Paradigme de Programare

S.I. dr. ing. Andrei Olaru
slides: Mihnea Muraru si Andrei Olaru

Catedra de Calculatoare

2013 – 2013, semestrul 2

Cursul 4

Programare funcţională în Scheme

Cuprins

Introducere

Deosebiri faţă de λ_0

- Tipat – dinamic/latent
  - Variabilele nu au tip
  - Valorile au tip (`3`, `#f`)
  - Verificarea se face la execuţie, în momentul aplicării unei funcţii (evaluare aplicativă)
- Permite recursivitate textuală
- Avem domenii de vizibilitate a variabilelor

Tipare

Modalităţi de tipare

- Rolul tipurilor: exprimare a intenției programatorului, abstractizare, documentare, optimizare, verificare
- Clasificare după momentul verificării:
  - statică
  - dinamică
- Clasificare după rigiditatea regulilor:
  - tare
  - slabă

Tipare statică vs. dinamică

Tipare statică:

- La compilare
- Valori şi variabile
- Rulare mai rapidă
- Rigidă: sancţionează orice construcţie
- Debugging mai facil
- Declaraţii explicite sau inferenţe de tip
- Pascal, C, C++, Java, Haskell

Tipare dinamică:

- La rulare
- Doar valori
- Rulare mai lentă (necesită verificarea tipurilor)
- Flexibilă: sancţionează doar când este necesar
- Debugging mai dificil
- Metaprogramare (v. `eval`)
- Python, Scheme, Prolog, JavaScript, PHP

Tipare tare vs. slabă

- Clasificare după libertatea de a agrega valori de tipuri diferite.

Exemplul 16.1 (Tipare tare).

`1 + "23"` $\rightarrow$ Eroare (Haskell)

Exemplul 16.2 (Tipare slabă).

`1 + "23" = 24` (Visual Basic)

`1 + "23" = "123"` (JavaScript)

Tiparea în Scheme

- este dinamică

Exemplul 16.3.

```
1 (if #t 'something (+ 1 #t)) → 'something
2 (if #f 'something (+ 1 #t)) → Eroare
```

- este tare

Exemplul 16.4.

```
1 (+ "1" 2) → Eroare
```

- Permite liste cu elemente de tipuri diferite.

Variabile

Variabile

Proprietăți

- Proprietăţi
  - tip – nu! (în Scheme)
  - identificator
  - valoarea legată (la un anumit moment)
  - domeniul de vizibilitate + durata de viaţă
- Stări
  - declarată: cunoaştem identificatorul
  - definită: cunoaştem şi valoarea

Exemplul 17.1.

```
1 int f(int x) {
2     int y = 0; // definitie
3     // domeniul de vizibilitate a lui y
4 }
```

Legarea variabilelor

Definiția 17.2 (Legarea variabilelor).

Modalitatea de asociere a apariţiei unei variabile cu definiţia acesteia.

Definiția 17.3 (Domeniu de vizibilitate – *scope*).

Mulţimea punctelor din program unde o definiţie este vizibilă. Este determinat de modalitatea de legare a variabilelor.

- Modalităţi de legare:
  - statică
  - dinamică

Legarea statică a variabilelor

Definiția 17.4 (Legare statică / lexicală).

Variabilele din corpul unei expresii sunt extrase din contextul în care aceasta a fost definită.

- Domeniu de vizibilitate determinat prin construcţiile limbajului, putând fi desprins la compilare

Exemplul 17.5.

Care sunt domeniile de vizibilitate a variabilelor de legare, în expresia $\lambda x.\lambda y.\lambda x.xy$?

$\lambda \underline{x}.\lambda y.\lambda x.xy \mid \lambda x.\lambda \underline{y}.\lambda x.xy \mid \lambda x.\lambda y.\lambda \underline{x}.xy$

Legarea dinamică a variabilelor

Definiția 17.6 (Legare dinamică).

Valorile variabilelor depind de momentul în care o expresie este evaluată.

- Domeniu de vizibilitate determinat la execuţie

Exemplul 17.7 (! Artificial).

```
1 int x = 0;
2 int f() { return x; }
3 int g(int x) { return f(); }
```

Ce va returna `g(2)`?

Legarea dinamică a variabilelor

Definiția 17.6 (Legare dinamică).

Valorile variabilelor depind de momentul în care o expresie este evaluată.

- Domeniu de vizibilitate determinat la execuţie

Exemplul 17.7 (! Artificial).

```
1 int x = 0;
2 int f() { return x; }
3 int g(int x) { return f(); }
```

Ce va returna `g(2)`?

`x=0` →`g(2)` →`x=2` →`f()` →2 (ultima valoare!)

Legarea variabilelor în Scheme

- Variabile declarate sau definite în expresii → static
  - `lambda`
  - `let`
  - `let*`
  - `letrec`
- Variabile *top-level* → dinamic
  - `define`

Construcţia `lambda`

Definiţie & Exemplu

- Leagă static parametrii formali ai unei funcţii
- Sintaxă:

```
1 (lambda (p1 ... pk ... pn) expr)
```

- Domeniul de vizibilitate a parametrului `pk`: mulţimea punctelor din corpul funcţiei – `expr`, în care apariţiile lui `pk` sunt libere (v. Exemplul 17.5)

Exemplul 17.8.

```
1 (lambda (x) (x (lambda (y) y)))
```

Construcţia `lambda`

Semantică

- Aplicaţie:

```
1 ((lambda (p1 ... pn) expr)
2   a1 ... an)
```

- Evaluare aplicativă: se evaluează argumentele `ak`, în ordine aleatoare
- Se evaluează corpul funcţiei, `expr`, ţinând cont de legările `pk` $\leftarrow$ *valoare*(`ak`)
- Valoarea aplicaţiei este valoarea lui `expr`

Construcţia `let`

Definiţie & Exemplu

- Leagă static variabile locale
- Sintaxă:

```
1  (let ((v1 e1) ... (vk ek) ... (vn en))
2    expr)
```

- Domeniul de vizibilitate a variabilei `vk` = mulţimea punctelor din corp – `expr`, în care apariţiile lui `vk` sunt libere (v. Exemplul 17.5)

Exemplul 17.9.

```
1  (let ((x 1) (y 2))
2    (+ x 2))
```

Construcţia `let`

Semantică

```
(let ((v1 e1) ... (vn en))
  expr)
```

echivalent cu

```
((lambda (v1 ... vn) expr)
  e1 ... en)
```

Construcţia `let*`

Definiţie & Exemplu

- Leagă static variabile locale
- Sintaxă:

```
1 (let* ((v1 e1) ... (vk ek) ... (vn en))
2    expr)
```

- Scope pentru variabila `vk` = mulţimea punctelor din
  - restul legărilor (legări anterioare) şi
  - corp – `expr`

  în care apariţiile lui `vk` sunt libere

Exemplul 17.10.

```
1 (let* ((x 1) (y x))
2   (+ x 2))
```

Construcţia `let*`

Semantică

```
(let* ((v1 e1) ... (vn en))
  expr)
```

echivalent cu

```
(let ((v1 e1))
  ...
    (let ((vn en))
      expr) ... )
```

- Evaluarea expresiilor `ei` se face în ordine!

Construcţia `letrec`

Definiţie

- Leagă static variabile locale
- Sintaxă:

```
1 (letrec ((v1 e1) ... (vk ek) ... (vn en))
2       expr)
```

- Domeniul de vizibilitate a variabilei `vk` = mulţimea punctelor din întreaga construcţie, în care apariţiile lui `vk` sunt libere

Construcţia `letrec`

Exemplu

Exemplul 17.11.

```
1 (letrec ((factorial
2             (lambda (n)
3               (if (zero? n) 1
4                   (* n (factorial (- n 1)))))))
5   factorial)
```

Construcţia `define`

Definiție & Exemplu

- Leagă dinamic variabile *top-level* (de obicei)

Exemplul 17.12.

```
1 (define x 0)
2 (define f (lambda () x))
3 (f)
4 (define x 1)
5 (f)
```

Output:

- Avantaje:
  - definirea variabilelor *top-level* în orice ordine
  - definirea de funcţii mutual recursive
- Dezavantaj: coruperea transparenţei referenţiale

Construcţia `define`

Definiție & Exemplu

- Leagă dinamic variabile *top-level* (de obicei)

Exemplul 17.12.

```
1 (define x 0)
2 (define f (lambda () x))
3 (f)
4 (define x 1)
5 (f)
```

Output: `0 1`

- Avantaje:
  - definirea variabilelor *top-level* în orice ordine
  - definirea de funcţii mutual recursive
- Dezavantaj: coruperea transparenţei referenţiale

Construcţia `define`

Exemplu obscur

Exemplul 17.13.

```
1  (define factorial (lambda (n)
2      (if (zero? n) 1
3          (* n (factorial (- n 1))))))
4
5  (factorial 5)
6
7  (define g factorial)
8  (define factorial (lambda (x) x))
9
10 (g 5)
```

Output:

Construcţia `define`

Exemplu obscur

Exemplul 17.13.

```
1  (define factorial (lambda (n)
2      (if (zero? n) 1
3          (* n (factorial (- n 1))))))
4
5  (factorial 5)
6
7  (define g factorial)
8  (define factorial (lambda (x) x))
9
10 (g 5)
```

Output: 120 20

Legarea variabilelor în Scheme

Exemplu mixt

Exemplul 17.14 (Variantă a Exemplului 17.7).

```
1  (define x 0)
2  (define f (lambda () x))
3  (define x 1)
4
5  (define g
6    (lambda (x)
7      (f)))
8
9  (g 2)
```

Output:

Legarea variabilelor în Scheme

Exemplu mixt

Exemplul 17.14 (Variantă a Exemplului 17.7).

```
1  (define x 0)
2  (define f (lambda () x))
3  (define x 1)
4
5  (define g
6    (lambda (x)
7      (f)))
8
9  (g 2)
```

Output: `1`

Evaluare

Evaluarea în Scheme

- Evaluare aplicativă: evaluarea parametrilor înaintea aplicării funcţiei asupra acestora (în ordine aleatoare)
- Transferul parametrilor: *call by sharing* – variantă a *call by value*
- Funcţii stricte (i.e.cu evaluare aplicativă)
- Excepţii: `if`, `cond`, `and`, `or`, `quote`

Contexte computaţionale

Definiţie

Definiția 18.1 (Context computaţional).

Contextul computaţional al unui punct P, dintr-un program, la momentul t, este mulţimea variabilelor ale căror domenii de vizibilitate îl conţin pe P, la momentul t.

- Legare statică $\rightarrow$ mulţimea variabilelor care îl conţin pe P în domeniul lexical de vizibilitate
- Legare dinamică $\rightarrow$ mulţimea variabilelor definite cel mai recent, la momentul t, şi referite din P

Contexte computaţionale

Exemplu

Exemplul 18.2.

Ce variabile locale conţine contextul computaţional al punctului *P*?

```
1 (lambda (x y)
2   (lambda (z)
3     (let ((x (car y)))
4       ; ...)))
```

Contexte computaţionale

Exemplu

Exemplul 18.2.

Ce variabile locale conţine contextul computaţional al punctului *P*?

```
1 (lambda (x y)
2   (lambda (z)
3     (let ((x (car y)))
4       ; ...)))
```

Închideri funcţionale

Motivaţie

- λ_0: evaluarea $\rightarrow$ substituţie textuală

Exemplul 18.3.

$((\lambda f.\lambda g.\lambda x.(f\ (g\ x))\ f_1)\ g_1) \rightarrow (\lambda g.\lambda x.(g_1\ (g\ x))\ f_1)$
$\rightarrow \lambda x.(g_1\ (f_1\ x))$

- Ineficienţa practică a procesului de substituţie
- Alternativă: salvarea contextului unei funcţii, în momentul creării acesteia
- Legarea variabilelor libere în contextul salvat $\rightarrow$ închidere funcţională

Închideri funcţionale

Definiţie

Definiția 18.4 (Închidere funcţională).

Funcţie care îşi salvează contextul, pe care îl va folosi, în momentul aplicării, pentru evaluarea corpului.

· Notaţie: închiderea funcţiei f în contextul $C \rightarrow < f; C >$

Exemplul 18.5.

$< \lambda x.z; \{z \leftarrow 2\} >$

· Utilizate în cazul legării statice!

Închideri funcţionale

Exemplu

Exemplul 18.6.

```
1  (define comp
2    (lambda (f) (lambda (g) (lambda (x) (f (g x))))))
3  (define inc (lambda (x) (+ x 1)))
4  (define comp-inc (comp inc))
5
6  (define double (lambda (x) (* x 2)))
7  (define comp-inc-double (comp-inc double))
8  (comp-inc-double 5) ; 11
9
10 (define inc (lambda (x) x))
11 (comp-inc-double 5) ; tot 11
```

Închideri funcţionale

Explicaţia exemplului

- $comp\text{-}inc \equiv < \lambda g.\lambda x.(f\ (g\ x)); \{f \leftarrow \lambda x.(+\ x\ 10)\} >$
- $comp\text{-}inc\text{-}double \equiv$
  $< \lambda x.(f\ (g\ x)); \{f \leftarrow \lambda x.(+\ x\ 10), g \leftarrow \lambda x.(*\ x\ 2)\} >$
- Inutilitatea redefinirii lui `inc`: valoarea sa fusese deja salvată în context, în momentul aplicării

Controlul evaluării

- `quote` sau ’
  - funcţie nestrictă
  - întoarce parametrul neevaluat
- `eval`
  - funcţie strictă
  - forţează evaluarea parametrului şi întoarce valoarea acestuia

Exemplul 18.7.

```
1 (define sum '(2 + 3))
2 sum ; (2 + 3)
3 (eval (list (cadr sum) (car sum) (caddr sum))) ; 5
```

Efecte laterale

Construcţia `set!`

- Modifică valoarea unei variabile

Exemplul 19.1.

```
1 (define x 0)
2 (define f (lambda (p)
3     (set! x p)
4     x))
5 (f 3) ; 3
6 x ; 3
```

- Diferenţă la nivel de intenţie faţă de `let`-uri şi `define`, care urmăresc definirea de variabile noi şi nu modificarea celor existente!

Atribuiri

- Avantaje:
  - Modelarea obiectelor a căror stare variază în timp
  - Evitarea pasării explicite a fiecărei modificări de stare

- Dezavantaj: pierderea transparenței referențiale (v. Cursul 1)

Sfârșitul cursului 4

Ce am învățat

· Tipare dinamică vs. statică, tare vs. slabă, legare dinamică vs statică; Scheme: tipare dinamică, tare; domeniu al variabilelor, construcții speciale în Scheme: lambda, let, let*, letrec, define; controlul evaluării, set! și efecte laterale.