Laboratorio 5
(Semantica delle dichiarazioni SmallC ed Implementazione)

Sommario: 20 aprile, 2018

- Implementazione: Memoria, Ambiente e loro operazioni
- SOS delle Dichiarazioni: Le Regole di Inferenza per Dcl
- SOS delle Espressioni: Le Regole di Inferenza per Exp
- SOS - Implementazione: La funzione semantica per Dcl
- Implementazione: La funzione di Interpretazione delle dichiarazioni

Semantica SOS di SmallC: Lo stato

- La definizione di **Stato**.

  - **Ambiente** per identificatori di costanti e di variabili (valori modificabili)
    - Lo rappresentiamo come sequenza finita di **bindings** separati da "," e racchiusi in parentesi quadre:
      $$[\mathtt{Ide}_1/\mathtt{Den}_1, ..., \mathtt{Ide}_k/\mathtt{Den}_k]$$

  - **Memoria** Statica per trattare variabili ed array a componenti modificabili di un solo tipo (interi)
    - Rappresentiamo la Memoria come sequenza finita di **words** separate da "," e racchiuse in parentesi quadre:
      $$[\mathtt{loc}_1 \leftarrow \mathtt{Mv}_1, ..., \mathtt{loc}_k \leftarrow \mathtt{Mv}_k]$$

  - **Stato** è rappresentato una coppia (ρ, μ), indicante Ambiente e Memoria.
    - Lo Stato è denotata con la variabile σ (anche con pedici)

Ocaml: Implementazione dello stato

- Implementazione dello **Stato**.
  - **Ambiente** $[\mathtt{Ide}_1/\mathtt{Den}_1, ..., \mathtt{Ide}_k/\mathtt{Den}_k]$
    Implementazione dell'Ambiente in Ocaml:
    — da dare oggi
  - **Memoria** $[\mathtt{loc}_1 \leftarrow \mathtt{Mv}_1, ..., \mathtt{loc}_k \leftarrow \mathtt{Mv}_k]$
    Implementazione dello Memoria in Ocaml (vedi Listing del 10/4)

```
type loc = Loc of int;;
type mval = Mval of int | Undef;;
(* Store:  Operations and Exceptions *)
let storeSize = 1000;;
```

  - **Stato** (ρ, μ)
    Implementazione in Ocaml:
    — da dare oggi
- Prima di procedere con l'implementazione guadiamo la semantica che ci dice come deve essere fatto l'esecutore che stiamo implementando

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

Dichiarazioni SmallC: Le Transizioni

$\texttt{Dcl} ::= \texttt{const Ide Num} \mid \texttt{var Ide Num} \mid \texttt{varN Ide} \mid \texttt{array Ide Num} \mid \texttt{Dcl;Dcl}$

- Il sistema di regole $\text{Sem}_{\texttt{Dcl}}$, sotto riportato, definisce il comportamento delle dichiarazioni SmallC durante la computazione dei Programmi.

$$\langle \texttt{const I N}, (\rho, \mu)\rangle \rightarrow ([\texttt{I}/\texttt{N}]\circ\rho, \mu)$$

$$\frac{\texttt{allocate}(\mu, 1) = (\texttt{loc}, \mu')}{\langle \texttt{var I N}, (\rho, \mu)\rangle \rightarrow ([\texttt{I}/\texttt{loc}]\circ\rho, \mu'[\texttt{loc} \leftarrow \texttt{N}])} \qquad \frac{\texttt{allocate}(\mu, 1) = (\texttt{loc}, \mu')}{\langle \texttt{varN I}, (\rho, \mu)\rangle \rightarrow ([\texttt{I}/\texttt{loc}]\circ\rho, \mu')}$$

$$\frac{\texttt{allocate}(\mu, \texttt{N}) = (\texttt{loc}, \mu')}{\langle \texttt{array I N}, (\rho, \mu)\rangle \rightarrow ([\texttt{I}/\texttt{loc}]\circ\rho, \mu')}$$

$$\frac{\langle \texttt{d}_1, (\rho, \mu)\rangle \rightarrow (\rho', \mu')}{\langle \texttt{d}_1\texttt{; d}_2, (\rho, \mu)\rangle \rightarrow \langle \texttt{d}_2, (\rho', \mu')\rangle}$$

Implementazione: Memoria, Ambiente e loro operazioni

Se esaminiamo le transizioni, vediamo le operazioni sulla Memoria e sull'Ambiente richieste per descrivere il comportamento delle dichiarazioni (semantica SOS).

- **Memoria**.
  - **allocate**(μ, **n**): alloca `n words` (per interi) in sequenza
  - **upd**(μ, **loc**, **n**): (alias, $\mu[\texttt{loc} \leftarrow \texttt{n}]$) modifica il valore di una locazione
  - **getStore**(μ, **loc**): (alias, $\mu(\texttt{loc})$) fornisce il valore di una locazione
  - **emptyStore**(): crea uno store iniziale con `words` libere, a valore indefinito
- **Ambiente**.
  - **bind**(ρ, **ide**, **den**): (alias, $[\texttt{ide}/\texttt{den}] \circ \rho$) aggiunge un nuovo binding
  - **getEnv**(ρ, **ide**): (alias, $\rho(\texttt{ide})$) valore denotabile di un binding
  - **emptyEnv**(): crea un'ambiente senza bindings

Per l'implementazione vedere il codice OCaml nel listing allegato all'attività di oggi

Implementazione: La funzione di Interpretazione delle dichiarazioni

Dobbiamo introdurre ed implementare in OCaml una funzione che esprima il comportamento del sistema di transizione definito dalla semantica SOS.

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

[1] $\rightarrow^*$ indica una computazione terminante in k-applicazioni delle regole $\mathtt{Sem}_{\mathtt{Dcl}}$, per qualche $k \geq 1$, ovvero:
$\langle \mathtt{d}, \sigma \rangle \rightarrow \langle \mathtt{d}_1, \sigma_1 \rangle \rightarrow \ldots \rightarrow \langle \mathtt{d}_k, \sigma_k \rangle \rightarrow \sigma'$