Laboratorio 7 - Small20.
Abstract Machine di Small20: Esecutore di Espressioni - II

SOMMARIO: 15 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 ...

- **Computazione** 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 $\mathrm{SEM}_{\mathrm{EXP}}$

Il Sistema di regole $\mathtt{Sem}_{\mathtt{EXP}}$ definisce il comportamento delle r-espressioni durante la computazione dei Programmi Small20 sulla Macchina Astratta AM20.
Controllo dei Tipi Dinamico: Il Sistema $\mathtt{Sem}_{\mathtt{EXP}}$ è Integrato con il Sistema Y (slide7, per le espressioni).

$\mathtt{Exp} ::= \mathtt{Exp}\ [\mathtt{op}]\ \mathtt{Exp} \mid [\mathtt{op}]\ \mathtt{Exp} \mid \mathtt{Exp}\ [=]\ \mathtt{Exp}$

$$\text{X9:}\ \frac{\begin{array}{c}\langle \mathtt{e}_1, \sigma\rangle \to \lfloor \mathtt{t}_1, \mathtt{v}_1, \sigma_1\rfloor \\ \langle \mathtt{e}_2, \sigma\rangle \to \lfloor \mathtt{t}_2, \mathtt{v}_2, \sigma_2\rfloor \\ \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}_1'[\mathtt{x}]\mathtt{t}_2' \\ \mathtt{t}_1' = \mathtt{t}_1 \qquad \mathtt{t}_2' = \mathtt{t}_2 \\ \mathtt{op} \in \mathcal{O}_2 \qquad \overline{\mathtt{op}}(\mathtt{v}_1, \mathtt{v}_2) = \mathtt{v}\end{array}}{\langle \mathtt{e}_1\ [\mathtt{op}]\ \mathtt{e}_2, \sigma\rangle \to \lfloor \mathtt{t}, \mathtt{v}, \sigma 2\rfloor}$$

$$\text{X10:}\ \frac{\begin{array}{c}\langle \mathtt{e}, \sigma\rangle \to \lfloor \mathtt{t}_\mathtt{e}, \mathtt{v}_\mathtt{e}, \sigma_\mathtt{e}\rfloor \\ \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}' \\ \mathtt{t}' = \mathtt{t}_\mathtt{e} \qquad \mathtt{op} \in \mathcal{O}_1 \\ \overline{\mathtt{op}}(\mathtt{v}_\mathtt{e}) = \mathtt{v}\end{array}}{\langle [\mathtt{op}]\ \mathtt{e}, \sigma\rangle \to \lfloor \mathtt{t}, \mathtt{v}, \sigma_\mathtt{e}\rfloor}$$

$$\text{X11:}\ \frac{\begin{array}{c}\langle \mathtt{e}_\mathtt{r}, \sigma\rangle \to \lfloor \mathtt{t}_\mathtt{r}, \mathtt{v}_\mathtt{r}, \sigma_\mathtt{r}\rfloor \\ \langle \mathtt{e}_\mathtt{l}, \sigma_\mathtt{r}\rangle \to_{\mathtt{DEN}} \lfloor \mathtt{t}_\mathtt{l}, \mathtt{loc}_\mathtt{t}, \sigma_\mathtt{l}\rfloor \\ \mathtt{t}_\mathtt{l} = [\mathtt{mut}]\ \mathtt{t} \qquad \mathtt{t} = \mathtt{t}_\mathtt{r} \\ \mathtt{t} \in \mathtt{Simple} \qquad \sigma_\mathtt{l} = (\rho_\mathtt{l}, \mu_\mathtt{l}) \\ \mu_\mathtt{l}[\mathtt{loc}_\mathtt{t} \leftarrow \mathtt{v}_\mathtt{r}] = \mu_\mathtt{F}\end{array}}{\langle \mathtt{e}_\mathtt{l}\ [=]\ \mathtt{e}_\mathtt{r}, \sigma\rangle \to \lfloor \mathtt{t}, \mathtt{v}_\mathtt{r}, (\rho_\mathtt{l}, \mu_\mathtt{F})\rfloor}$$

Notazione e Osservazioni

- $\mathtt{Y}_\rho(\mathtt{op})$ fornisce il tipo di operatori primitivi (forniti dallo RTS della Macchina Astratta che stiamo definendo)
- $\mathcal{O}_2 = \{+, -, *, \mathtt{div}, ==, >, <, \mathtt{or}, \mathtt{and}\}$; $\mathcal{O}_1 = \{-_1, \mathtt{not}\}$.
- $\overline{\mathtt{op}}$ è l'operazione dello RTS con cui implementiamo l'operatore op: Usiamo le corrispondenti op di Ocaml (se definite), implementazioni ad hoc, altrimenti.
- $\mathtt{Simple} = \{[\mathtt{int}], [\mathtt{bool}]\}$
- $=$ valuta gli argomenti da destra a sinistra come in C/C++

Sistema Y: Regole per EXP - Parte2

$$\text{Y13:}\ \frac{\langle \mathtt{e}_1, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_1, \mathtt{Y}_\rho) \quad \langle \mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_2, \mathtt{Y}_\rho) \quad \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}_1'[\mathtt{x}]\mathtt{t}_2' \quad \mathtt{op} \in \mathcal{O}_2 \quad \mathtt{t}_1' = \mathtt{t}_1 \quad \mathtt{t}_2' = \mathtt{t}_2}{\langle \mathtt{e}_1\ [\mathtt{op}]\ \mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

$$\text{Y14:}\ \frac{\langle \mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_e, \mathtt{Y}_\rho) \quad \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}' \quad \mathtt{op} \in \mathcal{O}_1 \quad \mathtt{t}' = \mathtt{t}_e}{\langle [\mathtt{op}]\ \mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

$$\text{Y15:}\ \frac{\langle \mathtt{e}_l, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} (\mathtt{t}_l, \mathtt{Y}_\rho) \quad \langle \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_r, \mathtt{Y}_\rho) \quad \mathtt{t}_l = [\mathtt{Mut}]\ \mathtt{t} \quad \mathtt{t} = \mathtt{t}_r \quad \mathtt{t} \in \mathtt{Simple}}{\langle \mathtt{e}_l\ [=]\ \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Gestione Errori di Tipo:

$$\text{E14:}\ \frac{\langle \mathtt{e}_1, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_1, \mathtt{Y}_\rho) \quad \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}_1'[\mathtt{x}]\mathtt{t}_2' \quad \mathtt{t}_1' \neq \mathtt{t}_1}{\langle \mathtt{e}_1[\mathtt{op}]\mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$\text{E15:}\ \frac{\langle \mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_2, \mathtt{Y}_\rho) \quad \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}_1'[\mathtt{x}]\mathtt{t}_2' \quad \mathtt{t}_2' \neq \mathtt{t}_2}{\langle \mathtt{e}_1[\mathtt{op}]\mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$\text{E16:}\ \frac{\langle \mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_e, \mathtt{Y}_\rho) \quad \mathtt{Y}_\rho(\mathtt{op}) = [\mathtt{abs}]\ \mathtt{t}\ \mathtt{t}' \quad \mathtt{t}' \neq \mathtt{t}_e}{\langle [\mathtt{op}]\mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$\text{E17:}\ \frac{\langle \mathtt{e}_l, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} ([\mathtt{mut}]\ \mathtt{t}, \mathtt{Y}_\rho) \quad \mathtt{t} \notin \mathtt{Simple}}{\langle \mathtt{e}_l\ [=]\ \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$\text{E18:}\ \frac{\langle \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} (\mathtt{t}, \mathtt{Y}_\rho) \quad \mathtt{t} \notin \mathtt{Simple}}{\langle \mathtt{e}_l\ [=]\ \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$\text{E19:}\ \frac{\langle \mathtt{e}_l, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} (\mathtt{t}_l, \mathtt{Y}_\rho) \quad \langle \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_r, \mathtt{Y}_\rho) \quad \mathtt{t}_l = [\mathtt{Mut}]\ \mathtt{t} \quad \mathtt{t} \neq \mathtt{t}_r}{\langle \mathtt{e}_l\ [=]\ \mathtt{e}_r, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Notazione e Osservazioni

- $\mathtt{Y}_\rho(\mathtt{op})$ fornisce il tipo di operatori primitivi (forniti dallo RTS della Macchina Astratta che stiamo definendo)
- $\mathcal{O}_2 = \{+, -, *, \mathtt{div}, ==, >, <, \mathtt{or}, \mathtt{and}\}$; $\mathcal{O}_1 = \{-_1, \mathtt{not}\}$.
- $\overline{\mathtt{op}}$ è l'operazione dello RTS con cui implementiamo l'operatore op.
- $\mathtt{Simple} = \{[\mathtt{int}], [\mathtt{bool}]\}$
- $=$ valuta gli argomenti da destra a sinistra come in C/C++

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)

Regola X9: Scrivere premessa e termine dx della conclusione

Esercizio (2)

Regola X9: L'implementazione della regola X9 può condurre ad introdurre un software di RunTimeSupport che implementa un **dispatcher** *che applicato ad un operatore, per espressioni Small20, fornisce un operatore primitivo di Ocaml, se presente, operante come atteso. Il Listing allegato contiene la realizzazione di un tale dispatcher, nella nuova sezione RunTimeSupport. Lo si commenti per l'uso nel successivo punto sotto.*
Funzione Semantica expSem: Estendere il Codice con l'implementazione di X9 e fornire Test Prima Verifica (vedi Listing allegato)

Esercizio (3)

Regola X11: Completare la regola per l'operatore di assegnamento.
Funzioni Semantiche dexpSem/expSem: Estendere il Codice e fornire Test Prima Verifica (vedi Listing allegato)

Esercizio (4)

Regola X10: Completare regola per espressioni Small20 ad operatore unario.
Funzione Semantica expSem:: Estendere Codice e fornire Test Prima Verifica (vedi Listing allegato)