Variabili ed Espressioni Array per Inizializzazione e Assegnamento di Array in Small21

Riccardo Daluiso

Sommario

Scopo del progetto

- La versione corrente di Small21 contiene gli array come valori allocabili solo staticamente e introducibili solo attraverso dichiarazioni di costanti.

Scopo del progetto

- La versione corrente di Small21 contiene gli array come valori allocabili solo staticamente e introducibili solo attraverso dichiarazioni di costanti.
- Essi sono dunque valori memorizzabili a componenti modificabili.

Scopo del progetto

Gli scopi del presente progetto sono dunque

Scopo del progetto

Gli scopi del presente progetto sono dunque

- Estendere Small21 con variabili di tipo array.
- Estendere Small21 con il costrutto di espressioni array della forma

$$\{\texttt{exp}_1 \ldots \texttt{exp}_n\}$$

Sintassi astratta

Espressioni Array:

Sintassi astratta

Espressioni Array:

```
Exp ::= ...| [arrayExp] ExpS
```

Sintassi astratta

Espressioni Array:

```
Exp ::= ...| [arrayExp] ExpS
  ExpS ::= Exp ExpS | Exp
```

Sintassi astratta

Espressioni Array:

```
Exp ::= ...| [arrayExp] ExpS
ExpS ::= Exp ExpS | Exp
```

Variabili Array:

Sintassi astratta

Espressioni Array:

```
Exp ::= ...| [arrayExp] ExpS
ExpS ::= Exp ExpS | Exp
```

Variabili Array:

```
Dcl ::= ...| [varArr] Type Num Ide Exp
```

Sintassi astratta

Espressioni Array:

```
Exp ::= ...| [arrayExp] ExpS
ExpS ::= Exp ExpS | Exp
```

Variabili Array:

```
Dcl ::= ...| [varArr] Type Num Ide Exp
```

Sintassi concreta

Espressioni Array:

$$\texttt{Exp} \rightarrow \ldots \mid \{\ \texttt{ExpS}\ \}$$
$$\texttt{ExpS} \rightarrow \texttt{Exp}, \texttt{ExpS} \mid \texttt{Exp}$$

Vincoli Contesuali:

Sintassi concreta

Espressioni Array:

$$\texttt{Exp} \rightarrow \ldots \mid \{\ \texttt{ExpS}\ \}$$
$$\texttt{ExpS} \rightarrow \texttt{Exp}, \texttt{ExpS} \mid \texttt{Exp}$$

Vincoli Contesuali:

- Le epressioni nelle espressioni array devono avere tutte lo stesso tipo.

Sintassi concreta

Variabili Array:

```
Type → ...| Simple |...
  Simple → Int | Bool
```

```
Dcl → ...| var Simple [Num] Ide
  | var Simple [Num] Ide = Exp
```

Sintassi concreta

Variabili Array:

$$\texttt{Type} \rightarrow \texttt{...|\ Simple\ |...}$$
$$\texttt{Simple} \rightarrow \texttt{Int\ |\ Bool}$$

$$\texttt{Dcl} \rightarrow \texttt{...|\ var\ Simple\ [Num]\ Ide}$$
$$\texttt{|\ var\ Simple\ [Num]\ Ide\ =\ Exp}$$

Vincoli Contesuali:

Sintassi concreta

Variabili Array:

$$\texttt{Type} \rightarrow \texttt{...|\ Simple\ |...}$$
$$\texttt{Simple} \rightarrow \texttt{Int\ |\ Bool}$$

$$\texttt{Dcl} \rightarrow \texttt{...|\ var\ Simple\ [Num]\ Ide}$$
$$\texttt{|\ var\ Simple\ [Num]\ Ide\ =\ Exp}$$

Vincoli Contesuali:

- Le variabili array possono essere usate come l-termini di assegnamenti aventi come r-termini espressioni array o identificatori.

Espressività

Queste estensioni aumentano l'espressività di Small21?

Espressività

Queste estensioni aumentano l'espressività di Small21?

- Allo stato attuale, gli Array di Small21 non sono valori esprimibili.

Espressività

Queste estensioni aumentano l'espressività di Small21?

- Allo stato attuale, gli Array di Small21 non sono valori esprimibili.
- Con questi nuovi costrutti saranno lecite dichiarazioni della forma `A` = $\{exp_1, \ldots, exp_n\}$

Espressività

Queste estensioni aumentano l'espressività di Small21?

- Allo stato attuale, gli Array di Small21 non sono valori esprimibili.
- Con questi nuovi costrutti saranno lecite dichiarazioni della forma `A` = $\{exp_1, \ldots, exp_n\}$
- Inoltre, sarà possibile vedere le variabili array come delle vere e proprie variabili a cui assegnare altre variabili o costanti array.

Espressività

Queste estensioni aumentano l'espressività di Small21?

- Allo stato attuale, gli Array di Small21 non sono valori esprimibili.
- Con questi nuovi costrutti saranno lecite dichiarazioni della forma `A` $= \{exp_1, \ldots, exp_n\}$
- Inoltre, sarà possibile vedere le variabili array come delle vere e proprie variabili a cui assegnare altre variabili o costanti array.
- Per portare un esempio,una dichiarazione di variabile array B senza inizializzazione, comporterà la possibilità di assegnarla ad un altro array A già inizializzato e operare su B come se stessimo operando su A.

Esempi

```
MyProg0 = {
  var int [3] B = {1;2;3};
  %B è un array inizializzato di 3 interi
}
```

Esempi

```
MyProg1 = {
  int [3] A;
  var int [3] B;
  A[0] = 1;
  A[1] = 2;
  A[2] = 3;
  B = A;
  B[0] = 7; %adesso anche A[0] = 7
}
```

Variabili Array: Sistema Y

Regole per DCL:

Variabili Array: Sistema Y

Regole per DCL:

$$\text{Y2.1}\ \frac{\begin{array}{c}\langle \mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}', \mathtt{Y}_\rho) \\ (\mathtt{t}' = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}') \,\mathtt{o}\, (\mathtt{t}' = [\mathtt{arr}](\mathtt{t}, \mathtt{N}')) \\ \mathtt{Y}_\rho|_{\mathtt{o}}(\mathtt{I}) = \bot \\ \mathtt{N} = \mathtt{N}',\ \mathtt{t} \in \mathtt{Simple}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{Void}], [\mathtt{I}/[\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}) \otimes \mathtt{Y}_\rho)}$$

Variabili Array: Sistema Y

Regole per DCL:

Variabili Array: Sistema Y

Regole per DCL:

$$\text{Y2.2}\ \frac{\begin{array}{c}\langle \mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}', \mathtt{Y}_\rho) \\ \mathtt{t}' = [\mathtt{Unit}] \\ \mathtt{Y}_\rho|_0(\mathtt{I}) = \bot \\ \mathtt{t} \in \mathtt{Simple}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{Void}], [\mathtt{I}/[\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}) \otimes \mathtt{Y}_\rho)}$$

Variabili Array: Sistema Y

Regole per EXP:

$$\text{Y12.1}\ \frac{\begin{array}{c}\mathtt{Y}_\rho(\mathtt{I}) = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})) \\ \mathtt{t} \in \mathtt{Simple}\end{array}}{\langle [\mathtt{val}]\,\mathtt{I}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} ([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})), \mathtt{Y}_\rho)}$$

$$\text{Y13.1}\ \frac{\begin{array}{c}\mathtt{Y}_\rho(\mathtt{I}) = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})) \\ \mathtt{t} \in \mathtt{Simple}\end{array}}{\langle [\mathtt{val}]\,\mathtt{I}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N}), \mathtt{Y}_\rho)}$$

Espressioni Array: Sistema Y

Introduciamo la regola per il nuovo costrutto ArrayExp precedentemente introdotto con la notazione $\{\mathtt{exp}_1 \dots \mathtt{exp}_n\}$:

$$\mathrm{Y19}\ \frac{\begin{array}{c}\langle \mathtt{e}_1, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_1, \mathtt{Y}_\rho) \\ \langle \mathtt{e}_2, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_2, \mathtt{Y}_\rho) \\ \dots \\ \langle \mathtt{e}_\mathtt{N}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_\mathtt{N}, \mathtt{Y}_\rho) \\ \mathtt{t}_1 = \mathtt{t}_2 = \dots = \mathtt{t}_\mathtt{N},\quad \mathtt{t}_1 \in \mathtt{SimpleN} > 0\end{array}}{([\mathtt{ArrayExp}]\{\mathtt{e}_1, \dots, \mathtt{e}_\mathtt{N}\}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} \langle([\mathtt{arr}](\mathtt{t}, \mathtt{N})), \mathtt{Y}_\rho)}$$

Espressioni Array: Sistema Y

Regole per STM

`STM ::= Exp [=] Exp`

Espressioni Array: Sistema Y

Regole per STM

`STM ::= Exp [=] Exp`

Per prima cosa è necessario modificare la regola Y15, che richiede un tipo Simple nell'l-termine di un assegnamento:

$$\text{Y15.1}\ \frac{\begin{array}{c}\langle \mathtt{e_l}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{t_l}, \mathtt{Y}_\rho) \\ \langle \mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t_r}, \mathtt{Y}_\rho) \\ \mathtt{t_l} = [\mathtt{Mut}]\mathtt{t} \\ \mathtt{t} = \mathtt{t_r} \\ \mathtt{t} \in \mathtt{Simple} \cup \{[\mathtt{arr}]\ \mathtt{t}\ \mathtt{N} | \mathtt{t} \in \mathtt{Simple}, \mathtt{N} > 0\}\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{EXP}}$

Aggiorniamo il sistema di regole $\mathrm{SEM}_{\mathrm{EXP}}$ coerentemente agli aggiornamenti sul sistema Y appena studiati

$$\text{X10}\ \frac{\begin{array}{c}\langle \mathtt{e}_1, (\Delta, \mu)\rangle \to \lfloor \mathtt{t}_1, \mathtt{v}_1, (\Delta, \mu_1)\rfloor \\ \langle \mathtt{e}_2, (\Delta, \mu_1)\rangle \to \lfloor \mathtt{t}_2, \mathtt{v}_2, (\Delta, \mu_2)\rfloor \\ \dots \\ \langle \mathtt{e}_N, (\Delta, \mu_{N-1})\rangle \to \lfloor \mathtt{t}_N, \mathtt{v}_N, (\Delta, \mu_N)\rfloor \\ \mathtt{t}_1 = \mathtt{t}_2 = \dots = \mathtt{t}_N, \quad \mathtt{t}_1 \in \mathtt{Simple} \\ \{\mathtt{e}_1, \dots, \mathtt{e}_N\} = \mathtt{e} \\ \{\mathtt{v}_1, \dots, \mathtt{v}_N\} = \mathtt{v}\end{array}}{\langle [\mathtt{ArrayExp}]\ \mathtt{e}, (\Delta, \mu)\rangle \to \lfloor [\mathtt{arr}](\mathtt{t}, \mathtt{N}), \mathtt{v}, (\Delta, \mu_N)\rfloor}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{EXP}}$

Ide di variabili array come l-termini

$$\text{X4.1}\ \frac{\begin{array}{c}\sigma = (\Delta, \mu) \\ \Delta(\mathtt{I}) = ([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})), \mathtt{loc}'_{\mathtt{t}}) \\ \mathtt{t} = [\mathtt{mut}]\mathtt{t}' \\ \mathtt{t}' \in \mathtt{Simple}\end{array}}{\langle [\mathtt{val}]\,\mathtt{I}, \sigma\rangle \rightarrow_{\mathtt{Den}} \lfloor([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})), \mathtt{loc}'_{\mathtt{t}}, \sigma\rfloor}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{EXP}}$

Ide di variabili array come r-termini

$$\text{X5.1}\ \frac{\begin{array}{c}\sigma = (\Delta, \mu) \\ \Delta(\mathtt{I}) = ([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})), \mathtt{loc}'_{\mathtt{t}}) \\ \mathtt{t} = [\mathtt{mut}]\mathtt{t}' \\ \mathtt{t}' \in \mathtt{Simple}\end{array}}{\langle [\mathtt{val}]\,\mathtt{I}, \sigma\rangle \rightarrow \lfloor([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})), \mathtt{loc}'_{\mathtt{t}}, \sigma\rfloor}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{DLC}}$

Dichiarazione di variabili array

$$\text{D2.1}\ \frac{\begin{array}{c}\langle \mathtt{e}, (\Delta, \mu)\rangle \to \lfloor \mathtt{t_e}, \mathtt{v}, \Delta, \mu_\mathtt{e}\rfloor \\ \mathtt{t_e} = [\mathtt{Unit}] \quad \Delta|_0(\mathtt{I}) = \bot \\ \triangleright(\mu_\mathtt{e}, \mathtt{t}, 1) = (\mathtt{loc_t}, \mu_\mathtt{a}) \\ [\mathtt{I}/[\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}))] \otimes \Delta = \Delta_\mathtt{I}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, (\Delta, \mu)\rangle \to ([\mathtt{Void}], (\Delta_\mathtt{I}, \mu_\mathtt{a}))}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{DLC}}$

$$\mathrm{D2.2}\ \frac{\begin{array}{c}\langle \mathtt{e}, (\Delta, \mu)\rangle \to \lfloor \mathtt{t_e}, \mathtt{v}, \Delta, \mu_\mathtt{e}\rfloor \\ \mathtt{t_e} = [\mathtt{arr}](\mathtt{t},\ \mathtt{N}) \quad \Delta|_0(\mathtt{I}) = \bot \\ \triangleright(\mu_\mathtt{e}, \mathtt{t}, \mathtt{N}) = (\mathtt{loc_t}, \mu_\mathtt{a}) \\ [\mathtt{I}/([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})), \mathtt{loc_t})] \otimes \Delta = \Delta_\mathtt{I}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, (\Delta, \mu)\rangle \to ([\mathtt{Void}], (\Delta_\mathtt{I}, \mu_\mathtt{a}))}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{DLC}}$

$$\text{D2.3}\ \frac{\begin{array}{c}\mathtt{e} = \mathtt{ValI}_2 \\ \langle \mathtt{e}, (\Delta, \mu)\rangle \to \lfloor \mathtt{t_e}, \mathtt{v}, \Delta, \mu_\mathtt{e} \rfloor \\ \mathtt{t_e} = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})\ \ \mathtt{o}\ \ \mathtt{t_e} = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t},\ \mathtt{N})) \\ \Delta(\mathtt{I}_2) = (\mathtt{t_e}, \mathtt{loc}_2) \\ \Delta|_0(\mathtt{I}) = \bot \\ [\mathtt{I}/([\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})), \mathtt{loc}_2)] \otimes \Delta = \Delta_\mathtt{I}\end{array}}{\langle [\mathtt{varArr}]\ \mathtt{t}\ [\mathtt{N}]\ \mathtt{I}\ \mathtt{e}, (\Delta, \mu)\rangle \to ([\mathtt{Void}], (\Delta_\mathtt{I}, \mu_\mathtt{a}))}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$

`STM ::= Exp [=] Exp`

Primo caso, l'r-termine è una variabile o costante Array, e dunque associata a un identificatore:

$$\text{S1.1}\ \frac{\begin{array}{c}\mathtt{e_l} = \mathtt{Val\ I} \\ \mathtt{e_r} = \mathtt{Val\ I_2} \\ \langle \mathtt{e_r}, \sigma \rangle \rightarrow \lfloor \mathtt{t_r}, \mathtt{v_r}, \sigma_\mathtt{r} \rfloor \\ \langle \mathtt{e_l}, \sigma_\mathtt{r} \rangle \rightarrow_{\mathtt{DEN}} \lfloor \mathtt{t_l}, \mathtt{loc_l}, (\Delta_\mathtt{l}, \mu_\mathtt{l}) \rfloor \\ \mathtt{t_l} = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})) \\ \mathtt{t_r} = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t}', \mathtt{N}')\ \mathtt{o}\ \mathtt{t_r} = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}', \mathtt{N}')) \\ \mathtt{t} = \mathtt{t}'\ \ \mathtt{N} = \mathtt{N}'\ \ \mathtt{t} \in \mathtt{Simple} \\ \Delta(\mathtt{I_2}) = (\mathtt{t_r}, \mathtt{loc_2}) \\ (\mathtt{I}/[\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})), \mathtt{loc_2}) \otimes \Delta_\mathtt{l} = \Delta_\mathtt{I}\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, (\Delta, \mu) \rangle \rightarrow ([\mathtt{Void}], (\Delta_\mathtt{I}, \mu_\mathtt{N}))}$$

Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$

`STM ::= Exp [=] Exp`

Secondo caso, l'r-termine è un'espressione Array:

$$\mathrm{S1.2}\ \frac{\begin{array}{c}\langle \mathtt{e_r}, \sigma\rangle \to \lfloor \mathtt{t_r}, \mathtt{v_r}, \sigma_\mathtt{r}\rfloor \\ \langle \mathtt{e_l}, \sigma_\mathtt{r}\rangle \to_{\mathtt{DEN}} \lfloor \mathtt{t_l}, \mathtt{loc_l}, (\Delta_\mathtt{l}, \mu_\mathtt{l})\rfloor \\ \mathtt{t_l} = [\mathtt{mut}]([\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N})) \\ \mathtt{t_r} = [\mathtt{arr}](\mathtt{t}', \mathtt{N}') \\ \mathtt{t} = \mathtt{t}' \quad \mathtt{N} = \mathtt{N}' \quad \mathtt{t} \in \mathtt{Simple} \\ \mathtt{v_r} = \{\mathtt{v_1}, \ldots, \mathtt{v_N}\} \\ \triangleright(\mu_\mathtt{l}, \mathtt{N}) = (\mathtt{loc_a}, \mu_\mathtt{a}) \\ \mu_\mathtt{a}[\mathtt{loc_a} \leftarrow \mathtt{v_1}] = \mu_1 \\ \ldots \\ \mu_{\mathtt{N}-1}[\mathtt{loc_l} \oplus \mathtt{N} \leftarrow \mathtt{v_N}] = \mu_\mathtt{N}\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, (\Delta, \mu)\rangle \to ([\mathtt{Void}], (\Delta_\mathtt{I}, \mu_\mathtt{N}))}$$

Gestione errori di tipo

A questo punto è necessario studiare i nuovi errori che possono presentarsi dopo l'introduzione dei due nuovi costrutti.

Gestione errori di tipo per Variabili Array

- Se $t \notin$ Simple, viene restituito errore:

Gestione errori di tipo per Variabili Array

- Se $t \notin$ Simple, viene restituito errore:

$$\text{E4.1}\ \frac{\begin{array}{c}\langle \mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}', \mathtt{Y}_\rho) \\ \mathtt{t}' = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}') \\ \mathtt{Y}_\rho|_{\mathtt{O}}(\mathtt{I}) = \bot \\ \mathtt{t} \notin \mathtt{Simple}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di tipo per Variabili Array

- Se i tipi non coincidono, viene restituito errore:

Gestione errori di tipo per Variabili Array

- Se i tipi non coincidono, viene restituito errore:

$$\text{E5.1}\ \frac{\begin{array}{c}\langle \mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}', \mathtt{Y}_\rho) \\ \mathtt{t}' = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t_r}, \mathtt{N}') \\ \mathtt{Y}_\rho|_{\mathtt{O}}(\mathtt{I}) = \bot \\ \mathtt{t} \neq \mathtt{t_r}\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di tipo per Variabili Array

- Se l'identificatore è gia stato usato, viene restituito errore:

Gestione errori di tipo per Variabili Array

- Se l'identificatore è gia stato usato, viene restituito errore:

$$\text{E6.1}\ \frac{\mathtt{Y}_\rho|_0(\mathtt{I}) \neq \bot}{\langle [\mathtt{varArr}]\ \mathtt{t}\ [\mathtt{N}]\ \mathtt{I}\ \mathtt{e}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di tipo per Variabili Array

- Se le dimensioni non coincidono, viene restituito errore:

Gestione errori di tipo per Variabili Array

- Se le dimensioni non coincidono, viene restituito errore:

$$\text{E7.1}\ \frac{\begin{array}{c}\langle \mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}', \mathtt{Y}_\rho) \\ \mathtt{t}' = [\mathtt{arr}]([\mathtt{mut}]\mathtt{t}, \mathtt{N}') \\ \mathtt{Y}_\rho|_{\mathtt{O}}(\mathtt{I}) = \bot \\ \mathtt{N} \neq \mathtt{N}'\end{array}}{\langle [\mathtt{varArr}]\,\mathtt{t}\,[\mathtt{N}]\,\mathtt{I}\,\mathtt{e}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di Tipo per Espressioni Array

Vediamo gli errori per le espressioni Array.

Gestione errori di Tipo per Espressioni Array

Vediamo gli errori per le espressioni Array.

- Se $t_1 \notin$ Simple, viene restituito errore:

Gestione errori di Tipo per Espressioni Array

Vediamo gli errori per le espressioni Array.

- Se $t_1 \notin$ Simple, viene restituito errore:

$$\text{E28.1}\ \frac{\begin{array}{c}\langle \mathtt{e}_1, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_1, \mathtt{Y}_\rho) \\ \langle \mathtt{e}_2, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_2, \mathtt{Y}_\rho) \\ \ldots \\ \langle \mathtt{e}_N, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_N, \mathtt{Y}_\rho) \\ \mathtt{t}_1 \notin \mathtt{Simple} \\ \{\mathtt{e}_1, \ldots, \mathtt{e}_N\} = \mathtt{e}\end{array}}{\langle [\mathtt{ArrayExp}]\ \mathtt{e}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} \langle ([\mathtt{terr}], \mathtt{Y}_\rho\rangle}$$

Gestione errori di Tipo per Espressioni Array

- Se esistono due tipi non coincidenti nella lista, viene restituito errore:

Gestione errori di Tipo per Espressioni Array

- Se esistono due tipi non coincidenti nella lista, viene restituito errore:

$$\text{E29.1}\ \frac{\begin{array}{c}\langle \mathtt{e}_1, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_1, \mathtt{Y}_\rho) \\ \langle \mathtt{e}_2, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_2, \mathtt{Y}_\rho) \\ \ldots \\ \langle \mathtt{e}_N, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}_N, \mathtt{Y}_\rho) \\ \exists\ \mathtt{i} > 1\ \mathtt{t.c}\ \mathtt{t}_\mathtt{i} \neq \mathtt{t}_1 \\ \{\mathtt{e}_1, \ldots, \mathtt{e}_N\} = \mathtt{e}\end{array}}{\langle [\mathtt{ArrayExp}]\ \mathtt{e}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} \langle ([\mathtt{terr}], \mathtt{Y}_\rho\rangle}$$

Gestione errori di Tipo per Espressioni Array

- Se la lista è vuota, viene restituito errore:

Gestione errori di Tipo per Espressioni Array

- Se la lista è vuota, viene restituito errore:

$$\text{E30.1}\ \frac{\{\mathtt{e_1}, \ldots, \mathtt{e_n}\} = \{\}}{\langle [\mathtt{ArrayExp}]\{\}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} \langle ([\mathtt{terr}], \mathtt{Y}_\rho \rangle}$$

Gestione errori di tipo per Variabili ed Espressioni Array

- Se le dimensioni non coincidono, viene restituito errore:

Gestione errori di tipo per Variabili ed Espressioni Array

- Se le dimensioni non coincidono, viene restituito errore:

$$\text{E17.1}\ \frac{\begin{array}{c}\langle \mathtt{e_l}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{t_l}, \mathtt{Y}_\rho) \\ \langle \mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t_r}, \mathtt{Y}_\rho) \\ \mathtt{t_l} = [\mathtt{Mut}]([\mathtt{arr}]([[\mathtt{Mut}]\mathtt{t}, \mathtt{N}])), \mathtt{t} \in \mathtt{Simple} \\ \mathtt{t_r} = [\mathtt{arr}](\mathtt{t}', \mathtt{N}') \\ \mathtt{t} = \mathtt{t}',\ \mathtt{N} \neq \mathtt{N}'\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di tipo per Variabili ed Espressioni Array

- Se i tipi degli array non coincidono, viene restituito errore:

Gestione errori di tipo per Variabili ed Espressioni Array

- Se i tipi degli array non coincidono, viene restituito errore:

$$\text{E18.1}\ \frac{\begin{array}{c}\langle \mathtt{e_l}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{t_l}, \mathtt{Y}_\rho) \\ \langle \mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t_r}, \mathtt{Y}_\rho) \\ \mathtt{t_l} = [\mathtt{Mut}]([\mathtt{arr}]([[\mathtt{Mut}]\mathtt{t}, \mathtt{N}])), \mathtt{t} \in \mathtt{Simple} \\ \mathtt{t_r} = [\mathtt{arr}](\mathtt{t}', \mathtt{N}') \\ \mathtt{t} \neq \mathtt{t}'\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Gestione errori di tipo per Variabili ed Espressioni Array

- Se l'r-termine non è di tipo array, viene restituito errore:

Gestione errori di tipo per Variabili ed Espressioni Array

- Se l'r-termine non è di tipo array, viene restituito errore:

$$\text{E19.1}\ \frac{\begin{array}{c}\langle \mathtt{e_l}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{t_l}, \mathtt{Y}_\rho) \\ \langle \mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t_r}, \mathtt{Y}_\rho) \\ \mathtt{t_l} = [\mathtt{Mut}]([\mathtt{arr}]([[\mathtt{Mut}]\mathtt{t}, \mathtt{N}]), \mathtt{t} \in \mathtt{Simple} \\ \mathtt{t_r} \neq [\mathtt{arr}](\mathtt{t}', \mathtt{N}')\end{array}}{\langle \mathtt{e_l}[=]\mathtt{e_r}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Implementazione

Per prima cosa aggiungiamo la nuova dichiarazione di variabile Array e le nuove espressioni Array:

Implementazione

Per prima cosa aggiungiamo la nuova dichiarazione di variabile Array e le nuove espressioni Array:

```
type dcl = ...
| VarArr of tye * num * ide * exp
...
and

exp = ...
| ArrayExp of expseq
...
and

expseq = exp list
...
```

Implementazione

Dato che stiamo operando con liste, possiamo sfruttare la sintassi di OCaml per effettuare la ricorsione dentro esse.
In particolare, presentiamo una funzione che ci risulterà utile nel codice; si tratta della funzione `List.fold_left`, di cui riportiamo il codice OCaml per completezza :

Implementazione

Dato che stiamo operando con liste, possiamo sfruttare la sintassi di OCaml per effettuare la ricorsione dentro esse.
In particolare, presentiamo una funzione che ci risulterà utile nel codice; si tratta della funzione `List.fold_left`, di cui riportiamo il codice OCaml per completezza :

```
let rec fold_left op acc = function
| [] -> acc
| h::t -> fold_left op (op acc h) t
```

Implementazione:DCLSEM

Passiamo all'aggiornamento della funzione dclSem con le nuove dichiarazioni di variabile Array. Il primo caso è quello in cui l'espressione sia un identificatore associato ad una costante array:

Implementazione:DCLSEM

Passiamo all'aggiornamento della funzione dclSem con le nuove dichiarazioni di variabile Array. Il primo caso è quello in cui l'espressione sia un identificatore associato ad una costante array:

```
let rec dclSem dcl (sk,(Store(d,g)as mu)) =
  match dcl with
  |VarArr(ty,num,ide,exp) when (isSimple ty) &&
                                  not(declared sk ide)
    ->(match exp with
          |Val ide2 ->
                (match expSem exp (sk,mu) with
                   |( (Arr(Mut te,num2),_,(ske,mue)) | (Mut(Arr(Mut
                   te,num2)),_,(ske,mue)) )
                        when (ysame ty te) &&
                                 (num = num2)
                        -> (match getS ske ide2 with
                               |DArry(_,loca) ->
                                    (let den = DArry (Mut(Arr(Mut ty,
                                    num)),loca) in
                                        let sk2 = bindS ske ide den in
                                        let st2 = (sk2,mue) in
                                        (Void,st2))
                               |_-> raise(TypeErrorI("Not array
                                      Binding",ide2)))
```

Implementazione:DCLSEM

```
|( (Arr(Mut te,num2),_,_) | (Mut(Arr(Mut te,num2)),_,_) )
          when (ysame ty te)
            -> raise(TypeErrorI("E7.1: dclSem-",ide))
|( (Arr(Mut te,num2),_,_) | (Mut(Arr(Mut te,num2)),_,_) )
    -> raise(TypeErrorI("E5.1: dclSem-",ide))
| _ -> raise(TypeErrorI("Binding not allowed",ide)))
```

Implementazione:DCLSEM

Secondo caso, l'espressione è un'espressione array:

Implementazione:DCLSEM

Secondo caso, l'espressione è un'espressione array:

```
|ArrayExp exps ->
        (let (tr,v,(ske,mue)) = expSem exp (sk,mu) in
                    match tr with
                    |Arr(te,num2) when (ysame ty te) && (num = num2)
                            -> (match v with
                                    |Listval listvalE ->
                                      (let (loca,mua) = allocate mue num in
                                       let muF = List.fold_left2 upd mua listloc listvalM in
                                       let den = DArry (Mut(Arr(Mut ty,num)),loca) in
                                       let sk2 = bindS ske ide den in
                                       let st2 = (sk2,muF) in
                                       (Void,st2))
                            |_ -> raise(SystemErrorE("expSem:-")))
                    | Arr(te,num2) when (ysame ty te) ->
                        (raise(TypeErrorI("E7.1: dclSem -",ide)))
                    | Arr(te,num2) -> (raise(TypeErrorI("E5.1: dclSem-",ide)))
                    |_ -> (raise(SystemErrorE("expSem:-")))
```

Implementazione:DCLSEM

Terzo ed ultimo caso, l'epressione è EE (Empty Expression). Stiamo dunque dichiarando una variabile array senza inizializzazione:

Implementazione:DCLSEM

Terzo ed ultimo caso, l'epressione è EE (Empty Expression). Stiamo dunque dichiarando una variabile array senza inizializzazione:

```
| EE ->
    (let (loca,muF) = allocate mu 1 in
    let den = DArry(Mut(Arr(Mut ty,num)),loca) in
    let skF = bindS sk ide den in
    (Void,(skF,muF)))
|_ -> (raise(TypeErrorE("expSem: Binding not allowed-",exp))))
```

Implementazione:DCLSEM

Terzo ed ultimo caso, l'epressione è EE (Empty Expression). Stiamo dunque dichiarando una variabile array senza inizializzazione:

```
| EE ->
    (let (loca,muF) = allocate mu 1 in
    let den = DArry(Mut(Arr(Mut ty,num)),loca) in
    let skF = bindS sk ide den in
    (Void,(skF,muF)))
|_ -> (raise(TypeErrorE("expSem: Binding not allowed-",exp))))
```

I successivi casi riportano errori di tipo:

Implementazione:DCLSEM

Terzo ed ultimo caso, l'epressione è EE (Empty Expression). Stiamo dunque dichiarando una variabile array senza inizializzazione:

```
| EE ->
    (let (loca,muF) = allocate mu 1 in
    let den = DArry(Mut(Arr(Mut ty,num)),loca) in
    let skF = bindS sk ide den in
    (Void,(skF,muF)))
|_ -> (raise(TypeErrorE("expSem: Binding not allowed-",exp))))
```

I successivi casi riportano errori di tipo:

```
|VarArr(ty,num,ide,exp) when (isSimple ty)  ->
    raise(TypeErrorI("E5.1: dclSem - ",ide))
|VarArr(ty,num,ide,exp) -> raise(TypeErrorI("E4.1: dclSem - ", ide))
```

Implementazione: EXPSEM

```
let expSem exp (sk,(Store(d,g)as mu)) =
  match exp with
   |ArrayExp expsq when (List.length expsq > 0) ->
      (let (t1,_, _) = expSem (nth expsq 0) (sk,mu) in
       match t1 with
       |_ when isSimple t1 ->
           (let evaluate (evlist, (sk,mu)) exp =
               (let (t,v,(ske,mue)) = expSem exp (sk,mu) in
               (match (t,v,(ske,mue)) with
                 |_ when ysame t t1 ->  (evlist@[v],(ske,mue))
                 |_ -> raise(TypeErrorE("E29.1: expSem -",ArrayExp expsq)))) in
              let (valuelist, (skF,muF)) = List.fold_left evaluate ([], (sk,mu)) expsq in
              (Arr(t1, List.length expsq),Listval valuelist,(skF,muF)))
              |_ -> raise(TypeErrorE("E28.1: expSem-",ArrayExp expsq)))
       |ArrayExp expsq -> raise(TypeErrorE("E30.1:emptyList",ArrayExp expsq))
```

Implementazione: EXPSEM

```
let expSem exp (sk,(Store(d,g)as mu)) =
  match exp with
   |ArrayExp expsq when (List.length expsq > 0) ->
      (let (t1,_, _) = expSem (nth expsq 0) (sk,mu) in
        match t1 with
        |_ when isSimple t1 ->
            (let evaluate (evlist, (sk,mu)) exp =
                (let (t,v,(ske,mue)) = expSem exp (sk,mu) in
                (match (t,v,(ske,mue)) with
                  |_ when ysame t t1 ->  (evlist@[v],(ske,mue))
                  |_ -> raise(TypeErrorE("E29.1: expSem -",ArrayExp expsq)))) in
               let (valuelist, (skF,muF)) = List.fold_left evaluate ([], (sk,mu)) expsq in
               (Arr(t1, List.length expsq),Listval valuelist,(skF,muF)))
               |_ -> raise(TypeErrorE("E28.1: expSem-",ArrayExp expsq)))
        |ArrayExp expsq -> raise(TypeErrorE("E30.1:emptyList",ArrayExp expsq))
```

Osserviamo che il `Listval` è stato aggiunto ai valori `Aval` come tipo atomico calcolabile da un'espressione. In questo modo possiamo restituire una lista di valori, corrispondente alla lista di valutazioni delle espressioni presenti nell'espressione Array.

Implementazione: DEXPSEM

```
dexpSem dexp (sk,(Store(d,g)as mu)) =
        match dexp with
          | Val ide
            ->(match getS sk ide with
            | DVar(Mut t,loct) when isSimple t
                        -> (Mut t,loct,(sk,mu))
               | DVar(Mut t,loct)
                        -> raise(TypeErrorE("E16: dexpSem - ",dexp))
               | DArry(Mut(Arr(Mut t,n)),loca) when (isSimple t)
                        -> (Mut(Arr(Mut t,n)), loca,(sk,mu))
               | DArry(Mut(Arr(Mut t,n)),loca)
                        -> raise(TypeErrorE("E9: dexpSem - ",dexp))
               | _
                        -> raise(TypeErrorE("E17: dexpSem - ",dexp)))
```

Implementazione: STMSEM

Aggiorniamo lo statement Update in accordo alle regole della semantica:

Implementazione: STMSEM

Aggiorniamo lo statement Update in accordo alle regole della semantica:

```
let stmSem stm (sk,(Store(d,g)as mu)) =
    | Upd(dexp,exp)
        ->(let (tr,vr,sgr) = expSem exp (sk,mu) in
           match dexpSem dexp sgr with
           | (Mut(Arr(Mut t,num)),loct,(skl,mul))
               ->(match dexp with
                  |Val ide ->
                   ->(match tr with
                         | ( Arr(Mut te,num2) | Mut(Arr(Mut te,num2)) )
                                    when (ysame t te) && (num=num2)
                          ->(match exp with
                                |Val ide2
                                       ->(match getS skl ide2 with
                                       |DArry(_,loca)
                                       -> (let sk2 = changeLocS skl ide loca in
                                          let st2 = (sk2,mul) in
                                          (Void,st2))
                                       |_-> raise(SystemError("Assignment not allowed")))
                              |_-> raise(SystemError("Assignment not allowed")))
```

Implementazione: STMSEM

```
    | ( Arr(Mut te,num2) | Mut(Arr(Mut te,num2)) )
                    when (ysame t te
      -> raise(TypeErrorE("E17.1:expSem",dexp))
    | ( Arr(Mut te,num2) | Mut(Arr(Mut te,num2)) )
      -> raise(TypeErrorE("E18.1:expSem",dexp))
    | ( Arr(te,num2)) when (ysame t te) && (num=num2)
          -> (match vr with
           |Listval listvalE
              -> (let (loca,mua) = allocate mul num in
                  let listloc = toLISTloc loca (Loc ((locTOint  loca)+num-1))  in
                  let listvalM = List.map eTOm listvalE in
                  let muF = List.fold_left2 upd mua listloc listvalM in
                          let sk2 = changeLocS skl ide loca in
                          (Void,(sk2,muF)))
            |_ -> raise(SystemErrorE("expSem:-")))
    | (Arr(te,num2)) when (ysame t te) -> raise(TypeErrorE("E17.1: expSem",exp))
    | (Arr(te,num2)) -> raise(TypeErrorE("E18.1: expSem",exp))
    |_-> raise(TypeErrorE("Is an array?",exp)))
|_-> raise(SystemError("Assignment not allowed"))
...
```

Riportiamo il codice di due funzioni utilizzate nello statement Update; il loro scopo è quello di cambiare la locazione associata ad un identificatore già dichiarato nell'AR corrente:

Riportiamo il codice di due funzioni utilizzate nello statement Update; il loro scopo è quello di cambiare la locazione associata ad un identificatore già dichiarato nell'AR corrente:

```
let changeLocEnv (Env(l,r)) ide loc =
        if mem ide l then
            (let den = getEnv (Env(l,r)) ide in
            match den with
                | DVar (t,loc2)
                -> (Env(l, fun i -> if i = ide then DVar(t,loc) else r i))
                | DArry (t,loc2)-> (Env(l, fun i -> if i = ide then DArry(t,loc) else r i))
                |_ -> (raise(NoLocFound("changeLocEnv", ide))))
                else raise (UnboundIde("changeLoc",ide))
;;


let changeLocS ((Stack ars)as sk) ide loc =
        match ars with
                | AR(h,s,env,c,v)::rest
                    when (defined env ide) -> ( let env2 = changeLocEnv env ide loc in
                    Stack (AR(h,s,env2,c,v)::rest) )
                | _ -> raise (NoLocFound ("noLoc",ide))
;;
```

Per ottenere un gradevole risultato output, rendiamo la funzione toStringExp compatbile con il nuovo tipo Espressione Array:

```
toStringExp = (function
...
    | ArrayExp expsq
        -> (let rec rangexp n m list =
        if n = m - 1 then toStringExp (nth list n)
        else toStringExp (nth list n)^","^(rangexp (n+1) m list) in
        "{" ^(rangexp 0 (List.length expsq) expsq  )^"}")
```

Eseguiamo alcuni programmi

```
(*primo blocco dichiarazioni*)
let d1 = Array(Arr(Int,3), "A");;
let d2 = VarArr(Int,3,"B",EE);;
let d3 = SeqD(a1,a2);;


(*secondo blocco statement*)
let s1 = Upd( Arrow1("A", N 0), N 1 ) ;;
let s2 = Upd( Arrow1("A", N 1), N 2 ) ;;
let s3 = Upd( Arrow1("A", N 2), N 3 ) ;;
let s4 = Upd(Val "B", Val "A");;


let bl1 = SeqS(s1,SeqS(s2,SeqS(s3,s4)));;

let bl = UnlC(bl1);;

let prog1 = Prog("myProg1",Block(d3,bl));;
```

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg1`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg1`

```
Program myProg1 {
    int[3] A;
    var int[3] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = A;
}
```

eseguiamo `progSem myProg1`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg1`

```
Program myProg1 {
    int[3] A;
    var int[3] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = A;
}
```

eseguiamo `progSem myProg1`

```
>Stack: {uno,0,[B/(M:Mint[3],L3); A/(:Mint[3],L0)],:cmdNext:,[N]}
 Store: [L0<-Undef,L1<-Undef,L2<-Undef,L3<-Undef]

>Stack: {uno,0,[B/(M:Mint[3],L0); A/(:Mint[3],L0)],:cmdNext:,[N]}
 Store: {L0<-1,L1<-2,L2<-3,L3<-Undef}
```

Eseguiamo alcuni programmi

- Osserviamo come la locazione inizialmente riservata a B rimanga inutilizzata dopo l'assegnamento.

Eseguiamo alcuni programmi

- Osserviamo come la locazione inizialmente riservata a B rimanga inutilizzata dopo l'assegnamento.
- In effetti, Small21 nella versione attuale non permette di deallocare memoria.

Eseguiamo alcuni programmi

```
(*primo blocco dichiarazioni *)
let a1 = Array(Arr(Int,3), "A");;
let a2 = VarArr(Int,3,"B",EE);;
let a3 = SeqD(a1,a2);;


(*secondo blocco statement*)
let b1 = Upd( Arrow1("A", N 0), N 1 ) ;;
let b2 = Upd( Arrow1("A", N 1), N 2 ) ;;
let b3 = Upd( Arrow1("A", N 2), N 3 ) ;;
let c1 = Upd(Val "B", ArrayExp [Arrow1("A",N 0);
              Arrow1("A",N 1); Arrow1("A",N 2)]);;

let bl = UnlC(SeqS(b1,SeqS(b2,SeqS(b3,c1))));;

let prog2 = Prog("myProg2",Block(a3,bl));;
```

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg2`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg2`

```
Program myProg2{
    int[3] A;
    var int[3] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = {A[0],A[1],A[2]};
}
```

eseguiamo `progSem myProg2`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg2`

```
Program myProg2{
    int[3] A;
    var int[3] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = {A[0],A[1],A[2]};
}
```

eseguiamo `progSem myProg2`

```
>Stack:
 {uno,0,[B/(M:Mint[3],L3); A/(:Mint[3],L0)],:cmdNext:,[N]}
 Store:
 [L0<-Undef,L1<-Undef,L2<-Undef,L3<-Undef]

>Stack: {uno,0,[B/(M:Mint[3],L4);  A/(:Mint[3],L0)],:cmdNext:,[N]}
 Store: [L0<-1,L1<-2,L2<-3,L3<-Undef,L4<-1,L5<-2,L6<-3]
```

Eseguiamo alcuni programmi

- Come previsto, il comportamento dei due programmi è completamente differente.

Eseguiamo alcuni programmi

- Come previsto, il comportamento dei due programmi è completamente differente.
- Nel primo caso, l'identificatore B è legato alla locazione del primo elemento di A, e dunque fino a inzializzazione successiva i due array sono inidistinguibili.

Eseguiamo alcuni programmi

- Come previsto, il comportamento dei due programmi è completamente differente.
- Nel primo caso, l'identificatore B è legato alla locazione del primo elemento di A, e dunque fino a inzializzazione successiva i due array sono inidistinguibili.
- Nel secondo caso, i due array sono completamente distinti, e le operazioni su uno non influenzano in alcun modo quelle sull'altro.

Eseguiamo alcuni programmi

Ovviamente, possiamo inzializzare B già durante la sua dichiarazione:

```
(*primo blocco dichiarazioni *)
let a1 = VarArr(Bool,3,"B",ArrayExp[B True; B True; B False]);;
let prog2 = Prog("myProg3",Block(a1,UnlC(ES)));;
```

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg3`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg3`

```
Program myProg3{
    var bool[3] B = {true,true,false};
}
```

eseguiamo `progSem myProg3`

Eseguiamo alcuni programmi

eseguiamo `toStringProg myProg3`

```
Program myProg3{
    var bool[3] B = {true,true,false};
}
```

eseguiamo `progSem myProg3`

```
>Stack: {myProg3,0,[B/(M:Mbool[3],L0)],:cmdNext:,[N]}
 Store: [L0<-true,L1<-true,L2<-false]
```

Gestione degli errori: Implementazione

- Vediamo adesso se gli errori presentati nelle regole del sistema Y sono stati implementati correttamente;

Gestione degli errori: Implementazione

- Vediamo adesso se gli errori presentati nelle regole del sistema Y sono stati implementati correttamente;
- Per brevità, omettiamo adesso il codice in sintassi astratta dei programmi.

Gestione degli errori: Implementazione

```
Program myProgErr1{
    var int[3] B = {true,true,false};
}

Exception: TypeErrorI ("E5.1: dclSem", "B").
```

Gestione degli errori: Implementazione

```
Program myProgErr2{
    int[3] A;
    var int[4] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = A;
}
Exception: TypeErrorS ("E17.1: stmSem", Upd(Val "B", Val "A")).
```

Gestione degli errori: Implementazione

```
Program myProgErr3{
    int[3] A;
    var bool[3] B;
    A[0] = 1;
    A[1] = 2;
    A[2] = 3;
    B = A;
}
Exception: TypeErrorS ("E18.1: stmSem", Upd(Val "B",Val "A")).
```

Gestione degli errori: Implementazione

Vediamo adesso cosa succede se proviamo ad assegnare a B una variabile di tipo int:

```
Program myProgErr4{
    int A = 3;
    var int[3] B;
    B = A;
}
Exception: TypeErrorE ("E19.1:stmSem", Upd(Val "B", Val "A")).
```

Gestione degli errori: Implementazione

Testiamo l'errore dovuto alla presenza di tipi diversi nell'espressione array:

```
Program myProgErr4{
    var int[3] B = {1; true; 3};
}
Exception: TypeErrorE ("E29.1: expSem", ArrayExp[N 1; B True; N 3]).
```

Gestione degli errori: Implementazione

Per ultimo, testiamo l'errore dovuto alla inizializzazione con lista vuota:

```
Program myProgErr4{
    var int[3] B = {};
}
Exception: TypeErrorE ("E30.1: EmptyList -", ArrayExp[]).
```

Grazie per l'attenzione