la **haos** funcţiile de apartenenţă acoperă cinci diapazoane.

Analogic pentru intrarea 3: nivel **relativ scurt –** de la **0** până la **15** automobile**, scurt – (10, 35), mediu – (18, 60), lung – (35, 88), foarte lung – (65, 100)** şi **mărginit – (85, 100)**. Funcţia de apartenenţă acoperă 6 diapazoane.

Valorile ieşirii (probabilitatea schimbării valorii curente a ciclului de comutare - p_c), asociate nivelelor vor fi: **nu – 0, probabil nu – 0.25**, **posibil – 0.5**, **probabil da – 0.75** şi **da – 1.0**, cinci valori pentru cinci nivele. Observăm, că pentru ieşire o valoare este asociată unui singur nivel şi nu unui diapazon de valori.

2º. ***Elaborarea semnalului fuzzy de comandă.*** Regulile de calculare a gradului de apartenenţă pot fi formulate utilizând o serie de afirmaţii logice de tipul *DACĂ-ATUNCI*, unite cu ajutorul conectorilor *ŞI-SAU*. De exemplu: *DACĂ durata ciclului este* ***medie*** *ŞI numărul de automobile pe roşu este* ***jos*** *ŞI numărul de automobile pe verde este* ***mediu****, ATUNCI modificarea valorii duratei ciclului –* ***probabil nu***. Pentru cazul nostru (3 intrări cu 5, 5 şi 6 diapazoane pentru funcţiile de apartenenţă) sunt posibile 150 de combinaţii. Studiind mai atent problema, folosind criterii speciale, numărul de combinaţii, care prezintă interes poate fi micşorat. La acest pas se va obţine valoarea funcţiei de apartenenţă pentru fiecare dintre cele 5 valori posibile ale ieşirii.

3º. ***Defuzzyficarea.*** Procesul de defuzzyficare transformă ieşirea fuzzy într-o valoare concretă non-fuzzy. Decizia finală (se va comuta sau nu culoarea semaforului) va fi luată în dependenţă de valoarea non-fuzzy a variabile p_c. De exemplu dacă valoarea calculată (care depinde de metoda adoptată pentru defuzzyficare) este mai mare ca 0,5, atunci culoarea va fi comutată, în caz contrar – nu. Prezentăm mai jos o formulă de dificultate medie de calculare a valorii numerice a variabilei p_c:

$$p_c = \sum_{i=1}^{n} (f_{a_i} * val(poz)) / \sum_{i=1}^{n} f_{a_i} ,$$

în care f_{a_i} este valoarea funcţiei de apartenenţă, calculată la pasul precedent pentru fiecare din nivelurile ieşirii, iar *val(poz)* – valoarea respectivă precisă din relaţia „nivel – valoare a ieşirii".

De exemplu, dacă după calcularea valorilor funcţiei de apartenenţă conform regulilor de mai sus obţinem tabelul următor

Valoarea fuzzy a ieşirii	**Valoarea f_a**
Nu se va schimba	0,1
Probabil nu	0,3
Posibil	0,9
Probabil da	0,6
Da	0,0

atunci valoarea precisă a ieşirii va fi

$$p_c=(0,1*0+0,3*0,25+0,9*0,5+0,6*0,75+0.0*1.0)/(0,1+0,3+0,9+0,6+0,0)=0,513,$$

şi decizia este *culoarea va fi comutată*.

Acest semafor a fost testat pentru 7 tipuri de trafic (de la trafic foarte slab până la foarte intens). În conformitate cu aceste condiţii de trafic au fost alese la întâmplare 35 de densităţi ale automobilelor şi ordonate în dependenţă de perioada zilei. Acelaşi lucru s-a făcut şi pentru un semafor obişnuit şi un expert în reglarea circulaţiei. Pentru comparare în calitate de criterii a fost luat numărul de automobile, cărora a fost permisă trecerea şi timpul mediu de aşteptare. A fost elaborat un indice de performanţă, care maximizează traficul şi minimizează timpul mediu de aşteptare.

Rezultatele comparării celor trei tipuri de reglare a traficului sunt prezentate în figura 3.6. Din aceste grafice poate fi făcută următoarea concluzie: controlerul fuzzy a permis trecerea unui număr de automobile cu 31% mai mare şi cu timpul mediu de aşteptare cu 5% mai mic decât un semafor obişnuit. Eficienţa a fost mai mare cu 71%, ceea ce era de aşteptat. Chiar şi în comparaţie cu un operator uman controlerul fuzzy a permis trecerea a unui număr mai mare de automobile (plus 14%), cu valoarea timpului mediu de aşteptare mai mică şi cu indicile de performanţă cu 36% mai înalt.

Logica fuzzy permite crearea unor maşini-automate mai performante, care sporesc nivelul de trai al omului.

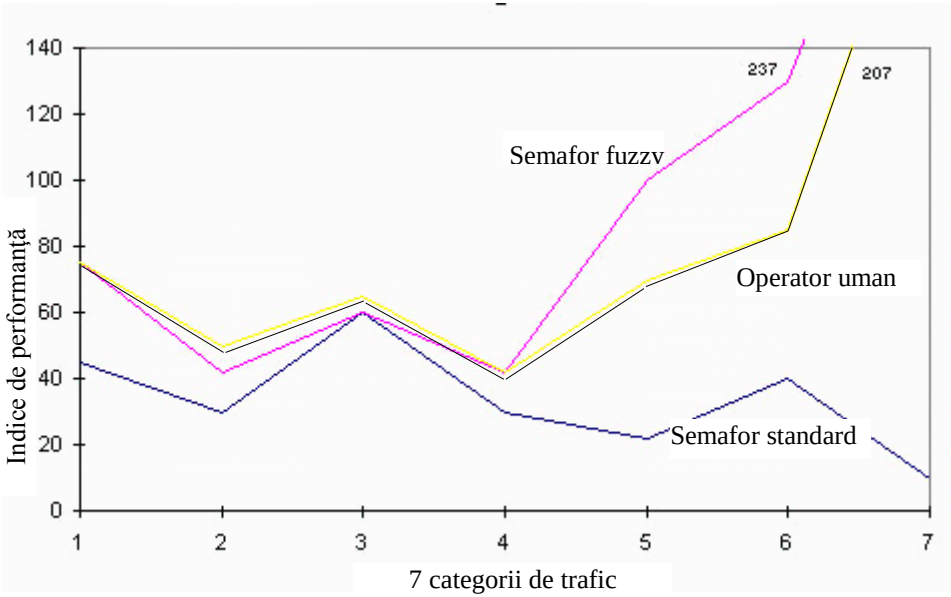


Fig.3.6. Compararea indicilor de performanţă

3.10. Exerciţii şi probleme

3.1. Construiţi tabelele de adevăr, determinaţi formele canonice disjunctivă şi conjunctivă, stabiliţi forma minimă şi implementaţi în bazele "ŞI-NU", "SAU-NU" funcţiile logice:

$$y_1 = \overline{((x_1x_2 \oplus x_3x_4) \vee (\overline{x_1x_3} \to x_2x_4)) \uparrow ((x_1 \mid x_4) \sim (x_2 \vee x_3))}$$

$$y_2 = \overline{((x_1x_2 \vee x_3x_4) \oplus (\overline{x_1x_3} \uparrow x_2x_4)) \to ((x_1 \mid x_4) \vee (x_2 \sim x_3))}$$

$$y_3 = \overline{((x_1x_2 \to x_3x_4) \oplus (\overline{x_1x_3} \uparrow x_2x_4)) \vee ((x_1 \sim x_4) \vee (x_2 \mid x_3))}$$

3.2. Într-o clădire cu *n* etaje există *m* lifturi, comandate la fiecare etaj cu ajutorul unui singur buton. Să se determine funcţia logică a butonului unui etaj dat, care având la intrare starea lifturilor, va apela cel mai apropiat lift liber. Să se construiască circuitul logic ataşat butonului unui etaj oarecare pentru *n=20* şi *m=5*.

3.3. Elaboraţi circuitul logic, care va vizualiza pe un dispozitiv cu 7 elemente - cristale lichide (exemplul unui ceas electronic) cifrele de la 0 la 9, prezentate în cod binar.

3.4. Exprimaţi în termeni ai mulţimilor fuzzy situaţia „temperatura într-o sală de studiu"

a) din punctul de vedere al studenţilor ☹,

b) din punctul de vedere al administraţiei facultăţii ☺.

3.5. Cum poate fi exprimată noţiunea „persoană tânără" în cadrul mulţimilor fuzzy?

3.6. Elaboraţi modelul matematic al procesului „examen" – obţinerea notei de examinare în dependenţă de nivelul cunoştinţelor, dificultatea cursului, stricteţea profesorului.

4. GRAFURI

Anul 1736 este considerat pe bună dreptate de început pentru teoria grafurilor. În acel an L.Euler a rezolvat problema despre podurile din Königsberg, stabilind criteriul de existenţă în grafuri a unui circuit special, denumit astăzi ciclu Euler. Acestui rezultat i-a fost hărăzit să fie mai bine de un secol unicul în teoria grafurilor. Doar la jumătatea secolului XIX inginerul G.Kirchof a elaborat teoria arborilor pentru cercetarea circuitelor electrice, iar matematicianul A.Caly a rezolvat problema enumerării pentru trei tipuri de arbori. În aceeaşi perioadă apare şi cunoscuta problemă despre patru culori.

Avându-şi începuturile în rezolvarea unor jocuri distractive (problema calului de şah şi reginelor, "călătoriei în jurul Pământului" despre nunţi şi haremuri, etc.), astăzi teoria grafurilor s-a transformat într-un aparat simplu şi accesibil, care permite rezolvarea unui cerc larg de probleme. Grafuri întâlnim practic peste tot. Sub formă de grafuri pot fi reprezentate sisteme de drumuri şi circuite electrice, hărţi geografice şi molecule chimice, relaţii dintre oameni şi grupuri de oameni.

Foarte fertile au fost pentru teoria grafurilor ultimele trei decenii, ceea ce a fost cauzat de creşterea spectaculoasă a domeniilor de aplicaţii. În termeni grafo-teoretici pot fi formulate o mulţime de probleme legate de obiecte discrete. Astfel de probleme apar la proiectarea circuitelor integrate şi sistemelor de comandă, la cercetarea automatelor finite, circuitelor logice, schemelor-bloc ale programelor, în economie şi statistică, chimie şi biologie, teoria orarelor şi optimizarea discretă. Teoria grafurilor a devenit o parte componentă a aparatului matematic al ciberneticii, limbajul matematicii discrete.

4.1. Noţiuni generale

4.1.1. Definiţia grafului

Se numeşte graf, ansamblul format dintr-o mulţime finită X şi o aplicaţie F a lui X în X. Se notează $G = (X,F)$. Numărul elementelor mulţimii X determină ordinul grafului finit. Dacă *card* $X = n$, graful $G = (X,F)$ se numeşte *graf finit de ordinul n*. Elementele mulţimii X se numesc *vârfurile* grafului. Geometric, vârfurile unui graf le reprezentăm prin puncte sau cerculeţe. Perechea de vârfuri (x,y) se numeşte *arc*; vârful x se numeşte originea sau extremitatea iniţială a arcului (x,y), iar vârful y se numeşte extremitatea finală sau terminală. Un arc (x,y) îl reprezentăm geometric printr-o săgeată orientată de la vârful x la vârful y.

Dacă un vârf nu este extremitatea nici unui arc el se numeşte *vârf izolat*, iar dacă este extremitatea a mai mult de două arce - *nod*. Un arc pentru care extremitatea iniţială coincide cu cea finală se numeşte *buclă*.

Arcele unui graf le mai notăm şi cu u_1, u_2,..., iar mulţimea arcelor grafului o notăm cu U. Se observă că mulţimea U a tuturor arcelor unui graf determină complet aplicaţia F, precum şi reciproc, aplicaţia F determină mulţimea U a arcelor grafului. Un graf G poate fi dat fie prin ansamblul (X,F) fie prin ansamblul (X,U).

Două *arce* se zic *adiacente* dacă sunt distincte şi au o extremitate comună. Două *vârfuri* se zic *adiacente* dacă sunt distincte şi sunt unite printr-un arc.

Un arc (x,y) se spune că este *incident* cu vârful x *spre exterior* şi este *incident* cu vârful y *spre interior*.

Fie $G = (X,F)$ şi $x \in X$. Mulţimea tuturor arcelor incidente cu x spre exterior (interior) se numeşte semigradul exterior (interior) a lui x şi se notează d^+x (d_x). Dacă pentru un vârf x, $d^+x=0$ sau $d\_x=0$ atunci el se numeşte vârf terminal

Într-un graf $G = (X,U)$ se numeşte drum un şir de arce $(u_1,...,u_k)$, astfel încât extremitatea terminală a fiecărui arc u_l coincide cu extremitatea iniţială a arcului următor u_{i+1}. Un drum care foloseşte o singură dată fiecare arc al său se numeşte drum simplu. Un drum care trece o singură dată prin fiecare vârf al său se numeşte drum elementar. Lungimea unui drum este numărul de arce din care este compus drumul.

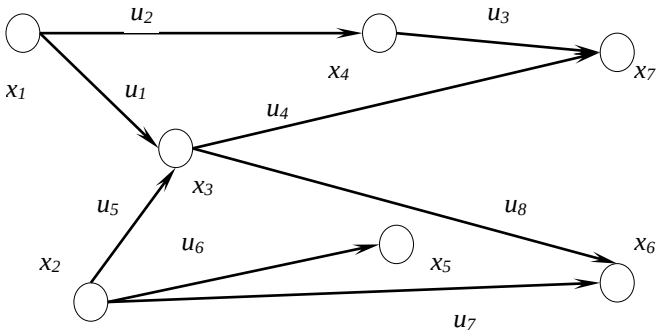


Fig. 4.1. Exemplu de graf

Un drum elementar ce trece prin toate vârfurile grafului se numeşte drum hamiltonian. Un drum simplu ce conţine toate arcele grafului se numeşte drum

eulerian. Un drum finit pentru care vârful iniţial coincide cu vârful terminal se numeşte circuit.

Graful obţinut din graful iniţial suprimând cel puţin un vârf al acestuia precum şi toate arcele incidente cu el se numeşte subgraf. Graful obţinut suprimând cel puţin un arc se numeşte graf parţial.

Un graf se numeşte complet dacă oricare ar fi x şi y din X există un arc de la x la y sau de la y la x.

Un graf $G = (X,F)$ se zice tare conex dacă pentru orice $x, y \in X$ (x diferit de y) există un drum de la x la y sau că oricare pereche de vârfuri x, y cu x diferit de y se află pe un circuit.

Fie $G = (X,F)$ şi $x \in X$. Mulţimea C_x formată din toate vârfurile $x_i \in X$ pentru care există un circuit ce trece prin x şi x_i se numeşte componentă tare conexă a lui G corespunzătoare vârfului x.

Componentele tari conexe ale unui graf $G = (X,F)$ constituie o partiţie a lui X.

Noţiunile introduse sunt valabile pentru grafurile orientate.

În cazul în care orientarea arcelor nu are nici o importanţă graful se va numi *neorientat*. Arcele se vor numi *muchii*, drumul - *lanţ*, circuitul - *ciclu*. La fel se introduce noţiunea de lanţ elementar şi simplu, lanţ Euler şi hamiltonian. Graful tare conex se va numi conex.

Cele două concepte de graf orientat şi graf neorientat se pot sprijini în practică unul pe altul. De la un graf orientat se poate trece la omologul său neorientat când se abordează o problemă ce nu presupune orientarea şi invers, dacă se precizează orientarea.

Unui graf orientat simetric i se poate asocia un graf neorientat, legătura dintre două vârfuri x şi y realizată de cele două arce (x,y) şi (y,x) de sensuri contrarii înlocuindu-se cu muchia $[x,y]$. De asemenea un graf neorientat poate fi identificat cu mai multe grafuri orientate înlocuind fiecare muchie cu două arce orientate în sens opus.

Fie $G = (X,U)$ un graf neorientat şi $x \in X$. Se numeşte gradul vârfului x numărul muchiilor care au o extremitate în vârful x. Se notează $g(x)$. Un vârf este izolat dacă $g(x) = 0$.

4.1.2. Număr cociclomatic şi număr ciclomatic

Fie $G = (X,F)$ un graf neorientat cu n vârfuri, m muchii şi p componente conexe. Numim număr cociclomatic asociat grafului G numărul

$$r(G) = n - p$$

iar numărul ciclomatic numărul

$$s(G) = m - r(G) = m - n + p.$$

Se numeşte multigraf un graf neorientat în care există perechi de vârfuri unite prin mai multe muchii. Se numeşte q - graf un multigraf pentru care numărul maxim de muchii ce unesc două vârfuri este q.

Teorema. Fie $G = (X,U)$ un multigraf, iar $G_1 = (X,U_1)$ un multigraf obţinut din G adăugând o muchie. Dacă $x, y \in X$ şi $[x,y]$ este muchia care se adaugă la mulţimea muchiilor grafului G, atunci:
(1) dacă $x = y$ sau x şi y sunt unite printr-un lanţ
$r(G_1) = r(G)$, $s(G_1) = s(G) + 1$
(2) în caz contrar
$r(G_1) = r(G) + 1$, $s(G_1) = s(G)$.

Demonstraţia este evidentă.

Consecinţă. Numerele cociclomatic şi ciclomatic sunt nenegative.

4.1.3. Număr cromatic. Grafuri planare. Arbori

Un graf $G = (X,U)$ se zice ca este *graf p-cromatic* daca vârfurile lui se pot colora cu p-culori distincte astfel ca două vârfuri adiacente să nu fie de aceeaşi culoare. *Cel mai mic* număr întreg şi pozitiv pentru care graful este *p-cromatic* se numeşte *număr cromatic*.

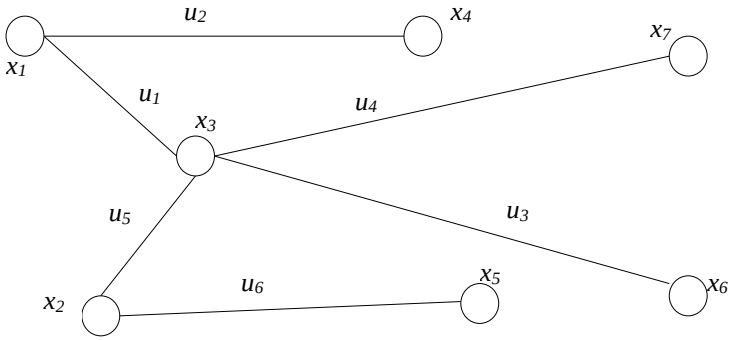


Fig. 4.2. Exemplu de arbore

Un graf se zice că este *planar* dacă poate fi reprezentat pe un plan astfel ca două muchii să nu aibă puncte comune în afară de extremităţile lor. Un graf planar determină regiuni numite *feţe*. O faţă este o regiune mărginită de muchii şi

care nu are în interior nici vârfuri, nici muchii. Conturul unei feţe este format din muchiile care o mărginesc. Două feţe sunt adiacente dacă contururile au o muchie comună. S-a demonstrat că numărul cromatic al unui graf planar este patru.

Cu privire la numărul cromatic s-a demonstrat următoarea

Teorema (König). Un graf este bicromatic dacă şi numai dacă nu are cicluri de lungime impară.

Se numeşte arbore oricare graf conex fără bucle şi cicluri.

Se numeşte arborescenţă un graf orientat fără circuite astfel ca să existe un vârf şi numai unul care nu e precedat de nici un alt vârf (rădăcina) şi orice alt vârf să fie precedat de un singur vârf.

4.2. Metode de reprezentare a grafului

Există trei metode de bază de definire a unui graf:

1. *Matricea de incidenţă;*
2. *Matricea de adiacenţă;*
3. *Lista de adiacenţă (incidenţă).*

Vom face cunoştinţă cu fiecare dintre aceste metode.

4.2.1. Matricea de incidenţă

Este o matrice de tipul *mxn*, în care *m* este numărul de muchii sau arce (pentru un graf orientat), iar *n* este numărul vârfurilor. La intersecţia liniei *i* cu coloana *j* se vor considera valori de *0* sau *1* în conformitate cu următoarea regulă:

- *1* - dacă muchia *i* este incidentă cu vârful *j* (dacă arcul *i* "intră" în vârful *j* în cazul unui graf orientat);
- *0* - dacă muchia (arcul) *i* şi vârful *j* nu sunt incidente;
- *-1* - numai pentru grafuri orientate, dacă arcul *i* "iese" din vârful *j*.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
u_1	-1	0	1	0	0	0	0
u_2	-1	0	0	1	0	0	0
u_3	0	0	0	-1	0	0	1
u_4	0	0	-1	0	0	0	1
u_5	0	-1	1	0	0	0	0
u_6	0	-1	0	0	1	0	0
u_7	0	-1	0	0	0	1	0
u_8	0	0	-1	0	0	1	0

Fig. 4.3. Exemplu de matrice de incidenţă (v.fig.1)

Este uşor de observat că această metodă este de o eficacitate mică în sensul utilizării memoriei calculatorului: fiecare linie conţine doar două elemente diferite de zero (o muchie poate fi incidentă cu nu mai mult de două vârfuri).

În limbajul Pascal matricea de incidenţă poate fi redată printr-un tablou bidimensional *mxn*:

```
Matr_Incd: array [1..LinksCount, 1..TopsCount] of integer;
```

4.2.2. Matricea de adiacenţă

Este o matrice pătrată *nxn*, aici *n* este numărul de vârfuri. Fiecare element poate fi *0*, dacă vârfurile respective nu sunt adiacente, sau 1, în caz contrar. Pentru un graf fără bucle putem observa următoarele:

- diagonala principală este formată numai din zerouri;
- pentru grafuri neorientate matricea este simetrică faţă de diagonala principală.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	0	0	1	1	0	0	0
x_2	0	0	1	0	1	1	0
x_3	0	0	0	0	0	1	1
x_4	0	0	0	0	0	0	1
x_5	0	0	0	0	0	0	0
x_6	0	0	0	0	0	0	0
x_7	0	0	0	0	0	0	0

Fig. 4.4. Exemplu de matrice de adiacenţă (v.fig.1)

În limbajul Pascal matricea de adiacenţă poate fi reprezentată în modul următor:

```
Matr_Ad :array [1..TopsCount,1..TopsCount] of byte,
```

sau

```
Matr_Ad :array [1..TopsCount,1..TopsCount] of boolean;
```

După cum este lesne de observat şi în acest caz memoria calculatorului este utilizată nu prea eficace din care cauză matricea de adiacenţă ca şi matricea de incidenţă se vor utiliza de obicei doar în cazul în care se va rezolva o problemă concretă pentru care reprezentarea grafului în această formă aduce unele facilităţi algoritmului respectiv.

Pentru păstrarea grafurilor în memoria calculatorului (în deosebi, memoria externă) se va utiliza una din posibilităţile de mai jos.

4.2.3. Lista de adiacenţă şi lista de incidenţă

Lista de adiacenţă este o listă cu *n* linii (după numărul de vârfuri *n*), în linia cu numărul *i* vor fi scrise numerele vârfurilor adiacente cu vârful *i*.

Lista de incidenţă se defineşte în mod analogic cu deosebirea că în linia *i* vor fi scrise numerele muchiilor (arcelor) incidente cu vârful *i*.

Reprezentarea grafurilor prin intermediul acestor liste permite utilizarea mai eficace a memoriei calculatorului, însă aceste forme sunt mai complicate atât în realizare, cât şi în timpul procesării. Pentru a lua în consideraţie lungimea variabilă a liniilor vor fi utilizate variabile dinamice şi pointeri.

Vom exemplifica pentru un graf cu *n* vârfuri. Deoarece fiecare element al listei conţine numere de vârfuri este evident să considerăm că vom avea un şir de variabile dinamice de tip INTEGER care se vor afla în relaţia respectivă de precedare (succedare). Această relaţie se va realiza prin pointeri, uniţi împreună cu variabila de tip întreg în înregistrarea (Pascal: record). Pentru a păstra indicatorii de intrare în aceste şiruri se va folosi un tablou unidimensional de indicatori de lungime *n*. În calitate de simbol de terminare a şirului se va utiliza un simbol care nu a fost folosit la numeraţia vârfurilor (de exemplu *0*), care va fi introdus în calitate de variabilă de tip întreg al ultimului bloc.

De exemplu, lista de adiacenţă (fig.1.1):

1 - 3, 4, 0
2 - 3, 5, 6, 0
3 - 6, 7, 0
4 – 7, 0
5 - 0
6 - 0
7 - 0

va avea reprezentare internă din fig.4.5.

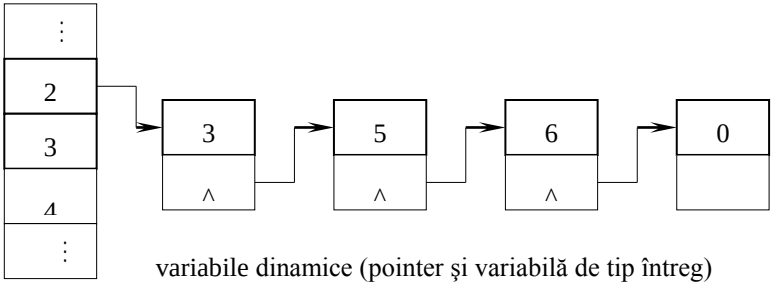


Fig. 4.5. Reprezentarea internă a listei de adiacenţă

Va fi necesar de realizat următoarea declaraţie:

```
Type
        BlockPtr = ^BlockType;
        BlockType = record;
        Number : integer;
        Point : BlockPtr;
        end;
Var In_Ptr :array [1..TopsCount] of BlockPtr;
```

Lista de adiacenţă (şi realizarea reprezentării interne) se va implementa cu ajutorul procedurii New(BlockPtr_Type_Variable), în care BlockPtr_Type_Variable este o variabilă de tip BlockPtr.

4.3. Structuri de date

4.3.1. Liste

Fiecare tip de listă defineşte o mulţime de şiruri finite de elemente de tipul declarat. Numărul de elemente care se numeşte lungimea listei poate varia pentru diferite liste de acelaşi tip. Lista care nu conţine nici un element se va numi vidă. Pentru listă sunt definite noţiunile începutul, sfârşitul listei şi respectiv primul şi ultimul element, de asemenea elementul curent ca şi predecesorul şi succesorul elementului curent. Element curent se numeşte acel unic element care este accesibil la momentul dat.

4.3.2. Fire de aşteptare

Firele de aşteptare (*FA*, rând, coadă, şir de aşteptare) se vor folosi pentru a realiza algoritmul de prelucrare a elementelor listei în conformitate cu care elementele vor fi eliminate din listă în ordinea în care au fost incluse în ea (primul sosit - primul servit).

Operaţiile de bază cu firele de aşteptare:

- Formarea unui FA vid;
- Verificare dacă FA nu este vid;
- Alegerea primului element cu eliminarea lui din FA;
- Introducerea unei valori noi în calitate de ultim element al FA.

4.3.3. Stive

Stiva se utilizează pentru a realiza algoritmul de prelucrare a elementelor după principiul "ultimul sosit - primul prelucrat" (LIFO).

Operaţiile de bază cu stivele sunt următoarele:

- Formarea unei stive vide;

- Verificare la vid;
- Alegerea elementului din topul stivei cu sau fără eliminare;
- Introducerea unui element nou în topul stivei.

4.3.4. Arbori

Se va defini o mulţime de structuri fiecare din care va consta dintr-un obiect de bază numit *vârf* sau *rădăcina arborelui* dat şi o listă de elemente din mulţimea definită, care (elementele) se vor numi *subarbori* ai arborelui dat. Arborele pentru care lista subarborilor este vidă se va numi *arbore trivial*, iar rădăcina lui - *frunză*.

Rădăcina arborelui se va numi tatăl vârfurilor care servesc drept rădăcini pentru subarbori; aceste vârfuri se vor mai numi copiii rădăcinii arborelui: rădăcina primului subarbore se va numi *fiul cel mai mare*, iar rădăcina fiecărui subarbore următor în listă se va numi frate.

Operaţiile de bază pentru arbori vor fi:

- Formarea unui arbore trivial;
- Alegerea sau înlocuirea rădăcinii arborelui;
- Alegerea sau înlocuirea listei rădăcinilor subarborilor;
- Operaţiile de bază care sunt valabile pentru liste.

4.4. Algoritmi pe grafuri

4.4.1. Căutare în adâncime

La căutarea în adâncime (parcurgerea unui graf în sens direct, în preordine) vârfurile grafului vor fi vizitate în conformitate cu următoarea procedură recursivă: *mai întâi va fi vizitată rădăcina arborelui q, apoi, dacă rădăcina arborelui nu este frunză - pentru fiecare fiu p al rădăcinii q ne vom adresa recursiv procedurii de parcurgere în adâncime pentru a vizita vârfurile tuturor subarborilor cu rădăcina p ordonate ca fii ai lui q*.

În cazul utilizării unei stive pentru păstrarea drumului curent pe arbore, drum care începe din rădăcina arborelui şi se termină cu vârful vizitat în momentul dat, poate fi realizat un algoritm nerecursiv de forma:

Vizitează rădăcina arborelui şi introdu-o în stiva vidă *S*;
WHILE stiva *S* nu este vidă DO
    BEGIN
    fie *p* - vârful din topul stivei *S*;
    IF fiii vârfului *p* încă nu au fost vizitaţi
    THEN vizitează fiul mai mare al lui *p* şi introduce-l în *S*

```
            ELSE BEGIN
                    elimină vârful p din stiva S
                    IF p are fraţi THEN vizitează pe fratele lui p şi
            introduce-l în stiva S
            END
    END
```

Acest algoritm poate fi modificat pentru a putea fi utilizat la parcurgerea tuturor vârfurilor unui graf arbitrar. În algoritmul de mai jos se va presupune că este stabilită o relaţie de ordine pe mulţimea tuturor vârfurilor grafului, iar mulţimea vârfurilor adiacente cu un vârf arbitrar al grafului este de asemenea ordonată:

```
WHILE va exista cel puţin un vârf care nu a fost vizitat DO
    BEGIN
    fie p - primul din vârfurile nevizitate;
    vizitează vârful p şi introduce-l în stiva vidă S;
            WHILE stiva S nu este vidă DO
                    BEGIN
                    fie p - vârful din topul stivei S;
                    IF m vârfuri ale lui p sunt vârfuri adiacente
                       nevizitate
                            THEN BEGIN
                               fie z primul vârf nevizitat din vârfurile
                               adiacente cu p;
                               parcurge muchia (p,z), vizitează vârful
                               z şi introduce-l în stiva S;
                            END
                       ELSE elimină vârful p din stiva S
                    END
    END
```

În cazul în care se va lucra cu un graf conex arbitrar cu relaţia de ordine lipsă, nu va mai avea importanţă ordinea de parcurgere a vârfurilor. Propunem un algoritm care utilizează mai larg posibilităţile stivei, cea ce face programul mai efectiv în sensul diminuării timpului de calcul necesar. De exemplu, acest algoritm în varianta recursivă este pe larg utilizat în programele de selectare globală în subdirectori (cazul programelor antivirus).

```
Introdu în stivă vârful iniţial şi marchează-l;
WHILE stiva nu este vidă DO
    BEGIN
    extrage un vârf din stivă;
    IF există vârfuri nemarcate adiacente cu vârful extras
```

THEN marchează-le şi introduce-le în stivă;
END

4.4.2. Algoritmul de căutare în lărgime

Parcurgerea grafului în lărgime, ca şi parcurgerea în adâncime, va garanta vizitarea fiecărui vârf al grafului exact o singură dată, însă principiul va fi altul. După vizitarea vârfului iniţial, de la care va începe căutarea în lărgime, vor fi vizitate toate vârfurile adiacente cu vârful dat, apoi toate vârfurile adiacente cu aceste ultime vârfuri ş.a.m.d. până vor fi vizitate toate vârfurile grafului. Evident, este necesar ca graful să fie conex. Această modalitate de parcurgere a grafului (în lărgime sau postordine), care mai este adesea numită parcurgere în ordine orizontală, realizează parcurgerea vârfurilor de la stânga la dreapta, nivel după nivel.

Algoritmul de mai jos realizează parcurgerea în lărgime cu ajutorul a două fire de aşteptare O_1 şi O_2.

Se vor forma două fire de aşteptare vide O_1 şi O_2;

Introduce rădăcina în FA O_1;

While cel puţin unul din firele de aşteptare O_1 sau O_2 nu va fi vid *do*

If O_1 nu este vid *then*

begin
fie *p* vârful din topul FA O_1;
vizitează vârful *p* eliminându-l din O_1;
vizitează pe toţi fiii lui *p* în FA O_2, începând cu cel mai mare;

END

ELSE
în calitate de O_1 se va lua FA O_2, care nu este vid,
iar în calitate de O_2 se va lua FA vid O_1;

Vom nota că procedura parcurgerii grafului în lărgime permite să realizăm arborele de căutare şi în acelaşi timp să construim acest arbore. Cu alte cuvinte, se va rezolva problema determinării unei rezolvări sub forma vectorului (a_1, a_2,...) de lungime necunoscută, dacă este cunoscut că există o rezolvare finită a problemei.

Algoritmul pentru cazul general este analogic cu cel pentru un graf în formă de arbore cu o mică modificare care constă în aceea că fiecare vârf vizitat va fi marcat pentru a exclude ciclarea algoritmului.

Algoritmul parcurgerii grafului în lărgime:

Se vor defini două FA O_1 şi O_2 vide;

Introdu vârful iniţial în FA O_1 şi marchează-l;

```
WHILE FA O1 nu este vid DO
BEGIN
        vizitează vârful din topul FA O1 şi elimină-l din FA;
        IF există vârfuri nemarcate adiacente cu vârful dat THEN
        introdu-le în FA O2;
END
```

4.4.3. Graf de acoperire

Fie *H* un subgraf care conţine toate vârfurile unui graf arbitrar *G*. Dacă pentru fiecare componentă de conexitate a lui *G* subgraful *H* va defini un arbore atunci *H* se va numi graf de acoperire (scheletul sau carcasă) grafului G. Este evident că graful de acoperire există pentru oricare graf: eliminând ciclurile din fiecare componentă de conexitate, adică eliminând muchiile care sunt în plus, vom ajunge la graful de acoperire.

Se numeşte graf aciclic orice graf care nu conţine cicluri. Pentru un graf arbitrar *G* cu *n* vârfuri şi *m* muchii sunt echivalente următoarele afirmaţii:

1. *G* este arbore;
2. *G* este un graf conex şi $m = n - 1$;
3. *G* este un graf aciclic şi $m = n - 1$;
4. oricare două vârfuri distincte (diferite) ale lui *G* sunt unite printr-un lanţ simplu care este unic;
5. *G* este un graf aciclic cu proprietatea că, dacă o pereche oarecare de vârfuri neadiacente vor fi unite cu o muchie, atunci graful obţinut va conţine exact un ciclu.

Consecinţă: numărul de muchii pentru un graf arbitrar G, care va fi necesar a fi eliminate spre a obţine un graf de acoperire nu depinde de ordinea eliminării lor şi este egal cu

$$m(G)-n(G)+k(G),$$

unde $m(G)$, $n(G)$ şi $k(G)$ sunt numărul de muchii, vârfuri şi componente conexe, respectiv.

Numărul $s(G) = m(G)-n(G)+ k(G)$ se numeşte *rang ciclic* sau număr *ciclomatic* al grafului *G*. Numărul $r(G) = n(G)-k(G)$ – *rang cociclomatic* sau *număr cociclomatic.*

Deci, $s(G)+r(G)=m(G)$.

Este adevărată următoarea afirmaţie: orice subgraf a unui graf arbitrar G se conţine într-un graf de acoperire a grafului G.

Există mai mulţi algoritmi de determinare a grafului de acoperire. Algoritmul de mai jos nu este un algoritm-standard, ci este unul elaborat în bază algoritmului de căutare în lărgime. Esenţa algoritmului constă în aceea că folosind două fire de aşteptare în unul din care sunt înscrise (pe rând) numerele vârfurilor adiacente cu vârfurile din celălalt FA (ca şi în cazul căutării în lărgime), vor fi eliminate muchiile dintre vârfurile unui FA şi toate muchiile în afară de una dintre fiecare vârf al FA curent şi vârfurile din FA precedent. în cazul În care ambele FA vor deveni vide procedura se va termina.

Pentru a nu admite ciclarea şi ca să fim siguri că au fost prelucrate toate componentele conexe se va utiliza marcarea vârfurilor. Dacă după terminarea unui ciclu ordinar nu au mai rămas vârfuri nemarcate procedura ia sfârşit, în caz contrar în calitate de vârf iniţial se va lua oricare din vârfurile nemarcate.

Descrierea algoritmului:

1. Se vor declara două FA (FA_1 şi FA_2) vide.
2. Se va lua în calitate de vârf iniţial un vârf arbitrar al grafului.
3. Se va introduce vârful iniţial în firul de aşteptare vid FA_1 şi se va marca acest vârf.
4. Se vor introduce în FA_2 toate vârfurile adiacente cu vârfurile din FA_1 şi se vor marca. Dacă FA_2 este vid se va trece la p.7, în caz contrar - la p. 4.
5. Se vor elimina toate muchiile care leagă vârfurile din FA_2.
6. Pentru toate vârfurile din FA_2 vor fi eliminate toate muchiile în afară de una care leagă vârful dat cu vârfurile din FA_1.
7. Se vor schimba cu numele FA_1 şi FA_2 (FA_1 va deveni FA_2 şi invers).
8. Dacă există cel puţin un vârf nemarcat se va lua în calitate de vârf iniţial oricare din acestea şi se va trece la p.1, altfel
9. STOP.

Graful obţinut este graful de acoperire.

4.4.4. Noţiune de drum minim. Algoritmul lui Ford pentru determinarea drumului minim

Pentru un graf orientat $G = (X,U)$ se va numi *drum* un şir de vârfuri $D = (x_0, x_1,..., x_r)$ cu proprietatea că (x_0, x_1), (x_1, x_2),..., (x_{r-1}, x_r) aparţin lui U, deci sunt arce ale grafului şi extremitatea finală a arcului precedent coincide cu extremitatea iniţială a arcului următor.

Vârfurile x_0 şi x_r se numesc extremităţile drumului *D*. Lungimea unui drum este dată de numărul de arce pe care le conţine. Dacă vârfurile $x_0, x_1,..., x_r$ sunt distincte două câte două drumul *D* este elementar.

Adeseori, fiecărui arc (muchii) i se pune în corespondenţă un număr real care se numeşte *ponderea* (lungimea) arcului. Lungimea arcului (x_i, x_j) se va nota $w(i,j)$, iar în cazul în care un arc este lipsă ponderea lui va fi considerată foarte mare (pentru calculator cel mai mare număr pozitiv posibil). În cazul grafurilor cu arce ponderate (grafuri ponderate) se va considera lungime a unui drum suma ponderilor arcelor care formează acest drum. Drumul care uneşte două vârfuri concrete şi are lungimea cea mai mică se va numi *drum minim* iar lungimea drumului minim vom numi *distanţă*. Vom nota distanţa dintre *x* şi *t* prin $d(x, t)$, evident, $d(x,x)=0$.

Permite determinarea drumului minim care începe cu un vârf iniţial x_i până la oricare vârf al grafului *G*. Dacă prin L_{ij} se va nota ponderea arcului (x_i, x_j) atunci algoritmul conţine următorii paşi:

1. Fiecărui vârf x_j al grafului *G* se va ataşa un număr foarte mare $H_j(\infty)$. Vârfului iniţial i se va ataşa $H_o = 0$;

2. Se vor calcula diferenţele $H_j - H_i$ pentru fiecare arc (x_i, x_j). Sunt posibile trei cazuri:

   a) $H_j - H_i < L_{ij}$,
   b) $H_j - H_i = L_{ij}$,
   c) $H_j - H_i > L_{ij}$.

Cazul "*c*" permite micşorarea distanţei dintre vârful iniţial şi x_j din care cauză se va realiza $H_j = H_i + L_{ij}$.

Pasul 2 se va repeta atâta timp cât vor mai exista arce pentru care are loc inegalitatea "c". La terminare, etichetele H_i vor defini distanţa de la vârful iniţial până la vârful dat x_i.

3. Acest pas presupune stabilirea secvenţei de vârfuri care va forma drumul minim. Se va pleca de la vârful final x_j spre cel iniţial. Predecesorul lui x_j va fi considerat vârful x_i pentru care va avea loc $H_j - H_i = L_{ij}$. Dacă vor exista câteva arce pentru care are loc această relaţie se va alege la opţiune.

4.4.5 Algoritmul Bellman - Calaba

Permite determinarea drumului minim dintre oricare vârf al grafului până la un vârf, numit vârf final.

Etapa inițială presupune ataşarea grafului dat *G* a unei matrice ponderate de adiacență, care se va forma în conformitate cu următoarele:

1. $M(i,j) = L_{ij}$, dacă există arcul (x_i, x_j) de pondere L_{ij};
2. $M(i,j) = \infty$, unde ∞ este un număr foarte mare (de tip întreg maximal pentru calculatorul dat), dacă arcul (x_i, x_j) este lipsă;
3. $M(i,j) = 0$, dacă $i = j$.

La etapa a doua se va elabora un vector V_0 în felul următor:

1. $V_{0(i)} = L_{in}$, dacă există arcul (x_i, x_n), unde x_n este vârful final pentru care se caută drumul minim, L_{in} este ponderea acestui arc;
2. $V_{0(i)} = \infty$, dacă arcul (x_i, x_n) este lipsă;
3. $V_{0(i)} = 0$, dacă $i = j$.

Algoritmul constă în calcularea iterativă a vectorului *V* în conformitate cu următorul procedeu:

1. $V_{k(i)} = min\{V_{k-1}; L_{ij}+V_{k-1(j)}\}$, unde $i = 1, 2, \ldots, n - 1$, $j = 1, 2, \ldots, n$; $i <> j$;
2. $V_{k(n)} = 0$.

Când se va ajunge la $V_k = V_{k-1}$ - STOP.

Componenta cu numărul *i* a vectorului V_k cu valoarea diferită de zero ne va da valoarea minimă a drumului care leagă vârful *i* cu vârful *n*.

4.5. Reţele de transport

4.5.1. Noţiuni generale

Un graf orientat $G = (X, U)$ se numeşte *reţea de transport* dacă satisface următoarele condiţii:

a) există un vârf unic *a* din *X* în care nu intră nici un arc sau $d_(a)=0$;
b) există un vârf unic *b* din *X* din care nu iese nici un arc sau $d^+(a)=0$;
c) *G* este conex şi există drumuri de la *a* la *b* în *G*;
d) s-a definit o funcţie *c*: $U \rightarrow R$ astfel încât $c(u) \geq 0$ pentru orice arc *u* din *U*.

Vârful *a* se numeşte intrarea reţelei, vârful *b* se numeşte ieşirea reţelei, iar $c(u)$ este capacitatea arcului u.

O funcţie $f: U \rightarrow R$ astfel încât $f(u) \geq 0$ pentru orice arc *u* se numeşte *flux* în reţeaua de transport *G* cu funcţia de capacitate *c*, care se notează $G = (X, U, c)$, dacă sunt îndeplinite următoarele două condiţii:

a) Condiţia de conservare a fluxului: Pentru orice vârf *x* diferit de *a* şi *b* suma fluxurilor pe arcele care intră în *x* este egală cu suma fluxurilor pe arcele care ies din *x*.

b) Condiţia de mărginire a fluxului: Există inegalitatea $f(u) \le c(u)$ pentru orice arc $u \in U$.

Dacă $f(u) = c(u)$ arcul se numeşte *saturat*. Un *drum* se va numi *saturat* dacă va conţine cel puţin un arc saturat. Fluxul, toate drumurile căruia sunt saturate se va numi *flux complet*. Cel mai mare dintre fluxurile complete se numeşte *flux maxim*.

Pentru orice mulţime de vârfuri $A \in U$ vom defini o *tăietură* $w\_(A) = \{(x, y) \mid x \notin A, y \in A, (x,y \in U\}$, adică mulţimea arcelor care intră în mulţimea A de vârfuri.

Prin $w^+(A)$ vom nota mulţimea arcelor care ies din mulţimea A de vârfuri.

Este justă afirmaţia: suma $f(u)$ pentru $u \in w^+(A)$ este egală cu suma $f(u)$ pentru arcele $u \in w\_(A)$. Această valoare comună se va nota f_b.

4.5.2. Algoritmul Ford-Fulkerson

Are loc următoarea **teoremă** (Ford-Fulkerson):

Pentru orice reţea de transport $G = (X, U, c)$ cu intrarea a şi ieşirea b valoarea maximă a fluxului la ieşire este egală cu capacitatea minimă a unei tăieturi, adică:

$$max\ f_b = min\ c(w\_(A)).$$

În baza acestei teoreme a fost elaborat următorul algoritm de determinare a fluxului maxim (Ford-Fulkerson) la ieşirea b a unei reţele de transport $G = (X, U, c)$, unde capacitatea c ia numai valori întregi [7]:

1º. Se defineşte fluxul iniţial având componente nule pe fiecare arc al reţelei, adică $f(u) = 0$ pentru orice arc $u \in U$;

2º. Se determină lanţurile nesaturate de la a la b pe care fluxul poate fi mărit, prin următorul procedeu de etichetare:

a) Se marchează intrarea a cu [+];

b) Un vârf x fiind marcat, se va marca:

cu [+x] oricare vârf y nemarcat cu proprietatea că arcul $u = (x, y)$ este nesaturat, adică $f(u)<c(u)$;

cu [-x] - orice vârf y nemarcat cu proprietatea că arcul $u = (x, y)$ are un flux nenul, adică $f(u)>0$.

Dacă prin acest procedeu de marcare se etichetează ieşirea b, atunci fluxul f_b obţinut la pasul curent nu este maxim. Se va considera atunci un lanţ format din vârfurile etichetate (ale căror etichete au respectiv semnele + sau -) care uneşte pe a cu b şi care poate fi găsit uşor urmărind etichetele vârfurilor sale în sensul de la b către a.

Dacă acest lanţ este v, să notăm cu v^+ mulţimea arcelor (x, y), unde marcajul lui y are semnul "+", deci care sunt orientate în sensul de la a către b şi cu $v\_$

mulţimea arcelor *(x, y)*, unde marcajul lui *y* are semnul "-", deci orientate în sensul de la *b* către *a*.

Determinăm cantitatea:

$e = min \{min(c(u) - f(u)), min\ f(u)\}.\ u \in v^+, u \in v\_$

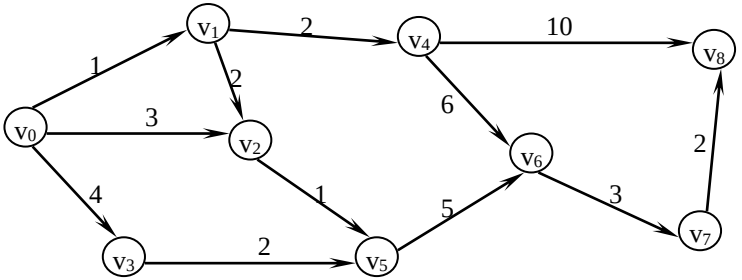
Din modul de etichetare rezultă $e > 0$.

Vom mări cu *e* fluxul pe fiecare arc *u* din v^+ şi vom micşora cu *e* fluxul pe fiecare arc $u \in v\_$, obţinând la ieşire un flux egal cu f_b+e. Se repetă aplicarea pasului 2 cu fluxul nou obţinut.

Dacă prin acest procedeu de etichetare nu putem marca ieşirea *b*, fluxul f_b are o valoare maximă la ieşire, iar mulţimea arcelor care unesc vârfurile marcate cu vârfurile care nu au putut fi marcate constituie o tăietură de capacitate minimă (demonstraţi că se va ajunge în această situaţie după un număr finit de paşi).

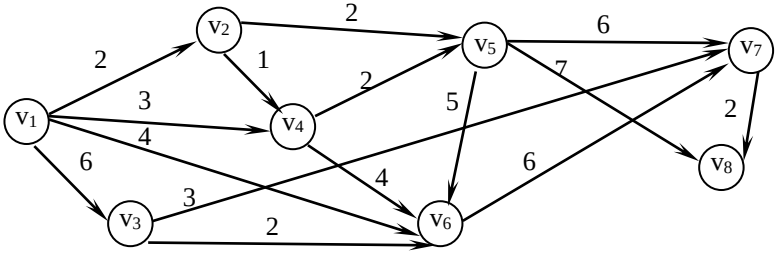
4.6. Exerciţii şi probleme

4.1. Utilizând metoda Bellman - Kalaba să se determine drumul minim între vârfurile v_0 şi v_8 în graful:



Care este drumul de valoare minimă între vârfurile v_2 şi v_8 ?

4.2. Într-o secţie a unei întreprinderi se aplică asupra unui produs anumite operaţii de prelucrare. Prelucrările ce se pot face, tehnologiile posibile de ordonare a lor şi costurile unitare asociate fiecărei prelucrări sunt date într-un model-graf ponderat cu reprezentarea geometrică dată prin figura:



Să se determine tehnologia corespunzătoare celui mai scăzut cost unitar total.

4.3. Prezentaţi sub formă de graf relaţiile de incluziune.

Fie date următoarele mulţimi:

$A = \{n \in \mathbf{Z} / n^2 \leq 17\}$

$B = \{-2, 0, 2\}$

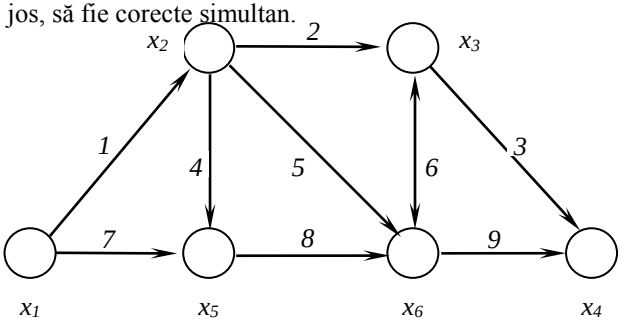
$C = \{E(x) / x \in \mathbf{R}\}$, aici $E(x)$ este funcţia *partea întreagă*

$D =$ intersecţia mulţimii B cu mulţimea R_+

$E = \{2p / p \in \mathbf{Z}\}$

$F =$ *mulţimea rădăcinilor polinomului* $(x^2-4)(x^2-1)(x^3-3x^2)$

Utilizând una din variantele relaţiilor de mai jos determinaţi locul corect al fiecărei mulţimi în unul din vârfurile grafului astfel ca toate relaţiile, alese din lista de mai jos, să fie corecte simultan.



Relaţiile *1 – 5, 7 - 9*:

a) x_i este inclusă în x_j şi nu sunt egale
b) x_i este egală cu x_j.

Relaţia *6*:

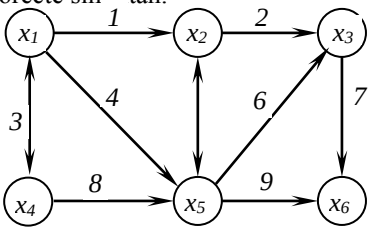
a) x_3 este inclusă în x_6 şi nu sunt egale
b) x_3 este egală cu x_6
c) x_3 nu este în nici o relaţie de incluziune cu x_6 şi invers.

Răspunsul va specifica care mulţime a fost pusă în x_1, care în x_2 şi aşa mai departe până la x_6 şi care este relaţia *1* din cele două posibile (arcul *1* al grafului), care este relaţia *2* (arcul *2* al grafului), până la arcul *9*.

4.4. Fie funcţiile reale definite pe [-1, 1] cu proprietăţile:

a) f_1 este continuă în *0*;
b) f_2 este derivabilă pe [-1, 1] cu derivata ≥ 0;
c) f_3 este derivabilă, crescătoare pe [-1, 1];
d) f_4 este continuă pe [-1, 1];
e) f_5 este derivabilă în *0*;
f) f_6 este continuă şi pară pe [-1, 1].

Utilizând una din variantele relaţiilor de mai jos determinaţi locul corect al fiecărei funcţii în unul din vârfurile grafului astfel ca toate relaţiile, alese din lista de mai jos, să fie corecte simultan.



Relaţiile 1, 2, 4, 6-9: f_i implică f_j, dar f_j nu implică f_i
f_i implică f_j şi f_j implică f_i
f_i nu implică f_j şi f_j nu implică f_i,

Relaţiile 3, 5: f_i implică f_j, dar f_j nu implică f_i
f_i implică f_j şi f_j implică f_i
f_i nu implică f_j şi f_j nu implică f_i,
f_j implică f_i, dar f_i nu implică f_j

Exemplu: funcţia f_4 implică funcţia f_1.

Răspunsul va specifica care funcţie a fost pusă în x_1, care în x_2 şi aşa mai departe până la x_6 şi care este relaţia *1* din cele posibile (arcul *1* al grafului), care este relaţia *2* (arcul *2* al grafului), până la arcul *9*.

5. MODELE ALGORITMICE

5.1. Precizarea noţiunii de algoritm

Algoritmii, înţelegând prin aceasta proceduri stricte şi eficiente de obţinere în mod univoc a rezultatului [2], plecând de la datele iniţiale, sunt prezenţi peste tot în matematică. Adunarea şi înmulţirea în "coloniţă", substituirea necunoscutelor la rezovarea unui sistem de ecuaţii liniare, construirea unui triunghi, fiind date cele trei laturi, sunt exemple de algoritmi. Însă până la momentul în care matematica opera doar cu numere, iar conceptul de algoritm era echivalent cu noţiunea de metodă de calcul, necesitatea studierii acestui concept nu putea să apară. Tradiţiile de organizare a calculelor s-au format pe parcursul secolelor, devenind parte componentă a culturii, în aceeaşi măsură cu aptitudinile elementare de a gândi logic. Toată mulţimea calculelor consta din câteva operaţii aritmetice, trigonometrice şi din analiza matematică, cauză din care noţiunea de "metodă de calcul" părea din start clară şi nu avea nevoie de cercetări speciale.

Până la mijlocul secolului XIX unica ramură matematică, care opera cu obiecte diferite de numere, era geometria. Neavând posibilitatea să folosească intuiţia omului de a calcula, geometria se deosebea în mod evident de restul matematicii prin strictеţea sporită a formulărilor. Apariţia în partea a doua a secolului XIX a geometriei lui Lobacevski, a teoriilor abstracte algebrice a sporit necesitatea abordării într-un mod strict a problemelor matematice. Un moment principial în schimbarea fundamentului matematicii a fost crearea de către Georg Cantor a teoriei mulţimulor. În scurt timp a devenit evident, că teoria mulţimilor stă la baza tuturor fenomenelor matematice. Însă la fel de repede a fost arătat, că unele raţionamente, aparent corecte în cadrul acestei teorii, conduc la contradicţii nerezolvabile – paradoxurile teoriei mulţimilor. Toate acestea au generat necesitatea unor studii stricte a principiilor raţionamentelor matematice (care până la acea perioadă păreau intuitiv clare) cu ajutorul unor mijloace matematice. A apărut astfel o nouă ramură a matematicii – bazele matematicii sau metamatematica. Teoria algoritmilor este partea componentă a metamatematicii în care se face încercarea de a clarifica, care obiecte şi operaţii asupra lor pot fi considerate strict definite, care sunt proprietăţile şi posibilităţile unor combinaţii ale operaţiilor elementare, etc. Aportul principal, adus de teoria algoritmilor matematicii a fost demonstrarea imposibilităţii rezolvării algoritmice (exacte şi univoce) a unor probleme matematice. Aceste demonstraţii, ca şi formularea precisă a afirmaţiilor, care sunt demonstrate, sunt imposibile fără o noţiune exactă a noţiunii de algoritm. Termenul "algoritm" a apărut în tehnică odată cu apariţia ciberneticii [8]. Noţiunea de proces de control (de comandă), partea componentă a acesteia – legea (algoritmul) de comandă trebuiau precizate strict

matematic. Era necesar să se înţeleagă, care sunt proprietăţile unei secvenţe de acţionări pentru a putea fi considerată definită exhaustiv, adică pentru avea posibilitatea să se numească algoritm de comandă. În conştientizarea acestei situaţii un rol extraordinar l-au jucat calculatoarele electronice, care au transformat noţiunea de algoritm într-o realitate palpabilă. Din punctul de vedere al practicii contemporane algoritmul este un program, iar criteriul că un proces este algoritmic este posibilitatea de a fi programat. Datorită acestei existenţe reale a algoritmilor, dar şi faptului că inginerii întotdeauna au abordat în mod constructiv metodele matematice, noţiunea de algoritm a devenit foarte populară în tehnică.

Necesitatea unei conştientizări a noţiunii de algoritm este importantă nu doar pentru o utilizare corectă, ci şi pentru elaborarea unor algoritmi concreţi, îndeosebi dacă se urmăreşte scopul creării ulterioare a unor programe în baza lor. Şi mai importantă este înţelegerea acestui concept pentru a face o ordine în mulţimea algoritmilor, mulţime în care se observă o creştere exponenţială. Pentru a fi posibil să ne orientăm în această diversitate trebuie să putem compara diferiţi algoritmi, care rezolvă aceeaşi problemă şi nu doar din punctul de vedere al calităţii soluţionării problemei, ci şi conform caracteristicilor algoritmilor – numărul de operaţii, memoria necesară, etc. O atare comparare este imposibilă fără a introduce un limbaj precis de cercetare a acestor concepte, adică chiar algoritmii trebuie să devină obiecte de studiu matematic, ca şi obiectele asupra cărora ei sunt destinaţi să acţioneze. Pentru a introduce un astfel de limbaj suntem obligaţi să cunoaştem, care sunt proprietăţile principale ale unui algoritm.

5.1.1. Proprietăţile algoritmilor

5.1.1.1. Obiecte algoritmice

Vom sublinia mai întâi că algoritmul prelucrează date iniţiale şi prezintă rezultate finale. În sens tehnic, un algoritm are intrări şi ieşiri, iar în rezultatul execuţiei unui algoritm apar date intermediare. Altfel spus, orice algoritm operează cu **date** – de intrare, de ieşire şi intermediare. Deoarece dorim să precizăm noţiunea de algoritm, va trebui să precizăm şi noţiunea de date, adică să indicăm care sunt proprietăţile la care vor răspunde obiectele cu care vor opera algoritmii.

Este evident că aceste obiecte trebuie să fie strict definite şi distincte atât unele de altele, cât şi de elemente, care nu pot fi considerate "obiecte". În multe cazuri totul este clar din start. Obiecte algoritmice pot fi considerate numerele, vectorii, matricele, formulele. Imaginile (de exemplu, reprezentarea grafică a unui graf) par obiecte algoritmice mai puţin evidente. Iar cu obiecte de tipul "un film interesant", "timp frumos", care nu prezintă probleme pentru o fiinţă umană

(fiecare înţelegând în felul său aceste noţiuni), algoritmul ar putea să "nu vrea să opereze".

În teoria algoritmilor în locul unei definiţii verbale stricte a conceptului de obiect sunt fixate seturi finite de obiecte iniţiale concrete (numite elementare) şi un set de mijloace de construire a altor obiecte din cele elementare şi, poate, cele intermediare. Setul de obiecte elementare formează alfabetul finit al simbolurilor iniţiale (cifre, litere, etc.) din care sunt obţinute alte obiecte (v. procedurile generatoare din 2.1.2, definiţiile inductive din 3.7.1 sau definiţia noţiunii de identificator într-un limbaj de programare). Cuvintele de lungime finită în alfabete finite (v. exemplul 2.12) reprezintă cel mai frecvent tip de date algoritmice, iar numărul de simboluri într-un cuvânt este unitatea naturală de măsurare a volumului de informaţii procesate.

Un alt exemplu, mai complicat, de obiecte algoritmice, pot fi formulele. Ele, la fel pot fi definite inductiv şi sunt cuvinte finite în alfabete finite. Însă nu orice cuvânt în acest alfabet este o formulă. În acest caz, de obicei înaintea algoritmilor de bază există un set de algoritmi auxiliari, care verifică, dacă datele iniţiale respectă cerinţele necesare, verificare numită analiză sintactică.

5.1.1.2. Memoria

Pentru păstrarea datelor este nevoie de memorie. Memoria, de obicei, este considerată omogenă şi discretă, adică constă din locaţiuni (celule) de acelaşi fel (banalizate), fiecare locaţiune păstrând un simbol al alfabetului de date. Am coordonat în acest mod unităţile de măsură pentru date şi memorie. Răspuns la întrebarea, dacă avem nevoie de mai multe tipuri de memorie în dependenţă de cele trei tipuri de date se rezolvă în diverse moduri şi nu prezintă interes special în cazul dat.

5.1.1.3. Paşii algoritmului

Un algoritm constă din *paşi elementari* distincţi, sau *operaţii*, mulţimea paşilor unui algoritm, este finită. Un exemplu tipic de mulţime de operaţii elementare este sistemul de instrucţiuni de bază al unui calculator electronic.

5.1.1.4. Determinism

Secvenţa paşilor unui algoritm este *determinată*, adică după executarea unui pas este în mod univoc indicat, care va fi pasul următor sau se va da comanda de oprire, după care algoritmul este considerat încheiat.

5.1.1.5. Rezultativitate

Este normal să cerem ca un algoritm să fie rezultativ, adică algoritmul trebuie să se oprească după un număr finit de paşi (care depinde de datele procesate), indicându-se rezultatul. Nu este obligator ca acest rezultat să ne "placă"; dacă nu s-a ajuns la ceea ce s-a sperat, algoritmul poate specifica cauza opririi.

5.1.1.6. Descrierea şi mecanismul de realizare a algoritmului

Va trebui să facem o deosebire între

a) descrierea algoritmului (set de instrucţii sau program);
b) mecanismul de realizare a algoritmului (de exemplu, calculatorul electronic), care include mijloacele de lansare, executarea paşilor elementari, garantarea determinismului şi extragerea rezultatelor (controlul procesului de execuţie);
c) procesul de realizare a algoritmului, adică secvenţa de paşi, care va fi generată de aplicarea algoritmilor pentru date concrete.

Vom considera, că descrierea unui algoritm şi mecanismul lui de realizare sunt finite.

5.1.2. Metode de precizare a noţiunii "algoritm"

Am subliniat mai sus proprietăţile, pe care trebuie să le posede un set de reguli pentru a putea fi considerat algoritm. Dar aceste proprietăţi (de exemplu, stricteţe, univocitate, paşi elementari, etc.) trebuie la rândul lor precizate. Definiţiile acestora vor conţine noţiuni noi, care de asemenea vor trebui precizate, ş.a.m.d. Pentru a exclude aceasta, în teoria algoritmilor a fost ales un set finit de obiecte iniţiale, declarate elementare şi un set finit de mijloace de construire din aceste obiecte a unor obiecte noi. De exemplu, vom preciza noţiunea de date, considerând datele mulţimi de cuvinte finite în alfabete finite. Pentru precizarea determinismului vom utiliza organigramele (schemele-bloc), descrierile verbale sau mecanismul de realizare a unui algoritm. În afară de acestea, trebuie să fixăm setul de paşi elementari şi să convenim în privinţa organizării memoriei. Vom obţine un model algoritmic concret.

Modelele algoritmice pot fi considerate entităţi matematice abstracte, destinate formalizării conceptului "algoritm", doar dacă vor fi universale, adică vor permite descrierea oricărui algoritm. Dar, alegând nişte instrumente concrete pentru formalizare, nu vom pierde oare cadrul general al formalizării? Reieşind din scopurile principale, care stau la baza creării teoriei algoritmilor – universalitatea şi posibilitatea cercetării proprietăţilor algoritmilor – acest semn de întrebare poate fi exclus în modul următor.

1. Se va demonstra convergenţa modelelor, adică orice algoritm, descris prin mijloacele unui model, poate fi descris şi cu resursele altui model.
2. Datorită convergenţei reciproce a modelelor în teoria algoritmilor a fost elaborat un sistem de noţiuni invariante de model, set care permite cercetarea proprietăţilor algoritmilor independent de tipul ales de formalizare. Acest sistem de noţiuni este bazat pe conceptul de funcţie calculabilă, adică funcţie, pentru calcularea valorilor căreia există un algoritm.

Totuşi, deşi cadrul general al formalizării nu este pierdut, diferite posibilităţi de alegere a resurselor iniţiale conduc la modele diferite. Pot fi evidenţiate trei tipuri de bază de modele algoritmice universale, în dependenţă de modul de înţelegere euristic iniţial al noţiunii de algoritm.

Primul tip leagă noţiunea de algoritm de cele mai tradiţionale concepte matematice – calcule şi funcţii numerice. Modelul algoritmic din această categorie - **funcţiile recursive** – cronologic este prima încercare de formalizare a noţiunii de algoritm.

Tipul al doilea reprezintă algoritmul sub forma unui dispozitiv ingineresc, capabil să îndeplinească la un moment dat de timp doar cele mai elemenare operaţii. În acest caz nu pot să apară semne de întrebare, legate de univocitatea algoritmului sau de simplitatea paşilor. Mai mult, euristica acestor modele este apropiată de principiile de funcţionare a calculatoarelor, adică şi de intuiţia inginerească, ceea ce le face foarte populare în acest mediu. Modelul teoretic principal din această categorie, propus la începutul anilor 1930, este **maşina Turing**.

Transformările cuvintelor în cadrul unor alfabete arbitrare reprezintă al treilea tip de modele algoritmice. În calitate de operaţie elementară este considerată substituirea, adică înlocuirea unei părţi de cuvânt cu alt cuvânt. Principalul avantaj al acestui tip este abstractizarea maximă şi posibilitatea utilizării noţiunii de algoritm pentru obiecte de natură arbitrară (nu obligator numerică). Exemplele cele mai elocvente sunt în acest caz **sistemele formale Post** şi **algoritmii normali Markov**.

5.2. Maşini Turing

Construcţia şi principiul de funcţionare a unei maşini Turing (MT) sunt elementare, din care cauză respectarea cadrului formal şi metodologic este evidentă. Totuşi vom sublinia, acolo unde vom considera necesar, modul în care este prezentă o proprietate anume în modelele de acest tip.

5.2.1. Componentele unei MT şi principiul de funcţionare

O maşină Turing este formată din următoarele părţi componente:

1. dispozitivul de comandă, care poate să se afle în una din stările, care aparţin mulţimii finite de stări $Q = \{q_1, q_2, \ldots, q_n\}$;
2. banda maşinii, împărţită în celule (locaţiuni), în fiecare celulă fiind posibilă scrierea unui simbol al alfabetului finit $A = \{a_1, a_2, \ldots, a_m\}$;
3. dispozitivul de accesare a benzii, adică un cap de citire şi scriere, care la fiecare moment de timp citeşte conţinutul unei celule a benzii şi, în dependenţă de simbolul citit şi starea dispozitivului de comandă, scrie în celula dată un simbol (care poate coincide cu simbolul precedent sau poate fi un simbol vid, adică şterge conţinutul celulei), se deplasează la stânga, sau la dreapta la una din celulele vecine, sau rămâne pe loc, iar dispozitivul de comandă poate trece într-o stare nouă sau poate rămâne în starea precedentă.

Prin descrierea dispozitivului de citire – scriere (DCS) a fost lămurit şi principiul de funcţionare a unei maşini Turing. Din mulţimea de stări ale dispozitivului de comandă vom evidenţia starea iniţială q_1 şi starea finală q_z. În starea iniţială maşina se va afla la începutul lucrului, iar nimerind în starea finală maşina se va opri.

Memoria maşinii Turing constă din memoria internă – mulţimea de stări ale dispozitivului de comandă, şi memoria externă – banda infinită în ambele părţi. Deşi banda este infinită, doar un număr finit de celule sunt completate cu simboluri nevide la momentul lansării MT, restul fiind vide, adică conţin simbolul vid λ (blanc). Din descrierea funcţionării MT reiese, că şi pentru oricare alt moment de timp viitor doar un fragment finit al benzii va fi ocupat de simboluri. Din această cauză prezintă interes nu infinitatea actuală a benzii, ci nemărginirea ei, adică posibilitatea de a înscrie cuvinte oricât de lungi, dar finite.

Datele unei MT sunt cuvinte în alfabetul benzii. Pe bandă se vor înscrie atât datele iniţiale, cât şi cele finale. Paşii elementari sunt citirea şi scrierea simbolurilor, deplasarea dispozitivului de citire-scriere la celula vecină, trecerea dispozitivului de comandă într-o stare nouă.

Determinismul MT constă în faptul că pentru orice stare internă q_i şi orice simbol a_j sunt cunoscute în mod univoc:

a) starea următoare q_i^1;
b) simbolul a_j^1, care va fi scris în aceiaşi celulă;
c) direcţia de deplasare a DCS $d_k = \{L \text{ (stânga)}, R \text{ (dreapta)}, E \text{ (pe loc)}\}$.

Aceasta poate fi descris cu ajutorul sistemului de comenzi de forma

$$q_i a_j \rightarrow q_i^1 a_j^1 d_k \qquad (5.1)$$

sau printr-un tabel, liniilor fiind ataşate stările, coloanelor – simbolurile de intrare, iar la intersecţia liniei q_i cu coloana a_j se va afla tripletul $q_i^1 a_j^1 d_k$. O MT mai poate fi descrisă cu ajutorul unui graf ponderat, stărilor fiind puse în

corespondenţă vârfurile grafului, unei comenzi de tipul (5.1) – un arc, care porneşte din q_i şi se termină în $\rightarrow q_i^1$ cu ponderea $a_j \rightarrow a_j^1 d_k$.

5.2.2. Configuraţia unei maşini Turing

Starea MT, care permite determinarea univocă a comportamentului maşinii, se numeşte **stare completă**. Starea completă este determinată de starea internă, starea benzii (cuvântul scris pe bandă) şi poziţia DCS. Starea completă se numeşte **configuraţia** MT sau cuvânt de maşină şi este notată prin tripletul $\alpha_1 q_i \alpha_2$, unde q_i este starea internă curentă, α_1 – cuvântul din stânga DCS, iar α_2 – cuvântul format de simbolul citit de DCS şi simbolurile din dreapta acestui dispozitiv, cu condiţia ca în stânga cuvântului α_1, ca şi în dreapta lui α_2 nu există simboluri nevide. De exemplu, configuraţia cu starea internă q_i, pentru care avem scris pe bandă *werty*, iar DCS citeşte simbolul *r*, se va nota $we\boldsymbol{q}_i rty$.

O configuraţie de tipul $\boldsymbol{q_1\alpha}$, adică configuraţia care conţine starea iniţială q_1, iar DCS citeşte cel mai din stânga simbol al benzii, se numeşte **configuraţie standard iniţială** (CSI). Analogic, numim **configuraţie standard finală** configuraţia de forma $\boldsymbol{q_z\alpha}$. Asupra unei configuraţii arbitrare K (cu excepţia configuraţiei finale) poate fi aplicată exact o comandă de tipul (5.1), care va transfera K în K^1, relaţia dintre configuraţii fiind notată în acest caz prin $K \rightarrow K^1$. Dacă din context nu este clar, care MT execută acest transfer, numele maşinii poate fi notat de asupra săgeţii. Dacă pentru K_1 şi K_n există un şir de configuraţii $K_1, K_2, \ldots K_n$, aceasta se va nota prin $K_1 \Rightarrow K_n$.

Exemplul 5.1. O MT cu alfabetul $A = \{1, \lambda\}$, cu stările $Q = \{q_1, q_z\}$ şi sistemul de instrucţiuni $q_1 1 \rightarrow q_1 1R$, $q_1 \lambda \rightarrow q_1 1R$, din orice configuraţie iniţială va lucra fără oprire, scriind *1* în toate casetele benzii, începând cu cea de plecare. ◄

Exemplul 5.2. Pentru orice MT, dacă $K_1 \Rightarrow K_i \Rightarrow K_j$ şi $K_i = K_j$, secvenţa $K_1 \Rightarrow K_i \Rightarrow K_j \Rightarrow \ldots$ este infinită, segmentul $K_i \Rightarrow K_j$ se va repeta până la infinit (ciclarea maşinii Turing). ◄

Vom spune că maşina Turing T procesează cuvântul $\alpha_1\alpha_2$ în cuvântul $\beta_1\beta_2$ (se va mai nota prin $T(\alpha_1\alpha_2) = \beta_1\beta_2$), dacă $\alpha_1 q_1 \alpha_2 \Rightarrow \beta_1 q_z \beta_2$. Notaţia $T(\alpha)$ va mai semnifica o maşină Turing cu valoarea iniţială α.

5.2.3. Funcţii calculabile Turing

Precizăm interpretarea comportamentului unei MT şi reprezentarea datelor. Date iniţiale ale unei MT sunt considerate cuvinte scrise pe bandă în alfabetul datelor iniţiale A_{in} ($A_{in} \subseteq A$) şi vectorii, formaţi din aceste cuvinte (vectori peste A_{in}), pentru fiecare maşină vor fi luate în consideraţie numai acele configuraţii iniţiale pentru care pe bandă sunt scrişi vectori de cuvinte peste A_{in}. Scrierea unui vector de cuvinte $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ se numeşte corectă, dacă este de forma $\alpha_1 \lambda \alpha_2 \lambda \ldots \alpha_{k-1} \lambda \alpha_k$ (cu condiţia, că $\lambda \notin A_{in}$) sau $\alpha_1 * \alpha_2 * \ldots \alpha_{k-1} * \alpha_k$, simbolul * fiind un simbol

special de separare (marcher), care la fel nu aparţine alfabetului A_{in}. Pentru orice vector V_{in} peste A_{in} maşina T sau lucrează până la infinit, sau acest vector este transformat într-un set de cuvinte, separate de blancuri, în alfabetul rezultatelor A_{rez}; A_{in} şi A_{rez} pot să se intersecteze şi chiar să coincidă. În procesul lucrului pe bandă pot să apară simboluri, care nu aparţin nici lui A_{in} şi nici la A_{rez} şi care formează un alfabet intermediar A_{int} (care conţine, de exemplu, separatorul). Alfabetul benzii poate fi definit ca reuniunea celor trei alfabete $A = A_{in} \cup A_{rez} \cup A_{int}$. Cel mai frecvent $A_{in} = A_{rez}$, iar $A = A_{in} \cup \{\lambda\}$.

Fie f o funcţie, care transformă o mulţime de vectori peste A_{in} într-o mulţime de vectori peste A_{rez}. Vom spune, că maşina T calculează corect funcţia f, dacă:

1) pentru oricare V şi W, pentru care $f(V) = W$ are loc $q_1V^* = q_zW^*$, aici V^* şi W^* sunt înregistrări corecte ale lui V şi W, respectiv;
2) pentru orice V pentru care $f(V)$ nu este definită, maşina Turing, lansată în CSI q_1V^*, lucrează până la infinit.

Dacă pentru f există maşina Turing, care o calculează corect, funcţia f se numeşte **corect calculabilă Turing**.

Pe de altă parte, fiecărei maşini Turing, care calculează corect, adică fiind lansată din CSI $q_1\alpha$ poate să se oprească doar în CSF $q_z\beta$, îi putem pune în corespondenţă funcţia pe care MT dată o calculează. Două maşini Turing cu acelaşi alfabet iniţial se numesc echivalente, dacă calculează aceeaşi funcţie. Maşinile din exemplele 5.1 şi 5.2 sunt echivalente, deoarece ambele calculează una şi aceeaşi funcţie – funcţia vidă (fără domeniu de definiţie).

Definiţiile de mai sus (calcularea valorilor unor funcţii definite peste vectori de cuvinte) sunt evident redundante, fiind legate de procesarea unor obiecte arbitrare. A fost demonstrat, că fără a pierde cadrul general, unele cercetări pot fi reduse la calcularea unor funcţii de tipul $N \to N$. Le vom reprezenta în cod unar, adică pentru orice funcţie numerică $A_{in} = \{1\}$ sau $A_{in} = \{1, *\}$, iar un număr x va fi reprezentat prin cuvântul $1...1 = 1^x$, care constă din x unităţi. În acest caz funcţia numerică $f(x_1,...,x_n)$ este calculată corect în sens Turing, dacă există o maşină T, care:

1) $q_1 1^{x1} * q_2 1^{x2} * ... * q_n 1^{xn} \Rightarrow q_z 1^y$, dacă $f(x_1,...,x_n) = y$ şi
2) T lucrează până la infinit, plecând din $q_1 1^{x1} * q_2 1^{x2} * ... * q_n 1^{xn}$, dacă $f(x_1,...,x_n)$ nu este definită.

5.2.3.1. Maşina Turing T_+

Adunarea a două numere a şi b, reprezentate în cod unar, este redusă la procesarea cuvântului $1^a * 1^b$ în cuvântul 1^{a+b}, adică să se elimine separatorul * şi unul din operanzi, de exemplu primul, să fie deplasat spre altul cu o poziţie. Această transformare poate fi realizată de o maşină Turing cu patru stări şi următorul sistem de comenzi: $q_1 * \to q_z \lambda R;$

$q_1 1 \rightarrow q_2 \lambda R;$
$q_2 1 \rightarrow q_2 1 R;$
$q_{2*} \rightarrow q_3 1 L;$
$q_3 1 \rightarrow q_3 1 L;$
$q_3 \lambda \rightarrow q_z \lambda R.$

Prima comandă este introdusă pentru cazul când $a=0$. Au fost omise acele combinaţii de stări şi simboluri, care, pornind din CSI, nu sunt posibile.

Propunem ca exerciţiu, elaborarea tabelului şi a grafului acestei maşini, ca şi controlarea corectitudinii funcţionării ei.

5.2.3.2. Maşina Turing T_{cop}

Copierea unui cuvânt este procesarea lui α în $\alpha * \alpha$. Pentru cazul numerelor problema poate fi rezolvată de maşina Turing cu sistemul de comenzi, prezentat în tabelul 5.1.

Tabelul 5.1. Maşina Turing T_{cop}

	1	λ	*	*0*
q_1	$q_2 0R$	$q_z \lambda R$	$q_1 * L$	$q_1 1L$
q_2	$q_2 1R$	$q_3 * R$	$q_3 * R$	
q_3	$q_3 1R$	$q_4 1L$		
q_4	$q_4 1L$		$q_4 * L$	$q_1 0R$

La fiecare trecere a cuvântului iniţial 1^a maşina T_{cop} înlocuieşte unitatea din stânga cu *0* şi scrie, în starea q_3, în prima celulă vidă din dreapta lui 1^a o unitate. La prima trecere, plus la aceasta, în starea q_2, scrie un marcher (*) înaintea scrierii unităţii. (Peste a treceri după marcher vom avea copia lui 1^a.) După scrierea unităţii ordinare, maşina trece în starea q_4, care deplasează DCS până în stânga celui mai apropiat *0*, după ce maşina trece în starea q_1 şi tot ciclul se repetă. Ciclul se va întrerupe, atunci când q_1 găseşte pe bandă marcherul, ceea ce înseamnă că toate unităţile au fost parcurse (au avut loc a treceri). DCS se întoarce în acest caz în poziţia iniţială, înlocuind pe parcurs toate zerourile cu unităţi.

5.2.4. Maşina matematică Turing

Strict matematic maşina Turing poate fi introdusă după cum urmează. Notăm prin λ simbolul blanc (spaţiu liber). Dacă x este un cuvânt oarecare peste alfabetul A, notăm prin $D(x)$ cuvântul obţinut prin eliminarea tuturor simbolurilor λ din faţa lui x, iar prin $F(x)$ - cuvântul obţinut prin eliminarea tuturor simbolurilor λ din spatele lui x. Notăm prin A^* mulţimea $A \cup \varepsilon$, unde ε este unicul cuvânt de lungime *0*.

O maşină Turing este un cvintet $M=(A, Q, q_1, Q^1, \tau)$ unde

- A este alfabetul lui M ($\lambda \in A$),
- Q este mulţimea stărilor maşinii M (mulţime finită),
- q_1 este starea iniţială a maşinii M (element privilegiat al lui Q),
- Q^1 reprezintă mulţimea stărilor finale ($Q^1 \subseteq Q$),
- τ este funcţia de tranziţie – o aplicaţie de tipul $(Q\text{-}Q^1)xA \to QxAxD$, iar $D = \{L, R, E\}$.

Setul de instrucţiuni ale maşinii M este mulţimea cvintetelor de forma (q,a,q^1,a^1,d) cu $d \in D$, $(q,a) \in (Q\text{-}Q^1)xA$ şi $(q^1,a^1,d) = \tau(q,a)$.

O configuraţie a maşinii M este un triplet de forma $(q,D(x),F(y))$, cu $q \in Q$, $x \in A^*$ şi $y \in A^*$. Introducem relaţia binară $\underset{\rightarrow}{M}$ definită peste mulţimea configuraţiilor maşinii M astfel:

$$(q, D(xa), F(by)) \underset{\rightarrow}{M} (q^1, D(xu), F(vy))$$

atunci şi numai atunci, când

$$(u, v) = \begin{cases} (\varepsilon, ab^1), \text{ dacă } d = L, \\ \\ (ab^1, \varepsilon), \text{ dacă } d = R \end{cases}$$

pentru toate elementele $a, b, b^1 \in A$, $x, y \in A^*$, $q, q^1 \in Q$ şi $d \in \{L, R\}$ astfel încât $\tau(q, b) = (b^1, q^1, d)$.

Notăm prin $\underset{\rightarrow}{M}^*$ închiderea reflexivă şi tranzitivă a lui $\underset{\rightarrow}{M}$, ceea ce înseamnă că $c \underset{\rightarrow}{M}^* c^1$ dacă şi numai dacă există $c_o, c_1, \ldots, c_n$ cu $n \geq 0$ şi $c = c_o$, $c_o \underset{\rightarrow}{M} c_1$, $c_1 \underset{\rightarrow}{M} c_2, \ldots, c_{n-1} \underset{\rightarrow}{M} c_n$, $c_n = c^1$.

Fie w o valoare, numită „nedefinită" cu $w \notin A^*$. Asociem maşinii Turing M funcţia f de tipul $A^* \cup \{\omega\} \to A^* \cup \{w\}$ definită prin expresia

$$f(x) = \begin{cases} y, & \text{dacă } x \in A^* \text{ şi există } (q_i, y^1, y) \text{ cu} \\ & (q_1, \varepsilon, D(x)) \underset{\rightarrow}{M}^* (q_i, y^1, y) \text{ şi } q_i \in Q^1, \\ \\ \omega, & \text{în caz contrar.} \end{cases}$$

Vom spune, că maşina Turing M este *fiabilă*, dacă nici pentru un $x \in A^*$ nu vom avea $f(x) = w$.

5.2.5. Operaţii cu maşinile Turing

Lucrul unei MT este în totalitate determinat de datele iniţiale şi sistemul de comenzi. Totuşi, pentru a fi mai uşor de înţeles ce face o maşină Turing sunt necesare anumite explicaţii, cum au fost cele din exemplele precedente. Folosind

limbajul organigramelor şi unele operaţii cu maşinile Turing aceste explicaţii pot deveni mai formale şi mai exacte.

Să ne amintim, că compunerea a două funcţii $f_1(x)$ şi $f_2(y)$ se numeşte funcţia $g(x) = f_2(f_1(x))$, care este obţinută prin aplicarea funcţiei f_2 la rezultatele, calculate de funcţia f_1. Unica condiţie este ca f_1 să fie definită pe x, iar f_2 pe $f_1(x)$.

Teorema 5.1. Dacă $f_1(x)$ şi $f_2(y)$ sunt funcţii calculabile Turing, atunci şi compunerea lor $f_2(f_1(x))$ este funcţie calculabilă Turing.

Fie T_1 maşina, care calculează funcţia f_1, iar T_2 calculează funcţia f_2, cu mulţimile de stări $Q_1=\{q_{11},..., q_{1n1}\}$ şi $Q_2=\{q_{21},..., q_{2n2}\}$, respectiv. Considerăm ambele funcţii, pentru simplitate, numerice de o singură variabilă. Construim graful unei maşini T din maşinile T_1 şi T_2, echivalând vârful iniţial q_{21} al maşinii T_2 cu vârful final al maşinii T_1 q_{1z}. Obţinem un graf cu n_1+n_2-1 vârfuri. Declarăm stare iniţială a maşinii T starea q_{11}, iar starea finală – q_{2z}. Dacă $f_2(f_1(x))$ este definită, atunci $T_1(1^x)=1^f1^{(x)}$ şi $q_{11}1^x \underset{T_1}{\Rightarrow} q_{1z}1^f1^{(x)}$.

Maşina T va parcurge acelaşi şir de configuraţii, doar în loc de $q_{1z}1^f1^{(x)}$ va trece în $q_{21}1^f1^{(x)}$. Această configuraţie este configuraţia standard iniţială pentru maşina T_2 din care cauză vom avea $q_{21}1^f1^{(x)} \underset{T_2}{\Rightarrow} q_{2z}1^f2^{(f}1^{(x))}$.

Deoarece toate comenzile maşinii T_2 sunt prezente în maşina T vom avea

$$q_{11}1^x \underset{T}{\Rightarrow} q_{21}1^f1^{(x)} \underset{T}{\Rightarrow} q_{2z}1^f2^{(f}1^{(x))},$$

adică $T(1^x) = 1^f2^{(f}1^{(x))}$. Dacă $f_2(f_1(x))$ nu este definită, una din maşinile T_1 sau T_2, sau ambele, nu se vor opri. Am demonstrat că maşina T calculează funcţia $f_2(f_1(x))$.

Maşina T, construită în acest mod, se numeşte compunerea maşinilor T_1 şi T_2 şi se notează prin $T_2(T_1)$. Deoarece o maşină Turing consideră un vector peste A_{in} drept cuvânt în alfabetul $A_{in} \cup \{*\}$, definiţia compunerii şi teorema 5.1 sunt valide şi pentru cazul în care T_1 şi T_2 calculează funcţii de mai multe variabile.

5.2.5.1. Maşina T_{2x}

Diagrama din figura 5.1 reprezintă maşina $T_+(T_{cop})$, care calculează funcţia $f(x)=2x$, pentru $x \neq 0$ şi poate servi ca exemplu de compunere a două maşini Turing.

Aici arcele multiple au fost reprezentate printr-un singur arc. De exemplu, instrucţiunile $q_{14}* \to q_{14}*L$ şi $q_{14}1 \to q_{14}1L$ sunt reprezentate printr-o singură buclă cu ponderea $*,1 \to L$, fără a repeta în partea dreapta simbolurile, care au rămas fără schimbare.

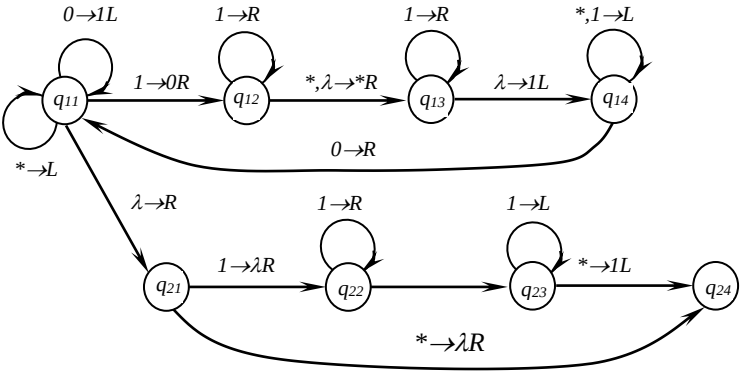


Fig.5.1. Diagrama maşinii T_{2x}

5.2.6. Calcularea predicatelor cu ajutorul maşinii Turing

Fie $\omega \in \{True, False\}$. Vom spune, că o maşină T calculează predicatul $P(\alpha)$ (α este cuvânt în A_{in}), dacă $T(\alpha)=\omega$, iar $\omega = True$, pentru $P(\alpha)=True$ şi $\omega = False$, dacă $P(\alpha) = False$. Dacă $P(\alpha)$ nu este definit, atunci T nu se va opri. La calcularea simplă a unui predicat α este distrus, ceea ce poate crea probleme, dacă după maşina T va lucra o altă maşină. Introducem noţiunea de calculare cu restabilire: maşina T calculează $P(\alpha)$ cu restabilire, dacă $T(\alpha)=\omega\alpha$. Poate fi demonstrat, că dacă există maşina T, care calculează $P(\alpha)$, atunci există şi maşina T^1, care calculează $P(\alpha)$ cu restabilire.

5.2.6.1. Maşina „ α - număr par"

Diagrama maşinii *„ α - număr par"* este prezentată în figura 5.2. Dispozitivul de citire-scriere ajunge la ultimul simbol în starea q_2, dacă numărul de unităţi este par, iar în starea q_3, când numărul acesta este impar. După aceasta DCS se va deplasa în poziţia iniţială în starea q_4 sau q_5, tipărind *True* sau *False*, respectiv.

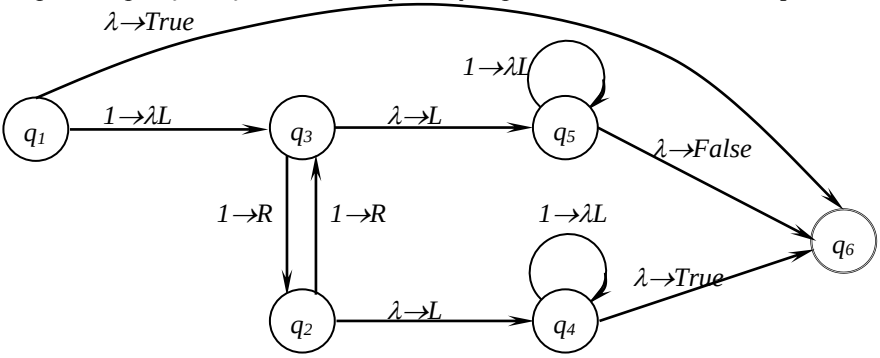


Fig.5.2. Diagrama maşinii *„ α - număr par"*

Pentru ca predicatul $P(\alpha)$ să fie calculat cu restabilire este suficient ca în buclele q_4 şi q_5 să fie păstrate unităţile ($1 \to 1L$ în loc de $1 \to \lambda L$).

5.2.6.2. Maşina „salt condiţionat"

Maşina Turing cu alfabetul $A_{in}=\{T, F\}$ (prin T am notat *True*, iar prin F – *False*) şi comenzile

$$q_1T \to q_zFE,$$
$$q_1F \to q_zTE$$

calculează negaţia unei variabile logice. Din calculabilitatea predicatului total definit $P(\alpha)$ rezultă calculabilitatea negaţiei lui $P(\alpha)$.

Fie funcţia $f(\alpha)$ definită de relaţia:

$$f(\alpha)=\begin{cases} g_1(\alpha), \text{ dacă } P(\alpha)=T, \\ g_2(\alpha), \text{ dacă } P(\alpha)=F, \end{cases}$$

iar dacă $P(\alpha)$ nu este definit, atunci $f(\alpha)$ la fel nu este definită. Funcţia $f(\alpha)$ se numeşte *salt condiţionat* (*ramificare*) *la* $g_1(\alpha)$ *sau la* $g_2(\alpha)$ conform condiţiei $P(\alpha)$.

Poate fi demonstrată

Teorema 5.2. Dacă $g_1(\alpha)$, $g_2(\alpha)$ şi $P(\alpha)$ sunt calculabile Turing, atunci ramificarea g_1 şi g_2 conform condiţiei P de asemenea este calculabilă.

Fie T_1 cu sistemul de comenzi Σ_1 şi mulţimea de stări $Q_1=\{q_{11},..., q_{1n1}\}$, calculează g_1; T_2 cu sistemul de comenzi Σ_2 şi mulţimea de stări $Q_2=\{q_{21},..., q_{2n2}\}$, calculează g_2; T_p calculează cu restabilire $P(\alpha)$. Maşina T, care calculează saltul condiţionat g_1, g_2 conform P este compunerea maşinii T_p şi a maşinii T_3, sistemul de comenzi al căreia este de forma:

$$\Sigma_3 = \Sigma_1 \cup \Sigma_2 \cup \{q_{31}T \to q_{11}\lambda R,\ q_{31}F \to q_{21}\lambda R,\ q_{1z} \to q_{2z}E\}.$$

Primele două comenzi transmit controlul sistemelor Σ_1 sau Σ_2 în dependenţă de valoarea de adevăr a predicatului $P(\alpha)$. Comanda a treia defineşte starea finală q_{2z} a maşinii T_3. Ea va fi îndeplinită pentru orice simbol, din care cauză simbolul benzii nu este indicat.

5.2.7. Maşina Turing universală

Sistemul de comenzi al unei maşini Turing poate fi interpretat şi ca descrierea modului de lucru a unui mecanism concret, şi ca program – set de indicaţii, care

conduc în mod univoc la un rezultat. Când sunt cercetate exemple concrete persoana în cauză de obicei se situează în rolul de mecanism, care poate îndeplini lucrul oricărei maşini Turing, având la dispoziţie setul de instrucţiuni ca şi program. Siguranţa că toţi vor îndeplini în acelaşi mod comenzile (adică nu vor face greşeli, ceea ce se consideră şi în cazul maşinii Turing) nu este altceva decât siguranţa, că există un algoritm care descrie modul de lucru a maşinii Turing conforma programului dat. Nu este complicat să formulăm acest algoritm: "Pentru configuraţia curentă $\alpha_1 a_k q_i a_j \alpha_2$ să se găsească în sistemul de comenzi comanda cu partea stângă $q_i a_j$. Dacă partea dreaptă a acestei comenzi este de forma $q_i^1 a_j^1 R$, să se schimbe în configuraţia curentă $q_i a_j$ cu $a_j^1 q_i^1$, vom avea configuraţia $\alpha_1 a_k\, a_j^1 q_i^1 \alpha_2$; dacă partea dreaptă este de forma $q_i^1 a_j^1 L$, să se schimbe $a_k q_i a_j$ cu $q_i^1 a_k\, a_j^1$. Poate fi demonstrat, că pentru cazul *E* situaţia poate fi redusă la una din cele prezentate anterior".

Dar, conform celor discutate chiar la începutul acestui capitol, descrierea verbală a unui algoritm poate fi imprecisă şi trebuie formalizată. Pentru aceasta poate fi utilizată chiar maşina Turing, adică să construim o maşină Turing, care va realiza algoritmul, descris mai sus. Pentru maşinile Turing, care calculează funcţii de un singur argument, enunţul problemei este următorul: să fie construită maşina Turing *U*, care calculează funcţii de două argumente, care pentru orice maşină *T* cu sistemul de comenzi Σ_T va verifica condiţia $U(\Sigma_T, \alpha)=T(\alpha)$, dacă $T(\alpha)$ este definită (*T* se va opri pentru datele iniţiale α), şi nu se va opri în caz contrar. Orice maşină *U* cu această proprietate se numeşte **maşina Turing universală** (MTU). Formularea conceptului de maşină Turing universală poate fi extinsă pentru orice număr de argumente.

MTU, ca şi orice altă maşină Turing, trebuie să posede un alfabet finit A_U şi o mulţime finită de stări Q_U. Nu este posibil de obţinut aceste două obiecte prin simpla reuniune a alfabetelor sau mulţimilor de stări ale maşinilor iniţiale. Oricând poate fi propusă o maşină nouă, care nu a fost luată în consideraţie.

Soluţia constă în codarea simbolurilor din A_U şi Q_U cu cuvinte din alfabetul A_U. Fie $|A_U|=m_T$, iar $|Q_U|=n_T$. Vom considera, că $a_1 = 1$, *iar* $a_{mT} = \lambda$ (aceste simboluri sunt întotdeauna prezente în alfabetul oricărei MT, care calculează funcţii numerice). Notăm codurile pentru q_i şi a_j prin $S(q_i)$ şi $S(a_i)$, definindu-le după cum urmează:

	$S(\lambda) = \lambda^m T,$
pentru orice alt simbol a_j	$S(a_j) = a\lambda^m T^{j-1} 1^j,$
pentru starea finală q_{Tz}	$S(q_{Tz}) = q\lambda^n T^{-1},$
dacă $i \neq n_T$	$S(q_i) = q\lambda^n T^{-i-1} 1^i.$

Codul $S(a_j)$ unei maşini concrete T întotdeauna are lungimea m_T, iar codul $S(q_i) - n_T$. Simbolurile R, L şi $\rightarrow$ sunt introduse în alfabetul A_U, adică $S(R) = R$, $S(L) = L$, $S(\rightarrow) = \rightarrow$. Notăm codul cuvântului α, format din codurile simbolurilor, care formează acest cuvânt, prin $S(\alpha)$.

Enunţul final al problemei maşinii universale Turing este: "pentru orice maşină T şi orice cuvânt în alfabetul A_T să se construiască maşina U, care va verifica relaţia $U(S(\Sigma_T), S(\alpha)) = T(\alpha)$".

Este posibil de arătat, că poate fi construită o maşină universală, care posedă doar două simboluri în alfabetul benzii (codarea binară !). Shanon a stabilit posibilitatea construirii unei maşini U cu doar două stări, iar Minsky şi Bobrow au demonstrat imposibilitatea construirii unei maşini universale cu doar două simboluri şi două stări.

Existenţa maşinii Turing universale permite tratarea sistemului de comenzi a unei maşini oarecare T în două moduri: sau ca descrierea funcţionării unui dispozitiv concret, sau ca program pentru maşina universală U. Analogic, un algoritm de comandă poate fi realizat sau cu ajutorul unor resurse tehnice ("în metal") sau prin program – elaborând programul unei maşini universale de comandă. Însă ideea unui dispozitiv algoritmic universal este absolut independentă de nivelul de dezvoltare a resurselor tehnice, care permit construirea acestui dispozitiv. Această idee nu este o realizare a tehnicii, ci un concept matematic, descris în termeni abstracţi independenţi de resursele tehnice şi bazat pe un număr extrem de mic de noţiuni iniţiale.

Este important de menţionat, că lucrările fundamentale din teoria algoritmilor au fost publicate înaintea apariţiei calculatoarelor electronice (1930 – 1940). Această interpretare dublă păstrează şi la nivel abstract avantajele şi dezavantajele principale ale celor două variante inginereşti de realizare: cu ajutorul unor circuite electronice sau prin program. O maşină concretă T lucrează mult mai repede, pe când dispozitivul de comandă al maşinii universale este relativ complicat, fiind relativ mare numărul de stări şi comenzi. Însă complexitatea lui este constantă şi, fiind odată construit, poate fi utilizat pentru realizarea oricăror algoritmi. Va fi nevoie doar de o bandă de capacitate mai mare, care este mai ieftină şi organizată mai simplu, în comparaţie cu dispozitivul de comandă. Mai mult, pentru algoritmi noi nu este necesar să fie construite dispozitive speciale. Va fi scris doar un alt program.

5.2.8. Teza lui Turing

Este oare posibil să se construiască o maşină Turing pentru orice procedură algoritmică? La această întrebare răspunde

Teza lui Turing: **Orice algoritm poate fi realizat de o maşină Turing.**

Teza lui Turing nu poate fi demonstrată, deoarece chiar noţiunea de algoritm (procedură eficientă) nu este exactă. Teza dată nu este nici teoremă, nici postulat, ci o afirmaţie, care leagă teoria cu obiectele, pentru descrierea cărora această teorie a fost creată. Confirmarea tezei lui Turing este pe de o parte adusă de practică, iar pe de altă parte, de faptul, că descrierea algoritmilor cu ajutorul oricăror altor modele cunoscute poate fi redusă la descrierea cu ajutorul maşinilor Turing.

Teza lui Turing permite să înlocuim afirmaţii inexacte despre existenţa unor proceduri eficiente (algoritmi) cu afirmaţii exacte despre existenţa maşinilor Turing, iar cazul în care o maşină Turing nu există să fie tratat ca şi inexistenţa algoritmului în general.

5.2.9. Problema opririi

Una dintre caracteristicile obligatorii ale algoritmilor este rezultativitatea, adică după un număr finit de paşi algoritmul trebuie să pună la dispoziţia noastră rezultatul calculelor. Formal, problema poate fi pusă astfel: pentru un algoritm oarecare A şi datele iniţiale α să se stabilească, dacă lansarea lui A va conduce sau nu la un rezultat. Adică, trebuie să se elaboreze un algoritm B, pentru care $B(A,\alpha)=T$, când $A(\alpha)$ este rezultativ şi $B(A,\alpha)=F$, când $A(\alpha)$ nu dă rezultat. În baza tezei lui Turing problema dată poate fi formulată în termenii maşinilor Turing: să se construiască maşina Turing T_0, care pentru orice maşină T şi orice date iniţiale α ale acestei maşini:

1) $T_0(\Sigma_T, \alpha)=T$, dacă maşina $T(\alpha)$ se opreşte şi
2) $T_0(\Sigma_T, \alpha)=F$, dacă maşina $T(\alpha)$ nu se opreşte.

Aceasta este problema opririi. Poate fi demonstrată teorema:

Teorema 5.3. Nu există o maşină Turing T_0, care poate rezolva problema opririi pentru o maşină Turing arbitrară T.

În baza tezei lui Turing imposibilitatea construirii maşinii Turing T_0 înseamnă lipsa unui algoritm de soluţionare a problemei date. Este un exemplu de problemă algoritmic nerezolvabilă (indecidabilă). Problema opririi maşinii Turing este algoritmic nerezolvabilă (problemă indecidabilă), adică nu poate fi algoritmic rezolvată problema determinării rezultativităţii algoritmilor.

Lipsa unei soluţii pentru cazul general nu înseamnă în mod obligator lipsa soluţiei pentru cazuri particulare. Indecidabilitatea problemei opririi poate fi interpretată ca inexistenţa unui algoritm general pentru depanarea programelor, adică un algoritm, care având la dispoziţie textul programului şi datele iniţiale

poate să stabilească, dacă programul conţine sau nu cicluri infinite pe aceste date. Pentru marea majoritate a programelor (cazuri particulare!) poate fi stabilit, dacă un program conţine sau nu cicluri infinite, poate fi determinată şi eliminată cauza acestui fenomen (programe depanate) şi obţinut rezultatul necesar.

5.3. Exerciţii şi probleme

5.1. Elaboraţi programul unei maşini Turing, care transformă un cuvânt iniţial 1^x în x numere egale cu 1 şi separate prin marchere.

5.2. Elaboraţi diagrama maşinii Turing, care transformă cuvântul $1*1*...*1*1$ cu x unităţi în 1^x.

5.3. Construiţi maşina Turing, care transformă numărul y dacă este impar, într-un număr par şi-l lasă nemodificat în caz contrar.

5.4. Pe banda unei maşini Turing sunt înscrise mai multe numere în cod unar $y_1 = 1^{x_1}$, $y_2 = 1^{x_2}$,…, $y_n=1^{x_n}$, separate prin cel mult trei blancuri. Propuneţi programul unei maşini, care va elimina toate blancurile dintre numere.

BIBLIOGRAFIE.

1. L.A.Zadeh. Fuzzy sets, Inf. Control 8, 338-353, 1965.
2. С. П. Кузнецов, В. Д. Адельсон-Вельский. Дискретная математика для инженера. Москва, Энергоатомиздат, 1988
3. Edgar Codd. Database Design: Relational, Distributed, & Object-Oriented Concepts. Addison-Wesley Longman, 1991
4. A.Valachi, F.Hoza, V.Onofrei, R.Silion. Analiza, sinteza şi testarea dispozitivelor numerice. Iaşi, „Nord-Est”, 1993
5. Chin-Liang Chang, Richard Char-Tung Lee. Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973
6. Peter Bauer, Stephan Nouak, Roman Winkler. A brief course in Fuzzy Logic and Fuzzy Control, Version: 1.2, http://www.flll.uni-linz.ac.at/pdw/fuzzy/index.html, 1996
7. T. Bânzaru ş.a. Matematici speciale. Bucureşti, E.D.P., 1981
8. Norbert Weiner. Cybernetics or Control and Communication in the Animal and the Machine. MIT Press 1948, 1961 (1948)