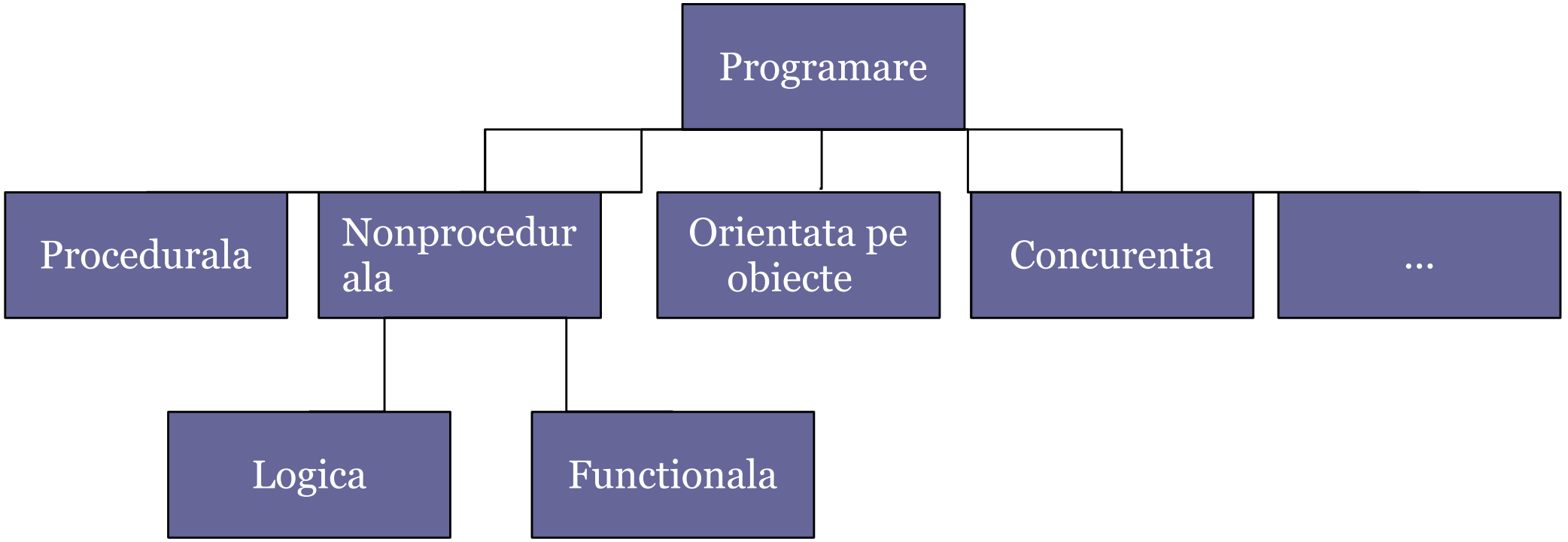
Introducere in programarea nonprocedurala (PNP)

Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Motivatia necesitatii PNP

- In cadrul paradigmelor de programare un loc aparte il ocupa programarea nonprocedurala.
- Programarea nonprocedurala cuprinde:
  - Programarea logica
  - Programarea functionala.
- Programarea nonprocedurala mai poarta numele de programare declarativa.

Plasarea PNP



Avantajele programelor PNP

- ✓Sunt orientate catre implementari logice.
- ✓Sunt extrem de potrivite pentru sistemele expert apartinand domeniului inteligentei artificiale.
- ✓Sunt usor de analizat, transformat, verificat, intretinut.
- ✓In general, sunt eficiente si competitive in comparatie cu programele nedeclarative.
- ✓Programul este mai degraba “intrebat” decat executat.

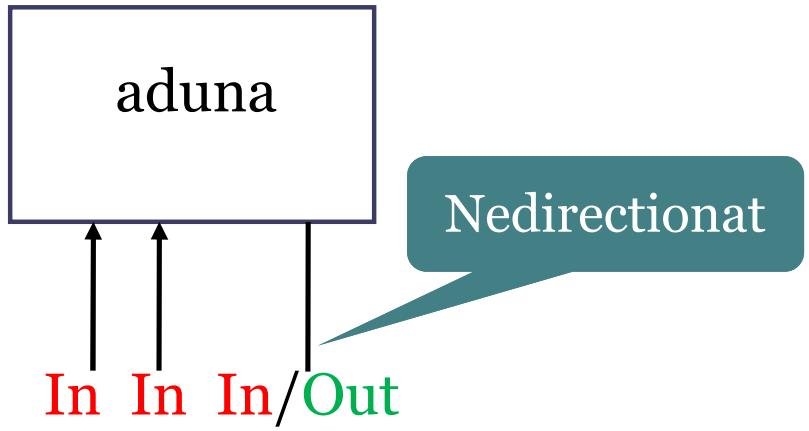
Dezavantajele programelor PNP

- Nu sunt usor implementabile si utilizabile pentru algoritmii matematici de cautare si optimizare din cadrul inteligentei artificiale.
- Nu sunt cele mai potrivite pentru exprimarea algoritmilor folositi in industrie.
- Au mecanisme mai putin directe decat ale celor nedeclarative si sunt considerate de multe ori mai “neprietenoase”.

Programare LOGICA versus programare FUNCTIONALA. Schema

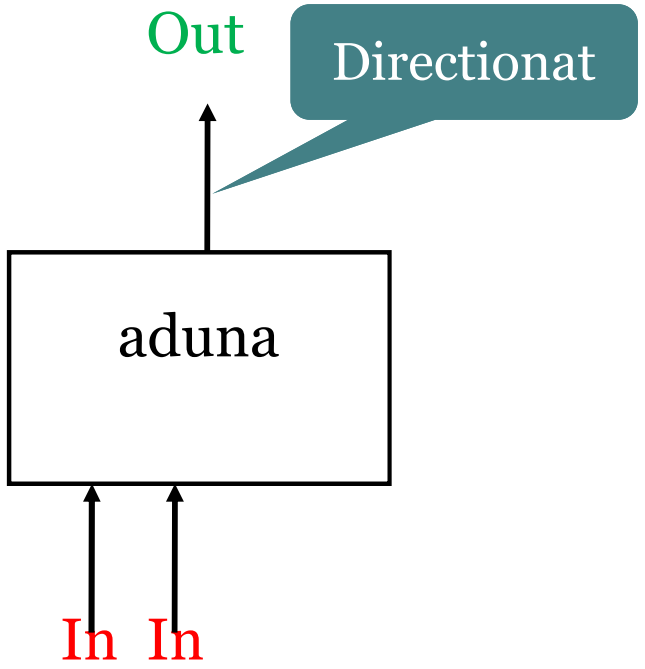
Programare logica

- Adunarea a doua numere



Programare functionala

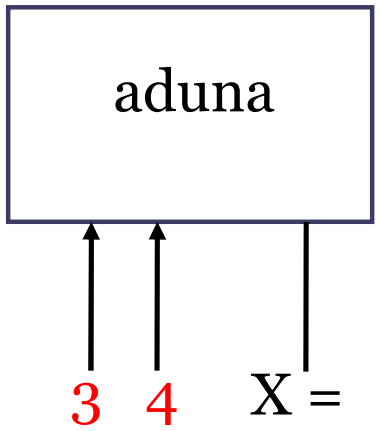
- Adunarea a doua numere



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

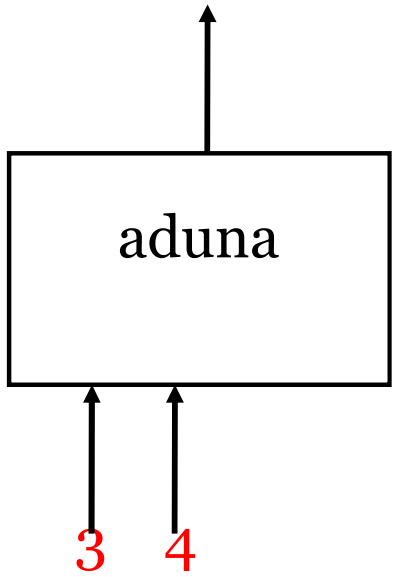
Programare logica

- Adunarea a doua numere



Programare functionala

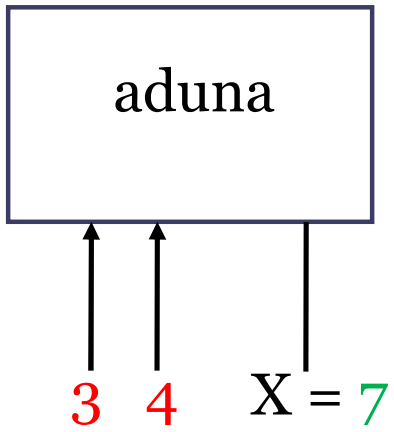
- Adunarea a doua numere



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

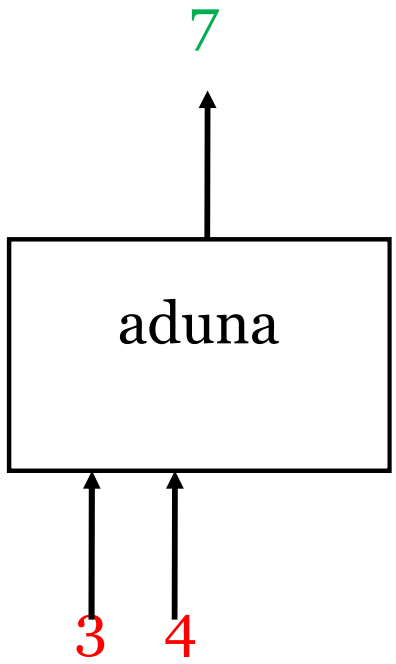
Programare logica

- Adunarea a doua numere



Programare functionala

- Adunarea a doua numere



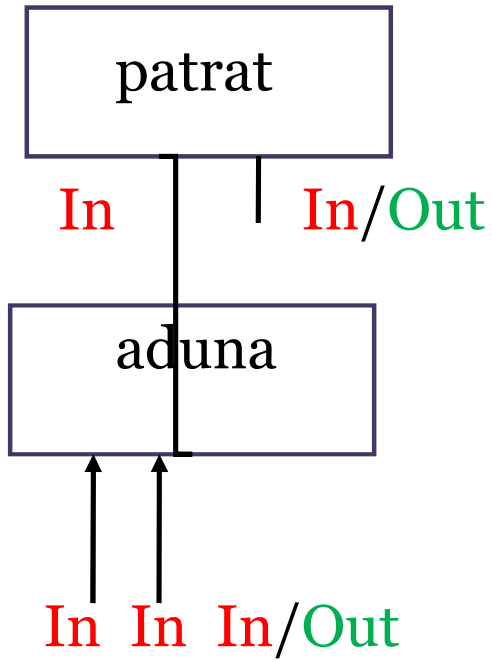
Programare LOGICA versus programare FUNCTIONALA. Scrierea simbolica

Programare logica	Programare functionala
• Adunarea a doua numere aduna $\subseteq$ In $\times$ In $\times$ In/Out aduna(3, 4, X) X = 7	• Adunarea a doua numere aduna(In, In) $\rightarrow$ Out aduna(3, 4) 7

Programare LOGICA versus programare FUNCTIONALA. Schema

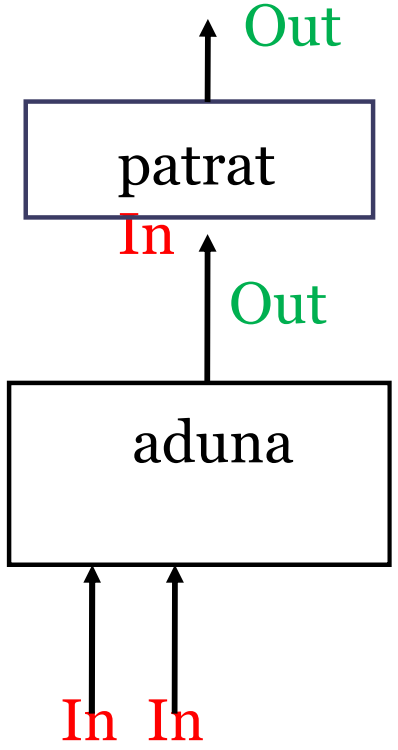
Programare logica

- Patratul sumei a doua numere



Programare functionala

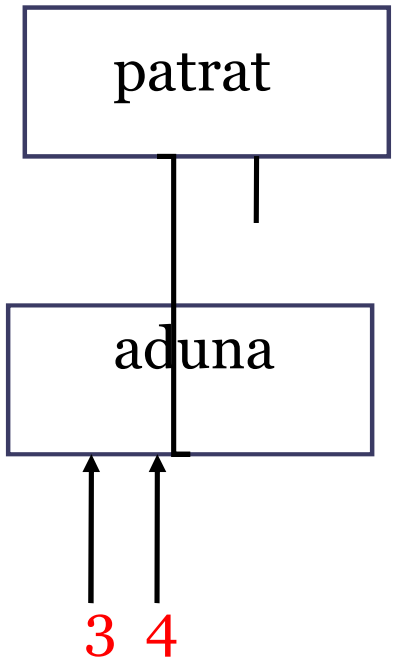
- Patratul sumei a doua numere



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

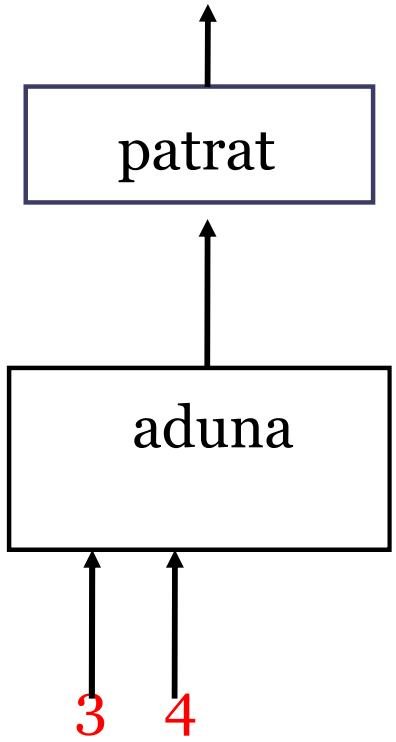
Programare logica

- Patratul sumei a doua numere



Programare functionala

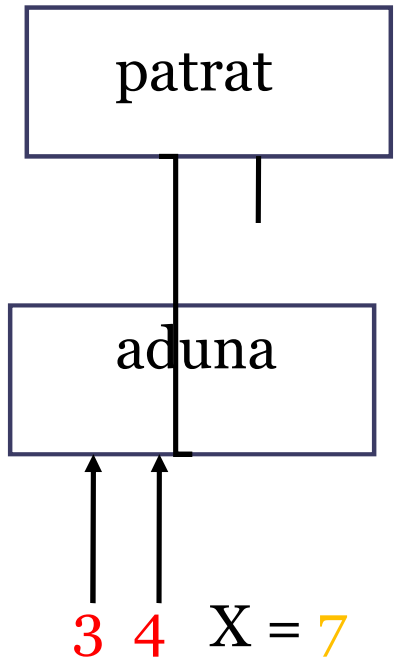
- Patratul sumei a doua numere



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

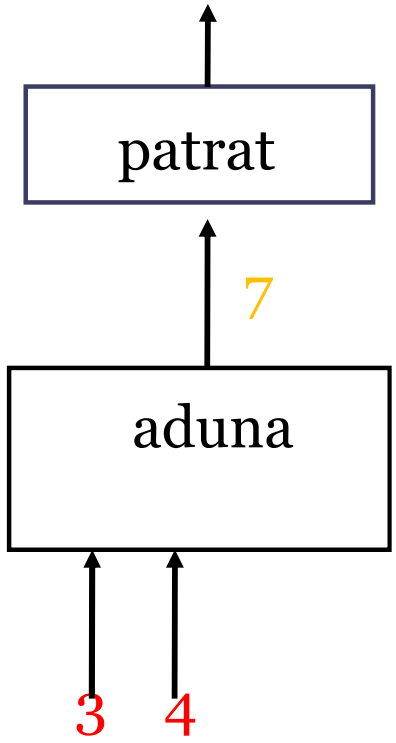
Programare logica

- Patratul sumei a doua numere



Programare functionala

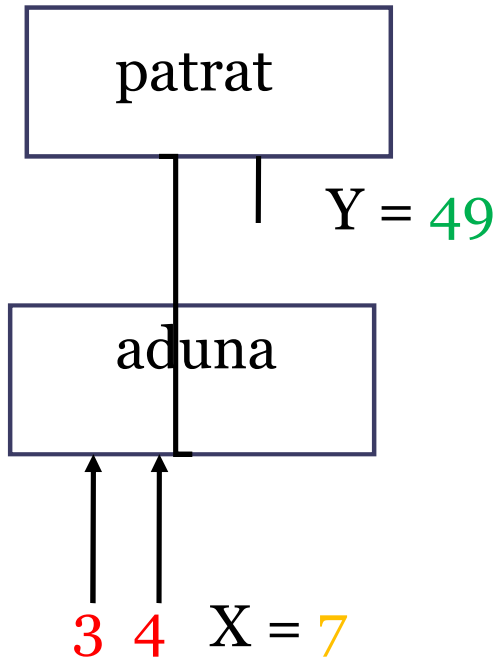
- Patratul sumei a doua numere



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

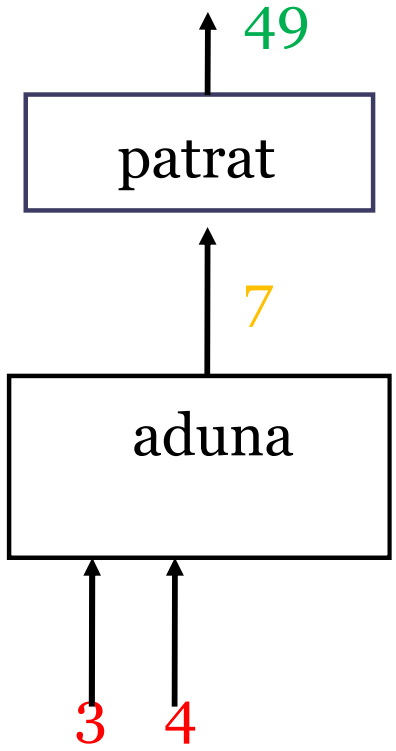
Programare logica

- Patratul sumei a doua numere



Programare functionala

- Patratul sumei a doua numere



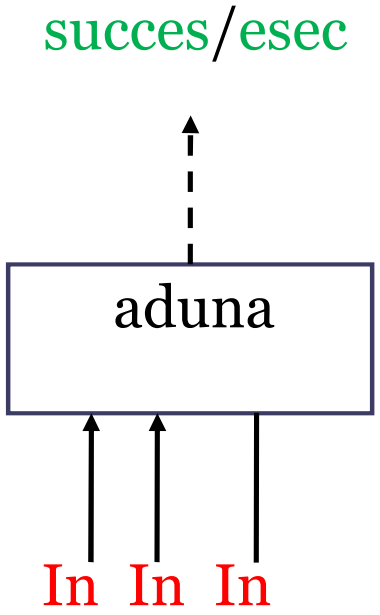
Programare LOGICA versus programare FUNCTIONALA. Scrierea simbolica

Programare logica	Programare functionala
• Patratul sumei a doua numere aduna(3, 4, X), patrat(X, Y) X = 7 patrat(7, Y) X = 7 Y = 49	• Patratul sumei a doua numere patrat(aduna(3, 4)) patrat(7) 49

Programare LOGICA versus programare FUNCTIONALA. Schema

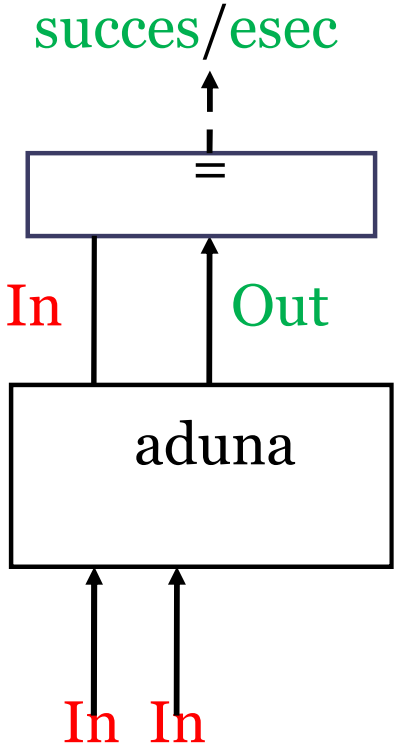
Programare logica

- Testarea egalitatii



Programare functionala

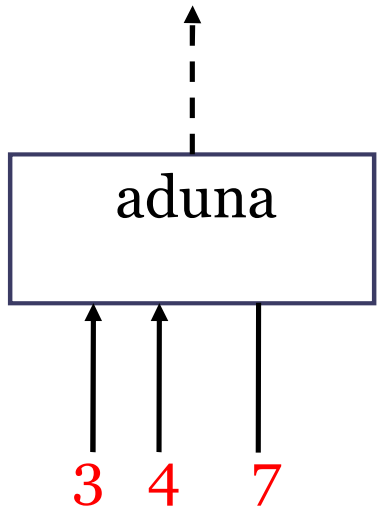
- Testarea egalitatii



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

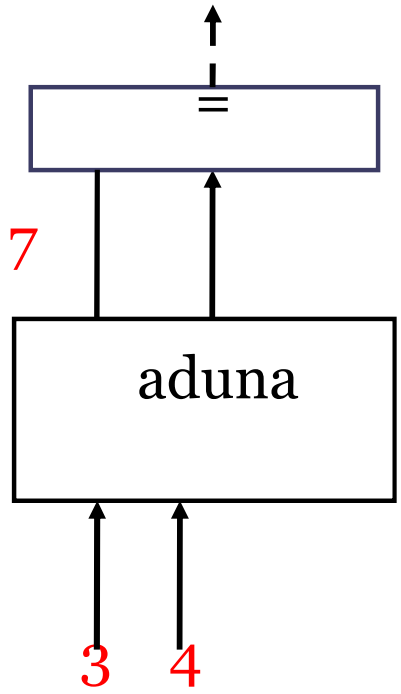
Programare logica

- Testarea egalitatii



Programare functionala

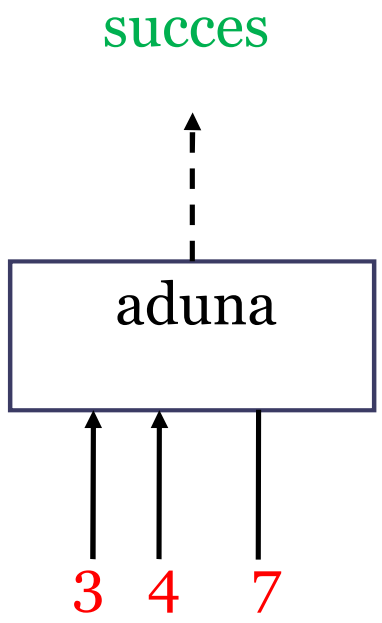
- Testarea egalitatii



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

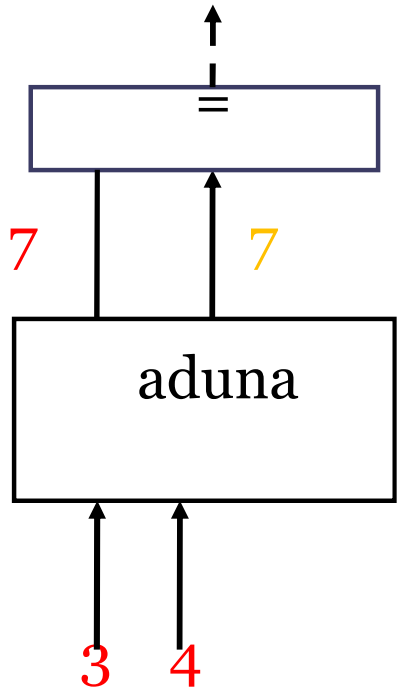
Programare logica

- Testarea egalitatii



Programare functionala

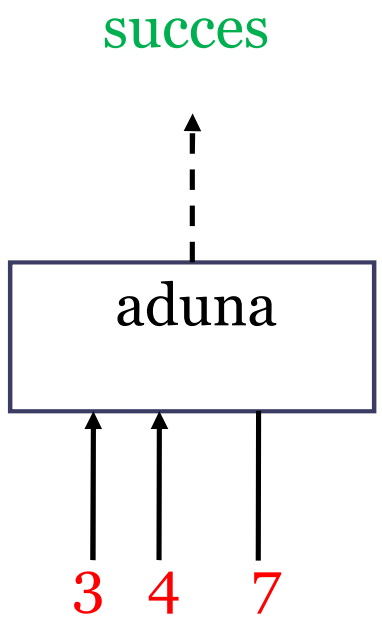
- Testarea egalitatii



Programare LOGICA versus programare FUNCTIONALA. Schema. Exemplu

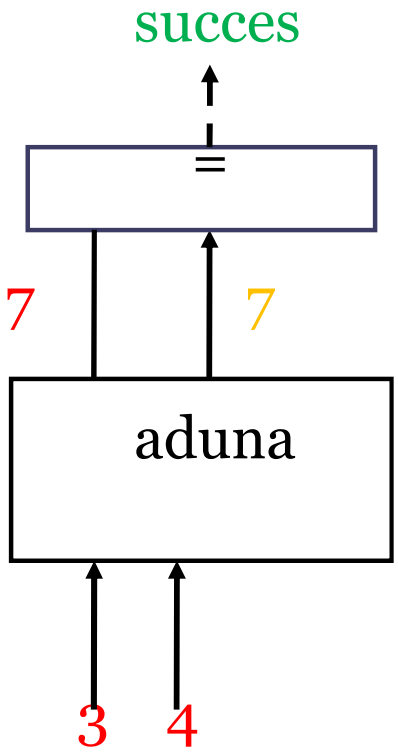
Programare logica

- Testarea egalitatii



Programare functionala

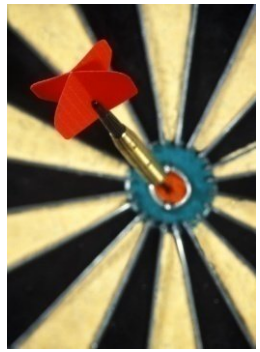
- Testarea egalitatii



Programare LOGICA versus programare FUNCTIONALA. Scrierea simbolica

Programare logica	Programare functionala
• Testarea egalitatii aduna(3, 4, 7) succes	• Testarea egalitatii 7 = aduna(3, 4) 7 = 7 succes

Scopurile acestui curs



Programare logica

- Notiuni avansate ale limbajului PROLOG.
- Implementari pretabile programarii logice.
- Intelegerea particularitatilor programarii logice.
- Observarea diferentelor fata de programarea functionala.

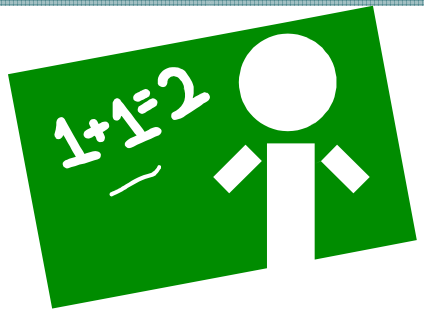
Programare functionala

- Notiuni de baza ale limbajului LISP.
- Implementari pretabile programarii functionale.
- Intelegerea particularitatilor programarii functionale.
- Observarea diferentelor fata de programarea logica.

Consecintele acestui curs

- Cunostinte elementare asupra programarii neprocedurale.
- Intelegerea diferentelor fata de programarea nedeclarativa.
- Observarea tipurilor de probleme care sunt pretabile programarii neprocedurale si a celor care sunt potrivite programarii nedeclarative.
- Obtinerea unei note bune la examen
  - Proba practica (proba laborator) 50%
    - Dezvoltarea unui program in Prolog
    - Dezvoltarea unui program in Lisp
  - Test grila (proba scrisa) 50%

PROLOG



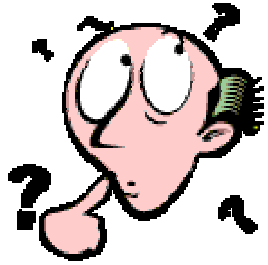
- Numele este o abreviere pentru **PRO**grammation en **LOG**ique.
- Prima versiune a fost creata in 1972 de catre Alain Colmerauer si Philippe Roussel, bazandu-se pe interpretarea clauzelor Horn data de catre Robert Kowalski.
- Vom folosi versiunea Swi-Prolog.
- Bibliografie:
  - Leon Sterling, Ehud Shapiro, *The Art of Prolog, Second Edition: Advanced Programming Techniques (Logic Programming)*, MIT Press, 1994.
  - Internet.

LISP

- Numele provine de la **LIS**t **P**rocessing.
- Codul LISP se scrie sub forma listelor parantetizate (sau s - expresii).
- Prima versiune a aparut in 1958 si a fost dezvoltata de catre Steve Russell, Timothy P. Hart si Mike Levin, bazandu-se pe calculul lambda al lui Alonzo Church.
- Vom folosi versiunea CLISP.
- Bibliografie:
  - Stuart C.Shapiro, *Common Lisp: An Interactive Approach*, Computer Science Press, 1992.
  - Internet.

I know what you did last summer… term

- A trecut mult de anul trecut…
- Nu prea mai tinem minte de atunci…
- Prolog, am mai facut noi Prolog?…
- A, Prolog, limbajul acela urat…
- Sa recapitulam asadar notiunile de baza din Prolog.



Ce trebuie sa ne amintim din Prolog?

- Fisierul Prolog
- Descrierea relatiilor
  - Fapte
  - Reguli
- Numere, atomi, variabile, operatii
- Unificare
- Recursivitate
- Liste
  - Accesare elemente
  - Unificare

Fisierul Prolog

- Se deschide programul Notepad.
- Se acceseaza File / Save As.
- In cadrul optiunii File Name se va scrie numele fisierului, urmat de extensia .pl:
  - nume.pl, de exemplu.
- In final, in campul Save as Type se completeaza cu All Files.
- Se face dublu-click pe fisier:
  - Directorul curent devine cel in care se afla fisierul
  - Fisierul este compilat
- Daca se mai opereaza modificari asupra sa, compilarea se face prin [nume].

Fara extensia .pl!

Descrierea relatiilor in Prolog

Numele relatiilor incep cu litera mica!

In Prolog, orice predicat se incheie cu punct!

Faptele – ceea ce stim despre problema:

- *nume_relatie(arg1, arg2, ..., argN).*
- nume_relatie este numele relaţiei (al predicatului) iar *arg1, arg2, ...* - argumentele.
- N reprezintă aritatea predicatului nume_relatie.
- Exemplu: frate(dan, maria).
- Un predicat de aritate 0 se poate defini simplu: nume_predicat.

Descrierea relatiilor in Prolog

- Regulile (legaturi logice) – definesc predicate pe baza altor predicate
  - *nume_relatie(arg1, ..., argN) :- nume_relatie_1(...), ..., nume_relatie_M(...).*
- Două predicate care au acelaşi nume, dar un număr diferit de argumente se consideră că sunt predicate diferite.
- Fiecare predicat dintr-un program este definit de existenţa uneia sau mai multor clauze.
- Clauzele care definesc acelasi predicat se afla una lângă alta în fişierul sursă.

Tipuri de termeni in Prolog

- Argumentele unui predicat Prolog se numesc termeni si pot avea urmatoarele tipuri:
  - Numere pozitive sau negative.
  - Atomii – text care începe cu literă mică.
  - Variabilele – încep cu literă mare sau *underline* (_).

Tipuri de termeni in Prolog

- Numerele:
  - -12, 7.1, 24

- Atomii:
  - Sunt alcatuiti de obicei din litere şi cifre, având primul caracter o literă mică.
  - salut, douaCuvinteAlaturate, un_atom, a2 sunt atomi
  - nu-este-atom, 5nu, _faraunderline, Literamare nu sunt atomi
  - 'acesta-este-atom', 'inca un atom', 'Atom' - atomi

Tipuri de termeni in Prolog

- Variabilele:
  - Sunt asemănatoare atomilor, cu excepţia că ele încep cu literă mare sau cu underline (_).
  - Variabila, _variabila, Alta_vaRiabila2 sunt variabile.
- Variabila anonima:
  - Sunt variabilele care incep cu underline (_).
  - Apare in doua cazuri:
    - Cand valoarea unei variabile nu este folosita in cadrul unui clauze.
    - In lucrul cu predicatul fail.

Variabila anonima - continuare

- Variabila nu este folosita in cadrul unui predicat:

```
semn(X, 1) :- X >= 0.
semn(_X, -1).
```

- In lucrul cu predicatul fail:

```
parinte(andrei, dan).
parinte(andrei, laura).
copii(Tata) :- parinte(Tata, Copil), tab(2),
   write_ln(Copil), fail.
copii(_OriceVariabila).
```

Operații matematice in Prolog

- Pentru evaluarea unei expresii aritmetice, se foloseste predicatul predefinit is:
  - X is <expresie aritmetică>.
  - suma(N1, N2, S) :- S is N1 + N2.

- Operatori matematici utilizati:
  - <, >, =<, >=, == (sau =), =\= (sau \=).
  - Ultimii doi operatori sunt pentru a verifica egalitatea dintre două numere, respectiv pentru a verifica dacă două numere sunt diferite.

Operaţii matematice in Prolog

- Scrierea 2+3 nu reprezintă o instrucţiune care păstrează rezultatul acestei adunări.
- Reprezintă mai degrabă atomul ‘adunarea lui 2 cu 3’.
- Termenul 2+3 este diferit de termenul 4+1.

```
numar(3).
numar(4).
numar(5).
? – numar(2 + 1).
No
? – X is 2 + 1, numar(X).
X = 3
Yes
```

Unificarea in Prolog

- Reprezinta modul în care Prologul realizează potrivirile între termeni:
  - X = marian.
    - Yes
  - marian = andrei.
    - No
  - place(maria, X) = place(Y, andrei).
    - Y = maria, X = andrei
  - f(X, a) = f(a, X).
    - X = a
  - place(maria, X) = place(X, andrei).
    - No

Recursivitatea in Prolog

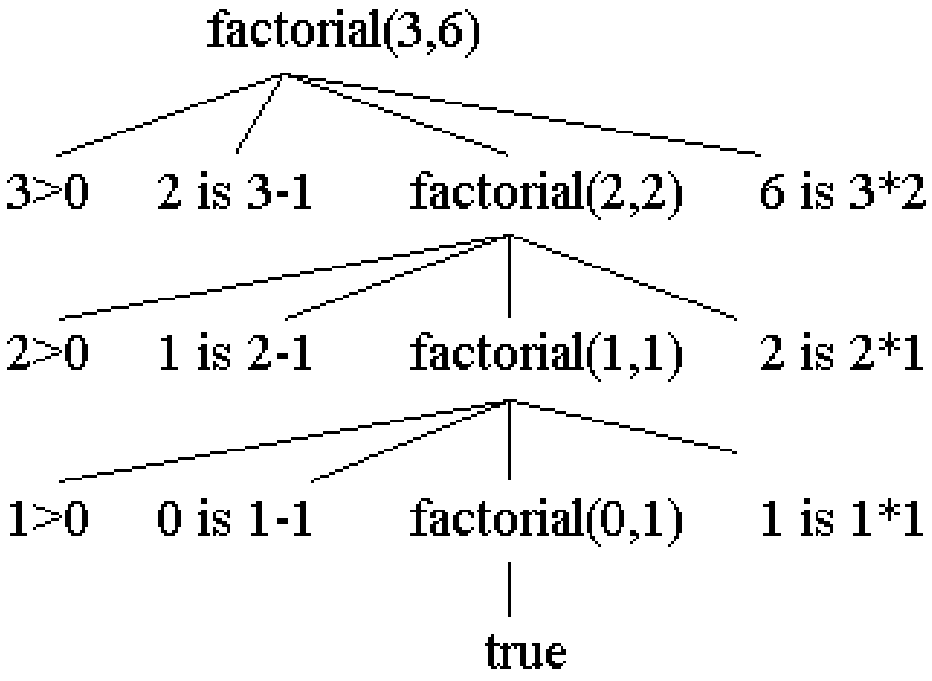
- Programarea în Prolog depinde foarte mult de acest principiu.
- Prolog-ul este important si fiindca ne invata sa gandim recursiv.
- Recursivitatea implică definirea unui predicat în funcţie de el însuşi.
- Mecanismul recursivitatii consta in faptul ca întotdeauna definim predicatul la o scală mai mică.
- Este echivalentă cu demonstrarea prin inducţie din matematică.

Recursivitatea in Prolog

- O definitie recursiva are doua parti:
  - Conditia elementara.
  - Partea recursiva.
- Condiţia elementară defineşte un caz simplu, care ştim că este întotdeauna adevărat.
- Partea recursivă simplifică problema eliminând iniţial un anumit grad de complexitate şi apoi apelându-se ea însăşi.
- La fiecare nivel, condiţia elementară este verificată:
  - dacă s-a ajuns la ea, recursivitatea se încheie;
  - altfel, recursivitatea continuă.

Recursivitatea in Prolog

- Factorialul:




```
factorial(0,1).
factorial(N,F) :- N>0, N1 is N-1, factorial(N1,F1), F
   is N * F1.

?- factorial(3, X).
X = 6
```

Recursivitatea in Prolog

- Implementarea functiilor recursive, de exemplu, Fibonacci:

f(1) = 1, f(2) = 1, f(n) = f(n - 1) + f(n - 2)

```
fibonacci(1, 1).
fibonacci(2, 1).
fibonacci(N, F):- N1 is N – 1, N2 is N – 2,
   fibonacci(N1, F1), fibonacci(N2, F2), F is F1 + F2.

?- fibonacci(3, X).
X = 2
```

Liste in Prolog

- Data viitoare...