Curs 1

Privire de ansamblu

Programare declarativă

- Programatorul spune ce vrea să calculeze, dar nu specifică concret cum calculează.
- Este treaba interpretorului (compilatorului) să identifice cum să efectueze calculul respectiv.
- Tipuri de programare declarativă:
  - Programare functională (e.g., Haskell)
  - Programare logică (e.g., Prolog)
  - Limbaje de interogare (e.g., SQL)

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.

- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

- Programarea logică poate fi privită ca o deductie controlată.

- Un program scris intr-un limbaj de programare logică este

  o listă de formule intr-o logică

  ce exprimă fapte și reguli despre o problemă.

- Limbajul Prolog are la bază *logica clauzelor Horn*, un fragment al calculului cu predicate.

Propoziții (informal)

- O propoziție este un enunț care poate fi adevărat (1) sau fals (0). Propozițiile sunt notate simbolic (p, q, φ, ψ, χ, $\cdots$) și sunt combinate cu ajutorul conectorilor logici ($\neg$, $\rightarrow$, $\vee$, $\wedge$, $\leftrightarrow$).
- Mulțimea valorilor de adevăr este $\{0, 1\}$ pe care considerăm următoarele operații:

x	$\neg x$
0	1
1	0

x	y	$x \rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

$$x \vee y := max\{x, y\}$$
$$x \wedge y := min\{x, y\}$$

Exemplu

Fie φ propoziția: *Azi este miercuri, deci avem curs de PLF*.
Cine este $\neg\varphi$?
$\neg\varphi$ este: *Azi este miercuri și nu avem curs de PLF*.

Propoziții și predicate (informal)

- O propoziție este un enunț care poate fi adevărat (1) sau fals (0). Propozițiile sunt notate simbolic (p, q, φ, ψ, χ, $\cdots$) și sunt combinate cu ajutorul conectorilor logici ($\neg$, $\rightarrow$, $\vee$, $\wedge$, $\leftrightarrow$).
- Un *predicat* este un enuț care conține variabile și a cărui valoare de adevăr depinde de valorile variabilelor.

Exemplu

Fie $P(x, y)$ predicatul: *In ziua x miercuri, anul III CTI are cursul y*.

Observăm că $P(miercuri, PLF)$ este adevărat dar $P(vineri, PLF)$ este fals.

Daca *Zi* este mulțimea zilelor săptămînii iar *Curs* este mulțimea cursurilor, putem asocia predicatului P o proprietate, adică o funcție $\mathbf{P} : Zi \times Curs \rightarrow \{0, 1\}$

Programare funcțională

Esență: funcții care relaționează intrările cu ieșirile

Caracteristici:
- funcții de ordin înalt – funcții parametrizate de funcții
- grad înalt de abstractizare (e.g., functori, monade)
- grad înalt de reutilizarea codului — polimorfism

Fundamente:
- Teoria funcțiilor recursive
- Lambda-calcul ca model de computabilitate (echivalent cu mașina Turing)

Programare funcțională

- Programare funcțională în limbajul vostru preferat de programare:
  - C++x11, Python, Java 8, JavaScript, ...
  - Funcții anonime (λ-abstracții)
  - Funcții de procesare a fluxurilor de date: filter, map, reduce

λ-calcul (A. Church, 1936)

$$
\begin{array}{rll}
t = & x & \text{(variabilă)} \\
 & \mid \lambda x.\, t & \text{(abstractizare)} \\
 & \mid t\ \ t & \text{(aplicare)}
\end{array}
$$

Funcții anonime în Haskell

În Haskell, $\backslash$ e folosit în locul simbolului λ și $->$ în locul punctului.

$\lambda x.x * x$ este `\x -> x * x`

$\lambda x.x > 0$ este `\x -> x > 0`

Mulțimi. Funcții. Relații

Mulțimi

- O mulțime este o colecție de obiecte distincte care se numesc *elementele* mulțimii.
- Dacă A este o mulțime, notăm cu $a \in A$ faptul că a este un element din A.
- Notăm cu $A \subseteq B$ faptul ca orice element din A este element al lui B și spunem că A este o submulțime a lui B.
- Submulțimile pot fi definite folosind proprietăți (predicate): $\{x \in A \mid P(x)\}$
- Există mulțimea fără elemente, care se numește *mulțimea vidă* și se notează cu $\emptyset$.

Criza fundamentelor matematicii

Conform teoriei naive a mulțimilor, orice colecție definibilă este mulțime. Fie U mulțimea tuturor mulțimilor.

Paradoxul lui Russel (1902)

Fie $R = \{A \in U \mid A \notin A\}$. Atunci R este mulțime, deci $R \in U$. Obținem că $R \notin R \Leftrightarrow R \in R$.

Criza fundamentelor matematicii

- a declanșat criza fundamentelor matematicii ("foundations of mathematics")
- s-a dezvoltat teoria axiomatică a mulțimilor: Zermelo-Fraenkel (ZF), ZFC: ZF+ Axioma alegerii (*Axiom of Choice*)

Operații cu mulțimi

Fie A, B, C, T mulțimi

- $A, B \subseteq T$
  $A \cup B = \{x \in T | x \in A \text{ sau } x \in B\}$
  $A \cap B = \{x \in T | x \in A \text{ și } x \in B\}$
  $\overline{A} = \{x \in T | x \notin A\}$
- $\mathcal{P}(T) = \{A | A \subseteq T\}$
- $A \times B = \{(a, b) | a \in A \text{ și } b \in B\}$

Exemplu. $\mathcal{P}(\emptyset) = \{\emptyset\}$, $\mathcal{P}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\}$,
$\mathcal{P}(\{\emptyset, \{\emptyset\}\}) = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$, ...

Exercițiu. A, B, $C \subseteq T$
$A \times (B \cup C) = (A \times B) \cup (A \times C)$
$A \times (B \cap C) = (A \times B) \cap (A \times C)$

Mulțimi. Operații și relații n-are

n număr natural, $n \geq 1$
$A_1, \ldots, A_n \subseteq T$ mulțimi

- $\bigcup_{i=1}^{n} A_i = \{x \in T | ex.\ i \in \{1, \ldots, n\}\ x \in A_i\}$
- $\bigcap_{i=1}^{n} A_i = \{x \in T | x \in A_i\ \ or.\ i \in \{1, \ldots, n\}\}$
- $\prod_{i=1}^{n} A_i = \{(x_1, \ldots, x_n) |\ x_i \in A_i\ \ or.\ i \in \{1, \ldots, n\}\}$

Notație. $A^n = \underbrace{A \times \cdots \times A}_{n}$

Definiție

O **relație** între $A_1, \ldots, A_n$ este o submulțime a produsului cartezian $\prod_{i=1}^{n} A_i$. O relație n-ară pe A este o submulțime a lui A^n.

Mulțimi. Relații n-are

Exemple

- $| \subseteq \mathbb{N} \times \mathbb{N}$
  $| = \{(k, n) |\ ex.\ m \in \mathbb{N}$ a.î. $mk = n\}$

- $< \subseteq \mathbb{N} \times \mathbb{N}$
  $<= \{(k, n) |\ ex.\ m \in \mathbb{N}$ a.î. $m \neq 0$ si $m + k = n\}$

- $R \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{Z}$
  $R = \{(n_1, n_2, n_3, z) \mid z = n_1 - n_2 + n_3\}$

Operații cu relațiii

A, B, C mulțimi

- relația inversă:
  dacă $R \subseteq A \times B$ atunci $R^{-1} \subseteq B \times A$ și
  $R^{-1} = \{(b, a) | (a, b) \in R\}$
- compunerea relațiilor:
  dacă $R \subseteq A \times B$ și $Q \subseteq B \times C$ atunci $R \circ Q \subseteq A \times C$ și
  $R \circ Q = \{(a, c) | ex.\ b \in B\ (a, b) \in R \text{ și } (b, c) \in Q\}$
- diagonala $\Delta_A = \{(a, a) | a \in A\}$

Exercițiu
(1) Compunerea relațiilor este asociativă.
(2) Dacă $R \subseteq A \times B$ atunci $\Delta_A \circ R = R$ și $R \circ \Delta_B = R$.

Relațiii

A, B mulțimi
$R \subseteq A \times B$ (relație între A și B)

Definiție

Relația R se numește:

- totală: or. $a \in A$ ex. $b \in B$ astfel încât $(a, b) \in R$
- surjectivă : or. $b \in B$ ex. $a \in A$ astfel încât $(a, b) \in R$
- injectivă: or. a_1, $a_2 \in A$ or. $b \in B$
  $(a_1, b) \in R$ și $(a_2, b) \in R$ implică $a_1 = a_2$
- funcțională: or. $a \in A$ or. b_1, $b_2 \in B$
  $(a, b_1) \in R$ și $(a, b_2) \in R$ implică $b_1 = b_2$

Funcții.

A, B mulțimi

- O **funcție** de la A la B este o relație totală și functională, deci $R \subseteq A \times B$ este funcție dacă pentru orice $a \in A$ există un unic $b \in B$ cu $(a, b) \in R$.

  Notația $f : A \to B$ are următoarea semnificație:
  $f(a)$ este unicul element din B care este în relație cu a.

  Astfel funcția $R \subseteq A \times B$ va fi notată prin $f_R : A \to B$, unde
  $$b = f_R(a) \text{ ddacă } (a, b) \in R.$$
  Invers, relația asociată lui $f : A \to B$ este $R_f = \{(a, f(a)) | a \in A\}$.

- Pentru orice mulțime A, vom nota prin $1_A : A \to A$ funcția $f(a) = a$ oricare $a \in A$. Evident, $R_{1_A} = \Delta_A$.

- Pentru orice mulțime A, există o unică funcție de la $\emptyset$ la A (relația asociată este $\emptyset$).

Compunerea funcțiilor

$f : A \to B$ și $g : B \to C$ funcții

Definim $g \circ f : A \to C$ unde $(g \circ f)(a) = g(f(a))$.

Exercițiu. $R_{g \circ f} = R_f \circ R_g$.

Spunem că o funcție $f : A \to B$ este **inversabilă** dacă există $g : B \to A$ astfel încât $g \circ f = 1_A$ și $f \circ g = 1_B$.

O **bijecție** este o funcție injectivă și surjectivă.

Exercițiu. O funcție este bijectivă ddacă este inversabilă.

Funcția caracteristică

T mulțime, $A \subseteq T$

Funcția caracteristică a lui A în raport cu T este
$\chi_A : T \to \{0,1\}$, $\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$

Proprietăți

Dacă A, $B \subseteq T$ și $x \in T$ atunci

(1) $\chi_{A \cap B}(x) = min\,\{\,\chi_A(x),\ \chi_B(x)\,\} = \chi_A(x) \cdot \chi_B(x)$

(2) $\chi_{A \cup B}(x) = max\,\{\ \chi_A(x), \chi_B(x)\,\} = \chi_A(x) + \chi_B(x) - \chi_A(x) \cdot \chi_B(x)$

(3) $\chi_{\overline{A}}(x) = 1 - \chi_A(x)$

Funcția caracteristică

T mulțime, $\{0,1\}^T = \{f : T \to \{0,1\} | f \text{ functie }\}$

Propoziție

Există o bijecție între $\mathcal{P}(T)$ și $\{0,1\}^T$.

Dem. Funcțiile care stabilesc bijecția sunt

$F : \mathcal{P}(T) \to \{0,1\}^T$, $F(A) = \chi_A$

$G : \{0,1\}^T \to \mathcal{P}(T)$, $G(f) = \{x \in T | f(x) = 1\}$

Se arată că $F(G(f)) = f$ și că $G(F(A)) = A$
oricare $A \subseteq T$ și $f : T \to \{0,1\}$.

Familii de elemente

I, A mulțimi

- O **familie** de elemente din A indexată de I este o funcție $f : I \to A$.

  Notăm cu $\{a_i\}_{i\in I}$ familia $f : I \to A$, $f(i) = a_i$ or. $i \in I$. Vom scrie $\{a_i\}_i$ atunci cand I poate fi dedus din context.

- Dacă $\{A_i\}_i$ și $\{B_i\}_i$ sunt familii de submulțimi ale unei mulțimi T atunci definim
  $\bigcup_{i\in I} A_i = \{x \in T |\ ex.\ i \in I \text{ astfel încât } x \in A_i\}$
  $\bigcap_{i\in I} A_i = \{x \in T |\ x \in A_i\ or.\ i \in I\}$
- Dacă $I = \emptyset$, atunci $\bigcup_{i\in I} A_i = \emptyset$ și $\bigcap_{i\in I} A_i = T$.

Exercițiu. $\overline{\bigcup_i A_i} = \bigcap_i \overline{A_i}$

Produsul cartezian

I mulțime, $\{A_i\}_i$ familie de mulțimi indexată de I.

Vom nota prin $(x_i)_i$ o familie de elemente a mulțimii $\bigcup_i A_i$ cu proprietatea că $x_i \in A_i$ or. $i \in I$.

Definim $\prod_i A_i = \{f : I \to \bigcup_i A_i | f(i) \in A_i \ or.\ i \in I\}$
$= \{(x_i)_i | x_i \in A_i \ or.\ i \in I\}$.

Exercițiu. Fie I, J mulțimi
$(\bigcup_{i\in I} A_i) \times (\bigcup_{j\in J} B_j) = \bigcup_{(i,j)\in I\times J}(A_i \times B_j)$
$(\bigcap_{i\in I} A_i) \times (\bigcap_{j\in J} B_j) = \bigcap_{(i,j)\in I\times J}(A_i \times B_j)$

Algebra Logicii

George Boole - *The Mathematical Analysis of Logic* (1847) Analiza raționamentelor prin metode asemănătoare calculului algebric.

Exemplu. Fie a, b, c și d proprietăți ale substanțelor care pot interac tiona în cadrul unui experiment. Se cunosc următoarele:
(1) a și b apar simultan $\Rightarrow$ apare numai una dintre c și d,
(2) b și c apar simultan $\Rightarrow$ fie a și d apar simultan,
fie nu apare nici una,
(3) nici una dintre a și b nu apare $\Rightarrow$ nici una dintre c și d nu apare,
(4) nici una dintre c și d nu apare $\Rightarrow$ nici una dintre a și b nu apare.

Demonstrați că:
(a) nici una dintre a și b nu apare $\Rightarrow$ nu apare nici c,
(b) a, b și c nu apar simultan.

Algebra Logicii

Fie A, B, C și D mulțimile substanțelor care au, respectiv, proprietățile a, b, c și d.

(1) a și b apar simultan $\Rightarrow$ cu siguranță apare una dintre c și d,
$A \cap B \subseteq (C \cap \overline{D}) \cup (D \cap \overline{C})$

(2) b și c apar simultan $\Rightarrow$ fie a și d apar simultan,
fie nu apare nici una,
$B \cap C \subseteq (A \cap D) \cup (\overline{A} \cap \overline{D})$

(3) nici una dintre a și b nu apare $\Rightarrow$ nici una dintre c și d nu apare,
$\overline{A} \cap \overline{B} \subseteq \overline{C} \cap \overline{D}$

(4) nici una dintre c și d nu apare $\Rightarrow$ nici una dintre a și b nu apare.
$\overline{C} \cap \overline{D} \subseteq \overline{A} \cap \overline{B}$

Algebra Logicii

Notăm $AB = A \cap B$. Folosim $A \subseteq B \Leftrightarrow A\overline{B} = \emptyset$.

Ipotezele:
(1) $AB(\overline{C} \cup D)(C \cup \overline{D}) = \emptyset$
(2) $BC(\overline{A} \cup \overline{D})(A \cup D) = \emptyset$
(3) $\overline{A}\ \overline{B}(C \cup D) = \emptyset$
(4) $\overline{C}\ \overline{D}(A \cup B) = \emptyset$

Concluzia:
(1) $\overline{A \cup B} \subseteq \overline{C} \Leftrightarrow \overline{A}\ \overline{B}C = \emptyset$

Demonstrația:
$\overline{A}\ \overline{B}(C \cup D) = \emptyset \Rightarrow \overline{A}\ \overline{B}C \cup \overline{A}\ \overline{B}D = \emptyset \Rightarrow \overline{A}\ \overline{B}C = \emptyset$ și $\overline{A}\ \overline{B}D = \emptyset$

(2) exercițiu.

Relații binare

A mulțime, $R \subseteq A \times A$

Relația binară R se numește:

- reflexivă: $(x, x) \in R$ or. $x \in A$
- simetrică: $(x, y) \in R$ implică $(y, x) \in R$ or. x, $y \in A$
- antisimetrică: $(x, y) \in R$ și $(y, x) \in R$ implică $x = y$
  or. x, $y \in A$
- tranzitivă: $(x, y) \in R$ și $(y, z) \in R$ implică $(x, z) \in R$
  or. x, y, $z \in A$
- relație de preordine: reflexivă, tranzitivă
- relație de ordine: reflexivă, antisimetrică, tranzitivă
- relație de echivalență: reflexivă, simetrică, tranzitivă

Relații binare

Dacă A mulțime și $R \subseteq A \times A$ definim:

$$\mathcal{R}(R) = R \cup \Delta_A$$
$$\mathcal{S}(R) = R \cup R^{-1}$$
$$\mathcal{T}(R) = \bigcup_{n \geq 1} R^n, \text{ unde } R^n = \underbrace{R \circ \cdots \circ R}_{n} \text{ pt. } n \geq 1,\ R^0 = \Delta_A$$

Propoziție

(1) $\mathcal{R}(R)$ este reflexivă, $\mathcal{S}(R)$ este simetrică, $\mathcal{T}(R)$ este tranzitivă.
(2) Dacă $Q \subseteq A \times A$ este reflexivă (simetrică, tranzitivă) și $R \subseteq Q$ atunci $\mathcal{R}(R) \subseteq Q$ ($\mathcal{S}(R) \subseteq Q$, $\mathcal{T}(R) \subseteq Q$).

$\mathcal{R}(R)$ ($\mathcal{S}(R)$, $\mathcal{T}(R)$) este cea mai *mică* (în sensul incluziunii) relație care include R și care este reflexivă (simetrică, tranzitivă).
Spunem că $\mathcal{R}(R)$ ($\mathcal{S}(R)$, $\mathcal{T}(R)$) este închiderea lui R la reflexivitate (simetrie, tranzitivitate).

Relații binare

Observație

Fie $\mathbb{N}$ mulțimea numerelor naturale și
$R = |$ (relația de divizibilitate). Atunci $\mathcal{S}(\mathcal{T}(R)) \neq \mathcal{T}(\mathcal{S}(R))$.

Dacă A mulțime și $R \subseteq A \times A$ definim $\mathcal{E}(R) = \mathcal{T}(\mathcal{S}(\mathcal{R}(R)))$.

Propoziție

(1) $\mathcal{E}(R)$ este relație de echivalență oricare $R \subseteq A \times A$.
(2) Dacă $Q \subseteq A \times A$ este relație de echivalență și
$R \subseteq Q$ atunci $\mathcal{E}(R) \subseteq Q$.

$\mathcal{E}(R)$ este cea mai *mică* (în sensul incluziunii) relație de echivalență care include R. Spunem că $\mathcal{E}(R)$ este echivalența generată de R.

Relații de echivalență

Exemple:

- Fie $k \geq 2$ și $\equiv \subseteq \mathbb{N} \times \mathbb{N}$
  $(n_1, n_2) \in \equiv$ ddacă ex. $x_1, x_2 \geq 0$, ex. $0 \leq r < k$ a.î. $n_1 = kx_1 + r$ și $n_2 = kx_2 + r$.

- Fie $f : A \to B$ o funcție și $\ker f \subseteq A \times A$
  $\ker f = \{(a_1, a_2) \in A \times A \mid f(a_1) = f(a_2)\}$

- (X, E) un graf neorientat ($E \subseteq X \times X$), $\sim \subseteq X \times X$
  $(x, y) \in \sim$ ddacă $x = y$ sau există un drum de la x la y.

- Fie *Var* o mulțime de variabile și *Form* mulțimea formulelor calculului propozițional clasic care se pot construi folosind variabilele din *Var*. Definim $\sim \subseteq Form \times Form$ prin
  $(\varphi, \psi) \in \sim$ ddacă formula $(\varphi \to \psi) \wedge (\psi \to \varphi)$ este teoremă.

Relații de echivalență

Fie A o mulțime și $\sim \subseteq A \times A$ o relație de echivalență.
Vom nota prin $x \sim y$ faptul că $(x, y) \in \sim$.

Pentru orice $x \in A$ definim $\hat{x} = \{y \in A | x \sim y\}$
(clasa de echivalență a lui x).

Un **sistem de reprezentanți** pentru $\sim$ este o submultime $X \subseteq A$ cu proprietatea că oricare $a \in A$ există un unic $x \in X$ astfel încât $a \sim x$.

Propoziție

Au loc următoarele proprietăți:
(1) $\hat{x} = \hat{y} \Leftrightarrow x \sim y$
(2) $\hat{x} \cap \hat{y} = \emptyset \Leftrightarrow x \not\sim y$
(3) $A = \bigcup\{\hat{x} | x \in X\}$ oricare ar fi $X \subseteq A$ un sistem de reprezentanți pentru $\sim$.

Partiții

A mulțime

O **partiție** a lui A este o familie $\{A_i\}_{i\in I}$ de submulțimi nevide ale lui A care verifică proprietățile:
(p1) $A_i \cap A_j = \emptyset$ oricare $i \neq j$,
(p2) $A = \bigcup_{i\in I} A_i$.

Propoziție

(1)Dacă $\{A_i\}_{i\in I}$ este o partiție a lui A atunci relația

$$x \sim y \Leftrightarrow \text{ex } i \in I \text{ astfel încât } x, y \in A_i$$

este relație de echivalență pe A.
(2) Dacă $\sim \subseteq A \times A$ este relație de echivalență și $X \subseteq A$ este un sistem de reprezentanți, atunci $\{\hat{x} | x \in X\}$ este o partiție a lui A.
(3) Există o bijecție între mulțimea relațiilor de echivalență pe A si mulțimea partițiilor lui A.

Dem. exercițiu

Mulțimea cât

A mulțime, $\sim \subseteq A \times A$ relație de echivalență pe A

Definim $A/_{\sim} = \{\hat{x} | x \in A\}$ (mulțimea claselor de echivalență).

Definim $p_{\sim} : A \to A/_{\sim}$, $p_{\sim}(x) = \hat{x}$ or. $x \in A$ (surjecția canonică).
Se observă că $\ker p_{\sim} = \sim$.

Proprietatea de universalitate a mulțimii cât

Fie B o mulțime și
$f : A \to B$ o funcție a.î. $\sim \subseteq \ker f$.
Atunci există o unică funcție
$\hat{f} : A/_{\sim} \to B$ a.î.
$\hat{f}(\hat{x}) = f(x)$ or. $x \in A$.

$$\begin{tikzcd} A \arrow[r, "p_{\sim}"] \arrow[d, "f"'] & A/_{\sim} \arrow[dl, dotted, "\hat{f}"] \\ B & \end{tikzcd}$$

Dem. exercițiu

Cardinali

Două mulțimi A și B sunt **echipotente** dacă există o funcție bijectivă $f : A \to B$. În acest caz scriem $A \simeq B$.

Propoziție. Următoarele proprietăți sunt adevărate:
(i) $A \simeq A$,
(ii) $A \simeq B$ implică $B \simeq A$,
(iii) $A \simeq B$ și $B \simeq C$ implică $A \simeq C$.

Relația de echipotență este o relație de echivalență. Pentru o mulțime A definim **cardinalul** lui A ca fiind $|A| = \{B \mid A \simeq B\}$.

Două mulțimi finite sunt echipotente dacă și numai dacă au același număr de elemente. Cardinalul unei mulțimi finite este numărul de elemente.

Există mulțimi infinite care nu sunt echipotente: $\mathbb{N} \not\simeq 2^{\mathbb{N}}$.

$|\mathbb{N}| = \aleph_0$ (*aleph-zero*) ,
$2^{\aleph_0} = |2^{\mathbb{N}}| = |\mathbb{R}| = \mathfrak{c}$ (*puterea continuului*)

Monoidul cuvintelor

Un **alfabet** este o mulțime de simboluri.
Un **cuvînt** este un șir finit de simboluri din alfabet.

Fie A un alfabet. Definim $A^+ = \{a_1 \dots a_n \mid n \geq 1, a_1, \dots, a_n \in A\}$ și $A^* = A^+ \cup \{\lambda\}$ unde λ este *cuvîntul vid*.

Operația de concatenare $\cdot : A^* \times A^* \to A^*$ se definește prin

$$(a_1 \dots a_n) \cdot (b_1 \dots b_k) = a_1 \dots a_n b_1 \dots b_k$$

$(A^*, \cdot, \lambda)$ este un monoid și se numește *monoidul cuvintelor peste alfabetul A*.

Pentru două cuvinte w, $w' \in A^*$ definim
$w \sim w'$ dacă și numai dacă au același număr de litere.

Exercițiu. $\sim$ este relație de echivalență pe A și $A/_\sim \simeq \mathbb{N}$.

Relații binare

Fie A mulțime și $R \subseteq A \times A$ o relație de preordine. Definim $x \sim y$ ddacă $(x, y) \in R$ și $(y, x) \in R$.

Propoziție. $\sim$ este relație de echivalență pe A.
Dem. exercițiu

Pe $A/\sim$ definim $\hat{x} \prec \hat{y} \Leftrightarrow (x, y) \in R$.

Lemă. $\prec$ este bine definită, adică
$x \sim x_1$, $y \sim y_1$ și $\hat{x} \prec \hat{y}$ implică $\hat{x_1} \prec \hat{y_1}$.
Dem. exercițiu

Propoziție. $\prec$ este relație de ordine pe $A/\sim$.
Dem. exercițiu

Exemple. Dezvoltați construcția anterioară pentru:
$A = \mathbb{Z} \times (\mathbb{N} \setminus \{0\})$
$R = \{((z_1, n_1), (z_2, n_2)) \in A \times A | z_1 \cdot n_2 \leq z_2 \cdot n_1\}$
Caracterizați $A/\sim$.

Programare logică & Prolog

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

- Programarea logică poate fi privită ca o deducție controlată.
- Un program scris într-un limbaj de programare logică este

  o listă de formule într-o logică

  ce exprimă fapte și reguli despre o problemă.
- Exemple de limbaje de programare logică:
  - Prolog
  - Answer set programming (ASP)
  - Datalog

Ce veți vedea la laborator

Prolog

- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet
- vom folosi implementarea SWI-Prolog

Limbajul Prolog a fost folosit pentru programarea sistemului IBM Watson!

Bibliografie:

Learn Prolog Now!http://www.let.rug.nl/bos/lpn/

Programare logică - în mod idealist

- Un ”program logic” este o colecție de proprietăți presupuse (sub formă de formule logice) despre lume (sau mai degrabă despre lumea programului).
- Programatorul furnizează și o proprietate (o formula logică) care poate să fie sau nu adevărată în lumea respectivă (întrebare, *query*).
- Sistemul determină dacă proprietatea aflată sub semnul întrebării este o consecință a proprietăților presupuse în program.
- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecință a programului.

Exemplu de program logic

$$\begin{aligned}
\texttt{oslo} &\rightarrow \texttt{windy}\\
\texttt{oslo} &\rightarrow \texttt{norway}\\
\texttt{norway} &\rightarrow \texttt{cold}\\
\texttt{cold} \land \texttt{windy} &\rightarrow \texttt{winterIsComing}\\
& \texttt{oslo}
\end{aligned}$$

Exemplu de întrebare

Este adevărat `winterIsComing`?

Putem să testăm în SWI-Prolog

Program:

```
windy :- oslo.
norway :- oslo.
cold :- norway.
winterIsComing :- windy, cold.
oslo.
```

Intrebare:

```
?- winterIsComing.
true
```

http://swish.swi-prolog.org/

Sintaxă: constante, variabile, termeni compuși

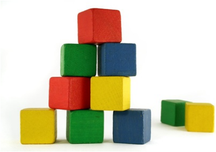
- Atomi: `brian`, `'Brian Griffin'`, `brian_griffin`

- Numere: `23`, `23.03`,`-1`

  Atomii și numerele sunt constante.

- Variabile: `X`, `Griffin`, `_family`

- Termeni compuși: `father(peter, stewie_griffin)`,
  `and(son(stewie,peter), daughter(meg,peter))`
  - forma generală: atom(termen,..., termen)
  - atom-ul care denumește termenul se numește functor
  - numărul de argumente se numește aritate



Un mic exercițiu sintactic

Care din următoarele șiruri de caractere sunt constante și care sunt variabile în Prolog?

- vINCENT – constantă
- Footmassage – variabilă
- variable23 – constantă
- Variable2000 – variabilă
- big_kahuna_burger – constantă
- 'big kahuna burger' – constantă
- big kahuna burger – nici una, nici alta
- 'Jules' – constantă
- _Jules – variabilă
- '_Jules' – constantă

Program în Prolog = bază de cunoștințe

Exemplu

Un program în Prolog:

```
father(thomas,emma).
father(thomas,arthur).

mother(jane,emma).
mother(jane,arthur).

watson(thomas).
watson(jane).

watson(X) :- father(Y,X),  watson(Y).
```

Un program în Prolog este o bază de cunoștințe (Knowledge Base).

Program în Prolog = mulțime de predicate

Practic, gândim un program în Prolog ca o mulțime de predicate cu ajutorul cărora descriem *lumea* (*universul*) programului respectiv.

Exemplu

```
father(thomas,emma).
father(thomas,arthur).

mother(jane,emma).
mother(jane,arthur).

watson(thomas).
watson(jane).

watson(X) :- father(Y,X),  watson(Y).
```

Predicate:
father/2
mother/2
watson/1

Un program în Prolog

Program

- Un program în Prolog este format din reguli de forma

  `Head :- Body.`

- Head este un predicat, iar Body este o secvență de predicate separate prin virgulă.
- Regulile fără `Body` se numesc fapte.

Exemplu

- Exemplu de regulă: `watson(X) :- father(Y,X), watson(Y).`
- Exemplu de fapt: `father(thomas,emma).`

Interpretarea din punctul de vedere al logicii

- operatorul :- este implicația logică $\leftarrow$

Exemplu

`cti(X) :- seria36(X)`

dacă `seria36(X)` este adevărat, atunci `cti(X)` este adevărat.

- virgula , este conjuncția $\wedge$

Exemplu

`seria36(X) :- coleg(Y,X), seria36(Y).`

dacă `coleg(Y,X)` și `seria36(Y)` sunt adevărate,
atunci `seria36(X)` este adevărat.

Interpretarea din punctul de vedere al logicii

- mai multe reguli cu același `Head` definesc același predicat, între defiții fiind un sau logic.

Exemplu

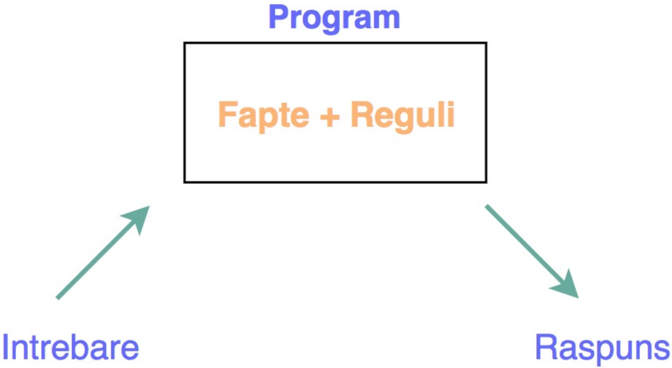
```
cti(X) :- fizica(X).
cti(X) :- proiectare_logica(X).
cti(X) :- electrotehnica(X).
```

dacă
`fizica(X)` este adevărat sau `proiectare_logica(X)` este adevărat sau `electrotehnica(X)` este adevărat,
atunci
`cti(X)` este adevărat.

Un program în Prolog

Cum folosim un program în Prolog?

Întrebări în Prolog



Întrebări și ținte în Prolog

- Prolog poate răspunde la întrebări legate de consecințele relațiilor descrise într-un program în Prolog.
- Întrebările sunt de forma:

$$\texttt{?- predicat}_1\texttt{(...),...,predicat}_n\texttt{(...).}$$

- Prolog verifică dacă întrebarea este o consecință a relațiilor definite în program.
- Dacă este cazul, Prolog caută valori pentru variabilele care apar în întrebare astfel încât întrebarea să fie o consecință a relațiilor din program.
- Un predicat care este analizat pentru a se răspunde la o întrebare se numește țintă (goal).

Întrebări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- `false` – în cazul în care întrebarea nu este o consecință a programului.
- `true` sau valori pentru variabilele din întrebare în cazul în care întrebarea este o consecință a programului.

Exemplu

```
?- watson(emma)
true
?- watson(ann)
false
```

```
?- watson(X)
X = thomas ;
X = jane ;
X = emma ;
X = arthur ;
false
```

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
X = a.
```

Pentru a răspunde la întrebare se caută o potrivire (unificator) între scopul `foo(X)` și baza de cunoștințe. Raspunsul este substituția care realizează potrivirea, în cazul nostru `X = a`.

Răspunsul la întrebare este găsit prin unificare!

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
X = a.

?- foo(d).
false
```

Dacaă nu se poate face potrivirea, răspunsul este false.

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
X = a.
```

Dacă dorim mai multe răspunsuri, tastăm ;

```
?- foo(X).
X = a ;
X = b ;
X = c.
```

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a).
foo(b).
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```

```
?- trace.
true.

[trace]  ?- foo(X).
   Call: (8) foo(_4556) ? creep
   Exit: (8) foo(a) ? creep
X = a ;
   Redo: (8) foo(_4556) ? creep
   Exit: (8) foo(b) ? creep
X = b ;
   Redo: (8) foo(_4556) ? creep
   Exit: (8) foo(c) ? creep
X = c.
```

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog redenumește variabilele.

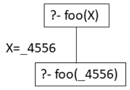
Exemplu

Să presupunem că avem programul:

```
foo(a).
foo(b).
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă regulile în ordinea apariției lor.

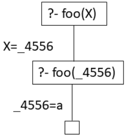
Exemplu

Să presupunem că avem programul:

```
foo(a).
foo(b).
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



În acest moment, a fost găsită prima soluție: X=_4556=a.

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă clauzele în ordinea apariției lor.

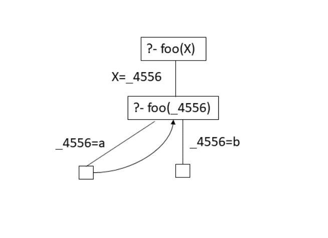
Exemplu

Să presupunem că avem programul:

```
foo(a).
foo(b).
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



Dacă se dorește încă un răspuns, atunci se face un pas înapoi în arborele de căutare și se încearcă satisfacerea țintei cu o nouă valoare.

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, Prolog încearcă clauzele în ordinea apariției lor.

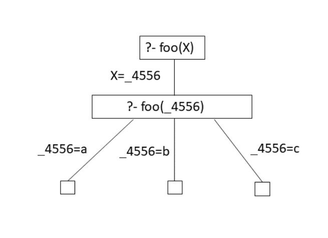
Exemplu

Să presupunem că avem programul:

```
foo(a).
foo(b).
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



arborele de căutare

Cum găsește Prolog răspunsul

Exemplu

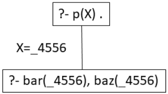
Să presupunem că avem programul:

```
bar(b).
bar(c).

baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```



Cum găsește Prolog răspunsul

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

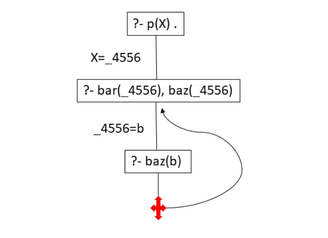
Exemplu

Să presupunem că avem programul:

```
bar(b).
bar(c).
baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```



Cum găsește Prolog răspunsul

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

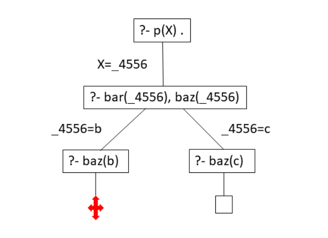
Exemplu

Să presupunem că avem programul:

```
bar(b).
bar(c).
baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```



Soluția găsită este: X=_4556=c.

Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

Exemplu

Să presupunem că avem programul:

```
bar(c).
bar(b).

baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```

Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

Exemplu

Să presupunem că avem programul:

```
bar(c).
bar(b).

baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
X = c ;
false
```

Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

Exemplu: program = baza de cunoștințe (kb)

Exemplu

Corect:

```
bigger(elephant, horse).
bigger(horse, donkey).
bigger(donkey, dog).
bigger(donkey, monkey).
is_bigger(X, Y) :-
      bigger(X, Y).
is_bigger(X, Y) :-
      bigger(X, Z),
      is_bigger(Z, Y).
```

Incorect:

```
bigger(elephant, horse).
bigger(horse, donkey).
is_bigger(X, Y) :-
      bigger(X, Y).
bigger(donkey, dog).
bigger(donkey, monkey).
is_bigger(X, Y) :-
      bigger(X, Z),
      is_bigger(Z, Y).
```

Faptele și regulile trebuie grupate după atomii folosiți în `Head`.

Exemple de întrebări și răspunsuri

Exemplu

```
bigger(elephant, horse).
bigger(horse, donkey).
bigger(donkey, dog).
bigger(donkey, monkey).

is_bigger(X, Y) :-
      bigger(X, Y).
is_bigger(X, Y) :-
      bigger(X, Z),
      is_bigger(Z, Y).
```

```
?- is_bigger(elephant, horse).
true

?- bigger(donkey, dog).
true

?- is_bigger(elephant, dog).
true

?- is_bigger(monkey, dog).
false

?- is_bigger(X, dog).
X = donkey ;
X = elephant ;
X = horse
```

Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
?- is_bigger(X, Y), is_bigger(Y,Z).
X = elephant,
Y = horse,
Z = donkey
X = elephant,
Y = horse,
Z = dog
X = elephant,
Y = horse,
Z = monkey
X = horse,
Y = donkey,
Z = dog
.....
```

Compararea termenilor: =,\=, ==,\==

`T = U`	reușește dacă există o potrivire (termenii se unifică)
`T \= U`	reușește dacă nu există o potrivire
`T == U`	reușește dacă termenii sunt identici
`T \== U`	reușește dacă termenii sunt diferiți

Exemplu

```
?- X = Y.
X = Y.

?- p(X,q(Z)) = p(Y,X).
X = Y, Y = q(Z).

?- 2 = 1 + 1
false
```

```
?- X == Y .
false

?- p(X,Y) == p(X,Y).
true

?- 2 == 1 + 1
false
```

- În exemplul de mai sus, `1+1` este privită ca o expresie, nu este evaluată. Există și predicate care forțează evaluarea.

Compararea cu evaluare

- Relația =:= este folosită pentru a compara (rezultatul evaluării) expresiilor aritmetice:

Exemplu

```
?- 2 ** 3 =:= 3 + 5.
true
?- 2 ** 3 = 3 + 5.
false
```

- Atenție la diferența dintre =:= și =
  - =:= compară două expresii aritmetice
  - = caută un unificator
- Exemple de relații disponibile:
  <, >, =<, >=, =\= (diferit), =:= (aritmetic egal)

Aritmetica în Prolog

Exemplu

```
?- 8 = 3 + 5.
false
?- 8 is 3 + 5.
true
```

Operatorul `is`:

- Primește două argumente. Primul argument este fie un număr, fie o variabilă. Al doilea argument trebuie să fie o expresie aritmetică validă, cu toate variabilele inițializate.
- Dacă primul argument este un număr, atunci rezultatul este `true` dacă este egal cu evaluarea expresiei aritmetice din al doilea argument. Dacă primul argument este o variabilă, răspunsul este pozitiv dacă variabila poate fi unificată cu evaluarea expresiei aritmetice din al doilea argument.

Negarea ca eșec: `\+ pred(X)`

Exemplu

```
animal(dog). animal(elephant). animal(sheep).

?- animal(cat).
false
?- \+ animal(cat).
true
```

- Clauzele din Prolog dau doar condiții suficiente, dar nu și necesare pentru ca un predicat să fie adevărat.
- Pentru a da un răspuns pozitiv la o țintă, Prolog trebuie să construiască o "demonstrație" pentru a arată că mulțimea de fapte și reguli din program implică acea țintă.
- Astfel, un răspuns `false` nu înseamnă neapărat că ținta nu este adevărată, ci doar că Prolog nu a reușit să găsească o demonstrație.

Un program mai complicat

Problema colorării hărților

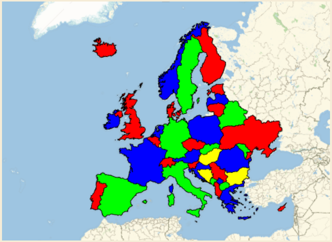
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?

Exemplu

Trebuie să definim:

- culorile
- harta
- constrângerile



Sursa imaginii

Problema colorării hărților

Definim culorile

Exemplu

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).
```

Problema colorării hărților

Definim culorile, harta

Exemplu

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).

harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),
                            vecin(RO,MD), vecin(RO,BG),
                            vecin(RO,HU), vecin(UA,MD),
                            vecin(BG,SE), vecin(SE,HU).
```

Problema colorării hărților

Definim culorile, harta și constrângerile. Cum punem întrebarea?

Exemplu

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).

harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),
                            vecin(RO,MD), vecin(RO,BG),
                            vecin(RO,HU), vecin(UA,MD),
                            vecin(BG,SE), vecin(SE,HU).
vecin(X,Y) :- culoare(X),
              culoare(Y),
              X \== Y.

?- harta(RO,SE,MD,UA,BG,HU).
```

Problema colorării hărților

Ce răspuns primim?

Exemplu

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).

harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),
                            vecin(RO,MD), vecin(RO,BG),
                            vecin(RO,HU), vecin(UA,MD),
                            vecin(BG,SE), vecin(SE,HU).
vecin(X,Y) :- culoare(X),
              culoare(Y),
              X \== Y.

?- harta(RO,SE,MD,UA,BG,HU).
```

Problema colorării hărților

Exemplu

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).
harta(RO,SE,MD,UA,BG,HU) :-   vecin(RO,SE), vecin(RO,UA),
                              vecin(RO,MD), vecin(RO,BG),
                              vecin(RO,HU), vecin(UA,MD),
                              vecin(BG,SE), vecin(SE,HU).
vecin(X,Y) :- culoare(X),
              culoare(Y),
              X \== Y.
```

```
?- harta(RO,SE,MD,UA,BG,HU).
RO = albastru,
SE = UA, UA = rosu,
MD = BG, BG = HU, HU = verde
```

Pe data viitoare!