Em busca de uma interoperabilidade de linguagens de programação

Fabricio Chalub

3 de setembro de 2003

Interoperabilidade

fat.lisp:

```
(defun fat (x) (if (> x 1) (* x (fat (- x 1))) 1))
```

Bar.java:

```
class Bar { void foo (int x) { return fat(x); } }
```

```
> Bar bar = new Bar();
> System.out.println(bar.foo(10));
3628800
```

Porque interoperabilidade?

- sistemas legados
- diversidade ideológica
- diversidade de paradigmas
- diferentes áreas de atuação (FORTRAN, COBOL, LISP, Pascal, C, Java)
- linguagens específicas de domínio bem difundidas: Matlab, Maple

Interoperabilidade: rápido survey

Interoperabilidade: alien objects (Common Lisp)

```
typedef struct { int quot; int rem; } div_t;
extern div_t div (int numer, int denom);
```

```
(def-c-struct div_t
  (quot int)
  (rem int))
(default-foreign-language :stdc)
(def-call-out div (:arguments (numer int) (denom int))
  (:return-type div_t))
```

```
> (div 20 3)
#S(DIV_T :QUOT 6 :REM 2)
```

Interoperabilidade: compiling ML to Java bytecodes

```
let type colour = "java.awt.Color"
    val grey = valOf (_getField colour "gray")
in Java.toInt (_invoke "getRed" (grey))
end
```

ML $\rightarrow$ Java $\rightarrow$ bytecodes

Interoperabilidade: blending SML with Java

```
val labels = map javax.swing.JLabel ["A", "B"]
val c = (JButton "My button") :> Component
```

Interoperabilidade como conseqüência da semântica de linguagens de programação

Semântica Definitiva (Definitive Semantics)

Definir de uma vez por todas semântica de linguagens de programação

Semântica Definitiva: hardware

- sintaxe abstrata
- semântica denotacional
- semântica estrutural operacional modular
- semântica de ações
- (+) rewriting logic

Semântica Definitiva: sintaxe abstrata

Linguagens de programação contêm construções com dinâmica semelhante, mas diferentes sintaticamente.

Java	E_1 ? E_2 : E_3	} cond(E_1, E_2, E_3)
S. ML	if E_1 then E_2 else E_3	
C. Lisp	(if E_1 E_2 E_3)	

Java	E_1 && E_3	} cond-conj(E_1, E_2)
S. ML	E_1 andalso E_2	
C. Lisp	(and E_1 E_2)	

Semântica Definitiva: semântica operacional estrutural modular

Generalized labelled transition system (estados, rótulos, transições)

$$\frac{E_1 \xrightarrow{x} E_1'}{\mathrm{cond}(E_1, E_2, E_3) \xrightarrow{x} \mathrm{cond}(E_1', E_2, E_3)}$$

$$\mathrm{cond}(\mathbf{t}, E_2, E_3) \xrightarrow{u} E_2$$

$$\mathrm{cond}(\mathbf{f}, E_2, E_3) \xrightarrow{u} E_3$$

rótulos modelam o processamento de informação

Semântica Definitiva: semântica denotacional

- denotações de frases são funções matemáticas que modelam o comportamento computacional

Exemplo: modelo de expressões com ambiente de variáveis.

$\mathcal{E}[\![\,\mathrm{E}\,]\!]$ é uma função que mapeia ambientes (ρ) em valores escrita da forma $\lambda\rho.z$.

$$\mathcal{E}[\![\,\mathrm{cond}(\mathrm{E}_1,\mathrm{E}_2,\mathrm{E}_3)\,]\!] = \lambda\rho.(\mathcal{E}[\![\,\mathrm{E}_1\,]\!]\rho \rightarrow \mathcal{E}[\![\,\mathrm{E}_2\,]\!]\rho, \mathcal{E}[\![\,\mathrm{E}_3\,]\!]\rho)$$

$$\mathrm{t} \rightarrow \nu_1, \nu_2 \qquad \mathcal{E}[\![\,\mathrm{I}\,]\!]$$

Semântica Definitiva: semântica de ações

- semântica operacional + semântica denotacional
- ações modelam o mapeamento de ASTs para entidades semânticas (denotational semantics)
- ações são definidas usando MSOS.
- geração de compiladores: (subset de) Ada $\rightarrow$ abstract RISC machine $\rightarrow$ SPARC (J. Palsberg. A provably correct compiler generator. PhD thesis, Univ. of Aarthus, 1992)

Semântica Definitiva: semântica de ações

evaluate cond(E_1, E_2, E_3) =

evaluate E_1 **then**
(**maybe check the** bool **then evaluate** E_2) **else**
evaluate E_3

Semântica Definitiva: semântica de ações vs. denotational semantics

$$A_1 \textbf{ then } A_2$$

$$\lambda\epsilon_1.\lambda\rho.\lambda\kappa.A_1\epsilon_1\rho(\lambda\epsilon_2.A_2\epsilon_2\rho\kappa)$$

Semântica de ações / Semântica modular de reescrita

Arrow Labeled Transition systems $\mapsto \mathcal{R}-$systems

Modular Rewriting Semantics (Braga & Meseguer)

$$\textbf{crl} \quad \langle\ \mathrm{cond}(E_1, E_2, E_3),\ X\ \rangle \Rightarrow \langle\ \mathrm{cond}(E_1', E_2, E_3),\ X'\ \rangle$$
$$\textbf{if}\ \langle E_1, X\rangle \Rightarrow \langle E_1', X'\rangle \wedge E_1 \neq E_1'.$$

$$\textbf{rl} \quad \langle\ \mathrm{cond}(\mathbf{t}, E_2, E_3), X\rangle \Rightarrow \langle E_2, X\rangle.$$

$$\textbf{rl} \quad \langle\ \mathrm{cond}(\mathbf{f}, E_2, E_3), X\rangle \Rightarrow \langle E_3, X\rangle.$$

(Ações estão num nível de abstração maior do que MRS)

Semântica Definitiva $\rightsquigarrow$ Interoperabilidade

Através das construções abstratas podemos chegar a uma interoperabilidade de linguagens.

Interoperabilidade: orientação a objetos (simplificada)

OO sem: herança, polimorfismo, atributos públicos, funções privadas

- $\mathrm{class}(\mathrm{bindings} + \mathrm{store}, \{\mathrm{constructor}, \mathrm{destructor}, \mathrm{m}_1, \mathrm{m}_2, \ldots\})$
- $\mathrm{object}(\mathrm{class}, \mathrm{bindings} + \mathrm{store})$

Interoperabilidade: orientação a objetos (simplificada)

```
class Fatorial {
  int numero; int fatorial = -1;
  Fatorial (int p) { numero = p; }
  void calcula() { ... }
  int getFatorial () { return fatorial; }
}
```

```
let val fat = construct (Fatorial, 10),
        calcula = get_method (fat, calcula),
        getfatorial = get_method (fat, getFatorial) in
    calcula();
    getFatorial();
end
```

Interoperabilidade: orientação a objetos (simplificada)

```
class Fatorial {
  int numero; int fatorial = -1;
  Fatorial (int p) { numero = p; }
  void calcula() { ... }
  int getFatorial () { return fatorial; }
}
```

```
let val fat = construct (Fatorial, 10) in
    call_method (fat, calcula) ;
    call_method (fat, getFatorial)
end
```

Interoperabilidade: orientação a objetos (simplificada)

- Notação abstrata para mini-Java
- Integrar Action Semantics, MSOS, RWL
- Implementar mini-Java e ML em Action Notation
- ?
- Interoperabilidade!