Laboratorio 4 - Small20.
Abstract Machine di Small20: Lo Stato

SOMMARIO: 24 Aprile, 2020

- Sintassi Astratta:
  - Rivista e Corretta (Ancora)
- Sintassi Concreta:
  - Una CFG per Small20 (Invariata)
- Attività.
  - Front-End: Stesura e Stampa Programmi (conclusioni)
  - Stato di AM: Definizione e Implementazione

Sintassi Astratta: Gli AST di Small20

```
Type ::= [int] | [bool] | [void]
       | [arr] Type Num | [mut] Type | [terr]
       | [abs] Type TypeSeq
TypeSeq ::= Type [x] Type | Type

Dcl ::= [const] Type Ide Exp | [var] Type Ide Exp
      | [varN] Type Ide
      | [array] Type Ide Num

Exp ::= [val] Ide | Num | Bool | Ide [↑] Exp | [−1]Exp
      | Exp [+] Exp | Exp [−] Exp | Exp [∗] Exp | Exp [div] Exp
      | Exp [==] Exp | Exp [<] Exp | Exp [>] Exp
      | [not] Exp | Exp [or] Exp | Exp [and] Exp
      | Exp [=] Exp | [emptyE]

Cmd ::= Cmd [seqC] Cmd
      | [ifE] Exp Cmd Cmd | [ifT] Exp Cmd
      | [for] Stm Exp Stm Cmd | Exp | [emptyC]

Stm ::= Dcl | Cmd | Stm [seqM] Stm | [emptyS]

Prog ::= [prog] Ide Stm
```

Sintassi Concreta: Una CFG per Small20

$$
\begin{aligned}
&\texttt{Type} ::= \texttt{Simple} \mid \texttt{void} \mid \texttt{Simple}\ [\texttt{Num}]\\
&\texttt{Simple} ::= \texttt{int} \mid \texttt{bool}\\
\\
&\texttt{Dcl} ::= \texttt{final Simple ide} = \texttt{Exp} \mid \texttt{var Simple ide} = \texttt{Exp}\\
&\qquad\qquad \mid \texttt{var Simple ide} \mid \texttt{Simple array ide}\,[\texttt{num}]\\
\\
&\texttt{Exp} ::= \texttt{Exp or ExpB1} \mid \texttt{ExpB1}\\
&\texttt{ExpB1} ::= \texttt{ExpB1 and ExpB} \mid \texttt{ExpB}\\
&\texttt{ExpB} ::= \texttt{not ExpB} \mid \texttt{truth} \mid \texttt{ExpR}\\
&\texttt{ExpR} ::= \texttt{ExpR} == \texttt{ExpA2} \mid \texttt{ExpR} < \texttt{ExpA2} \mid \texttt{ExpR} > \texttt{ExpA2} \mid \texttt{ExpA2}\\
&\texttt{ExpA2} ::= \texttt{ExpA2} + \texttt{ExpA1} \mid \texttt{ExpA2} - \texttt{ExpA1} \mid \texttt{ExpA1}\\
&\texttt{ExpA1} ::= \texttt{ExpA1} * \texttt{ExpA} \mid \texttt{ExpA1} \,/\, \texttt{ExpA} \mid \texttt{ExpA}\\
&\texttt{ExpA} ::= \texttt{num} \mid \texttt{DExp} \mid \texttt{DExp} = \texttt{Exp} \mid (\texttt{Exp}) \mid -\texttt{ExpA2}\\
&\texttt{DExp} ::= \texttt{ide} \mid \texttt{ide}\,[\texttt{ExpA2}] \mid \epsilon\\
\\
&\texttt{Cmd} ::= \texttt{if}\ (\texttt{ExpB2})\ \texttt{Cmd} \mid \texttt{OtherCmd}\\
&\texttt{OtherCmd} ::= \texttt{if}\ (\texttt{ExpB2})\ \texttt{OtherCmd else Cmd} \mid \texttt{NonConditionalCmd}\\
&\texttt{NonConditionalCmd} ::= \texttt{for}\ (\texttt{Stm};\ \texttt{Exp};\ \texttt{Stm})\ \texttt{Cmd} \mid \texttt{Exp} \mid \{\texttt{cmds}\}\\
&\texttt{Cmds} ::= \texttt{Cmd};\ \mid \texttt{Cmd};\ \texttt{Cmds}\\
\\
&\texttt{Stm} ::= \texttt{Dcl} \mid \texttt{Cmd}\\
&\texttt{Stms} ::= \texttt{Stm};\ \mid \texttt{Stm};\ \texttt{Stms}\\
\\
&\texttt{Prog} ::= \texttt{Program ide}\ \{\texttt{Stms}\}
\end{aligned}
$$

Semantica SOS di SMall20: Stato di AM20

- La definizione di **Stato** della Macchina Astratta di Small20.

  - **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 tipi diversi (interi e booleani)
    - 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)

Semantica SOS di Small120: 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)
  - 0^{ρ} crea un ambiente vuoto: Senza bindings.
  - 0^{μ} crea una memoria vuota: Tutte le words sono indefinite e allocabili.
  - $\perp_{mem}$ memorizzabile indefinito, contenuto di word indefinita: Indica valore misleading o ingannevole
  - N (anche con pedici) intero, B (anche con pedici) booleano, I (anche con pedici) identificatore, D (anche con pedici), una coppia (t, d_t), indicante un tipo e una denotazione di tale tipo.
  - $[I/D]\circ\rho$ crea un nuovo ambiente che estende ρ con il binding [I/D]
  - $\rho(I)$ denotazione del binding di I in ρ, se esiste
  - $\triangleright(\mu, n)$ alloca in μ, n locazioni libere, in sequenza da loc, modifica μ in μ', restituisce (loc, μ')
  - $\triangleleft(\mu, \rho)$ dealloca da μ, le locazioni in ρ, che tornano allocabili. Restituisce la memoria modificata.
  - $\mu(loc)$ (loc deve essere una locazione giá allocata) restituisce il valore di loc
  - $\mu[loc \leftarrow v]$ (loc deve essere giá allocata) modifica il valore di loc con il memorizzabile v
  - $loc \oplus k$ locazione k posizioni dopo loc.
  - Introduzione o Vincolo. Posto in premessa con forma: espressione = pattern.

Dichiarazioni Small20: Le Transizioni $\mathtt{SEM}_{\mathtt{DCL}}$

Il sistema di regole $\mathtt{Sem}_{\mathtt{Dcl}}$ definisce il comportamento delle dichiarazioni durante la computazione dei Programmi Small20 sulla Macchina Astratta AM20.

$\mathtt{Dcl} ::= [\mathtt{const}]\ \mathtt{Type}\ \mathtt{Ide}\ \mathtt{Exp} \mid [\mathtt{var}]\ \mathtt{Type}\ \mathtt{Ide}\ \mathtt{Exp}$
$\quad \mid [\mathtt{varN}]\ \mathtt{Type}\ \mathtt{Ide} \mid [\mathtt{array}]\ \mathtt{Type}\ \mathtt{Ide}\ \mathtt{Num}$

$$\frac{\begin{array}{c}\langle \mathtt{e}, (\rho, \mu)\rangle \rightarrow \langle \mathtt{t_e}, \mathtt{v}, (\rho, \mu_\mathtt{e})\rangle \\ \mathtt{t} \in \mathtt{Simple} \qquad \mathtt{t} = \mathtt{t_e} \\ [\mathtt{I}/(\mathtt{t}, \mathtt{v})] \circ \rho = \rho_\mathtt{I}\end{array}}{\langle [\mathtt{const}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{e}, (\rho, \mu)\rangle \rightarrow ([\mathtt{void}], (\rho_\mathtt{I}, \mu_\mathtt{e}))}$$

$$\frac{\ldots.}{\langle [\mathtt{var}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{e}, (\rho, \mu)\rangle \rightarrow ([\mathtt{void}], (\rho_\mathtt{I}, \mu_\mathtt{a}))} \qquad \frac{\ldots.}{\langle [\mathtt{varN}]\ \mathtt{t}\ \mathtt{I}, (\rho, \mu)\rangle \rightarrow ([\mathtt{void}], (\rho_\mathtt{I}, \mu_\mathtt{a}))}$$

$$\frac{\ldots.}{\langle [\mathtt{array}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{N}, (\rho, \mu)\rangle \rightarrow ([\mathtt{void}], (\rho_\mathtt{I}, \mu_\mathtt{a}))}$$

Notazione, Osservaioni

- [xxx] è un costruttore di AST (i.e, Albero di Sintassi Astratta). I rimanenti identificatori (inessenziale se minuscoli o Maiuscoli o se con indici) che occorrono in una regola sono nomi di variabile quantificate universalmente se introdotte, i.e. bound, nel termine sinistro della conclusione, esistenzialmente se introdotte nel termine destro di una premessa o di una uguaglianza.
- $\mathtt{Simple} = \{[\mathtt{Int}], [\mathtt{Bool}]\}$

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**$(\mu, \mathbf{t}, \mathbf{n})$: (alias, $\triangleright(\mu, \mathtt{t}, \mathtt{n})$) alloca `n words` di tipo `t` in sequenza
  - **upd**$(\mu, \mathbf{loc}, \mathbf{v})$: (alias, $\mu[\mathtt{loc} \leftarrow \mathtt{v}]$) modifica il valore di una locazione
  - **getStore**$(\mu, \mathbf{loc})$: (alias, $\mu(\mathtt{loc})$) fornisce il valore di una locazione
  - **emptyStore**(): (alias, 0^{μ}) crea uno store iniziale con `words` libere e contenuto indefinito

- **Ambiente**.
  - **bind**$(\rho, \mathbf{ide}, \mathbf{den})$: (alias, $[\mathtt{ide}/\mathtt{den}] \circ \rho$) aggiunge un nuovo binding
  - **getEnv**$(\rho, \mathbf{ide})$: (alias, $\rho(\mathtt{ide})$) valore denotabile di un binding
  - **emptyEnv**(): (alias, 0^{ρ}) crea un'ambiente senza bindings

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

Implementazione Stato di AM20: 6 Esercizi

Esercizio (1)

Definizione di allocate: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi)

Esercizio (2)

Definizione di upd: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi).

Esercizio (3)

Definizione di getStore: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi).

Esercizio (4)

Definizione di emptyEnv: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi).

Esercizio (5)

Definizione di bind: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi).

Esercizio (6)

Definizione di getEnv: Correggere il codice e Test Prima Verifica (rimuovendo opportunamente i commenti nella sezione Tests relativi).