Semantică operațională

Limbajul IMP—Implementare și decizii de proiectare

Traian Florin Șerbănuță

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

21 octombrie 2014

Implementare

Mai multe opțiuni

- Directă: reprezentarea regulilor direct într-un limbaj cu suport pentru unificare/matching și căutare
  - Maude
  - Prolog
- Interpretor care definește funcții de evaluare într-un pas pornind de la structura programelor
  - de exemplu, în OCaml (fișierul sursă e accesibil pe Moodle)
- Compilare bazată pe transformări de limbaj
  - Mașină virtuală bazată pe stive (progam/valori)
  - Cod nativ
  - Vedeți proiectul CompCert `http://compcert.inria.fr/`: un compilator pentru C cu demonstratie de corectitudine.

Tip abstract de date pentru sintaxă

```
type I = string
type op = Plus | Mic
type e =
  | Int of int                    (* op ::= + | <=          *)
  | Bool of bool                  (* e ::=                  *)
  | Op of e * op * e              (*      n                 *)
  | If of e * e * e               (*    | b                 *)
  | Loc of I                      (*    | e op e            *)
  | Atrib of I * e                (*    | if e then e else e *)
  | Skip                          (*    | ! I               *)
  | Secv of e * e                 (*    | I := e            *)
  | While of e * e                (*    | skip              *)
                                  (*    | e ; e             *)
                                  (*    | while e do e      *)
```

Starea memoriei

```
type state = (l * int) list

val lookup : l -> state -> int option
val update : l * int -> state -> state option
```

- Reprezentăm memoria ca o lista de perechi locație-valoare
  - Rezonabil pentru programe cu puține locații
  - Ușor modificabil la un alt tip (e.g., Map)
- Operațiile cu memoria sunt parțiale (tipul „option“)

```
type 'a option = None | Some of 'a
```

Un pas ca funcție de tranziție

- Configurația e o pereche expresie-stare

  type config = e * state

- Funcția de tranziție e parțială datorită configurațiilor ireductibile

  val reduce : config -> config option

Codarea regulilor de tranziție

Expresii aritmetice

```
let rec reduce = function
  | (Op(Int n1,Plus,Int  n2),s)  -> Some (Int (n1+n2),s)          (*Op+*)
  | (Op(Int n1,Mic,Int  n2),s)  -> Some (Bool (n1<=n2),s)        (*Op<=*)
  | (Op(Int n1,op,e2),s)  ->                                     (*OpD*)
    (match reduce (e2,s) with
       | None -> None
       | Some (e2',s')  -> Some (Op(Int n1,op,e2'),s')
    )
  | (Op(e1,op,e2),s) ->                                          (*OpS*)
    (match reduce (e1,s) with
       | None -> None
       | Some (e1',s')  -> Some (Op(e1',op,e2),s')
    )
```

Codarea regulilor de tranziție

Operații cu memoria

```
| (Loc l, s) ->                                                    (*Loc*)
  (match lookup l s with
     | None -> None
     | Some n -> Some (Int n, s)
  )
| (Atrib(l, Int n),s) ->                                          (*Atrib *)
  (match update (l,n) s with
     | None -> None
     | Some s' -> Some (Skip, s')
  )
| (Atrib(l,e),s) ->                                              (*AtribD*)
  (match reduce (e,s) with
     | None -> None
     | Some (e',s') -> Some (Atrib(l,e'), s')
  )
```

Codarea regulilor de tranziție

Programare „imperativă“

```
| (Secv(Skip,e),s) -> Some (e,s)                                   (*Secv*)
| (Secv(e1,e2),s) ->                                              (*SecvS*)
  (match reduce (e1,s) with
     | None -> None
     | Some (e1',s') -> Some (Secv(e1',e2),s')
  )
| ( If (Bool true ,e1,e2),s) -> Some (e1,s)                       (* IfTrue *)
| ( If (Bool false ,e1,e2),s) -> Some (e2,s)                     (* IfFalse *)
| ( If (e,e1,e2),s) ->                                           (* IfS *)
  (match reduce (e,s) with
     | None -> None
     | Some (e',s') -> Some (If(e',e1,e2),s')
  )
| (While(e1,e2),s) -> Some (If(e1,Secv(e2,While(e1,e2)),Skip),s) (*While*)
| _ -> None                                                      (* default *)
```

Codarea regulilor de tranziție

Observații

- Codarea regulilor de tranziție ca funcție e permisă de faptul că semantica IMP e puternic deterministă
  - Rezultatul asigură existența a cel mult un succesor pentru o configurație
  - În caz contrar funcția de un pas ar trebui să calculeze o mulțime (listă)
- Ordinea cazurilor cu același constructor contează
  - mai întâi cel mai concret (e.g., Op(Int n1, Plus, Int n2))
  - la sfârșit cel mai abstract (e.g. Op(e1, op, e2))
- Ordinea cazurilor ortogonale (e.g., cu constructori diferiți) nu contează
- Pentru regulile structurale aplicăm funcția de tranziție recursiv expresiei reductibile
- Ultima regulă (`_ -> None`) spune că funcția e nedefinită în rest

Execuția unui program

- Execuția se obține prin iterarea unui pas atât timp cât e posibil

```
let rec evaluate c = match reduce c with
  | None –> c
  | Some c' –> evaluate c'
```

- Observație: pentru programe care ciclează, e posibil ca evaluate sa nu se termine

Ordinea de evaluare a operanzilor

- Ce ordine de evaluare specifica regulile din semantica limbajului IMP?

$$(\text{OpS}) \quad \frac{\langle e_1, s\rangle \to \langle e_1', s'\rangle}{\langle e_1 \ op \ e_2, s\rangle \to \langle e_1' \ op \ e_2, s'\rangle}$$

$$(\text{OpD}) \quad \frac{\langle e_2, s\rangle \to \langle e_2', s'\rangle}{\langle n_1 \ op \ e_2, s\rangle \to \langle n_1 \ op \ e_2', s'\rangle}$$

- Cum specificăm o ordine de la dreapta la stânga?

$$(\text{OpS}) \quad \frac{\langle e_2, s\rangle \to \langle e_2', s'\rangle}{\langle e_1 \ op \ e_2, s\rangle \to \langle e_1 \ op \ e_2', s'\rangle}$$

$$(\text{OpD}) \quad \frac{\langle e_2, s\rangle \to \langle e_2', s'\rangle}{\langle e_1 \ op \ n_2, s\rangle \to \langle e_1' \ op \ n_2', s'\rangle}$$

- Cum specificăm că ordinea e complet arbitrară?

Rezultatul atribuirii

- Pentru limbajul IMP rezultatul atribuirii e skip

  (Atrib) $\langle l := n, s\rangle \to \langle \texttt{skip}, s[l \mapsto n]\rangle \quad$ *dacă* $l \in Dom(s)$

  (Secv) $\langle \texttt{skip; } e_2, s\rangle \to \langle e_2, s\rangle$

- Dar dacă am vrea să fie ca în C (`l1 := l2 := 5`)?

  (Atrib) $\langle l := n, s\rangle \to \langle n, s[l \mapsto n]\rangle \quad$ *dacă* $l \in Dom(s)$

  (Secv) $\langle v \texttt{ ; } e_2, s\rangle \to \langle e_2, s\rangle$

- Dar dacă vrem să întoarcă vechea valoare
  - Care ar fi efectul excuției codului `l1 := l2 := !l1` ?

inițializarea locațiilor de memorie

- Semantica IMP impune să fie inițializate în configurația inițială

$$(\text{Loc}) \quad \langle !\ l, s\rangle \to \langle n, s\rangle \quad \textit{dacă } l \in \mathit{Dom}(s), n = s(l)$$

$$(\text{Atrib}) \quad \langle l := n, s\rangle \to \langle \texttt{skip}, s[l \mapsto n]\rangle \quad \textit{dacă } l \in \mathit{Dom}(s)$$

- Dacă am vrea ca toate locațiile să fie „inițializate“ cu 0?

- Dacă am vrea ca locațiile să fie inițializate la prima atribuire?

Ce conține memoria?

- Chei: locații, valori: întregi

(Loc) $\langle !\ l, s\rangle \to \langle n, s\rangle \quad$ *dacă* $l \in Dom(s), n = s(l)$

(Atrib) $\langle l := n, s\rangle \to \langle \texttt{skip}, s[l \mapsto n]\rangle \quad$ *dacă* $l \in Dom(s)$

(AtribD) $$\frac{\langle e, s\rangle \to \langle e', s'\rangle}{\langle l := e, s\rangle \to \langle l := e', s'\rangle}$$

- Dar dacă am vrea să putem stoca orice fel de valoare? (`l := true`)
- Ar putea locațiile să fie și ele valori? (`l1 := l2`)
- Bucăți de cod (funcții/fragmente de program) ca valori?
- Declarații locale și domenii de vizibilitate?

Tipurile de date primitive

- tipul de date boolean nu e identificat cu cel întreg. De ce? Se poate?
- Semantica folosește întregi nemărginiți, implementarea întregi pe 31 de biți.
  - Se poate remedia problema? Cum?

Expresivitate

Este IMP suficient de expresiv?

DA E Turing complet

NU Nu are funcții, tipuri complexe, ...

Este IMP prea expresiv?

Adică, sunt programe care n-aș vrea să le pot scrie, dar le pot?

- Putem detecta programe greșite înainte de rulare? Cum?
- Scop: evitarea blocării programului în timpul execuției
- Soluție: Asocierea de tipuri și verificarea statică a lor