Laboratorio 6 - Small20.
Abstract Machine di Small20: Esecutore di Espressioni

SOMMARIO: 8 Maggio, 2020

- Sintassi Astratta: AST in Ocaml (invariata)
- Sintassi Concreta: Una CFG per Small20 (Invariata)
- Abstract Machine: AM20 - Stato (Invariato)
- Attività.
  - Semantica SOS con Controllo Dinamico dei Tipi:
    - Espressioni: Il Sistema $\texttt{Sem}_{\texttt{EXP}}$
  - AM20 - Implementazione Esecutore:
    - Espressioni: definizione Ocaml di `expSem`

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

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

Semantica SOS di SMall20: Stato di AM20

- **Memoria** Statica per variabili ed array di tipi diversi (interi e booleani)
  Sequenza finita di **words** separate da "," e racchiusa in parentesi quadre:
  $$[\texttt{loc}_1 \leftarrow \texttt{Mv}_1, ..., \texttt{loc}_k \leftarrow \texttt{Mv}_k]$$
  - **allocate**$(\mu, \textbf{n})$: (alias, $\triangleright(\mu, \texttt{n})$) alloca `n words` in sequenza;
  - **upd**$(\mu, \textbf{loc}, \textbf{v})$: (alias, $\mu[\texttt{loc} \leftarrow \texttt{v}]$) modifica il contenuto di una word;
  - **getStore**$(\mu, \textbf{loc})$: (alias, $\mu(\texttt{loc})$) fornisce il contenuto di una word;
  - **emptyStore**(): (alias, 0^{μ}) crea uno store con `words` di contenuto indefinito.

- **Ambiente** per identificatori di costanti e di variabili (valori modificabili)
  Sequenza finita di **bindings** separati da "," e racchiusa in parentesi quadre:
  $$[\texttt{Ide}_1/\texttt{Den}_1, ..., \texttt{Ide}_k/\texttt{Den}_k]$$
  - **bind**$(\rho, \textbf{ide}, \textbf{den})$: (alias, $[\texttt{ide}/\texttt{den}] \circ \rho$) aggiunge un nuovo binding;
  - **getEnv**$(\rho, \textbf{ide})$: (alias, $\rho(\texttt{ide})$) valore denotabile di un binding;
  - **emptyEnv**(): (alias, 0^{ρ}) crea un'ambiente senza bindings.

- **Stato**: Coppia (ρ, μ), indicante Ambiente e Memoria.
  - Activation Record (Stack di AR):
    - Unico: Si riduce al solo frame ρ (del blocco programma).
    - Altri componenti:
      Assenti: Static/Dynamic chain, Return Address,...
      Associati allo AST: Program Counter, Intermediate Results,..

Semantica SOS di Small120: Transizione e Computazione

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

Espressioni con Stato: Le Transizioni SEM_{EXP} e SEM_{DEN}

Il Sistema di regole Sem_{EXP} (risp. Sem_{DEN}) definisce il comportamento delle r-espressioni (risp. l-espressioni) durante la computazione dei Programmi Small20 sulla Macchina Astratta AM20.
Controllo dei Tipi Dinamico: I Sistemi Sem_{EXP} e Sem_{DEN} sono Integrati con il Sistema Y (slide7, per le espressioni).

$\texttt{Exp} ::= [\texttt{val}]\ \texttt{Ide} \mid \texttt{Num} \mid \texttt{Bool} \mid \texttt{Ide}\ [\uparrow]\ \texttt{Exp} \mid \ldots$

$$\text{X1:}\ \frac{\ldots}{\langle [\texttt{num}]\ \texttt{N}, \sigma\rangle \rightarrow \lfloor \ldots \rfloor} \qquad \text{X2:}\ \frac{\ldots}{\langle [\texttt{truth}]\ \texttt{B}, \sigma\rangle \rightarrow \lfloor \ldots \rfloor} \qquad \text{X3:}\ \frac{\ldots}{\langle [\texttt{emptyE}]\ , \sigma\rangle \rightarrow_{\texttt{Y}} \lfloor \ldots \rfloor}$$

$$\text{X4:}\ \frac{\begin{array}{c}\sigma = (\rho, \mu)\\ \ldots \\ \texttt{t}' \in \texttt{Simple} \\ \texttt{t} = [\texttt{mut}]\ \texttt{t}'\end{array}}{\langle [\texttt{val}]\ \texttt{I}, \sigma\rangle \rightarrow_{\texttt{Den}} \lfloor \ldots, \texttt{loc}'_{\texttt{t}}, \ldots \rfloor} \qquad \text{X5:}\ \frac{\begin{array}{c}\sigma = (\rho, \mu)\\ \rho(\texttt{I}) = ([\texttt{mut}]\ \texttt{t}, \texttt{loc}_{\texttt{t}}) \\ \texttt{t} \in \texttt{Simple} \\ \ldots\end{array}}{\langle [\texttt{val}]\ \texttt{I}, \sigma\rangle \rightarrow \lfloor \ldots, \texttt{v}_{\texttt{t}}, \ldots \rfloor} \qquad \text{X6:}\ \frac{\begin{array}{c}\sigma = (\rho, \mu)\\ \rho(\texttt{I}) = \ldots \\ \texttt{t} \in \texttt{Simple}\end{array}}{\langle [\texttt{val}]\ \texttt{I}, \sigma\rangle \rightarrow \lfloor \ldots \rfloor}$$

$$\text{X7:}\ \frac{\begin{array}{c}\rho(\texttt{I}) = \ldots \\ \ldots \rightarrow \ldots \\ \texttt{N} \in [0..\texttt{N}_{\texttt{k}} - 1] \quad \texttt{t} \in \texttt{Simple} \\ \texttt{loc}_{\texttt{a}} \oplus \texttt{N} = \ldots\end{array}}{\langle \texttt{I}\ [\uparrow]\ \texttt{e}, \sigma\rangle \rightarrow_{\texttt{Den}} \lfloor \ldots, \texttt{loc}_{\texttt{t}}, \ldots \rfloor} \qquad \text{X8:}\ \frac{\ldots}{\langle \texttt{I}\ [\uparrow]\ \texttt{e}, \sigma\rangle \rightarrow \lfloor \ldots \rfloor}$$

Notazione e Osservazioni

- $\rightarrow$ ha termine destro una tripla indicante tipo, valore (esprimibile/denotabile), stato, racchiusa da $\lfloor\ \rfloor$.
- ? è l'unico valore di tipo `void`: La sua presenza ha solo ragioni tecniche.
- $\rightarrow_{\texttt{DEN}}$ è la relazione per di Tipo-ValoreDenotato di espressioni con doppio significato.
- $\texttt{N} \in [0..\texttt{N}_{\texttt{k}} - 1]$ è il controllo a run-time sui bounds inferiore e superiore per i componenti di array statico.

Sistema Y: Regole per EXP - Parte1

$$\text{Y5:}\ \frac{}{\langle [\texttt{num}]\,\texttt{N}, \sigma\rangle \rightarrow_{\texttt{Y}} ([\texttt{int}], \texttt{Y}_\rho)} \qquad \text{Y6:}\ \frac{}{\langle [\texttt{truth}]\,\texttt{B}, \sigma\rangle \rightarrow_{\texttt{Y}} ([\texttt{bool}], \texttt{Y}_\rho)} \qquad \text{Y7:}\ \frac{}{\langle [\texttt{emptyE}]\ , \sigma\rangle \rightarrow_{\texttt{Y}} ([\texttt{void}], \texttt{Y}_\rho)}$$

$$\text{Y8:}\ \frac{\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]\,\texttt{t} \qquad \texttt{t} \in \texttt{Simple}}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{mut}]\,\texttt{t}, \texttt{Y}_\rho)} \qquad \text{Y9:}\ \frac{\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]\,\texttt{t} \qquad \texttt{t} \in \texttt{Simple}}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{t}, \texttt{Y}_\rho)} \qquad \text{Y10:}\ \frac{\texttt{Y}_\rho(\texttt{I}) = \texttt{t} \qquad \texttt{t} \in \texttt{Simple}}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{t}, \texttt{Y}_\rho)}$$

$$\text{Y11:}\ \frac{\begin{array}{c}\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]([\texttt{arr}]\,\texttt{t}\,\texttt{N}) \\ \langle \texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} ([\texttt{int}], \texttt{Y}_\rho) \qquad \texttt{t} \in \texttt{Simple} \\ \textit{InBoundDinamycCheck}\end{array}}{\langle \texttt{I}\,[\uparrow]\,\texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{mut}]\,\texttt{t}, \texttt{Y}_\rho)} \qquad \text{Y12:}\ \frac{\begin{array}{c}\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]([\texttt{arr}]\ \texttt{t}\ \texttt{N}) \\ \langle \texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} ([\texttt{int}], \texttt{Y}_\rho) \qquad \texttt{t} \in \texttt{Simple} \\ \textit{InBoundDinamycCheck}\end{array}}{\langle \texttt{I}\,[\uparrow]\,\texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{t}, \texttt{Y}_\rho)}$$

Gestione Errori di Tipo:

$$\text{E8:}\ \frac{\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]\,\texttt{t} \qquad \texttt{t} \notin \texttt{Simple}}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{terr}], \texttt{Y}_\rho)} \qquad \text{E9:}\ \frac{\texttt{Y}_\rho(\texttt{I}) = \texttt{t} \qquad \texttt{t} \neq [\texttt{mut}]\,\texttt{t}'}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{terr}], \texttt{Y}_\rho)} \qquad \text{E10:}\ \frac{\texttt{Y}_\rho(\texttt{I}) \neq [\texttt{mut}]\,\texttt{t}}{\langle [\texttt{val}]\,\texttt{I}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} ([\texttt{terr}], \texttt{Y}_\rho)}$$

$$\text{E11:}\ \frac{\begin{array}{c}\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]([\texttt{arr}]\,\texttt{t}\,\texttt{N}) \\ \langle \texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{t}_\texttt{e}, \texttt{Y}_\rho) \quad \texttt{t}_\texttt{e} \neq [\texttt{int}]\end{array}}{\langle \texttt{I}\,[\uparrow]\,\texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{terr}], \texttt{Y}_\rho)} \qquad \text{E12:}\ \frac{\begin{array}{c}\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]([\texttt{arr}]\,\texttt{t}\,\texttt{N}) \\ \texttt{t} \notin \texttt{Simple}\end{array}}{\langle \texttt{I}\,[\uparrow]\,\texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{DY}} ([\texttt{terr}], \texttt{Y}_\rho)} \qquad \text{E13:}\ \frac{\begin{array}{c}\texttt{Y}_\rho(\texttt{I}) = [\texttt{mut}]([\texttt{arr}]\ \texttt{t}\ \texttt{N}) \\ \langle \texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} ([\texttt{int}], \texttt{Y}_\rho) \\ \textit{OutOfBoundDinamycCheck}\end{array}}{\langle \texttt{I}\,[\uparrow]\,\texttt{e}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} ([\texttt{terr}], \texttt{Y}_\rho)}$$

Notazione

$\rightarrow_{\texttt{DY}}$ è l'inferenza di Tipo del Valore Denotabile di espressioni con doppio significato.

$\texttt{Simple} = \{[\texttt{int}], [\texttt{bool}]\}$

Implementazione: La Funzione di Interpretazione expSem

Dobbiamo introdurre ed implementare in OCaml una funzione che esprima il comportamento del sistema $\mathtt{SEM}_{\mathtt{EXP}}$ definito per la semantica SOS delle r-Espressioni Small20

- Introduciamo una funzione che chiameremo `expSem`.
- `expSem` deve definire una trasformazione che data una espressione e ed uno Stato σ calcola la tripla (Tipo,r-Valore,Stato) prodotto usando le transizioni di $\mathtt{SEM}_{\mathtt{EXP}}$:

$$(\forall \mathtt{e}, \sigma) \quad \mathtt{expSem}(\mathtt{e}, \sigma) = (\mathtt{t_e}, \mathtt{v_e}, \sigma_\mathtt{e}) \quad \mathtt{iff} \quad \langle \mathtt{e}, \sigma \rangle \rightarrow \lfloor \mathtt{t_e}, \mathtt{v_e}, \sigma_\mathtt{e} \rfloor \in \mathtt{Sem}_{\mathtt{EXP}}$$

- `expSem` ha segnatura:

$$\mathtt{expSem} : \mathtt{e} \rightarrow \mathtt{State} \rightarrow \mathtt{Type} * \mathtt{rValue} * \mathtt{State}$$

  La presenza di premesse contenenti $\rightarrow_{\mathtt{EXP}}$ nelle regole di inferenza conduce a definizioni ricorsive della funzione semantica.

- `expSem` associa ad ogni espressione una funzione:

$$\mathtt{State} \rightarrow \mathtt{Type} * \mathtt{rValue} * \mathtt{State}$$

  dove il primo componente dell'immagine è il tipo, associato, dal Sistema Y, alla espressione e i cui identificatori hanno tipo e valori come specificato dallo Stato sorgente.

Implementazione: La Funzione di Interpretazione dexpSem

Dobbiamo introdurre ed implementare in OCaml una funzione che esprima il comportamento del sistema $\mathtt{SEM}_{\mathtt{DEN}}$ definito per la semantica SOS delle l-Espressioni Small20

- Introduciamo una funzione che chiameremo `dexpSem`.
- `dexpSem` definisce una trasformazione che data una espressione e ed uno Stato σ calcola la tripla (Tipo,l-Valore,Stato) prodotto usando le transizioni di $\mathtt{SEM}_{\mathtt{DEXP}}$:

$$(\forall \mathtt{e}, \sigma) \quad \mathtt{dexpSem}(\mathtt{e}, \sigma) = (\mathtt{t_e}, \mathtt{l_e}, \sigma_\mathtt{e}) \quad \mathtt{iff} \quad \langle \mathtt{e}, \sigma \rangle \rightarrow \lfloor \mathtt{t_e}, \mathtt{l_e}, \sigma_\mathtt{e} \rfloor \in \mathtt{Sem}_{\mathtt{DEN}}$$

- `dexpSem` ha segnatura:

$$\mathtt{dexpSem} : \mathtt{e} \rightarrow \mathtt{State} \rightarrow \mathtt{Type} * \mathtt{lValue} * \mathtt{State}$$

  La presenza di premesse contenenti $\rightarrow_{\mathtt{DEN}}$ nelle regole di inferenza conduce a definizioni ricorsive della funzione semantica.

- `dexpSem` associa ad ogni l-espressione una funzione:

$$\mathtt{State} \rightarrow \mathtt{Type} * \mathtt{lValue} * \mathtt{State},$$

  dove il primo componente dell'immagine è il tipo, associato dal Sistema Y, alla espressione e, i cui identificatori hanno tipo e valori come specificato dallo Stato sorgente.

Implementazione Espressioni con Stato di AM20

Esercizio (1)

Regole X1, X2, X3: Completare premessa e termine dx della conclusione
Funzione Semantica expSem: Correggere il Codice e fornire Test Prima Verifica (vedi Listing allegato)

Esercizio (2)

Regola X4, X5, x6: Completare le regole per il calcolo del valore di un identificatore.
Funzioni Semantiche dexpSem/expSem: Estendere il Codice e fornire Test Prima Verifica (vedi Listing allegato)

Esercizio (3)

Regola X7: Completare regola per valore denotabile di un componente array.
Funzione Semantica dexpSem: Estendere Codice e fornire Test Prima Verifica (vedi Listing allegato)

Esercizio (4)

Regola X8: Completare regola per valore di un componente array.
Funzione Semantica expSem: Estendere Codice e fornire Test Prima Verifica (vedi Listing allegato)