Paradigme de programare

2010-2011, semestrul 2

Curs 1

Cuprins

- Obiectivele cursului
- Ce este o paradigma (de programare)
- Paradigme existente
- Concepte ale programarii si limbajelor de programare
- Aplicatii ale paradigmelor studiate
- Criterii de popularitate a limbajelor de programare
- Exemple de programare in Scheme, Haskell, Clips si Prolog

Motto

"*The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.*" -- Edsger Dijkstra, How do we tell truths that might hurt

Obiectivele cursului - abstract

- Noi moduri de gandire in rezolvarea problemelor (think out of the box)
- Paradigma = joc nou cu reguli, provocari, facilitati si constrangeri noi (este nevoie de flexibilitate, adaptabilitate si placerea de a fi stimulat mintal) (ex: pf nu are atribuiri)
- Usurinta de a invata concepte si limbaje noi (mereu necesar unui programator) (teorie conform careia ce putem gandi este limitat de limbaj)

Obiectivele cursului – (mai) concret

- Alegerea unui limbaj adecvat problemei
- Analiza facilitatilor si limitarilor din fiecare limbaj, capacitatea de a profita de aceasta diversitate (de ex: simuland facilitati in limbaje care nu ofera acele facilitati)
- Perfectionarea stilului de programare (structura, eficienta)
- Cultura de programator

Obiectivele cursului – intre noi fie vorba ☺

- Sa fie o experienta placuta, provocatoare si palpitanta
- Sa aveti cel putin cateva revelatii
- Sa punem programarea in perspectiva si sa putem privi limpede in intercorelarea conceptelor cu care jonglam

Cum vom invata?

- Elemente teoretice strict necesare
- Exemple
- Analiza de cod
- Implementare de cod
- Citirea documentatiei recomandate
- Discutii

Paradigma - filozofic

- Kuhn: practicile care definesc o disciplina stiintifica la un anumit moment in timp
- Dicteaza:
  - ce se studiaza
  - ce fel de probleme se pun si cum se formuleaza ele
  - cum sunt interpretate rezultatele
- Exemplu: fizica (Newton/Einstein)
- Paradigma = “box” (din “think out of the box”)

“Paradigm shift”

- Stiinta “normala” – se bazeaza pe acumularea de informatie peste ceea ce se stie deja (in paradigma existenta) (Aristotel)
- Stiinta “revolutionara” – pune la indoiala insasi paradigma, isi pune problema “ce altceva s-ar putea sti” (Platon)
- Stiinta revolutionara duce la paradigm shift

Paradigma de programare

- Stil fundamental de a programa
- Dicteaza:
  - Cum se reprezinta datele problemei (variabile, functii, obiecte, fapte, constrangeri etc)
  - Cum se prelucreaza reprezentarea (atribuiri, evaluari, fire de executie, continuari, fluxuri etc)
- Favorizeaza un set de concepte si tehnici de programare
- Influenteaza felul in care sunt ganditi algoritmii de rezolvare a problemelor
- Limbaje – in general multiparadigma (ex: Python – imperativ, functional, orientat pe obiecte)

Principalele paradigme de programare existente

- Programare imperativa
- Programare declarativa
  - Programare functionala
  - Programare asociativa
  - Programare logica
- Programare orientata pe obiecte

Alte paradigme de programare

- Programare paralela
- Programare bazata pe constrangeri

Programarea imperativa!!!

- Bazata pe ideile lui Von Neumann
- Starea programului variaza in functie de timp
- Executie secventiala (reteta)
- Variabile reprezentate ca locatii in memorie si modificate prin atribuiri
- Abstractiunea specifica: procedura
- C, Pascal, Fortran, Basic, Algol

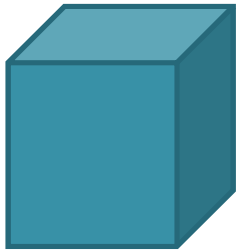
(((Programarea functionala)))

- Bazata pe matematica si teoria functiilor (calcul lambda)
- Atemporala
- Valori imutabile (nu exista atribuiri, nu exista efecte laterale)
- Functiile = valori de ordinul 1
- Toate prelucrarile = compunere si aplicare de functii (“need driven”)
- Abstractiunea specifica: functia
- Lisp, Scheme, Haskell, ML

Programarea asociativa \/ logica

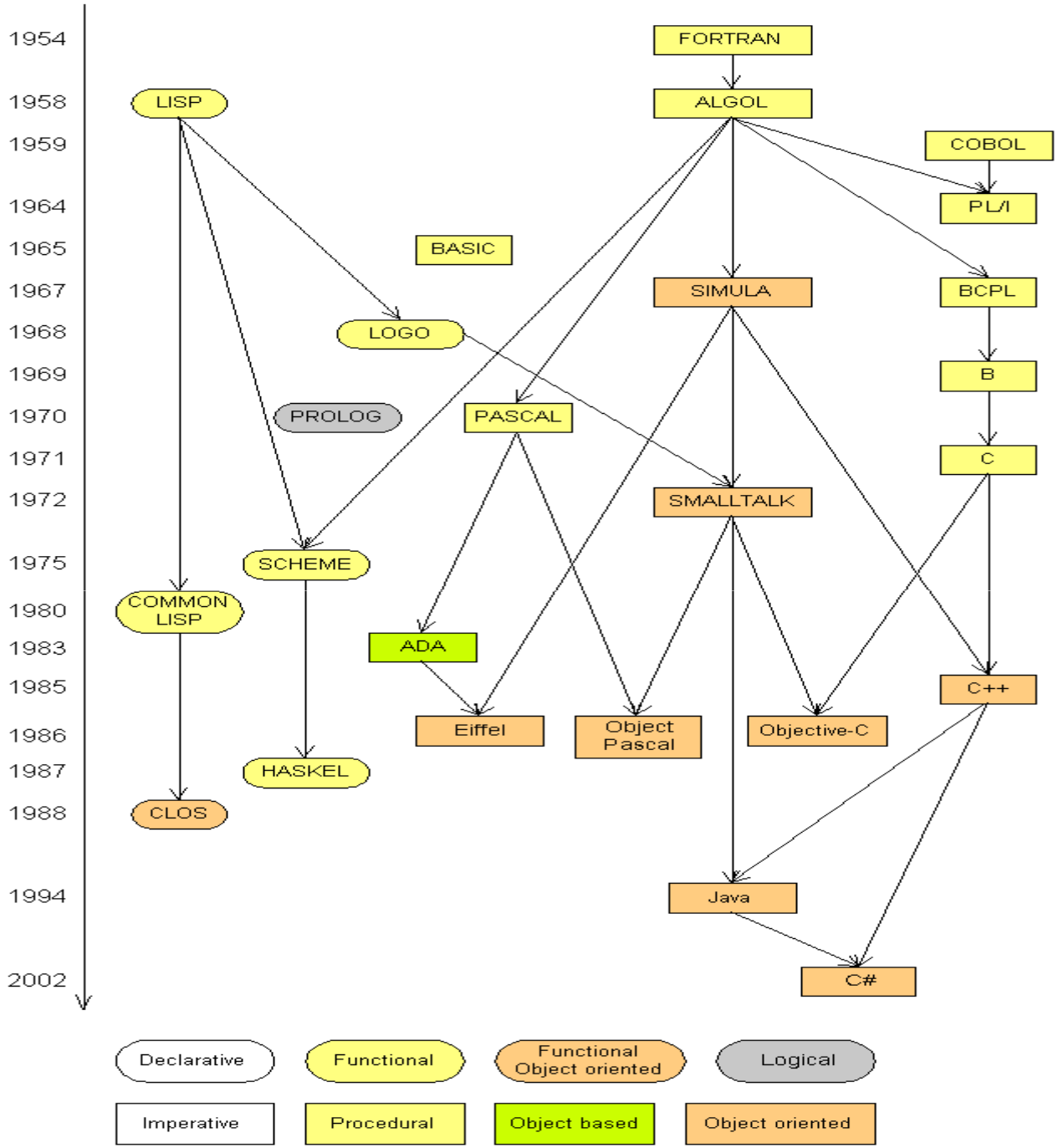
- Se bazeaza pe demonstratii automate (inteligenta artificiala)
- Fapte/axiome, reguli de inferenta, interogari
- Nu conteaza ordinea regulilor
- Se descrie CE anume e o solutie, nu CUM se ajunge la ea
- Cautarea solutiei = cautare in multimea de fapte, condusa de reguli
- Clips, Prolog

Programarea orientata pe obiecte



- Bazata pe teoria conceptelor si pe interactiunea umana cu mediul
- Obiecte cu proprietati interne (intruparea conceptelor), clase (concepte)
- Ierarhie bazata pe mostenire
- Comunicare prin transmitere de mesaje (“message passing”)
- Abstractiunea specifica: obiectul
- Java, C++, Smalltalk

Istoric


1954
1958
1959
1964
1965
1967
1968
1969
1970
1971
1972
1975
1980
1983
1985
1986
1987
1988
1994
2002
FORTRAN
ALGOL
LISP
COBOL
PL/I
BASIC
SIMULA
BCPL
LOGO
B
PROLOG
PASCAL
C
SMALLTALK
SCHEME
COMMON LISP
ADA
C++
Eiffel
Object Pascal
Objective-C
HASKEL
CLOS
Java
C#
Declarative
Functional
Functional Object oriented
Logical
Imperative
Procedural
Object based
Object oriented

Concepte caracteristice limbajelor de programare

- **Sisteme de tip**
  - Tipare tare/slaba/statica/dinamica
  - Echivalenta/compatibilitate/conversie de tip
  - Inferenta de tip
- **Abstractiuni procedurale**
  - Functii curry/uncurry
  - Functii ca valori de ordinul 1
  - Functionale
  - Inchideri functionale (closures)
  - Continuari
  - Corutine

Mai multe concepte

- **Strategii de evaluare**
  - Functii stricte/nestricte
  - Evaluare normala/aplicativa/lenesa etc
- **Reprezentarea datelor**
  - Tipuri de date abstracte
  - Niveluri de abstractizare
  - Liste si list comprehensions
  - Fluxuri
  - Efecte laterale
  - Transparenta/opacitate referentiala
- **Variabile**
  - Legare pe lant static/dinamic a variabilelor
  - Domeniu de vizibilitate (scope)
  - Durata de viata (extent)
  - Context computational al unui punct din program

Si mai multe concepte

- **Controlul fluxului**
  - Recursivitate pe stiva/coada/arborescenta
  - Exceptii
  - Continuari
  - Control condus de date
  - Forward/backward chaining
- **Gestiunea memoriei**
  - Garbage collection

…etc

- Intelegerea acestor concepte => o mai buna intelegere a oricarui limbaj de programare
  - => o mai buna intelegere a programelor scrise
  - => o alegere informata a limbajului adecvat rezolvarii fiecarei probleme

Alte subiecte interesante

- Criterii pentru evaluarea limbajelor
  - lizibilitate
  - usurinta/viteza de programare
  - gradul de predispozitie la erori
  - eficienta (eficienta executiei versus eficienta dezvoltarii codului)
  - cod interpretat versus cod compilat

…etc

Aplicatii “serioase” ale paradigmelor studiate

- Compilatoare
  - Ghc = 90K linii de Haskell
  - Caml = implementat in Caml
  - Unele versiuni de Scheme = implementate in Scheme
  - Erlang = implementat in Erlang

Aplicatii “serioase” ale paradigmelor studiate

- Demonstratoare de teoreme
  - LCF (1972), HOL, Isabelle (scrise in ML)
  - Folosesc functionale si sisteme de tip
  - Teoremele din sistem sunt instantieri ale unui TDA
  - Sistemul de tip se asigura ca pot fi demonstrate doar folosind reguli de inferenta specificate de operatorii TDA-ului
  - Tactica de demonstratie = functie care primeste un scop de demonstrat si intoarce o lista de sbscopuri si o justificare
  - Justificare = functie de la demonstratii de subscopuri la demonstratii de scopuri
  - Dezvoltarea de tactici complexe = compunere de functii (de tactici simple)

Aplicatii “serioase” ale paradigmelor studiate

- Erlang (limbaj functional pentru aplicatii din telecomunicatii)
  - Ex: Ericsson Mobility Server
  - foloseste functii ca valori de ordinul 1, recursivitate pe coada, garbage collection
  - Exista primitive care trimit orice fel de valoare ca mesaj catre un proces, asigurand si compresia acesteia
  - Server Erlang = mica functie cu argumente reprezentand starea serverului; corpul functiei primeste un mesaj, realizeaza evaluarea ceruta si intoarce rezultatul, terminand cu un apel recursiv pe coada avand ca argumente noua stare a serverului

Aplicatii “serioase” ale paradigmelor studiate

- Pdiff
- CPL/Kleisli
- Natural Expert

etc…

Criterii de popularitate (sau de ce aceste paradigme nu au fost folosite mai intens pana acum)

- Compatibilitate (cu componente deja scrise in limbaje mainstream)
- Biblioteci
- Portabilitate si instalare
- Stabilitate versus dezvoltare pentru a sustine cercetarea
- Unelte (ex: debuggere)
- Obisnuinta si inertie
- Killer Apps

Fibonacci in Scheme

```
(define (fibonacci n)
  (if (< n 2)
      1
      (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))
```

```
> (map fibonacci '(0 1 2 3 4 5 6 7 8 9 10))
(1 1 2 3 5 8 13 21 34 55 89)
```

Fibonacci in Haskell

```
fibonacci 0 = 1
fibonacci 1 = 1
fibonacci n = fibonacci (n-1) + fibonacci (n-2)
```

```
Main> map fibonacci [0..10]
[1,1,2,3,5,8,13,21,34,55,89]
```

Un Fibonacci spectaculos in Haskell

```
fib = 1 : 1 : [ x+y | (x,y)<-zip fib (tail fib)]
```

```
Main> take 10 fib
[1,1,2,3,5,8,13,21,34,55]
```

Clips… nu a fost facut pentru Fibonacci

```
(deffacts fibo-facts
    (fib 0 1)
    (fib 1 1)
    (fib-needed 5)
    (fib-done no))

(defrule next-fibo
    (fib-done no)
    (fib ?n ?fibn)
    (fib ?m&:(= ?m (- ?n 1)) ?fibm)
=>
    (assert (fib (+ ?n 1) (+ ?fibm ?fibn))))

(defrule stop-fibo
    (declare (salience 1))
    ?f <- (fib-done no)
    (fib-needed ?n)
    (fib ?n ?fibn)
=>
    (retract ?f))
```

Output Clips

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (fib 0 1)
f-2     (fib 1 1)
f-3     (fib-needed 5)
f-5     (fib 2 2)
f- 6    (fib 3 3)
f-7     (fib 4 5)
f-8     (fib 5 8)
For a total of 8 facts.
```

Fibonacci in Prolog

```
fib(0,1):-!.
fib(1,1):-!.
fib(X,Rez):-
   Y is X-1, Z is X-2,
   fib(Y,R1), fib(Z,R2),
   Rez is R1+R2.
```

```
1 ?- fib(10,Rez).
Rez=89.
```

De la specificatie formala la cod functional

```
data Natural =
    Zero |
    Succ Natural
        deriving Show

add Zero n = n
add (Succ m) n = Succ (add m n)
```

```
Main> add (Succ (Succ Zero)) (Succ (Succ (Succ
   Zero)))
Succ (Succ (Succ (Succ (Succ Zero))))
```