FUN-IMP

Funcții

Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UNIBUC
traian.serbanuta@fmi.unibuc.ro

2 decembrie 2014

Exemple

Adunare și incrementare ca funcții de ordinul I

C/C++/...

```
int addition(int x, int y) { return x+y; }
int increment(int x) { return x+1; }
```

OCaml

```
let addition = fun (x,y) -> x + y
let increment = fun x -> x + 1
```

Javascript

```
function addition(x,y) { return x + y; }
function increment(x) { return x + 1; }
```

Exemple

Stil funcțional

Javascript
```
addition = function (x) { return function(y) { return x+y; } }
increment = addition (1)
```

OCaml
```
let adddition = fun x -> fun y -> x+y
let increment = addition 1
```

C#
```
delegate int IntToInt (int y);
delegate IntToInt IntToIntToInt (int x);
class O {
    public static void Main() {
        IntToIntToInt addition = x => y => x + y;
        IntToInt increment = addition (1);
        System.Console.WriteLine(increment(3));
    }
}
```

Exemple

Capturarea variabilelor din mediu în C#

```
delegate int IntThunk();
class M {
    public static void Main() {
        IntThunk[] funcs = new IntThunk[11];
        for (int i = 0; i <= 10; i++) {
            funcs[i] = () => i;
        }
        for (int i = 0; i <= 10; i++) {
            System.Console.WriteLine(f());
        }
    }
}
```

Exemple

Capturarea variabilelor din mediu în C#

```
delegate int IntThunk();
class N {
    public static void Main() {
        IntThunk[] funcs = new IntThunk[11];
        int i;
        for (i = 0; i <= 10; i++) {
            funcs[i] = () => i;
        }
        for (i = 0; i <= 10; i++) {
            System.Console.WriteLine(funcs[i]());
        }
    }
}
```

Exemple

Capturarea variabilelor de mediu în Javascript

```
var bar = function (x) {
  return function() { var x = 5; return x; };
}
var f = bar(200);
f ()
```

Funcții + IMP = FUN-IMP

Exemple

```
fun (x:int) -> x+1

(fun (x:int) -> x+1) 7

fun (x:int) -> fun (y:int) -> x + y

(fun (x:int) -> fun (y:int) -> x + y) 1

fun (x:int->int) -> fun (y:int) -> x (x y)

(fun (x:int->int) -> fun (y:int) -> x (x y)) (fun (x:int) -> x+1)

((fun (x:int->int) -> fun (y:int) -> x (x y)) (fun (x:int) -> x+1)) 7
```

FUN-IMP

Sintaxă

Extindem sintaxa limbajului IMP cu variabile, funcții și aplicația funcțiilor

Variabile $x \in \mathbb{X}$, unde $\mathbb{X} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots\}$ disjunctă de mulțimea locațiilor de memorie $\mathbb{L}$

Expresii

$$e ::= \ldots \mid x \mid \mathtt{fun}\,(x : T) \to e \mid e\ \ e$$

Tipuri

$$T ::= \mathtt{int} \mid \mathtt{bool} \mid \mathtt{unit} \mid T \to T$$

$$T_{loc} ::= \mathtt{intref}$$

Sintaxă concretă

La fel ca în OCaml

- Aplicarea funcțiilor se grupează la stânga

```
(fun (x:int) -> fun (y:int) -> x + y) 3 5
  = ((fun (x:int) -> fun (y:int) -> x + y) 3) 5
```

- Tipul săgeată se grupează la dreapta

```
int -> int -> int  = int -> (int -> int)
```

- `fun` se extinde cât de mult posibil la dreapta

```
fun (x:unit) -> x ; x   = fun (x:unit) -> (x ; x)
```

Domeniul de vizibilitate al variabilelor

C#

```
class P {
    public static void Main() {
        int i = 7 ;
        System.Console.WriteLine(i);
        {
          System.Console.WriteLine(i);
          int i = 13 ;
          System.Console.WriteLine(i);
        }
        System.Console.WriteLine(i);
    }
}
```

Domeniul de vizibilitate al variabilelor

C++

```
#include <iostream>
using namespace std;
int main() {
  int i = 7;
  cout << i << endl;
  {
     cout << i << endl;
     int i = 13;
     cout << i << endl;
  }
  cout <<i << endl;
}
```

Legare

$\texttt{fun}\ (x : T) \to e$

- x se numește parametru formal / variabilă de legare
- e se numește corpul funcției / domeniul legării lui x
- Orice apariție a lui x în e (care nu este parametru formal al altei funcții și nu este în corpul unei funcții $\texttt{fun}\ (x : T') \to e'$ cu același parametru formal x)
  - se referă la parametrul formal x al funcției definită mai sus
  - și se numește legată de acesta.
- Vizibilitatea variabilei de legare este limitată strict la definiția funcției
  - În afara definiției funcției, nu contează numele variabilei de legare
  - Matematic vorbind, $f(x) = x + 1$ e același lucru cu $f(y) = y + 1$.

α-echivalență

Definiție intuitivă

- Variabila de legare x din definiția $\texttt{fun}\ (x : T) \to e$ poate fi redenumită împreună cu toate aparițiile legate de ea (din e).
- Procesul de redenumire a argumentului formal al unei definiții de funcție se numește α-conversie.
- α-conversia determină o relație de echivalență numită α-echivalență.

$$\texttt{fun}\ (x : \texttt{int}).x + y \equiv_\alpha \texttt{fun}\ (z : \texttt{int}).z + y \not\equiv_\alpha \texttt{fun}\ (y : \texttt{int}).y + y$$

Apariție liberă a unei variabile

Apariție liberă/legată

O apariție a variabilei x în expresia e este liberă dacă nu este în corpul nici unei definiții de funcție $fun(x : T) \rightarrow \ldots$.
Orice altă apariție a lui x este legată de cea mai apropiată declarație $fun(x : T) \rightarrow \ldots$ care o conține.

Exemple

- 17
- $x + y$
- $\texttt{fun}\ (x : \texttt{int}) \rightarrow x + 2$
- $\texttt{fun}\ (x : \texttt{int}) \rightarrow x + z$
- $\texttt{if}\ y\ \texttt{then}\ 2 + x\ \texttt{else}\ (\texttt{fun}\ (x : \texttt{int}) \rightarrow x + 2)\ z$
- $\texttt{fun}\ (x : \texttt{unit}) \rightarrow \texttt{fun}\ (x : \texttt{int}) \rightarrow x$

Variabile libere

Definiție formală

Funcția *var* care dă mulțimea variabilelor care apar libere într-o expresie e definită recursiv pe structura expresiilor:

- $var(x) = \{x\}$
- $var(\mathtt{fun}\ (x : T) \to e) = var(e) \setminus \{x\}$
- $var(e_1\ \ e_2) = var(e_1) \cup var(e_2)$
- $var(n) = var(b) = var(l) = var(\mathtt{skip}) = var(!\ l) = \emptyset$
- $var(e_1\ o\ e_2) = var(e_1) \cup var(e_2)$, $o \in \{+, \leq\}$
- $var(\mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2) = var(e_b) \cup var(e_1) \cup var(e_2)$
- $var(l := e) = var(e)$
- $var(e_1\ ;\ e_2) = var(e_1) \cup var(e_2)$
- $var(\mathtt{while}\ e_b\ \mathtt{do}\ e\ \mathtt{done}) = var(e_b) \cup var(e)$

Evaluarea funcțiilor ca substituție

- Dată fiind $f : \mathbb{Z} \longrightarrow \mathbb{Z}$, $f(x) = x * x$, simplificăm $f(y + 5)$ înlocuind în definiția funcției f variabila x cu valoarea dată $y + 5$

$$f(y + 5) = (y + 5) * (y + 5)$$

- Folosind ideea de substituție, $f(y + 5) = (x * x)[(y + 5)/x]$
- Unde $e[e'/x]$ se definește intuitiv ca fiind expresia e în care aparițiile libere ale lui x se înlocuiesc cu e'.

Substituție

Exemple

Definiția intuitivă a substituției $e\ [e'/x]$

Se obține prin înlocuirea cu e' a tuturor aparițiilor libere ale lui x în e

- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [4/y]$

Substituție

Exemple

Definiția intuitivă a substituției $e \; [e'/x]$

Se obține prin înlocuirea cu e' a tuturor aparițiilor libere ale lui x în e

- $(\texttt{fun}\ (x : \texttt{int}) \rightarrow x + y)\ \ [4/y] = \texttt{fun}\ (x : \texttt{int}) \rightarrow x + 4$
- $(\texttt{fun}\ (x : \texttt{int}) \rightarrow x + y)\ \ [4/x]$

Substituție

Exemple

Definiția intuitivă a substituției $e\ [e'/x]$

Se obține prin înlocuirea cu e' a tuturor aparițiilor libere ale lui x în e

- $(\texttt{fun}\ (x : \texttt{int}) \to x + y)\ [4/y] = \texttt{fun}\ (x : \texttt{int}) \to x + 4$
- $(\texttt{fun}\ (x : \texttt{int}) \to x + y)\ [4/x] = \texttt{fun}\ (x : \texttt{int}) \to x + y$
  - Deoarece x nu apare liber
- $(\texttt{fun}\ (x : \texttt{int}) \to x + y)\ [x/y]$

Substituție

Exemple

Definiția intuitivă a substituției $e\ [e'/x]$

Se obține prin înlocuirea cu e' a tuturor aparițiilor libere ale lui x în e

- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [4/y] = \mathtt{fun}\ (x : \mathtt{int}) \to x + 4$
- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [4/x] = \mathtt{fun}\ (x : \mathtt{int}) \to x + y$
  - Deoarece x nu apare liber
- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [x/y] \neq \mathtt{fun}\ (x : \mathtt{int}) \to x + x$
  - Deoarece variabila x ar fi capturată (incorect) de operatorul de legare.

Substituție

Exemple

Definiția intuitivă a substituției $e\ [e'/x]$

Se obține prin înlocuirea cu e' a tuturor aparițiilor libere ale lui x în e

- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [4/y] = \mathtt{fun}\ (x : \mathtt{int}) \to x + 4$
- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [4/x] = \mathtt{fun}\ (x : \mathtt{int}) \to x + y$
    - Deoarece x nu apare liber
- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [x/y] \neq \mathtt{fun}\ (x : \mathtt{int}) \to x + x$
    - Deoarece variabila x ar fi capturată (incorect) de operatorul de legare.
- $(\mathtt{fun}\ (x : \mathtt{int}) \to x + y)\ [x/y] \equiv_\alpha (\mathtt{fun}\ (z : \mathtt{int}) \to z + y)\ [x/y] = \mathtt{fun}\ (z : \mathtt{int}) \to z + x$

Substituție simultană

Definiție

O substituție este o funcție parțială de domeniu finit $\sigma : \mathbb{X} \overset{\circ}{\to} \mathbb{E}$ care asociază expresii variabilelor.

Notații

- $\sigma = [e_1/x_1, e_2/x_2, \ldots, e_n/x_n]$
  se citește e_1 înlocuiește pe x_1, ș.a.m.d
- $Dom\,\sigma = \{x_1, x_2, \ldots, x_n\}$
- $Im\,\sigma = \{e_1, e_2, \ldots, e_n\}$
- $e\ \ \sigma$ — expresia obținută în urma aplicării substituției σ expresiei e

Definiție formală (inductivă)

$$x\ \ \sigma = \sigma(x) \quad \textit{dacă } x \in \mathit{Dom}\,\sigma$$

$$x\ \ \sigma = x \quad \textit{dacă } x \notin \mathit{Dom}\,\sigma$$

$$n\ \ \sigma = n$$ și la fel pentru b, $!l$, și `skip`

$$\frac{e_1\ \ \sigma = e_1' \quad e_2\ \ \sigma = e_2'}{(e_1\ \ e_2)\ \ \sigma = e_1'\ \ e_2'}$$

La fel și pentru ceilalți constructori de limbaj, mai puțin `fun`

$$\frac{e\ \ \sigma = e'}{(\mathtt{fun}\ (x : T) \to e)\ \ \sigma = \mathtt{fun}\ (x : T) \to e'} \quad \textit{dacă } x \notin \mathit{Dom}\,\sigma \cup \mathit{var}(\mathit{Im}\,\sigma)$$

Definiție formală (inductivă)

$$x\ \ \sigma = \sigma(x) \quad \textit{dacă } x \in \textit{Dom}\,\sigma$$
$$x\ \ \sigma = x \quad \textit{dacă } x \notin \textit{Dom}\,\sigma$$

$$n\ \ \sigma = n$$ și la fel pentru b, $!l$, și `skip`

$$\frac{e_1\ \ \sigma = e_1' \quad e_2\ \ \sigma = e_2'}{(e_1\ \ e_2)\ \ \sigma = e_1'\ \ e_2'}$$

La fel și pentru ceilalți constructori de limbaj, mai puțin `fun`

$$\frac{e\ \ \sigma = e'}{(\mathtt{fun}\ (x : T) \to e)\ \ \sigma = \mathtt{fun}\ (x : T) \to e'} \quad \textit{dacă } x \notin \textit{Dom}\,\sigma \cup \textit{var}(\textit{Im}\,\sigma)$$

Ce facem dacă $x \in \textit{Dom}\,\sigma \cup \textit{var}(\textit{Im}\,\sigma)$?

Substituție

Evitarea capturării variabilelor

$$\frac{e\ \ \sigma' = e'}{(\texttt{fun}\ (x : T) \to e)\ \ \sigma = \texttt{fun}\ (x' : T) \to e'} \quad \textit{dacă}\ \begin{array}{l} x \in Dom\,\sigma \cup var(Im\,\sigma) \\ x' \notin var(e) \cup var(Im\,\sigma) \\ \sigma' = [x'/x]\ \ \sigma \end{array}$$

Dacă x apare liber în $Im\,\sigma$

- Alegem o variabilă nouă x' (care nu apare liberă în e sau $Im\,\sigma$)
- e' se obține din e aplicând substituția obținută din σ prin (re)definirea lui x ca x'

$$\sigma'(y) = \begin{cases} \sigma(x), \text{ dacă } y \neq x \\ x', \text{ dacă } y = x \end{cases}$$

- Putem să-l redefinim pe x în σ' pentru că toate aparițiile libere ale lui x în e sunt legate de parametrul formal al funcției.

Substituție

Exemplu

$$(\mathtt{fun}\ (y : \mathtt{int}) \to (\mathtt{fun}\ (y : \mathtt{int}) \to x + y)\ y)\ \ [y + 2/x]$$

Substituție

Exemplu

$$
\begin{aligned}
& (\texttt{fun}\ (y : \texttt{int}) \to (\texttt{fun}\ (y : \texttt{int}) \to x + y)\ y)\ \ [y + 2/x] \\
= \ & \texttt{fun}\ (y' : \texttt{int}) \to (((\texttt{fun}\ (y : \texttt{int}) \to x + y)\ y)\ \ [y'/y, y + 2/x]) \\
= \ & \texttt{fun}\ (y' : \texttt{int}) \to \\
& \qquad ((\texttt{fun}\ (y : \texttt{int}) \to x + y)\ \ [y'/y, y + 2/x])\ (y\ \ [y'/y, y + 2/x]) \\
= \ & \texttt{fun}\ (y' : \texttt{int}) \to (\texttt{fun}\ (y'' : \texttt{int}) \to ((x + y)\ \ [y''/y, y + 2/x]))\ y' \\
= \ & \texttt{fun}\ (y' : \texttt{int}) \to \\
& \quad (\texttt{fun}\ (y'' : \texttt{int}) \to (x\ \ [y''/y, y + 2/x]) + (y\ \ [y''/y, y + 2/x]))\ y' \\
= \ & \texttt{fun}\ (y' : \texttt{int}) \to (\texttt{fun}\ (y'' : \texttt{int}) \to ((y + 2) + y'')\ y'
\end{aligned}
$$

α-echivalență

Definiție inductivă

$$(\alpha R)\quad e \equiv_\alpha e \qquad\qquad (\alpha S)\quad \frac{e \equiv_\alpha e'}{e' \equiv_\alpha e} \qquad\qquad (\alpha T)\quad \frac{e \equiv_\alpha e' \quad e' \equiv_\alpha e''}{e \equiv_\alpha e''}$$

$$(\alpha @)\quad \frac{e_1 \equiv_\alpha e_1' \quad e_2 \equiv_\alpha e_2'}{e_1\ \ e_2 \equiv_\alpha e_1'\ \ e_2'} \qquad\qquad (\alpha :=)\quad \frac{e \equiv_\alpha e'}{l := e \equiv_\alpha l := e'}$$

La fel și pentru ceilalți constructori de limbaj, mai puțin `fun`

$$(\alpha\textsc{fun})\quad \frac{e \equiv_\alpha e'}{\texttt{fun}\ (x : T) \to e \equiv_\alpha \texttt{fun}\ (x : T) \to e'}$$

$$(\alpha\textsc{conv})\quad \texttt{fun}\ (x : T) \to e \equiv_\alpha \texttt{fun}\ (y : T) \to (e\ \ [y/x]) \quad \textit{dacă } y \notin var(e)$$

Proprietăți de compatibilitate

Compatibilitate cu substituția

$$e_1 \equiv_\alpha e_1' \text{ și } e_2 \equiv_\alpha e_2' \implies e_1\ [e_2/x] \equiv_\alpha e_1'\ [e_2'/x]$$

Substituția e funcțională modulo α-echivalență

$$e\ \ \sigma = e_1 \text{ și } e\ \ \sigma = e_2 \implies e_1 \equiv_\alpha e_2$$

Compatibilitate cu *var*

$$e \equiv_\alpha e' \implies var(e) = var(e')$$

Termeni FUN-IMP

Definiție

Un termen FUN-IMP este o clasă de echivalență modulo $\equiv_\alpha$.

- Identificăm termenii obținuți prin redenumirea variabilelor legate
- Simplificare: notăm clasa de echivalență prin oricare din reprezentanți.

Evaluarea funcțiilor

Fie expresia e dată de

```
(fun (x : unit) -> l := 5 ; x ; x) (l := !l + 1)
```

Care este starea finală corespunzătoare stării inițiale $\langle e, \{l \mapsto 0\}\rangle$?

$$\langle e, \{l \mapsto 0\}\rangle \longrightarrow^* \langle \mathtt{skip}, \{l \mapsto ???\}\rangle$$

Evaluare strictă (Cam toate limbajele, inclusiv OCaml)

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\mathtt{fun}\ (x : T) \to e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\mathtt{fun}\ (x : T) \to e) \;\; v$ la $e \;\; [v/x]$

Evaluare strictă (Cam toate limbajele, inclusiv OCaml)

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\mathtt{fun}\ (x : T) \to e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\mathtt{fun}\ (x : T) \to e) \;\; v$ la $e \;\; [v/x]$

$$
\begin{array}{ll}
 & \langle (fun(x : unit) \to l := 5; x; x)(l := !l + 1), \{l \mapsto 0\}\rangle \\
\to & \langle (fun(x : unit) \to l := 5; x; x)(l := 0 + 1), \{l \mapsto 0\}\rangle \\
\to & \langle (fun(x : unit) \to l := 5; x; x)(l := 1), \{l \mapsto 0\}\rangle \\
\to & \langle (fun(x : unit) \to l := 5; x; x)\ \mathtt{skip}, \{l \mapsto 1\}\rangle \\
\to & \langle l := 5; \mathtt{skip}; \mathtt{skip}, \{l \mapsto 1\}\rangle \\
\to & \langle \mathtt{skip}; \mathtt{skip}; \mathtt{skip}, \{l \mapsto 5\}\rangle \\
\to & \langle \mathtt{skip}; \mathtt{skip}, \{l \mapsto 5\}\rangle \\
\to & \langle \mathtt{skip}, \{l \mapsto 5\}\rangle
\end{array}
$$

Evaluare non-strictă (Limbaje pur funcționale)

Pentru a reduce $e_1 \; e_2$:

- Reducem e_1 până la o funcție $\texttt{fun}\ (x : T) \to e$
- Apoi reducem $(\texttt{fun}\ (x : T) \to e)\ \ e_2$ la $e\ \ [e_2/x]$

Evaluare non-strictă (Limbaje pur funcționale)

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\texttt{fun}\ (x : T) \to e$
- Apoi reducem $(\texttt{fun}\ (x : T) \to e)\ \ e_2$ la $e\ \ [e_2/x]$

$$
\begin{array}{ll}
 & \langle (fun(x : unit) \to l := 5; x; x)(l :=!l + 1), \{l \mapsto 0\}\rangle \\
\to & \langle l := 5; l :=!l + 1; l :=!l + 1, \{l \mapsto 0\}\rangle \\
\to & \langle \texttt{skip}; l :=!l + 1; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle l :=!l + 1; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle l := 5 + 1; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle l := 6; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle \texttt{skip}; l :=!l + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l :=!l + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l := 6 + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l := 7, \{l \mapsto 6\}\rangle \\
\to & \langle \texttt{skip}, \{l \mapsto 7\}\rangle
\end{array}
$$

Evaluare leneșă

Implementări ale limbajelor pure gen Haskell

Pentru a reduce $e_1 \ \ e_2$:

- Reduc e_1 până la o funcție $\texttt{fun}\ (x : T) \to e$
- Apoi reduc corpul funcției e până la un e' care are nevoie de x
- Apoi reduc e_2 până la o valoare v
- Apo reduc $(\texttt{fun}\ (x : T) \to e')\ \ v$ la $e'\ \ [v/x]$

Evaluare leneșă

Implementări ale limbajelor pure gen Haskell

Pentru a reduce $e_1 \; e_2$:

- Reduc e_1 până la o funcție $\texttt{fun}\ (x : T) \to e$
- Apoi reduc corpul funcției e până la un e' care are nevoie de x
- Apoi reduc e_2 până la o valoare v
- Apo reduc $(\texttt{fun}\ (x : T) \to e')\ v$ la $e'\ [v/x]$

$$
\begin{array}{rl}
 & \langle (fun(x : unit) \to l := 5; x; x)(l :=!l + 1), \{l \mapsto 0\}\rangle \\
\to & \langle (fun(x : unit) \to \texttt{skip}; x; x)(l :=!l + 1), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x : unit) \to x; x)(l :=!l + 1), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x : unit) \to x; x)(l := 5 + 1), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x : unit) \to x; x)(l := 6), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x : unit) \to x; x)\ \texttt{skip}, \{l \mapsto 6\}\rangle \\
\to & \langle \texttt{skip}; \texttt{skip}, \{l \mapsto 6\}\rangle \\
\to & \langle \texttt{skip}, \{l \mapsto 6\}\rangle
\end{array}
$$

Evaluare nerestricționată

Pentru a reduce $e_1 \; e_2$

- Reducem fie e_1 fie e_2
- Putem reduce corpurile funcțiilor
- Oricând avem $(\mathtt{fun}\ (x : T) \rightarrow e')e''$, o putem (sau nu) reduce la $e' \; [e''/x]$

Evaluare „normală“

Pentru a reduce $e_1 \; e_2$

- Reducem mereu cel mai din stânga redex din cele de mai sus
- Reducem $(\texttt{fun}\ (x : T) \to e')e''$ la $e' \; [e''/x]$
- Reducem e_1 (putem reduce și corpurile funcțiilor)
- Dacă $e_1 \not\rightarrow$, reducem e_2

Evaluare „normală“

Pentru a reduce $e_1 \;\; e_2$

- Reducem mereu cel mai din stânga redex din cele de mai sus
- Reducem $(\mathtt{fun}\ (x : T) \to e')e''$ la $e' \;\; [e''/x]$
- Reducem e_1 (putem reduce și corpurile funcțiilor)
- Dacă $e_1 \not\rightarrow$, reducem e_2

$$
\begin{array}{rl}
 & \langle (fun(x:unit) \to l := 5; x; x)(l :=!l + 1), \{l \mapsto 0\}\rangle \\
\to & \langle (fun(x:unit) \to \mathtt{skip}; x; x)(l :=!l + 1), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x:unit) \to \mathtt{skip}; x; x)(l :=!l + 1), \{l \mapsto 5\}\rangle \\
\to & \langle (fun(x:unit) \to x; x)(l :=!l + 1), \{l \mapsto 5\}\rangle \\
\to & \langle l :=!l + 1; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle l := 5 + 1; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle l := 6; l :=!l + 1, \{l \mapsto 5\}\rangle \\
\to & \langle \mathtt{skip}; l :=!l + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l :=!l + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l := 6 + 1, \{l \mapsto 6\}\rangle \\
\to & \langle l := 7, \{l \mapsto 6\}\rangle
\end{array}
$$

Evaluare strictă

Pentru a reduce $e_1 \ \ e_2$:

- Reducem e_1 până la o funcție $\texttt{fun}\ (x : T) \to e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\texttt{fun}\ (x : T) \to e)\ \ v$ la $e\ \ [v/x]$

Reguli

(S@S) $$\frac{\langle e_1, s\rangle \to \langle e_1', s'\rangle}{\langle e_1\ \ e_2, s\rangle \to \langle e_1'\ \ e_2, s'\rangle}$$

(S@D) $$\frac{\langle e_2, s\rangle \to \langle e_2', s'\rangle}{\langle(\texttt{fun}\ (x : T) \to e_1)\ \ e_2, s\rangle \to \langle(\texttt{fun}\ (x : T) \to e_1)\ \ e_2', s'\rangle}$$

(S@) $$\langle(\texttt{fun}\ (x : T) \to e_1)\ \ v_2, s\rangle \to \langle e, s\rangle \quad \textit{dacă}\ e = e_1\ \ [v_2/x]$$

Evaluare non-strictă

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\mathtt{fun}\ (x : T) \rightarrow e$
- Apoi reducem $(\mathtt{fun}\ (x : T) \rightarrow e)\;\; e_2$ la $e\;\;[e_2/x]$

Reguli

$$\text{(NS@S)} \quad \frac{\langle e_1, s\rangle \rightarrow \langle e_1', s'\rangle}{\langle e_1 \;\; e_2, s\rangle \rightarrow \langle e_1' \;\; e_2, s'\rangle}$$

$$\text{(NS@)} \quad \langle(\mathtt{fun}\ (x : T) \rightarrow e_1)\;\; e_2, s\rangle \rightarrow \langle e, s\rangle \quad \textit{dacă}\ e = e_1\;\;[e_2/x]$$

Evaluare nerestricționată

Pentru a reduce $e_1 \ \ e_2$

- Reducem fie e_1 fie e_2
- Putem reduce corpurile funcțiilor
- Oricând avem $(\texttt{fun}\ (x : T) \rightarrow e')e''$, o putem (sau nu) reduce la $e' \ \ [e''/x]$

Reguli

(NR@S) $$\frac{\langle e_1, s\rangle \rightarrow \langle e_1', s'\rangle}{\langle e_1 \ \ e_2, s\rangle \rightarrow \langle e_1' \ \ e_2, s'\rangle}$$

(NR@D) $$\frac{\langle e_2, s\rangle \rightarrow \langle e_2', s'\rangle}{\langle e_1 \ \ e_2, s\rangle \rightarrow \langle e_1 \ \ e_2', s'\rangle}$$

(NRFunD) $$\frac{\langle e, s\rangle \rightarrow \langle e', s'\rangle}{\langle \texttt{fun}\ (x : T) \rightarrow e, s\rangle \rightarrow \langle \texttt{fun}\ (x : T) \rightarrow e', s\rangle}$$

(NR@) $\langle (\texttt{fun}\ (x : T) \rightarrow e_1) \ \ e_2, s\rangle \rightarrow \langle e, s\rangle$ $\quad$ *dacă* $e = e_1 \ \ [e_2/x]$

Evaluare „normală“

Pentru a reduce $e_1 \;\; e_2$

- Reducem mereu cel mai din stânga redex din cele de mai sus
- Reducem $(\mathtt{fun}\ (x : T) \rightarrow e')e''$ la $e' \;\; [e''/x]$
- Reducem e_1 (putem reduce și corpurile funcțiilor)
- Dacă $e_1 \nrightarrow$, reducem e_2

Reguli

(Nor@) $\langle(\mathtt{fun}\ (x : T) \rightarrow e_1)\ \ e_2, s\rangle \rightarrow \langle e, s\rangle \quad \textit{dacă } e = e_1 \;\; [e_2/x]$

(Nor@S) $$\frac{\langle e_1, s\rangle \rightarrow \langle e_1', s'\rangle}{\langle e_1 \;\; e_2, s\rangle \rightarrow \langle e_1' \;\; e_2, s'\rangle}$$ *dacă* e_1 nu e încă funcție

(NorFunD) $$\frac{\langle e, s\rangle \rightarrow \langle e', s'\rangle}{\langle \mathtt{fun}\ (x : T) \rightarrow e, s\rangle \rightarrow \langle \mathtt{fun}\ (x : T) \rightarrow e', s\rangle}$$

(Nor@D) $$\frac{\langle e_1, s\rangle \nrightarrow \qquad \langle e_2, s\rangle \rightarrow \langle e_2', s'\rangle}{\langle e_1 \;\; e_2, s\rangle \rightarrow \langle e_1 \;\; e_2', s'\rangle}$$

Evaluare leneșă

Implementări ale limbajelor pure gen Haskell

Pentru a reduce $e_1 \;\; e_2$:

- Reduc e_1 până la o funcție $\mathtt{fun}\ (x : T) \to e$
- Apoi reduc corpul funcției e până la un e' care are nevoie de x
- Apoi reduc e_2 până la o valoare v
- Apo reduc $(\mathtt{fun}\ (x : T) \to e') \;\; v$ la $e' \;\; [v/x]$

Reguli?

E mai complicat decât pare, deoarece trebuie să ne dăm seama că e' are nevoie de x.

Lenevire în Maude

Lista numerelor naturale

```
fmod LISTA-LENESA is including NAT .
  sort ListaLenesa .
  op nil : -> ListaLenesa .
  op __ : Nat ListaLenesa -> ListaLenesa [strat (1)] .

  var M N : Nat . var L L' : ListaLenesa .

  op _.. infinit : Nat -> ListaLenesa .
  eq N .. infinit = N s(N) .. infinit .

  op primele_din_ : Nat ListaLenesa -> ListaLenesa .
  ceq primele s(N) din M L = M L'
   if L' := primele N din L .
  eq primele 0 din L = nil .
endfm
red 2 .. infinit .                ***> result ListaLenesa : 2 3 .. infinit
red primele 4 din 2 .. infinit .  ***> result ListaLenesa: 2 3 4 5 nil
```

Lenevire în Maude

Ciurul lui Eratostene

```
fmod ERATOSTENE is including LISTA-LENESA .
  var M N : Nat .   var L L' : ListaLenesa .

  op cerne : ListaLenesa -> ListaLenesa .
  eq cerne(N L) = N cerne(elimina multiplii lui N din L) .

  op elimina multiplii lui_din_ : Nat ListaLenesa -> ListaLenesa .
  eq elimina multiplii lui N din M L
   = if N divides M
     then elimina multiplii lui N din L
     else M elimina multiplii lui N din L
     fi .
endfm
red elimina multiplii lui 2 din 2 .. infinit .
***> result ListaLenesa : 3 elimina multiplii lui 2 din 4 .. infinit
red primele 10 din cerne(2 .. infinit ) .
***> result ListaLenesa: 2 3 5 7 11 13 17 19 23 29 nil
```

Contexte de evaluare

- Găsirea redex-ului se aseamănă cu un algoritm de analiză sintactică
- Putem înlocui regulile structurale cu reguli gramaticale

Sintaxă: $e ::= n \mid l \mid e \; op \; e$

Reguli structurale

$$\frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma\rangle}{\langle e_1 \; op \; e_2, \sigma\rangle \longrightarrow \langle e_1' \; op \; e_2, \sigma\rangle}$$

$$\frac{\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma\rangle}{\langle n_1 \; op \; e_2, \sigma\rangle \longrightarrow \langle n_1 \; op \; e_2', \sigma\rangle}$$

Contexte de evaluare

$$\begin{array}{lll} c & ::= & \blacksquare \\ & \mid & c \; op \; e \\ & \mid & n \; op \; c \end{array}$$

Instanțierea unui context c cu expresia e

$$c[e] = c[e/\blacksquare]$$

Contexte de evaluare

Exemple

Sintaxă: $e ::= n \mid l \mid e\ op\ e$

Contexte: $c ::= \blacksquare \mid c\ op\ e \mid n\ op\ c$

Exemple de contexte

Corecte	Greșite
$\blacksquare$	5
$3 * \blacksquare$	$x + \blacksquare$
$9 + 3 * (\blacksquare + 7)$	$3 * 3 + 3 * (\blacksquare + 7)$

Exemple de contexte instanțiate

- $\blacksquare[x + 1] = x + 1$
- $(9 + 3 * (\blacksquare + 7))[x] = 9 + 3 * (x + 7)$
- $(9 + 3 * (\blacksquare + 7))[5] = 9 + 3 * (5 + 7)$

Semantica Operațională Contextuală

[Felleisen, 1992]

Un pas de execuție folosind contexte de evaluare

- Descompune expresia în contextul c și redex-ul r
- Aplică o regulă operațională asupra lui r obținând e
- Pune e în contextul inițial, obținând $c[e]$

$$\frac{\langle r, \sigma \rangle \longrightarrow \langle e, \sigma \rangle}{\langle c[r], \sigma \rangle \longrightarrow \langle c[e], \sigma \rangle}$$

Semantica: definiții de contexte și reguli de reducție

Contexte: $c ::= \blacksquare \mid c \; op \; e \mid n \; op \; c$ Sintaxă: $e ::= n \mid l \mid e \; op \; e$

Reguli: $\langle l, \sigma \rangle \longrightarrow \langle n, \sigma \rangle$ *dacă* $n = \sigma(l)$

$\langle n_1 \; op \; n_2, \sigma \rangle \longrightarrow \langle n, \sigma \rangle$ *dacă* $n = n_1 \; op_{int} \; n_2$

Evaluare leneșă folosind Semantica Contextuală

Idee de bază

Contexte de evaluare pentru aplicație

$$
\begin{array}{rcl}
c & ::= & \blacksquare \\
& \mid & \ldots \\
& \mid & c\ e \\
& \mid & (\mathtt{fun}\ (x : T) \to c)\ e \\
& \mid & (\mathtt{fun}\ (x : T) \to c[x])\ c
\end{array}
$$

Regulă de evaluare pentru aplicație

$$\langle(\mathtt{fun}\ (x : T) \to c[x])\ v, s\rangle \to \langle(\mathtt{fun}\ (x : T) \to c[v])\ v, s\rangle$$