Esercizio 1

Data la seguente BNF relativa alla descrizione di numeri in base cinque:

pentanum → pentanum **0** | pentanum **1** | pentanum **2** | pentanum **3** | pentanum **4** | **0** | **1** | **2** | **3** | **4**

definire la semantica denotazionale che esprime il valore di una frase (sequenza di cifre).

Esercizio 1

Data la seguente BNF relativa alla descrizione di numeri in base cinque:

pentanum → pentanum **0** | pentanum **1** | pentanum **2** | pentanum **3** | pentanum **4** | **0** | **1** | **2** | **3** | **4**

definire la semantica denotazionale che esprime il valore di una frase (sequenza di cifre).

N = { naturali } = dominio degli oggetti matematici associati

M = funzione di mapping:

M(‘0’) = 0	M(pentanum ‘0’) = 5 * M(pentanum)
M(‘1’) = 1	M(pentanum ‘1’) = 5 * M(pentanum) + 1
M(‘2’) = 2	M(pentanum ‘2’) = 5 * M(pentanum) + 2
M(‘3’) = 3	M(pentanum ‘3’) = 5 * M(pentanum) + 3
M(‘4’) = 4	M(pentanum ‘4’) = 5 * M(pentanum) + 4

Esercizio 2

Specificare la semantica denotazionale della seguente istruzione (ciclo a condizione finale):

repeat L **until** B

in cui la lista L di istruzioni viene ripetuta (almeno una volta) finchè la condizione booleana B risulta vera (il ciclo termina quando B = **true**). Specificatamente, si definisca la funzione M_r(**repeat** L **until** B, *s*), in cui *s* rappresenta lo stato del programma, assumendo di aver a disposizione le seguenti funzioni:

- M_b(B, *s*) : espressione booleana $\rightarrow$ {**true**, **false**, **errore**}
- M_l(L, *s*): lista di istruzioni $\rightarrow$ nuovo stato o **errore**.

Esercizio 2

Specificare la semantica denotazionale della seguente istruzione (ciclo a condizione finale):

repeat L **until** B

in cui la lista L di istruzioni viene ripetuta (almeno una volta) finchè la condizione booleana B risulta vera (il ciclo termina quando B = **true**). Specificatamente, si definisca la funzione M_r(**repeat** L **until** B, *s*), in cui *s* rappresenta lo stato del programma, assumendo di aver a disposizione le seguenti funzioni:

- M_b(B, *s*) : espressione booleana $\rightarrow$ {**true**, **false**, **errore**}
- M_l(L, *s*): lista di istruzioni $\rightarrow$ nuovo stato o **errore**.

M_r(**repeat** L **until** B, s) =
  if M_l(L, s) = **errore then**
    errore
  elsif M_b(B, M_l(L, s)) = **errore then**
    errore
  elsif M_b(B, M_l(L, s)) = **true then**
    M_l(L, s)
  else M_r(**repeat** L **until** B, M_l(L, s)).

Esercizio 3

Specificare la semantica denotazionale di una espressione logica definita dalla seguente grammatica:

$expr \rightarrow \textbf{true} \mid \textbf{false} \mid \textbf{id} \mid and\text{-}expr$
$and\text{-}expr \rightarrow expr_1 \ \textbf{and} \ expr_2$

Si assume che:

- **id** rappresenti il nome di una variabile logica;
- la valutazione di *and-expr* sia in corto circuito, con valutazione degli operandi da sinistra a destra;
- sia disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = **?**, qualora il valore della variabile **id** non sia definito;
- il linguaggio di specifica denotazionale non disponga della congiunzione logica (**and**).

Esercizio 3

Specificare la semantica denotazionale di una espressione logica definita dalla seguente grammatica:

expr → **true** | **false** | **id** | *and-expr*
and-expr → $expr_1$ **and** $expr_2$

Si assume che:

- **id** rappresenti il nome di una variabile logica;
- la valutazione di *and-expr* sia in corto circuito, con valutazione degli operandi da sinistra a destra;
- sia disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = **?**, qualora il valore della variabile **id** non sia definito;
- il linguaggio di specifica denotazionale non disponga della congiunzione logica (**and**).

M_e(**true**, *s*) = **true**.
M_e(**false**, *s*) = **false**.
M_e(**id**, *s*) = **if** μ(**id**, *s*) == **?** **then errore else** μ(**id**, *s*) **endif**.
M_e(*and-expr*, *s*) = M_{and}(*and-expr*, *s*).
M_{and}(*and-expr*, *s*) = **if** $M_e(expr_1, s)$ == **errore then errore**
elsif $M_e(expr_1, s)$ == **true then** $M_e(expr_2, s)$
else false
endif.

Esercizio 4

Specificare la semantica denotazionale di una espressione logica definita dalla seguente grammatica:

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **and** $expr_2$ | $expr_1$ **or** $expr_2$

sulla base delle seguenti assunzioni:

- **id** rappresenta il nome di una variabile logica;
- la valutazione degli operatori **and** ed **or** è in corto circuito;
- la valutazione degli operandi dell'operatore **and** è da sinistra a destra;
- la valutazione degli operandi dell'operatore **or** è da destra a sinistra;
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = UNDEF, qualora il valore della variabile **id** non sia definito;
- il linguaggio di specifica denotazionale non dispone di operatori logici.

Esercizio 4

Specificare la semantica denotazionale di una espressione logica definita dalla seguente grammatica:

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **and** $expr_2$ | $expr_1$ **or** $expr_2$

sulla base delle seguenti assunzioni:

- **id** rappresenta il nome di una variabile logica;
- la valutazione degli operatori **and** ed **or** è in corto circuito;
- la valutazione degli operandi dell'operatore **and** è da sinistra a destra;
- la valutazione degli operandi dell'operatore **or** è da destra a sinistra;
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = UNDEF, qualora il valore della variabile **id** non sia definito;
- il linguaggio di specifica denotazionale non dispone di operatori logici.

M_e(**true**, *s*) = `TRUE`.
M_e(**false**, *s*) = `FALSE`.

M_e(**id**, *s*) = **if** μ(**id**, *s*) == `UNDEF` **then** `ERRORE` **else** μ(**id**, *s*) **endif**.

M_e($expr_1$ **and** $expr_2$, *s*) = **if** M_e($expr_1$, *s*) == `ERRORE` **then** `ERRORE`
elsif M_e($expr_1$, *s*) == `TRUE` **then** M_e($expr_2$, *s*)
else `FALSE`
endif.

M_e($expr_1$ **or** $expr_2$, *s*) = **if** M_e($expr_2$, *s*) == `ERRORE` **then** `ERRORE`
elsif M_e($expr_2$, *s*) == `TRUE` **then** `TRUE`
else M_e($expr_1$, *s*)
endif.

Esercizio 5

Specificare la semantica denotazionale di una espressione aritmetica che coinvolge gli operatori di moltiplicazione ed esponenziazione:

$$expr \rightarrow expr_1 * expr_2 \mid expr_1 \text{ ^ } expr_2 \mid \textbf{id} \mid \textbf{num}$$

sulla base delle seguenti assunzioni:

- **num** rappresenta una costante intera;
- **id** rappresenta il nome di una variabile intera;
- la valutazione degli operandi è da destra a sinistra;
- la valutazione degli operatori è in corto circuito, secondo le seguenti regole:
- $expr_1 * expr_2 = 0$ quando $expr_2 = 0$
- $expr_1$ ^ $expr_2 = 1$ quando $expr_2 = 0$
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = UNDEF, qualora il valore della variabile **id** non sia definito;
- è disponibile una funzione M_n(**num**) che restituisce il valore di **num**;
- il linguaggio di specifica denotazionale dispone degli operatori $*$ e **^**.

Esercizio 5

Specificare la semantica denotazionale di una espressione aritmetica che coinvolge gli operatori di moltiplicazione ed esponenziazione:

$$expr \rightarrow expr_1 * expr_2 \mid expr_1 \text{ ^ } expr_2 \mid \textbf{id} \mid \textbf{num}$$

sulla base delle seguenti assunzioni:

- **num** rappresenta una costante intera;
- **id** rappresenta il nome di una variabile intera;
- la valutazione degli operandi è da destra a sinistra;
- la valutazione degli operatori è in corto circuito, secondo le seguenti regole:
- $expr_1 * expr_2 = 0$ quando $expr_2 = 0$
- $expr_1 \text{ ^ } expr_2 = 1$ quando $expr_2 = 0$
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
- μ(**id**, *s*) = UNDEF, qualora il valore della variabile **id** non sia definito;
- è disponibile una funzione M_n(**num**) che restituisce il valore di **num**;
- il linguaggio di specifica denotazionale dispone degli operatori $*$ e **^**.

$M_e(\textbf{num}, s) = M_n(\textbf{num})$.

$M_e(\textbf{id}, s) = \textbf{if } \mu(\textbf{id}, s) == \texttt{UNDEF } \textbf{then } \texttt{ERRORE } \textbf{else } \mu(\textbf{id}, s) \textbf{ endif}$.

$M_e(expr_1 * expr_2, s) = \textbf{if } M_e(expr_2, s) == \texttt{ERRORE } \textbf{then } \texttt{ERRORE}$
$\textbf{elsif } M_e(expr_2, s) == 0 \textbf{ then } 0$
$\textbf{elsif } M_e(expr_1, s) == \texttt{ERRORE } \textbf{then } \texttt{ERRORE}$
$\textbf{else } M_e(expr_1, s) * M_e(expr_2, s)$
$\textbf{endif}$.

$M_e(expr_1 \text{ ^ } expr_2, s) = \textbf{if } M_e(expr_2, s) == \texttt{ERRORE } \textbf{then } \texttt{ERRORE}$
$\textbf{elsif } M_e(expr_2, s) == 0 \textbf{ then } 1$
$\textbf{elsif } M_e(expr_1, s) == \texttt{ERRORE } \textbf{then } \texttt{ERRORE}$
$\textbf{else } M_e(expr_1, s) \text{ ^ } M_e(expr_2, s)$
$\textbf{endif}$.

Esercizio 6

È dato il seguente frammento di grammatica BNF, relativo alla specifica di una lista di istruzioni di un linguaggio imperativo:

$$\textit{stat-list} \rightarrow \textit{stat stat-list} \mid \textit{stat}$$
$$\textit{stat} \rightarrow \textit{assign-stat} \mid \textit{if-stat} \mid \textit{while-stat}$$

Si richiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo che siano disponibili le seguenti funzioni di mapping, che mappano una istruzione ed un certo stato s in un nuovo stato (eventualmente **errore**):

- $M_{assign}(\textit{assign-stat}, s)$
- $M_{if}(\textit{if-stat}, s)$
- $M_{while}(\textit{while-stat}, s)$

In particolare, si richiede la specifica delle seguenti funzioni (che computano il nuovo stato, eventualmente errore):

- $M_{stat}(\textit{stat}, s)$
- $M_{list}(\textit{stat-list}, s)$

Esercizio 6

stat-list $\rightarrow$ *stat stat-list* | *stat*
stat $\rightarrow$ *assign-stat* | *if-stat* | *while-stat*

$M_{stat}(assign\text{-}stat, s) = M_{assign}(assign\text{-}stat, s)$.

$M_{stat}(if\text{-}stat, s) = M_{if}(if\text{-}stat, s)$.

$M_{stat}(while\text{-}stat, s) = M_{while}(while\text{-}stat, s)$.

$M_{list}(stat, s) = M_{stat}(stat, s)$.

$M_{list}(stat\ stat\text{-}list_2, s) =$ **if** $M_{stat}(stat, s)$ **= errore then**
errore
else
$M_{list}(stat\text{-}list_2, M_{stat}(stat, s))$.

Esercizio 7

Specificare la semantica denotazionale della seguente istruzione (ciclo *for C-like*):

for (I_1; B; I_2) L

la cui semantica operazionale può essere espressa mediante il ciclo *while* nel seguente modo:

I_1;
while B **do** L; I_2 **end.**

dove I_1 ed I_2 sono istruzioni, B è una espressione booleana ed L è una lista di istruzioni (corpo del ciclo). Si assume che siano disponibili le seguenti funzioni semantiche (di cui non è richiesta l'implementazione):

- $\mathbf{M_{stat}(I, s)}$, che computa il nuovo stato (eventualmente ERRORE) dopo l'esecuzione della istruzione **I** nello stato **s**;
- $\mathbf{M_{bool}(B, s)}$, che computa il valore dell'espressione booleana **B** (eventualmente ERRORE) nello stato **s**;
- $\mathbf{M_{list}(L, s)}$, che computa il nuovo stato (eventualmente ERRORE) dopo l'esecuzione della lista **L** di istruzioni nello stato **s**.

Esercizio 7

Specificare la semantica denotazionale della seguente istruzione (ciclo *for C-like*):

for (I_1; B; I_2) L

la cui semantica operazionale può essere espressa mediante il ciclo *while* nel seguente modo:

I_1;
while B **do** L; I_2 **end.**

dove I_1 ed I_2 sono istruzioni, B è una espressione booleana ed L è una lista di istruzioni (corpo del ciclo). Si assume che siano disponibili le seguenti funzioni semantiche (di cui non è richiesta l'implementazione):

- $\mathbf{M_{stat}(I, s)}$, che computa il nuovo stato (eventualmente ERRORE) dopo l'esecuzione della istruzione **I** nello stato **s**;
- $\mathbf{M_{bool}(B, s)}$, che computa il valore dell'espressione booleana **B** (eventualmente ERRORE) nello stato **s**;
- $\mathbf{M_{list}(L, s)}$, che computa il nuovo stato (eventualmente ERRORE) dopo l'esecuzione della lista **L** di istruzioni nello stato **s**.

M_{for}(**for**(I_1 ; B ; I_2) L, s) = **if** $M_{stat}(I_1, s)$ == `ERRORE` **then** `ERRORE`
else $M_{while}(B, L, I_2, M_{stat}(I_1, s))$.

$M_{while}(B, L, I_2, s)$ = **if** $M_{bool}(B, s)$ == `ERRORE` **then** `ERRORE`
else if $M_{bool}(B, s)$ == `FALSE` **then** s
else if $M_{list}(L, s)$ == `ERRORE` **then** `ERRORE`
else if $M_{stat}(I_2, M_{list}(L, s))$ == `ERRORE` **then** `ERRORE`
else $M_{while}(B, L, I_2, M_{stat}(I_2, M_{list}(L, s)))$.

Esercizio 8

E' dato il seguente frammento di grammatica BNF relativo alla specifica del ciclo a condizione iniziale in un linguaggio imperativo:

while-stat → **while** *expr* **do** *stat*
expr → **true** | **false** | **id** | *expr* **and** *expr* | **(** *expr* **)**
stat → *assign-stat* | *while-stat*

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo che:

- **id** rappresenti il nome di una variabile logica;
- la valutazione dell'operatore **and** sia in corto circuito, con valutazione degli operandi da sinistra a destra;
- sia disponibile una funzione M_{id}(**id**, s) che restituisce il valore della variabile **id** nello stato s;
- M_{id}(**id**, s) = `ERRORE`, qualora il valore della variabile **id** non sia definito;
- sia disponibile una funzione M_{assign}(*assign-stat*, s) che restituisce il nuovo stato (eventualmente `ERRORE`);
- il linguaggio di specifica denotazionale non disponga di operatori logici.

In particolare, si richiede la specifica della seguenti funzioni semantiche:

M_{while}(*while-stat*, s)
M_{expr}(*expr*, s)
M_{stat}(*stat*, s)

Esercizio 8

$M_{while}(\textit{while-stat}, s)$ =

if $M_{expr}(\textit{expr}, s)$ == ERRORE **then** ERRORE
elsif $M_{expr}(\textit{expr}, s)$ == **false** **then** s
elsif $M_{stat}(\textit{stat}, s)$ == ERRORE **then** ERRORE
else $M_{while}(\textit{while-stat}, M_{stat}(\textit{stat}, s))$
endif.

while-stat → **while** *expr* **do** *stat*
expr → **true** | **false** | **id** | *expr* **and** *expr* | **(** *expr* **)**
stat → *assign-stat* | *while-stat*

$M_{expr}(\textbf{true}, s)$ = TRUE.
$M_{expr}(\textbf{false}, s)$ = FALSE.
$M_{expr}(\textbf{id}, s)$ = $M_{id}(\textbf{id}, s)$.
$M_{expr}(\textit{expr}_1 \textbf{ and } \textit{expr}_2, s)$ = **if** $M_{expr}(\textit{expr}_1, s)$ == ERRORE **then** ERRORE
elsif $M_{expr}(\textit{expr}_1, s)$ == TRUE **then** $M_{expr}(\textit{expr}_2, s)$
else FALSE
endif.
$M_{expr}(\textbf{(}\ \textit{expr}_1\ \textbf{)}, s)$ = $M_{expr}(\textit{expr}_1, s)$.

$M_{stat}(\textit{assign-stat}, s)$ = $M_{assign}(\textit{assign-stat}, s)$.
$M_{stat}(\textit{while-stat}, s)$ = $M_{while}(\textit{while-stat}, s)$.

Esercizio 9

Specificare la semantica denotazionale della seguente istruzione (ciclo a conteggio):

for n **do** L

in cui n è una costante intera. L'esecuzione della lista L di istruzioni viene ripetuta n volte (zero volte se $n \leq 0$). Specificatamente, si definisca la funzione semantica

$$M_{for}(\textbf{for}\ n\ \textbf{do}\ L, s)$$

in cui s rappresenta lo stato del programma, assumendo di avere a disposizione la funzione

$M_{list}(L, s)$: lista di istruzioni $\rightarrow$ nuovo stato o **errore**.

Esercizio 9

Specificare la semantica denotazionale della seguente istruzione (ciclo a conteggio):

for n **do** L

in cui n è una costante intera. L'esecuzione della lista L di istruzioni viene ripetuta n volte (zero volte se $n \leq 0$). Specificatamente, si definisca la funzione semantica

$$M_{for}(\textbf{for}\ n\ \textbf{do}\ L, s)$$

in cui s rappresenta lo stato del programma, assumendo di avere a disposizione la funzione

$$M_{list}(L, s) : \text{lista di istruzioni} \rightarrow \text{nuovo stato o } \textbf{errore}.$$

$M_{for}(\textbf{for}\ n\ \textbf{do}\ L, s) = M_f(n, L, s).$

$M_f(n, L, s) =$ **`if`** $n \leq 0$ **`then s`**

`else if` $M_{list}(L, s) =$ ***errore*** **`then`** ***errore***

`else` $M_f(n-1, L, M_{list}(L, s))$.

Esercizio 10

Specificare la semantica denotazionale di una espressione aritmetica con operatori di addizione ed assegnamento:

$expr \rightarrow$ **num** | **id** | $expr$ **+** $expr$ | **(id =** $expr$**)**

sulla base delle seguenti assunzioni:

- **num** rappresenta una costante intera;
- **id** rappresenta il nome di una variabile intera;
- la valutazione degli operandi della addizione è da sinistra a destra;
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s* (eventualmente UNDEF);
- è disponibile una funzione M_n(**num**) che restituisce il valore di **num**.

Esercizio 10

Specificare la semantica denotazionale di una espressione aritmetica con operatori di addizione ed assegnamento:

$expr \rightarrow$ **num** | **id** | $expr$ **+** $expr$ | **(id =** $expr$**)**

sulla base delle seguenti assunzioni:

- **num** rappresenta una costante intera;
- **id** rappresenta il nome di una variabile intera;
- la valutazione degli operandi della addizione è da sinistra a destra;
- è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s* (eventualmente UNDEF);
- è disponibile una funzione M_n(**num**) che restituisce il valore di **num**.

Dominio associato = $(Z \times S) \cup \{$ ***errore*** $\}$, in cui $\begin{cases} Z = \{ \text{interi} \} \\ S = \{ \text{stati} \} \end{cases}$

M_e(**num**, s) = (M_n(**num**), s).

M_e(**id**, s) = **if** μ(**id**, *s*) == UNDEF **then** ***errore*** **else** (μ(**id**, *s*), s) **endif**.

$M_e(expr_1$ **+** $expr_2$, s) =

if $M_e(expr_1$, s) == ***errore*** **then** ***errore***
elsif $M_e(expr_2$, s_1) == ***errore*** , **where** (_, s_1) = $M_e(expr_1$, s), **then** ***errore***
else (v', s'), **where** v' = v_1+v_2, (v_1, s_1) = $M_e(expr_1$, s), (v_2, s') = $M_e(expr_2$, s_1)
endif.

M_e((**id =** $expr$), s) =

if M_e($expr$, s) == ***errore*** **then** ***errore***
else (v', s''), **where** M_e($expr$, s) = (v', s'), s'' = $\{(i_1, v_1''),\ldots, (i_n, v_n'')\}$, $\forall$ k $\in$ [1 .. *n*] $\left[v_k'' = \begin{cases} \mu(i_k, s') \text{ if } i_k \neq \textbf{id} \\ v' \quad \textbf{otherwise} \end{cases} \right]$
endif.

Esercizio 11

Specificare la semantica denotazionale di una espressione relazionale di selezione:

relexpr → **select [** *pred* **]** *relexpr* | **id**

in cui *relexpr* rappresenta una operazione di selezione, *pred* un predicato di selezione, ed **id** il nome di una tabella. Si assume che l'istanza (anche vuota) della tabella sia rappresentata da una lista (anche vuota) di tuple. Il risultato della selezione è costituito dalla sottolista di tuple della tabella per le quali il predicato di selezione risulta vero.
In particolare, si chiede di specificare la funzione semantica **Mr**(*relexpr*, s), in cui s rappresenta lo stato del programma, assumendo di avere a disposizione le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- μ(**id**, s): computa la lista di tuple della tabella di nome **id** (se è definita) oppure **error** (se non è definita).
- σ(*p*, *t*): computa il valore booleano del predicato *p* applicato alla tupla *t*.
- `head(lista)`: restituisce la testa di `lista`.
- `tail(lista)`: restituisce la coda di `lista`.
- `cons(testa,coda)`: restituisce la lista `(testa:coda)`, in cui `testa` è una tupla e `coda` una lista.

Esercizio 11

Specificare la semantica denotazionale di una espressione relazionale di selezione:

relexpr → **select [** *pred* **]** *relexpr* | **id**

in cui *relexpr* rappresenta una operazione di selezione, *pred* un predicato di selezione, ed **id** il nome di una tabella. Si assume che l'istanza (anche vuota) della tabella sia rappresentata da una lista (anche vuota) di tuple. Il risultato della selezione è cositituito dalla sottolista di tuple della tabella per le quali il predicato di selezione risulta vero.

M_r(**id**, s) = μ(**id**, s).

M_r(**select [** *pred* **]** *relexpr*, s) = M_{sel}(*pred*, M_r(*relexpr*, s)).

M_{sel}(_, **error**) = **error**.
M_{sel}(_, [ ]) = [ ].
M_{sel}(*p*, *T*) = **if** σ(*p*, head(*T*)) **then**
cons(head(*T*), M_{sel}(*p*, tail(*T*)))
else
M_{sel}(*p*, tail(*T*)).

Esercizio 12

Specificare la semantica denotazionale di una espressione relazionale di intersezione definita dalla seguente BNF:

$$intexpr \rightarrow \mathbf{(}\ intexpr \cap intexpr\ \mathbf{)} \mid \mathbf{table}$$

in cui *intexpr* rappresenta l'operazione insiemistica di intersezione e **table** il nome di una tabella. Si assume che l'istanza (anche vuota) di ogni tabella sia rappresentata da una lista (anche vuota) di tuple senza duplicati.
In particolare, si chiede di specificare la funzione semantica $\mathbf{M}_{int}$(*intexpr*) assumendo di avere a disposizione la funzione ausiliaria $\mathbf{M}_{id}$(**table**), di cui non è richiesta la specifica, che restituisce la lista di tuple della tabella di nome **table** (se è definita) oppure **errore** (se non è definita). Si richiede che la specifica denotazionale sia definita mediante la notazione di pattern-matching. In particolare, è possibile utilizzare il pattern `(testa:coda)` per una lista non vuota, ed il pattern `[]` per una lista vuota. Il linguaggio di specifica denotazionale non contiene operatori insiemistici, ad eccezione dell'appartenenza, $\in$. Non è richiesto il controllo di compatibilità dei due operandi.

Esercizio 12

Specificare la semantica denotazionale di una espressione relazionale di intersezione definita dalla seguente BNF:

$$intexpr \rightarrow \mathbf{(}\ intexpr \cap intexpr\ \mathbf{)} \mid \mathbf{table}$$

in cui *intexpr* rappresenta l'operazione insiemistica di intersezione e **table** il nome di una tabella. Si assume che l'istanza (anche vuota) di ogni tabella sia rappresentata da una lista (anche vuota) di tuple senza duplicati.
In particolare, si chiede di specificare la funzione semantica $\mathbf{M}_{int}(intexpr)$ assumendo di avere a disposizione la funzione ausiliaria $\mathbf{M}_{id}(\mathbf{table})$, di cui non è richiesta la specifica, che restituisce la lista di tuple della tabella di nome **table** (se è definita) oppure **errore** (se non è definita). Si richiede che la specifica denotazionale sia definita mediante la notazione di pattern-matching. In particolare, è possibile utilizzare il pattern `(testa:coda)` per una lista non vuota, ed il pattern `[]` per una lista vuota. Il linguaggio di specifica denotazionale non contiene operatori insiemistici, ad eccezione dell'appartenenza, $\in$. Non è richiesto il controllo di compatibilità dei due operandi.

$M_{int}(\mathbf{table}) = M_{id}(\mathbf{table})$.

$M_{int}(intexpr_1 \cap intexpr_2) = \text{Inter}(M_{int}(intexpr_1), M_{int}(intexpr_2))$

Inter(_, **errore**) = **errore**.
Inter(**errore**, _) = **errore**.
Inter([ ], _) = [ ].
Inter(testa:coda, T) = **if** testa $\in$ T **then** testa**:**(Inter(coda,T)) **else** Inter(coda, T).

Esercizio 13

Specificare la semantica denotazionale di una espressione insiemistica che coinvolge operatori di unione ($\cup$) e differenza ($-$), il cui linguaggio è definito dalla seguente BNF:

$$expr \rightarrow \mathbf{(}\ expr \cup expr\ \mathbf{)} \mid \mathbf{(}\ expr - expr\ \mathbf{)} \mid \mathbf{set}$$

in cui *expr* rappresenta una espressione insiemistica e **set** il nome di un insieme. Si assume che l'istanza (anche vuota) di ogni insieme sia rappresentata da una lista (anche vuota) di elementi senza duplicazioni. In particolare, si chiede di specificare la funzione semantica $\mathbf{M}_e(expr)$ assumendo di avere a disposizione la funzione ausiliaria $\mathbf{M}_{id}(\mathbf{set})$, di cui non è richiesta la specifica, che restituisce la lista di elementi dell'insieme di nome **set** (se è definito) oppure **errore** (se non è definito). Si richiede che la specifica denotazionale sia definita mediante la notazione di pattern-matching. In particolare, è possibile utilizzare il pattern `(testa:coda)` per una lista non vuota, ed il pattern `[]` per una lista vuota. Il linguaggio di specifica denotazionale non contiene operatori insiemistici, ad eccezione dell'appartenenza, $\in$. Non è richiesto il controllo di compatibilità dei due operandi.

Esercizio 13

Specificare la semantica denotazionale di una espressione insiemistica che coinvolge operatori di unione ($\cup$) e differenza ($-$), il cui linguaggio è definito dalla seguente BNF:

$expr \rightarrow$ **(** $expr \cup expr$ **)** | **(** $expr - expr$ **)** | **set**

M_e(**set**) = M_{id}(**set**).

$M_e(expr_1 \cup expr_2)$ = Union($M_e(expr_1)$, $M_e(expr_2)$)
$M_e(expr_1 - expr_2)$ = Diff($M_e(expr_1)$, $M_e(expr_2)$)

Union(_, **errore**) = **errore**.
Union(**errore**, _) = **errore**.
Union([ ], S) = S.
Union(testa:coda, S) = **if** testa $\notin$ S **then** testa**:**(Union(coda,S)) **else** Union(coda, S).

Diff(_, **errore**) = **errore**.
Diff (**errore**, _) = **errore**.
Diff ([ ], S) = [ ].
Diff (testa:coda, S) = **if** testa $\notin$ S **then** testa**:**(Diff (coda,S)) **else** Diff(coda, S).

Esercizio 14

È dato il seguente frammento di grammatica BNF relativo alla specifica di una istruzione condizionale in un linguaggio imperativo:

if-stat → **if** *expr* **then** *stat elsif-part* **else** *stat* **endif**
elsif-part → **elsif** *expr* **then** *stat elsif-part* | **ε**

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- $M_{expr}(expr, s)$: restituisce il valore logico di *expr* allo stato s oppure `ERRORE`;
- $M_{stat}(stat, s)$: restituisce il nuovo stato oppure `ERRORE` dopo l'esecuzione di *stat*.

In particolare, si richiede la specifica della seguente funzione semantica:

$M_{if\text{-}stat}(if\text{-}stat, s)$.

Esercizio 14

È dato il seguente frammento di grammatica BNF relativo alla specifica di una istruzione condizionale in un linguaggio imperativo:

$$\textit{if-stat} \rightarrow \textbf{if}\ \textit{expr}\ \textbf{then}\ \textit{stat}_1\ \textit{elsif-part}\ \textbf{else}\ \textit{stat}_2\ \textbf{endif}$$
$$\textit{elsif-part} \rightarrow \textbf{elsif}\ \textit{expr}\ \textbf{then}\ \textit{stat}\ \textit{elsif-part}_2 \mid \boldsymbol{\varepsilon}$$

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- $M_{expr}(expr, s)$: restituisce il valore logico di *expr* allo stato s oppure `ERRORE`;
- $M_{stat}(stat, s)$: restituisce il nuovo stato oppure `ERRORE` dopo l'esecuzione di *stat*.

In particolare, si richiede la specifica della seguente funzione semantica:

$M_{if\text{-}stat}(\textit{if-stat}, s)$.

$$M_{if\text{-}stat}(\textit{if-stat}, s) = \textbf{if}\ M_{expr}(expr, s) == \texttt{ERRORE}\ \textbf{then}\ \texttt{ERRORE}$$
$$\textbf{elsif}\ M_{expr}(expr, s) == \texttt{TRUE}\ \textbf{then}\ M_{stat}(stat_1, s)$$
$$\textbf{elsif}\ M_{elsif\text{-}part}(\textit{elsif-part}, s) == \texttt{NULL}\ \textbf{then}\ M_{stat}(stat_2, s)$$
$$\textbf{else}\ M_{elsif\text{-}part}(\textit{elsif-part}, s).$$

$$M_{elsif\text{-}part}(\textbf{elsif}\ expr\ \textbf{then}\ stat\ elsif\text{-}part_2, s) =$$
$$\textbf{if}\ M_{expr}(expr, s) == \texttt{ERRORE}\ \textbf{then}\ \texttt{ERRORE}$$
$$\textbf{elsif}\ M_{expr}(expr, s) == \texttt{TRUE}\ \textbf{then}\ M_{stat}(stat, s)$$
$$\textbf{else}\ M_{elsif\text{-}part}(elsif\text{-}part_2, s).$$
$$M_{elsif\text{-}part}(\boldsymbol{\varepsilon}, s) = \texttt{NULL}.$$

Esercizio 15

Specificare la semantica denotazionale di una istruzione condizionale a due vie definita dalla seguente grammatica:

if-stat → **if** *expr* **then** *stat-list*$_1$ **else** *stat-list*$_2$
stat-list → *stat stat-list*$_1$ | *stat*
expr → **true** | **false** | **id** | *expr*$_1$ **and** *expr*$_2$ | *expr*$_1$ **or** *expr*$_2$ | **not** *expr*$_1$

Si assume che:

- **id** rappresenti il nome di una variabile logica;
- la valutazione degli operandi nelle operazioni logiche sia da sinistra a destra;
- la valutazione della disgiunzione (**or**) sia in corto circuito;
- la valutazione della congiunzione (**and**) non sia in corto circuito;
- il linguaggio di specifica denotazionale disponga degli operatori logici $\wedge$ (congiunzione), $\vee$ (disgiunzione), $\neg$ (negazione), e degli operatori aritmetici +, -, *, /, applicabili unicamente ad operandi semanticamente corretti;
- siano disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

  μ(**id**, s): restituisce il valore della variabile **id** nello stato s (o **errore** nel caso **id** non sia istanziata);
  M_{stat}(*stat*, s): restituisce lo stato raggiunto dalla esecuzione di *stat* allo stato s (eventualmente **errore**).

Esercizio 15

Specificare la semantica denotazionale di una istruzione condizionale a due vie definita dalla seguente grammatica:

if-stat → **if** *expr* **then** *stat-list*$_1$ **else** *stat-list*$_2$
stat-list → *stat stat-list*$_1$ | *stat*
expr → **true** | **false** | **id** | *expr*$_1$ **and** *expr*$_2$ | *expr*$_1$ **or** *expr*$_2$ | **not** *expr*$_1$

$M_{\text{if-stat}}$(*if-stat*, *s*) = **if** M_e(*expr*, s) == **errore then errore**
elsif M_e(*expr*, s) == **true then** M_{list}(*stat-list*$_1$, s)
else M_{list}(*stat-list*$_2$, s).

M_{expr}(**true**, s) = **true**.
M_{expr}(**false**, s) = **false**.
M_{expr}(**id**, *s*) = μ(**id**, s).

M_{expr}(*expr*$_1$ **and** *expr*$_2$, s) = **if** M_e(*expr*$_1$, s) == **errore then errore**
elsif M_e(*expr*$_2$, s) == **errore then errore**
else M_e(*expr*$_1$, s) ∧ M_e(*expr*$_2$, s);

M_{expr}(*expr*$_1$ **or** *expr*$_2$, s) = **if** M_e(*expr*$_1$, s) == **errore then errore**
elsif M_e(*expr*$_1$, s) == **true then true**
else M_e(*expr*$_2$, s)
endif.

M_{expr}(**not** *expr*$_1$, s) = **if** M_e(*expr*$_1$, s) == **errore then errore**
else ¬M_e(*expr*$_1$, s)
endif.

M_{list}(*stat stat-list*$_1$, s) = **if** M_{stat}(*stat*, s) == **errore then errore**
else M_{list}(*stat-list*$_1$, M_{stat}(*stat*, s))
endif.

M_{list}(*stat*, s) = M_{stat}(*stat*, s).

Esercizio 16

Specificare la semantica denotazionale di una funzione *Haskell*-like definita dalla seguente grammatica (mediante il costrutto a guardie):

$$
\begin{aligned}
&\textit{function} \rightarrow \textbf{id}\ \textit{param-list}\ \ \textit{equation-list}\\
&\textit{param-list} \rightarrow \textbf{id}\ \textit{param-list}_1 \mid\ \textbf{id}\\
&\textit{equation-list} \rightarrow\ \textit{equation}\ \textit{equation-list}_1 \mid \textit{equation}\\
&\textit{equation} \rightarrow\ \text{'|'}\ \textit{cond} = \ \textit{expr}
\end{aligned}
$$

esempio di frase

```
alfa x y z
   | x > y    = x + y
   | x == z   = z - 1
   | y < z    = x * (y - z)
```

Si richiede la specifica del valore computato dalla funzione, assumendo che:

- non ci siano errori semantici nella valutazione delle condizioni (guardie) e delle espressioni;
- i parametri della funzione abbiano un valore intero;
- siano disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

  $M_{expr}(expr)$: restituisce il valore (intero) di *expr*,
  $M_{cond}(cond)$: restituisce il valore (booleano) di *cond*;

- il valore computato dalla funzione corrisponda alla prima espressione la cui condizione (guardia) risulti vera.
- nel caso in cui nessuna condizione (guardia) risulti vera, il valore computato sia **nil**.

Esercizio 16

Specificare la semantica denotazionale di una funzione *Haskell*-like definita dalla seguente grammatica (mediante il costrutto a guardie):

function → **id** *param-list equation-list*
param-list → **id** *param-list*$_1$ | **id**
equation-list → *equation equation-list*$_1$ | *equation*
equation → '|' *cond* = *expr*

esempio di frase

```
alfa x y z
   | x > y    = x + y
   | x == z   = z - 1
   | y < z    = x * (y - z)
```

M_f(**id** *param-list equation-list*) = $M_{eq\text{-}list}$(*equation-list*).

$M_{eq\text{-}list}$(*equation equation-list*$_1$) =
if M_{eq}(*equation*) == (**true**, val) **then** val **else** $M_{eq\text{-}list}$(*equation-list*$_1$) **endif**.

$M_{eq\text{-}list}$(*equation*) =
if M_{eq}(*equation*) == (**true**, val) **then** val **else nil endif**.

M_{eq}(*equation*) =
if M_{cond}(*cond*) == **false then** (**false**, _) **else** (**true**, M_{expr}(*expr*)) **endif**.

Esercizio 17

Specificare la semantica denotazionale di un frammento di codice definito dalla seguente BNF:

frammento → **id = [** *num-list* **] ; num in id.**
num-list → **num ,** *num-list* | **num**

```
S = [1,3,6,9,24];
4 in S.
```

Il frammento è costituito da due operazioni: istanziazione di una lista di interi e appartenenza di un intero a tale lista. La lista non deve contenere duplicati e il nome della lista nella espressione di appartenenza deve coincidere con il nome della lista istanziata precedentemente. In particolare, si richiede la specifica delle seguenti funzioni semantiche:

- M_f(*frammento*): restituisce il risultato della operazione di appartenenza in *frammento*, (eventualmente `error`);
- M_{nl}(*num-list*): restituisce la lista di interi relativa a *num-list*, (eventualmente `error`).

Nel linguaggio di specifica denotazionale, è possibile esprimere l'operazione di appartenenza mediante il simbolo ∈ . Sono inoltre disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- `name(`**id**`)`: restituisce il valore lessicale (nome) di **id**;
- `value(`**num**`)`: restituisce il valore lessicale (valore) di **num.**

Esercizio 17

Specificare la semantica denotazionale di un frammento di codice definito dalla seguente BNF:

frammento → **id = [** *num-list* **] ; num in id.**
num-list → **num ,** *num-list* | **num**

```
S = [1,3,6,9,24];
4 in S.
```

M_f(**id$_1$ = [** *num-list* **] ; num in id$_2$.**) =
  if name(**id$_1$**) != nome(**id$_2$**) **then** error
  elsif M_{nl}(*num-list*) == error **then** error
  else value(**num**) ∈ M_{nl}(*num-list*)
  endif.

M_{nl}(**num**) = [value(**num**)].
M_{nl}(*num* **,** *num-list*) =
  if M_{nl}(*num-list*) == error **then** error
  elsif value(**num**) ∈ M_{nl}(*num-list*) **then** error
  else (value(**num**) : M_{nl}(*num-list*))
  endif.

Esercizio 18

Specificare la semantica denotazionale del ciclo a conteggio definito dalla seguente BNF:

loop-stat → **for id = num$_1$ to num$_2$ do** *stat*

in cui [**num$_1$** .. **num$_2$**] costituisce il range di valori interi per la variabile di conteggio **id**. In particolare, si richiede la specifica delle funzione semantica M_{loop}(*loop-stat*, `s`), in cui `s` rappresenta lo stato del programma. Sono disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- `name`(**id**): restituisce il nome (attributo lessicale) di **id**;
- `value`(**num**): restituisce il valore (attributo lessicale) di **num.**
- `value`(`nome`, `s`): restituisce il valore della variabile `nome` nello stato `s`.
- `inscope`(`nome`, `s`): restituisce un booleano, vero quando la variabile `nome` è visibile nello stato `s`.
- M_{assign}(`nome`, `val`, `s`): restituisce lo stato raggiunto (eventualmente `error`) dopo l'assegnamento della variabile `nome` (nello stato `s`) con il valore `val`.
- M_{stat}(*stat*, `s`): restituisce lo stato raggiunto (eventualmente `error`) dopo l'esecuzione di *stat*.

Esercizio 18

Specificare la semantica denotazionale del ciclo a conteggio definito dalla seguente BNF:

loop-stat → **for id = num$_1$ to num$_2$ do** *stat*

in cui [**num$_1$** .. **num$_2$**] costituisce il range di valori interi per la variabile di conteggio **id**. In particolare, si richiede la specifica delle funzione semantica M_{loop}(*loop-stat*, `s`), in cui `s` rappresenta lo stato del programma.

M_{loop}(*loop-stat,* `s`) =
    if not `inscope(name(`**id**`), s)` **then** `error`
    else M_f(`name(`**id**`), value(`**num$_2$**`)`, *stat*, M_{assign}(`name(`**id**`), value(`**num$_1$**`)`, `s`))
    endif.

M_f(`nome, val`, *stat*, `s`) =
    if `value(nome, s) > val` **then** `s`
    elsif M_{stat}(*stat,* `s`) == `error` **then** `error`
    else M_f(`nome, val`, *stat*, M_{assign}(`nome, value(nome)`+1, M_{stat}(*stat,* `s`)))
    endif.

Esercizio 19

Specificare la semantica denotazionale di un frammento di codice definito dalla seguente BNF:

codice → **id = [** *string-list* **] ; id [num].**
string-list → **string ,** *string-list* | **string**

```
v = ["alfa","beta","gamma","delta","epsilon];
v[3].
```

Il frammento è costituito da due operazioni: assegnamento di un vettore di stringhe e indicizzazione di tale vettore. Il nome del vettore nella espressione di indicizzazione deve coincidere con il nome del vettore assegnato precedentemente. Inoltre, il valore dell'indice non deve uscire dai limiti di indicizzazione $[0 .. n\text{-}1]$, in cui n è la dimensione del vettore. Si richiede la specifica delle seguenti funzioni semantiche:

- M_c(*codice*): restituisce il risultato della operazione di indicizzazione in *codice*, (eventualmente `errore`);
- M_{sl}(*string-list*): restituisce la lista di stringhe relativa a *string-list*.

Nel linguaggio di specifica denotazionale sono disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- `name(`**id**`)`: restituisce il valore lessicale (nome) di **id**;
- `ivalue(`**num**`)`: restituisce il valore lessicale (valore) di **num.**
- `svalue(`**string**`)`: restituisce il valore lessicale (valore) di **string.**
- `length(L)`: restituisce la lunghezza della lista `L`**.**
- `elem(L,i)`: restituisce l'elemento i-esimo della lista `L`**.**
- `cons(e,L)`: restituisce la lista `(e:L)`**.**

Esercizio 19

Specificare la semantica denotazionale di un frammento di codice definito dalla seguente BNF:

codice → **id = [** *string-list* **] ; id [num].**
string-list → **string ,** *string-list* | **string**

```
v = ["alfa","beta","gamma","delta","epsilon];
v[3].
```

M_c(**id_1 = [** *string-list* **] ; id_2[num].**) =
    if `name`(**id_1**) != `name`(**id_2**) **then** `errore`
    elsif `ivalue`(**num**) < 0 **or** `ivalue`(**num**) >= `length`(M_{sl}(*string-list*)) **then** `errore`
    else `elem`(M_{sl}(*string-list*), `ivalue`(**num**))
    endif.

M_{sl}(**string**) = [`svalue`(**string**)].
M_{sl}(**string ,** *string-list*) = `cons`(`svalue`(**string**), M_{sl}(*string-list*)).

Esercizio 20

È dato un linguaggio delle espressioni logiche definito dalla seguente grammatica BNF:

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **entails** $expr_2$

in cui:

- **id** rappresenta il nome di una variabile logica;
- **entails** rappresenta l'operatore di implicazione logica;
- la valutazione dell'operatore di implicazione è in corto circuito (da sinistra a destra).

Si chiede di:

- rappresentare la tabella di verità dell'operatore **entails**;
- sulla base della relativa tabella di verità, definire la regola di corto circuito per l'operatore **entails**;
- specificare la semantica denotazionale di una espressione logica.

Si assume che:

- sia disponibile una funzione `value`(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*; (nel caso in cui la variabile non abbia un valore, `value` restituisce `ERRORE`);
- il linguaggio di specifica denotazionale non disponga di alcun operatore logico.

Esercizio 20

È dato un linguaggio delle espressioni logiche definito dalla seguente grammatica BNF:

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **entails** $expr_2$

A	B	A **entails** B
F	F	T
F	T	T
T	F	F
T	T	T

A entails B ≡ **(A ? B : true)**

M_e(**true**, s) = `TRUE`.

M_e(**false**, s) = `FALSE`.

M_e(**id**, s) = `value`(**id**, s).

M_e($expr_1$ **entails** $expr_2$, s) = **if** M_e($expr_1$, s) == `ERRORE` **then** `ERRORE`
elsif M_e($expr_1$, s) == `TRUE` **then** M_e($expr_2$, s)
else `TRUE`
endif.

Esercizio 21

È dato un linguaggio delle espressioni logiche definito dalla seguente grammatica BNF :

$$expr \rightarrow \textbf{true} \mid \textbf{false} \mid \textbf{id} \mid expr_1 \ \textbf{xor} \ expr_2$$

in cui:

- **id** rappresenta il nome di una variabile logica;
- **xor** rappresenta l'operatore di disgiunzione logica esclusiva (vero se e solo se un solo operando è vero);
- la valutazione dell'operatore **xor** è da sinistra a destra.

Si chiede di:

- rappresentare la tabella di verità dell'operatore **xor**;
- sulla base della relativa tabella di verità, specificare l'espressione condizionale (non triviale) che definisce l'operatore **xor**;
- specificare la semantica denotazionale di una espressione logica sulla base della espressione condizionale.

Si assume che:

- sia disponibile una funzione `value`(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*; (nel caso in cui la variabile non abbia un valore, `value` restituisce `ERRORE`);
- il linguaggio di specifica denotazionale non disponga di alcun operatore logico ad eccezione della negazione (**!**).

Esercizio 21

È dato un linguaggio delle espressioni logiche definito dalla seguente grammatica BNF :

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **xor** $expr_2$

A	B	A xor B
T	T	F
T	F	T
F	T	T
F	F	F

A xor B ≡ (A ? ¬B : B)

M_e(**true**, s) = TRUE.

M_e(**false**, s) = FALSE.

M_e(**id**, s) = value(**id**, s).

M_e($expr_1$ **xor** $expr_2$, s) = **if** M_e($expr_1$, s) == ERRORE **then** ERRORE
elsif M_e($expr_1$, s) == TRUE **then**
if M_e($expr_2$, s) = ERRORE **then** ERRORE
else ! M_e($expr_2$, s)
endif
else M_e($expr_2$, s)
endif.

Esercizio 22

È dato il seguente frammento di grammatica BNF relativo alla specifica di una istruzione case in un linguaggio imperativo:

case-stat → **case** *expr* **of** *case-list default* **end**
case-list → *case case-list* | *case*
case → **const :** *stat-list*
default → **otherwise** *stat-list* | **ε**

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- M_{expr}(*expr*, `s`): restituisce il valore di *expr* (eventualmente `ERROR`) allo stato `s`;
- $M_{stat\text{-}list}$(*stat-list*, `s`): restituisce il nuovo stato (eventualmente `ERROR`) dopo l'esecuzione delle istruzioni in *stat-list* allo stato `s`;
- M_{const}(**const**): restituisce il valore della costante **const**.

Esercizio 22

È dato il seguente frammento di grammatica BNF relativo alla specifica di una istruzione case in un linguaggio imperativo:

case-stat → **case** *expr* **of** *case-list default* **end**
case-list → *case case-list* | *case*
case → **const :** *stat-list*
default → **otherwise** *stat-list* | **ε**

$M_{case\text{-}stat}(case\text{-}stat, \mathtt{s}) =$ **if** $M_{expr}(expr, s) ==$ `ERROR` **then**
`ERROR`
elsif $M_{case\text{-}listt}(case\text{-}list, M_{expr}(expr, \mathtt{s}), \mathtt{s}) \neq$ `NULL` **then**
$M_{case\text{-}listt}(case\text{-}list, M_{expr}(expr, \mathtt{s}), \mathtt{s})$
else
$M_{default}(default, \mathtt{s})$
endif.

$M_{case\text{-}list}(case\ case\text{-}list, \mathtt{val}, \mathtt{s}) =$ **if** $M_{case}(case, \mathtt{val}, \mathtt{s}) \neq$ NULL **then**
$M_{case}(case, \mathtt{val}, \mathtt{s})$
else $M_{case\text{-}listt}(case\text{-}list, \mathtt{val}, \mathtt{s})$ **endif**.

$M_{case}(case, \mathtt{val}, \mathtt{s}) =$ **if** $M_{const}(\mathbf{const}) ==$ `val` **then**
$M_{stat\text{-}list}(stat\text{-}list, \mathtt{s})$
else `NULL` **endif**.

$M_{case\text{-}list}(case, \mathtt{val}, \mathtt{s}) = M_{case}(case, \mathtt{val}, \mathtt{s})$.

$M_{default}(\mathbf{otherwise}\ stat\text{-}list, \mathtt{s}) = M_{stat\text{-}list}(stat\text{-}list, \mathtt{s})$.

$M_{default}(\varepsilon, \mathtt{s}) = \mathtt{s}$.

Esercizio 23

È dato il seguente frammento di grammatica BNF, relativo alla specifica di un insieme di numeri:

$$set \rightarrow \{\ numbers\ \}$$
$$numbers \rightarrow \mathbf{num}\ ,\ numbers \mid\ \mathbf{num}$$

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, definita come la somma di tutti i numeri positivi nell'insieme. Si assume la disponibilità della funzione ausiliaria $M_n(\mathbf{num})$, che restituisce il valore lessicale di **num**.

Esercizio 23

È dato il seguente frammento di grammatica BNF, relativo alla specifica di un insieme di numeri:

$$\textit{set} \rightarrow \{\ \textit{numbers}\ \}$$
$$\textit{numbers} \rightarrow \textbf{num}\ ,\ \textit{numbers}\ |\ \textbf{num}$$

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, definita come la somma di tutti i numeri positivi nell'insieme. Si assume la disponibilità della funzione ausiliaria $M_n(\textbf{num})$, che restituisce il valore lessicale di **num**.

$$M_s(\textit{set}) = M_{sum}(\textit{numbers}).$$
$$M_{sum}(\textbf{num}) = \textbf{if}\ M_n(\textbf{num}) > 0\ \textbf{then}\ M_n(\textbf{num})\ \textbf{else}\ 0\ \textbf{endif}.$$
$$M_{sum}(\textbf{num}\ ,\ \textit{numbers}) = M_{sum}(\textbf{num}) + M_{sum}(\textit{numbers}).$$

Esercizio 24

È dato il seguente frammento di grammatica BNF, relativo alla specifica di una espressione di tipo intero in un linguaggio funzionale, che coinvolge costanti (**num**), identificatori (**id**) che hanno un binding con un valore intero, somme e chiamate di funzioni unarie (**id(***expr***)**):

$$expr \rightarrow \mathbf{num} \mid \mathbf{id} \mid expr + expr \mid \mathbf{id(}expr\mathbf{)}$$

Assumendo che l'esecuzione di una funzione possa generare errore, si chiede di specificare la funzione semantica $M_e(expr)$, relativa al corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- $M_n(\mathbf{num})$, che restituisce il valore lessicale (intero) di **num**;
- $M_{id}(\mathbf{id})$, che restituisce il valore lessicale (stringa) di **id**;
- $M_s(\texttt{s})$, che restituisce il valore intero associato alla costante simbolica di nome `s`, oppure **errore** (nel caso in cui `s` non sia definita).
- $M_f(\texttt{f})$, che restituisce la λ-espressione associata al nome `f`, oppure **errore** (nel caso in cui `f` non sia definito).

Esercizio 24

È dato il seguente frammento di grammatica BNF, relativo alla specifica di una espressione di tipo intero in un linguaggio funzionale, che coinvolge costanti (**num**), identificatori (**id**) che hanno un binding con un valore intero, somme e chiamate di funzioni unarie (**id(***expr***)**):

$$expr \rightarrow \textbf{num} \mid \textbf{id} \mid expr + expr \mid \textbf{id(}expr\textbf{)}$$

Assumendo che l'esecuzione di una funzione possa generare errore, si chiede di specificare la funzione semantica $M_e(expr)$, relativa al corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- $M_n(\textbf{num})$, che restituisce il valore lessicale (intero) di **num**;
- $M_{id}(\textbf{id})$, che restituisce il valore lessicale (stringa) di **id**;
- $M_s(\texttt{s})$, che restituisce il valore intero associato alla costante simbolica di nome `s`, oppure **errore** (nel caso in cui `s` non sia definita).
- $M_f(\texttt{f})$, che restituisce la λ-espressione associata al nome `f`, oppure **errore** (nel caso in cui `f` non sia definito).

$$M_e(\textbf{num}) = M_n(\textbf{num}).$$

$$M_e(\textbf{id}) = M_s(M_{id}(\textbf{id})).$$

$$M_e(expr_1 + expr_2) = \textbf{if } M_e(expr_1) == \textbf{errore then errore}$$
$$\textbf{elsif } M_e(expr_2) == \textbf{errore then errore}$$
$$\textbf{else } M_e(expr_1) + M_e(expr_2).$$

$$M_e(\textbf{id(}expr\textbf{)}) = \textbf{if } M_e(expr) == \textbf{errore then errore}$$
$$\textbf{elsif } M_f(M_{id}(\textbf{id})) == \textbf{errore then errore}$$
$$\textbf{else } M_f(M_{id}(\textbf{id}))(M_e(expr)).$$

Esercizio 25

È dato il seguente frammento di grammatica BNF, relativo alla specifica di un insieme di numeri:

insieme → { *numeri* }
numeri → **numero** , *numeri* | **numero**

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, definita come la media aritmetica dei numeri nell'insieme. Si assume la disponibilità della funzione ausiliaria M_n(**numero**), che restituisce il valore lessicale di **numero**.

Esercizio 25

È dato il seguente frammento di grammatica BNF, relativo alla specifica di un insieme di numeri:

insieme → { *numeri* }
numeri → **numero** , *numeri* | **numero**

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, definita come la media aritmetica dei numeri nell'insieme. Si assume la disponibilità della funzione ausiliaria M_n(**numero**), che restituisce il valore lessicale di **numero**.

$M_{media}(insieme) = M_{somma}(numeri) \,/\, M_{card}(numeri)$

$M_{somma}(\textbf{numero}) = M_n(\textbf{numero})$.
$M_{somma}(\textbf{numero}\ ,\ numeri) = M_n(\textbf{numero}) + M_{somma}(numeri)$.

$M_{card}(\textbf{numero}) = 1$.
$M_{card}(\textbf{numero}\ ,\ numeri) = 1 + M_{card}(numeri)$.

Esercizio 26

È dato il seguente frammento di grammatica BNF, relativo di una lista di istruzioni di un linguaggio imperativo:

instruction-list → *instruction instruction-list instruction* | *instruction*
instruction → *declaration* | *conditional* | *iteration*

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo la disponibilità delle seguenti funzioni ausiliarie, che mappano una istruzione ed un certo stato in un nuovo stato (eventualmente **errore):**

- M_d(*declaration*, s);
- M_c(*conditional*, s);
- M_i(*iteration,* s).

In particolare, si richiede la specifica della seguente funzione (che computa il nuovo stato, eventualmente **errore**):

- M_{list}(*instruction-list*, s).

Esercizio 26

È dato il seguente frammento di grammatica BNF, relativo di una lista di istruzioni di un linguaggio imperativo:

instruction-list → *instruction instruction-list instruction* | *instruction*
instruction → *declaration* | *conditional* | *iteration*

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo la disponibilità delle seguenti funzioni ausiliarie, che mappano una istruzione ed un certo stato in un nuovo stato (eventualmente **errore):**

- $M_d(declaration, s)$;
- $M_c(conditional, s)$;
- $M_i(iteration, s)$.

In particolare, si richiede la specifica della seguente funzione (che computa il nuovo stato, eventualmente **errore**):

- $M_{list}(instruction\text{-}list, s)$.

$M_{inst}(declaration, s) = M_d(declaration, s)$.

$M_{inst}(conditional, s) = M_c(conditional, s)$.

$M_{inst}(iteration, s) = M_i(iteration, s)$.

$M_{list}(instruction, s) = M_{inst}(instruction, s)$.

$M_{list}(instruction_1\ instruction\text{-}list\ instruction_2, s) =$
if $M_{inst}(instruction_1, s) =$ **errore then errore**
elsif $M_{list}(instruction\text{-}list, M_{inst}(instruction_1, s)) =$ **errore then errore**
else $M_{inst}(instruction_2, M_{list}(instruction\text{-}list, M_{inst}(instruction_1, s)))$.

Esercizio 27

È dato il seguente frammento di grammatica BNF:

for-stat→ **foreach id$_1$ in id$_2$ do** *stat*

che corrisponde all'iterazione su tutti gli elementi di un array di nome **id$_2$**. Ad ogni iterazione, viene eseguita l'istruzione *stat* (che non altera l'array), in cui **id$_1$** rappresenta di volta in volta l'elemento corrente dell'array.

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (in cui `s` rappresenta lo stato del programma):

- μ(**id**, `s`): restituisce la lista di elementi dell'array **id**, se questo è definito, altrimenti restituisce **undef**;
- M_{stat}(*stat*, `s`): restituisce lo stato raggiunto dalla computazione di *stat* (eventualmente **errore**).

In particolare, si richiede la specifica della funzione M_{for}(*for-stat*, `s`), che computa il nuovo stato, eventualmente **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

Esercizio 27

È dato il seguente frammento di grammatica BNF:

for-stat→ **foreach id$_1$ in id$_2$ do** *stat*

che corrisponde all'iterazione su tutti gli elementi di un array di nome **id$_2$**. Ad ogni iterazione, viene eseguita l'istruzione *stat* (che non altera l'array), in cui **id$_1$** rappresenta di volta in volta l'elemento corrente dell'array.

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (in cui `s` rappresenta lo stato del programma):

- μ(**id**, `s`): restituisce la lista di elementi dell'array **id**, se questo è definito, altrimenti restituisce **undef**;
- M_{stat}(*stat*, `s`): restituisce lo stato raggiunto dalla computazione di *stat* (eventualmente **errore**).

In particolare, si richiede la specifica della funzione M_{for}(*for-stat*, `s`), che computa il nuovo stato, eventualmente **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

M_{for}(**foreach id$_1$ in id$_2$ do** *stat*, `s`) = **if** μ(**id$_2$**, `s`) == **undef then errore**
else iterate(μ(**id$_2$**, `s`), *stat*, `s`).

iterate(`[]`, _, `s`) = `s`.

iterate(`_:coda`, *stat*, `s`) = **if** M_{stat}(*stat*, `s`) == **errore then errore**
else iterate(`coda`, *stat*, M_{stat}(*stat*, `s`)).

Esercizio 28

È dato il seguente frammento di grammatica BNF:

$$vexpr \rightarrow vexpr + vexpr \mid \mathbf{id}$$

in cui **id** è il nome di un vettori di numeri. L'operazione di somma vettoriale genera un nuovo vettore con dimensione pari alla dimensione minore dei due vettori, in cui ogni elemento corrisponde alla somma aritmetica dei due elementi nella nella stessa posizione nei due vettori. Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità della funzione ausiliaria `instance`(**id**, `s`), in cui s rappresenta lo stato del programma, la quale restituisce la lista di numeri del vettore **id**, se questo è definito, altrimenti restituisce **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

Esercizio 28

È dato il seguente frammento di grammatica BNF:

$vexpr \rightarrow vexpr + vexpr \mid \mathbf{id}$

in cui **id** è il nome di un vettori di numeri. L'operazione di somma vettoriale genera un nuovo vettore con dimensione pari alla dimensione minore dei due vettori, in cui ogni elemento corrisponde alla somma aritmetica dei due elementi nella nella stessa posizione nei due vettori. Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità della funzione ausiliaria `instance`(**id**, `s`), in cui s rappresenta lo stato del programma, la quale restituisce la lista di numeri del vettore **id**, se questo è definito, altrimenti restituisce **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

$M_{vexpr}(\mathbf{id}, \mathtt{s}) = \mathtt{instance}(\mathbf{id}, \mathtt{s})$.

$M_{vexpr}(vexpr_1 + vexpr_2, \mathtt{s}) = M_{sum}(M_{vexpr}(vexpr_1, \mathtt{s}), M_{vexpr}(vexpr_2, \mathtt{s}))$.

$M_{sum}(\mathbf{errore}, \_) = \mathbf{errore}$.

$M_{sum}(\_, \mathbf{errore}) = \mathbf{errore}$.

$M_{sum}(\mathtt{[]}, \_) = \mathtt{[]}$.

$M_{sum}(\_, \mathtt{[]}) = \mathtt{[]}$.

$M_{sum}(\mathtt{x:xs}, \mathtt{y:ys}) = \mathtt{(x+y):}M_{sum}\mathtt{(xs, ys)}$

Esercizio 29

È dato il seguente frammento di grammatica BNF, relativo alla specifica del ciclo a condizione finale in un linguaggio imperativo:

do-stat → **do** *stat* **while** *expr*
expr → **not** *expr* | **id**
stat → *assign-stat* | *do-stat*

Specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo che **id** rappresenti il nome di una variabile logica, il linguaggio di specifica non disponga di operatori logici, siano disponibili le funzioni ausiliarie M_{id}(**id**, `s`), che restituisce il valore di **id** allo stato `s` (eventualmente **errore**), e M_{assign}(*assign-stat*, `s`). In particolare, specificare la funzione semantica M_{stat}(*stat*, `s`).

Esercizio 29

È dato il seguente frammento di grammatica BNF, relativo alla specifica del ciclo a condizione finale in un linguaggio imperativo:

do-stat → **do** *stat* **while** *expr*
expr → **not** *expr* | **id**
stat → *assign-stat* | *do-stat*

Specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo che **id** rappresenti il nome di una variabile logica, il linguaggio di specifica non disponga di operatori logici, siano disponibili le funzioni ausiliarie M_{id}(**id**, s), che restituisce il valore di **id** allo stato s (eventualmente **errore**), e M_{assign}(*assign-stat*, s). In particolare, specificare la funzione semantica M_{stat}(*stat*, s).

M_{stat}(*assign-stat*, s) = M_{assign}(*assign-stat*, s).

M_{stat}(*do-stat*, s) = M_{do}(*do-stat*, s).

M_{do}(**do** *stat* **while** *expr*, s) =
if M_{stat}(*stat*, s) == **errore then errore**
elsif M_{expr}(*expr*, M_{stat}(*stat*, s)) == **errore then errore**
elsif M_{expr}(*expr*, M_{stat}(*stat*, s)) == **false then** M_{stat}(*stat*, s)
else M_{do}(**do** *stat* **while** *expr*, M_{stat}(*stat*, s)).

M_{expr}(**not** *expr*, s) = **if** M_{expr}(*expr*, s) == **errore then errore**
elsif M_{expr}(*expr*, s) == **true then false**
else true
endif.

M_{expr}(**id**, s) = M_{id}(**id**, s).