Laboratorio 6 - Transizioni SOS di Espressioni, Comandi Implementazione

Sommario: 10 maggio, 2019

- Raccomandazioni e Preliminari
- Semantica SOS - Transizioni e Computazione
- Espressioni: Le Transizioni Ultimi 6 casi
- Comandi: Le Transizioni Ultimi 5 casi
- L'implementazione: Le proprietà delle Funzioni Semantiche
- Attività di Oggi: Esercizi 15, 16, 17 (e 14)

SmallC: All'inizio di Ogni Sessione

- Caricare il file SmallC, distribuito per la sessione, sulla WAD.
  Verifica: Caricarlo sul TLE OCaml e Controllare Esecuzione Tests Precedenti.
- Copiarlo in SmallC5.
- Seguire la Presentazione delle Attività della Sessione.
- Svogere le Attività modificando il file SmallC distribuito.

Semantica SOS di SmallC: Transizione e Computazione

- **Transizione**.
  - Esecuzione di un costrutto c nello stato σ della Macchina Astratta
  - La esprimiamo con:
    - $\langle c, \sigma\rangle \rightarrow \sigma'$, indicante ...
    - $\langle c, \sigma\rangle \rightarrow \langle c', \sigma'\rangle$, indicante ...
    - $\langle c_1, \sigma_1\rangle \rightarrow \langle c'_1, \sigma'_1\rangle, \cdots, \langle c_k, \sigma_k\rangle \rightarrow \langle c'_k, \sigma'_k\rangle \ / \ \langle c, \sigma\rangle \rightarrow \langle c', \sigma'\rangle$, k-premesse/1-conclusione, indicante ...

- **Compurazione** La sequenza $\sigma_1, ..., \sigma_k, ...$ degli stati effettivamente calcolati dalle transizioni usate nella valutazione/esecuzione del programma.

- **Notazione**.
  - (ρ, μ) stato con ambiente ρ (anche con apici/pedici) e memoria μ (anche con apici/pedici)
  - N (anche con pedici) intero, I (anche con pedici) identificatore, D (anche con pedici) valore denotabile
  - $[I/D]\circ\rho$ crea un nuovo ambiente che estende ρ con il binding [I/D]
  - $\rho(I)$ denotazione del binding di I in ρ, se esiste
  - $\perp_{mem}$ indefinito nei memorizzabili
  - allocate(μ, n) alloca n locazioni libere, in sequenza, a partire da loc, modificando μ in μ' e restituisce (loc, μ')
  - μ(loc) (loc deve essere una locazione giá allocata) restituisce il valore di loc
  - μ[loc $\leftarrow$ n] (loc deve essere giá allocata) modifica il valore di loc con il valore n
  - loc $\oplus$ k locazione k posizioni dopo loc.
  - true, false le costanti booleane, 1,0 gli interi utilizzati in SmallC per le costanti booleane
  - [+],[-],[*],[div],[==],[>],[<] operazione somma, sottrazione,..., minore_di su interi

Semantica SOS: Le Transizioni $\mathtt{Sem}_{\mathtt{Exp}}$ - ultimi 6 casi.

$$\begin{aligned}\mathtt{Exp} ::= {} & [\mathtt{val}]\ \mathtt{Ide} \mid \mathtt{Num} \mid \mathtt{Ide}\,[\uparrow]\,\mathtt{Exp} \\ & \mid \mathtt{Exp}\,[+]\,\mathtt{Exp} \mid \mathtt{Exp}\,[-]\,\mathtt{Exp} \mid \mathtt{Exp}\,[*]\,\mathtt{Exp} \mid \mathtt{Exp}\,[\mathtt{div}]\,\mathtt{Exp} \\ & \mid \mathtt{Exp}\,[==]\,\mathtt{Exp} \mid \mathtt{Exp}\,[<]\,\mathtt{Exp} \mid \mathtt{Exp}\,[>]\,\mathtt{Exp} \\ & \mid [\mathtt{not}]\,\mathtt{Exp} \mid \mathtt{Exp}\,[\mathtt{or}]\,\mathtt{Exp} \mid \mathtt{Exp}\,[\mathtt{and}]\ \mathtt{Exp}\end{aligned}$$

- Il sistema di regole $\mathtt{Sem}_{\mathtt{Exp}}$, sotto riportato, definisce il comportamento delle espressioni durante la computazione dei Programmi SmallC.

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1\,[\mathtt{op}]\,\mathtt{N}_2 = \mathtt{true} \qquad \mathtt{op} \in \{==, <, >\}}{\langle \mathtt{e}_1\,[\mathtt{op}]\,\mathtt{e}_2, \sigma\rangle \to 1}$$

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1\,[\mathtt{op}]\,\mathtt{N}_2 = \mathtt{false} \qquad \mathtt{op} \in \{==, <, >\}}{\langle \mathtt{e}_1\,[\mathtt{op}]\,\mathtt{e}_2, \sigma\rangle \to 0}$$

$$\frac{\langle \mathtt{e}, \sigma\rangle \to 0}{\langle [\mathtt{not}]\ \mathtt{e}, \sigma\rangle \to 1} \qquad \frac{\langle \mathtt{e}, \sigma\rangle \to 1}{\langle [\mathtt{not}]\ \mathtt{e}, \sigma\rangle \to 0}$$

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1 = 1}{\langle \mathtt{e}_1\,[\mathtt{or}]\,\mathtt{e}_2, \sigma\rangle \to 1} \qquad \frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_2 = 1}{\langle \mathtt{e}_1\,[\mathtt{or}]\,\mathtt{e}_2, \sigma\rangle \to 1}$$

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1 = 0 \qquad \mathtt{N}_2 = 0}{\langle \mathtt{e}_1\,[\mathtt{or}]\,\mathtt{e}_2, \sigma\rangle \to 0}$$

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1 = 0}{\langle \mathtt{e}_1\,[\mathtt{and}]\,\mathtt{e}_2, \sigma\rangle \to 0} \qquad \frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_2 = 0}{\langle \mathtt{e}_1\,[\mathtt{and}]\,\mathtt{e}_2, \sigma\rangle \to 0}$$

$$\frac{\langle \mathtt{e}_1, \sigma\rangle \to \mathtt{N}_1 \qquad \langle \mathtt{e}_2, \sigma\rangle \to \mathtt{N}_2 \qquad \mathtt{N}_1 = 1 \qquad \mathtt{N}_2 = 1}{\langle \mathtt{e}_1\,[\mathtt{and}]\,\mathtt{e}_2, \sigma\rangle \to 1}$$

Implementazione: La funzione di Interpretazione delle Espressioni

Dobbiamo introdurre ed implementare in OCaml una funzione che esprima il comportamento del sistema $\texttt{SEM}_{\texttt{EXP}}$ definito per la semantica SOS delle Espressioni OCaml

- Introduciamo una funzione che chiameremo `expSem`.
- `expSem` deve definire una trasformazione che data una espressione e e uno stato σ calcola il valore prodotto usando le transizioni di $\texttt{SEM}_{\texttt{EXP}}$. Ovvero:
$$(\forall \texttt{e}, \sigma) \quad \texttt{expSem}(\texttt{e}, \sigma) = \texttt{N} \quad \texttt{iff} \quad \langle \texttt{e}, \sigma \rangle \rightarrow \texttt{N} \in \texttt{Sem}_{\texttt{EXP}}$$
- `expSem` ha segnatura:
$$\texttt{expSem} : \texttt{Exp} \rightarrow \texttt{State} \rightarrow \texttt{Num}$$
La presenza di premesse contenenti $\rightarrow$ nelle regole di inferenza conduce a definizioni ricorsive della funzione semantica.
- `expSem` associa ad ogni espressione una funzione di tipo $\texttt{State} \rightarrow \texttt{Num}$
- In effetti l'attuale esclusione dell'assegnamento dalle espressioni rende le espressioni di SmallC, diversamente da quelle di C, prive di side-effects. in particolare, la presenza di side-effects condurrebbe a modifiche della memoria e quindi alla seguente segnatura per `expSem`: $\texttt{expSem} : \texttt{Exp} \rightarrow \texttt{State} \rightarrow (\texttt{Num} \times \texttt{State})$

Semantica SOS: Le Transizioni $\mathtt{Sem_{Cmd}}$ - ultimi 5 casi.

$$\begin{aligned}\mathtt{Cmd} ::= {} & \mathtt{Ide}\ [=]\ \mathtt{Exp} \mid \mathtt{Ide}\ \mathtt{Exp}\ [\leftarrow]\ \mathtt{Exp} \\ & \mid [\mathtt{ifE}]\ \mathtt{Exp}\ \mathtt{Cmd}\ \mathtt{Cmd} \mid [\mathtt{ifT}]\ \mathtt{Exp}\ \mathtt{Cmd} \\ & \mid [\mathtt{while}]\ \mathtt{Exp}\ \mathtt{Cmd} \mid \mathtt{Cmd}\ [\mathtt{seqC}]\ \mathtt{Cmd} \mid [\mathtt{emptyCMD}]\end{aligned}$$

- Il sistema di regole $\mathtt{Sem_{Cmd}}$, sotto riportato, definisce il comportamento dei comandi durante la computazione dei Programmi SmallC.

$$\frac{\langle \mathtt{e}, (\rho, \mu)\rangle \to 1 \qquad \langle \mathtt{c}_1, (\rho, \mu)\rangle \to (\rho, \mu')}{\langle [\mathtt{ifE}]\ \mathtt{e}\ \mathtt{c}_1\ \mathtt{c}_2, (\rho, \mu)\rangle \to (\rho, \mu')}$$

$$\frac{\langle \mathtt{e}, (\rho, \mu)\rangle \to 0 \qquad \langle \mathtt{c}_2, (\rho, \mu)\rangle \to (\rho, \mu')}{\langle [\mathtt{ifE}]\ \mathtt{e}\ \mathtt{c}_1\ \mathtt{c}_2, (\rho, \mu)\rangle \to (\rho, \mu')}$$

$$\frac{\langle \mathtt{e}, (\rho, \mu)\rangle \to 1 \qquad \langle \mathtt{c}, (\rho, \mu)\rangle \to (\rho, \mu')}{\langle [\mathtt{ifT}]\ \mathtt{e}\ \mathtt{c}, (\rho, \mu)\rangle \to (\rho, \mu')}$$

$$\frac{\langle \mathtt{e}, \sigma\rangle \to 0}{\langle [\mathtt{ifT}]\ \mathtt{e}\ \mathtt{c}, (\rho, \mu)\rangle \to \sigma}$$

$$\frac{\langle \mathtt{e}, (\rho, \mu)\rangle \to 0}{\langle [\mathtt{while}]\ \mathtt{e}\ \mathtt{c}, (\rho, \mu)\rangle \to (\rho, \mu)}$$

$$\frac{\langle \mathtt{e}, (\rho, \mu)\rangle \to 1 \qquad \langle \mathtt{c}, (\rho, \mu)\rangle \to (\rho, \mu_1) \qquad \langle [\mathtt{while}]\ \mathtt{e}\ \mathtt{c}, (\rho, \mu_1)\rangle \to (\rho, \mu')}{\langle [\mathtt{while}]\ \mathtt{e}\ \mathtt{c}, (\rho, \mu)\rangle \to (\rho, \mu')}$$

$$\frac{}{\langle [\mathtt{emptyCMD}], (\rho, \mu)\rangle \to (\rho, \mu)}$$

Implementazione: La funzione di Interpretazione dei Comandi

Dobbiamo introdurre ed implementare in OCaml una funzione che esprima il comportamento del sistema $\texttt{SEM}_{\texttt{CMD}}$ definito dalla semantica SOS data per le dichiarazioni.

- Introduciamo una funzione che chiameremo `cmdSem`.
- `cmdSem` deve definire una trasformazione che data un comando c e uno stato σ calcola lo stato prodotto usando le transizioni di $\texttt{SEM}_{\texttt{CMD}}$. Ovvero:
  $$(\forall \texttt{c}, \sigma) \quad \texttt{cmdSem}(\texttt{c}, \sigma) = \sigma' \quad \texttt{iff} \quad \langle \texttt{c}, \sigma \rangle \rightarrow^* \sigma' \in \texttt{Sem}_{\texttt{CMD}}$$ [1]
- `cmdSem` ha segnatura:
  $$\texttt{cmdSem} : \texttt{cmd} \rightarrow \texttt{State} \rightarrow \texttt{State}$$
  La presenza di premesse contenenti $\rightarrow$ nelle regole di inferenza conduce a definizioni ricorsive della funzione semantica.
- `cmdSem` associa ad ogni comando una funzione di tipo $\texttt{State} \rightarrow \texttt{State}$

[1] $\rightarrow^*$ indica una computazione terminante in k-applicazioni delle regole $\texttt{Sem}_{\texttt{CMD}}$, per qualche $\texttt{k} \geq 1$, ovvero:
$$\langle \texttt{c}, \sigma \rangle \equiv \langle \texttt{c}_1, \sigma_1 \rangle \rightarrow \ldots \rightarrow \langle \texttt{c}_{\texttt{k}-1}, \sigma_{\texttt{k}-1} \rangle \rightarrow \sigma_{\texttt{k}} \equiv \sigma'$$

Gli Esercizi di Oggi: 3 esercizi

Esercizio (15)

(a) Si completi la definizione per casi della funzione "expSem" con gli utlimi 6 casi di cui è stata data la semantica.
(b) Si verfichi, utilizzando opportuni tests (da mostrare) la correttezza delle definizioni date.
(c) Si mostri quanto ottenuto: Check

Esercizio (16)

(a) Si completi la definizione della funzione "cmdSem" con gli ultimi 5 casi di cui è stata data la semantica.
(b) Si verfichi, utilizzando opportuni tests (da mostrare) la correttezza delle definizioni date.
(c) Si mostri quanto ottenuto: Check

Esercizio (17)

Nello spazio riservato ai "Semantic Functions: expSem e cmdSem", in coda alle soluzioni presenti, si fornisca la definizione per un AST del programma prog6, la cui sintassi concreta è mostrata sotto:

```
{
 x = 7;
 var y = 10;
 var z;
 A[12];
 y = (x + y);
 z = 0;
 while (z < 12) {
        A[z] = (y + z);
        z = (z + 1);
        }
}
```

Si verifichi la corrispondenza di quanto scritto: Check;

Per quanti abbiano superato il check di tutti gli esercizi

Esercizio (14)

(a) Si scrivano le transizioni SOS della semantica dei programmi.
(b) Si fornisca la definizione per casi della funzione "progSem".
(c) Si mostri che la definizione data soddisfa quanto espresso dalla semantica SOS. Check

Tests di Controllo Run-Time

- Usiamo Tests Semplici che si limitino oggi, a controllare il comportamento principale (trascurando i vari, possibili, casi) atteso.
- Possiamo costruire uno schema generale a modificare via via per i vari costrutti
- Possiamo limitarci oggi, a 3 soli tests per le espressioni e 3 per i comandi.
- E completare i tests successivamente

```
(* tests  -- campione da modificare  *)
let d1 = Var("x",10);;
let d2 = Var("y",10);;
let dd = SeqDcl(d1,d2);;
let c1 = UpdVar("x",Eq(Val "x",Val "y"));;
let sigma1 = dclSem dd sigma0;;
showState sigma1;;
let sigma2 = cmdSem c1 sigma1;;
showState sigma2;;
```

SmallC: Al termine di Ogni Sessione

- Riportare nel file SmallC.ml ogni progresso nel codice sviluppato durante la sessione.
- La prossima sessione:
  - Sviluppiamo Programmi in SmallC ed Eseguiamoli
  - Un interfaccia interattiva per l'Interprete: Quali Modifiche?
  - Estendiamo il Linguaggio SmallC con Nuovi Costrutti