Corso di Programmazione

Note (estorte) sulla Semantica dei Linguaggi

Versione preliminarissima a.a.'94-'95

Abstract

A grande richiesta rendiamo disponibili le note su

- aspetti algebrici delle grammatiche (e strutture a regole)
- semantica dei linguaggi
- introduzione degli identificatori e problemi correlati

nonostante siano incomplete e non riviste. Ringrazio anticipatamente chi mi segnalerà errori o darà suggerimenti e critiche sull'esposizione.

1 Linguaggi: dalla sintassi strutturata alla semantica

Avendo affrontato il problema della definizione induttiva dei *linguaggi*, cioè degli aspetti *sintattici* della comunicazione, dedichiamo questo capitolo all'analisi della *semantica* dei linguaggi stessi, ovvero data una frase quale sia il suo significato.

Si tratta dunque di individuare anzitutto un universo di "valori concreti" che siano rappresentati dai termini del linguaggio e quindi di definire una funzione di *valutazione semantica* che ad ogni stringa del linguaggio faccia corrispondere un valore. Si pensi ad esempio al caso delle espressioni numeriche: ogni espressione (stringa del linguaggio) denota un numero (valore semantico). Naturalmente si vorrebbe che se e_1 è valutata in $5 \in \mathbb{N}$ e e_2 è valutata in 7, allora $e_1 + e_2$ deve essere valutata in 12 e $e_1 * e_2$ deve essere valutata in 35. Questo corrisponde all'intuizione che il simbolo sintattico '+' è una notazione per rappresentare un'operazione, quella di somma, e come tale dovrebbe essere valutato.

Più in generale, come si vedrà nella prima sezione, ogni qualvolta un linguaggio è definito induttivamente da una grammatica BNF, viene dotato implicitamente di una *struttura algebrica*, la stessa già presente nella semantica che si vuole descrivere e la funzione di valutazione è un *omomorfismo* di tale struttura.

La "preservazione" della struttura algebrica da parte della funzione di valutazione viene, nella maggior parte dei casi, ottenuta immediatamente mediante l'applicazione di tecniche induttive standard.

In appendice sono riportate le definizioni essenziali sulle algebre (eterogenee, parziali...). Maggiori dettagli sono contenuti nelle dispense di algebra per l'a.a. 91/92 di Astesiano-Rossi.

1.1 Aspetti algebrici delle BNF

L'idea intuitiva alla base di una grammatica BNF è che le produzioni servono a "generare" gli elementi di ciascun tipo, cioè sono in un certo senso operazioni che, presi elementi del tipo giusto per sostituire le metavariabili, producono come risultato un elemento "nuovo" del tipo.

Ad esempio nella seguente variante della definizione delle espressioni numeriche:

$$e ::= 0 \mid 1 \mid (e_1 + e_2) \mid e_1 * e_2.$$

il linguaggio generato contiene 0 e 1, perché sono dati come assiomi, e le ultime due produzioni permettono di generare $(0+0)$, $(0+1)$, $(1+0)$, $(1+1)$, $0*0$, $0*1$, $1*0$, $1*1$, da cui poi si ottengono ad esempio $((1+1)+1)$, $(1+1)*(1+1)\ldots$

Questo corrisponde a pensare che il linguaggio che stiamo definendo serve a descrivere (ed è) una struttura algebrica con due elementi costanti, chiamati 0 e 1, e due operazioni binarie, la cui applicazione ad argomenti a e b viene rispettivamente rappresentata da $(a+b)$ e $a*b$. La scelta della rappresentazione è relativamente poco importante, ed è guidata dal voler parlare dei numeri con somma e prodotto. Fin dal primo esempio si vuole sottolineare che, benché apparentemente la teoria descriva come dato un linguaggio se ne possa definire in modo "ragionevole" la semantica, nella pratica si procede a ritroso: si ha un'idea informale della struttura che si vuole impiegare e si cerca di definire un linguaggio per poterne parlare; poi, definito un linguaggio "candidato", si deve verificare, mediante strumenti formali, che la sintassi proposta è appropriata.

Def. 1.1 Data la struttura a regole $\mathcal{L} = (T, S, MR)$, la *segnatura associata a* $\mathcal{L}$, $\Sigma_{\mathcal{L}} = (\bar{S}, \bar{F})$, consiste di:

Tipi: i tipi della segnatura sono i non-terminali della struttura a regole: $\bar{S} = S$;

Operazioni: la famiglia $\bar{F} = \{\bar{F}_{w,s}\}_{w \in S^*, s \in S}$ è definita da:

$$\begin{aligned}\bar{F}_{w,s} \;=\; \{op_{s::=e} \mid\ & s ::= e \in MR, w = s_1 \ldots s_n, \text{ e le metavariabili che compaiono in } e \text{ sono,}\\ & \text{nell'ordine, di tipo } s_1 \ldots s_n\}. \qquad \square\end{aligned}$$

Esercizio 1.2 Sia data la seguente struttura a regole $\mathcal{L} = (T, S, MR)$, dove

- $T = \{a, b, c\}$;
- $S = \{\alpha, \beta\}$;
- MR consiste delle seguenti produzioni: $$\begin{aligned}\alpha &::= a \mid a\alpha \mid \alpha_1\alpha_2\alpha_3\\ \beta &::= \alpha \mid b\beta \mid b\beta b\end{aligned}$$

Definire la segnatura associata a $\mathcal{L}$.

Svolgimento La segnatura $\Sigma_{\mathcal{L}} = (\bar{S}, \bar{F})$, consiste di

Tipi: $\bar{S} = \{\alpha, \beta\}$;

Operazioni: la famiglia $\bar{F} = \{\bar{F}_{w,s}\}_{w \in \bar{S}^*, s \in \bar{S}}$ è definita da:

$$\begin{array}{lcllcl}\bar{F}_{\Lambda,\alpha} & = & \{op_{\alpha::=a}\} & \bar{F}_{\alpha,\alpha} & = & \{op_{\alpha::=a\alpha}\}\\ \bar{F}_{\alpha\alpha\alpha,\alpha} & = & \{op_{\alpha::=\alpha_1\alpha_2\alpha_3}\} & \bar{F}_{\alpha,\beta} & = & \{op_{\beta::=\alpha}\}\\ \bar{F}_{\beta,\beta} & = & \{op_{\beta::=b\beta}, op_{\beta::=b\beta b}\} & \bar{F}_{w,s} & = & \emptyset \text{ altrimenti}\end{array}$$

ovvero in notazione "compatta"

sig $\Sigma_{\mathcal{L}} =$
sorts α, β

opns $op_{\alpha::=a} \colon \to \alpha$

$op_{\alpha::=a\alpha} \colon \alpha \to \alpha$

$op_{\alpha::=\alpha_1\alpha_2\alpha_3} \colon \alpha \times \alpha \times \alpha \to \alpha$

$op_{\beta::=\alpha} \colon \alpha \to \beta$

$op_{\beta::=b\beta} \colon \beta \to \beta$

$op_{\beta::=b\beta b} \colon \beta \to \beta$

Esercizio Proposto 1.3 Sia data la seguente struttura a regole $\mathcal{L} = (T, S, MR)$, dove

- $T = \{\square, \diamond, \circ\}$;
- $S = \{\text{spig}, \text{circ}\}$;
- MR consiste delle seguenti produzioni:

$$\begin{array}{lcl} \text{spig} & ::= & \square \mid \diamond \mid \square\text{spigcirc} \\ \text{circ} & ::= & \circ \mid \circ\text{circ} \mid \text{circ}_1\text{circ}_2 \end{array}$$

Definire la segnatura associata a $\mathcal{L}$.

Oltre alla segnatura, cioè alla struttura algebrica che intuitivamente si vuole avere sul linguaggio che si sta definendo, una struttura a regole contiene un'altra informazione, cioè il modo "concreto" con cui si vuole rappresentare, a livello sintattico l'applicazione dell'operazione. Pertanto una struttura a regole definisce, implicitamente, non solo una segnatura Σ, ma anche una Σ-algebra i cui supporti sono il linguaggio definito dalla struttura a regole e in cui l'interpretazione dell'operazione $op_{s::=w}$, dove in w compaiono le metavariabili $s_1, \ldots s_n$, sugli elementi del linguaggio $e_i \in L_{s_i}$ darà come risultato la stringa ottenuta da w sostituendo le metavariabili s_i con i corrispondenti argomenti e_i, cioè $w[e_1/s_1, \ldots e_n/s_n]$.

Ad esempio nel caso del linguaggio delle espressioni avremo che l'operazione $op_{e::=(e_1+e_2)}$ applicata a due elementi del linguaggio, cioè a due espressioni $\bar{e}_1$ e $\bar{e}_2$ darà come risultato la stringa (del linguaggio) $(\bar{e}_1 + \bar{e}_2)$, cioè, chiamando f l'interpretazione dell'operatore $op_{e::=(e_1+e_2)}$ nell'algebra sintattica, $f(\bar{e}_1, \bar{e}_2) = (\bar{e}_1 + \bar{e}_2)$.

Def. 1.4 Sia $\mathcal{L} = (S, T, MR)$ una struttura a regole; si dice *sintassi concreta* di $\mathcal{L}$ l'algebra $\mathcal{S}$ su $\Sigma_{\mathcal{L}}$ definita da

$$\begin{array}{ll} s^{\mathcal{S}} = L_s & \text{per ogni } s \in S \\ op^{\mathcal{S}}_{s::=w}(e_1, \ldots, e_n) = w[e_i/m_i \mid i \in [1,n]] & \text{per ogni } op_{s::=w} \in F_{s_1 \ldots s_n, s} \text{ e ogni } e_i \in L_{s_i} \end{array}$$

dove $w[e_i/m_i \mid i \in [1,n]]$ è la stringa ottenuta sostituendo ad ogni metavariabile m_i la stringa e_i. $\square$

Esempio 1.5 Se si considera la struttura a regole e la segnatura ad essa associata dell'esercizio 1.2, l'algebra della sintassi concreta è definita da

$$\begin{array}{lcl} \alpha^A & = & \{a^{n+1} \mid n \in \mathbb{N}\} \\ \beta^A & = & \{b^{i+j}a^{k+1}b^i \mid i,j,k \in \mathbb{N}\} \\ op^A_{\alpha::=a} & = & a \\ op^A_{\alpha::=a\alpha}(a^{n+1}) & = & a^{n+2} \text{ per ogni } n \in \mathbb{N} \\ op^A_{\alpha::=\alpha_1\alpha_2\alpha_3}(a^{n+1}, a^{k+1}, a^{m+1}) & = & a^{n+k+m+3} \text{ per ogni } n,k,m \in \mathbb{N} \\ op^A_{\beta::=\alpha}(a^{n+1}) & = & a^{n+1} \text{ per ogni } n \in \mathbb{N} \\ op^A_{\beta::=b\beta}(b^{i+j}a^{k+1}b^i) & = & b^{i+j+1}a^{k+1}b^i \text{ per ogni } i,j,k \in \mathbb{N} \\ op^A_{\beta::=b\beta b}(b^{i+j}a^{k+1}b^i) & = & b^{i+1+j}a^{k+1}b^{i+1} \text{ per ogni } i,j,k \in \mathbb{N} \end{array}$$

Ad esempio un'altra algebra sulla stessa segnatura è la seguente algebra N:

$$\begin{array}{lcl} \alpha^N & = & \mathbb{N} \\ \beta^N & = & \mathbb{N} \times \mathbb{N} \times \mathbb{N} \\ op^N_{\alpha::=a} & = & 0 \\ op^N_{\alpha::=a\alpha}(n) & = & n+1 \text{ per ogni } n \in \mathbb{N} \\ op^N_{\alpha::=\alpha_1\alpha_2\alpha_3}(n,k,m) & = & n+k+m \text{ per ogni } n,k,m \in \mathbb{N} \\ op^N_{\beta::=\alpha}(n) & = & (0,n,0) \text{ per ogni } n \in \mathbb{N} \\ op^N_{\beta::=b\beta}(n,m,k) & = & (n+1,m,k) \text{ per ogni } n,m,k \in \mathbb{N} \\ op^N_{\beta::=b\beta b}(n,m,k) & = & (n,m,k+1) \text{ per ogni } n,m,k \in \mathbb{N} \end{array}$$

L'omomorfismo di valutazione semantica $h \colon A \to N$ è così definito:

$$h_\alpha(a^{n+1}) = n \quad h_\beta(b^{i+j}a^{k+1}b^i) = (j,k,i)$$

Il fatto che la definizione di un linguaggio mediante una struttura a regole ne dia implicitamente anche la struttura algebrica permette di rendere rigoroso il concetto di *semantica composizionale*.

Se le produzioni (ovvero le metaregole che esse rappresentano) avessero condizioni a lato, l'algebra della sintassi concreta sarebbe un'algebra *parziale*, ma le altre nozioni resterebbero inalterate.

In molti manuali si descrive la sintassi concreta del linguaggio, ma se ne dà la semantica facendo riferimento ad una rappresentazione disambiguata della sintassi assolutamente astratta, lasciando al lettore il compito di accorgersi dei problemi e di capire come si applichi la soluzione.

2 Valutazione semantica di linguaggi

Per attribuire una semantica ad un linguaggio definito mediante una grammatica si procede per induzione, seguendo lo stesso schema di metaregole usato per definire il linguaggio. Ad esempio consideriamo un linguaggio per definire i numeri interi nella solita notazione decimale.

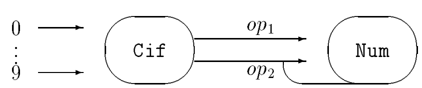
Esempio 2.1 La struttura a regole che definisce il nostro linguaggio è

$$S = \{\texttt{Cif}, \texttt{Num}\}$$
$$T = \{0,1,2,3,4,5,6,7,8,9\}$$
$$\texttt{Cif} ::= 0|1|2|3|4|5|6|7|8|9.$$
$$\texttt{Num} ::= \texttt{Cif}|\texttt{NumCif}.$$

Corrispondente al sistema induttivo

$$\frac{}{0:\texttt{Cif}} \quad \frac{}{1:\texttt{Cif}} \quad \frac{}{2:\texttt{Cif}} \quad \frac{}{3:\texttt{Cif}} \quad \frac{}{4:\texttt{Cif}} \quad \frac{}{5:\texttt{Cif}} \quad \frac{}{6:\texttt{Cif}}$$
$$\frac{}{7:\texttt{Cif}} \quad \frac{}{8:\texttt{Cif}} \quad \frac{}{9:\texttt{Cif}} \quad \frac{c:\texttt{Cif}}{c:\texttt{Num}} \quad \frac{c:\texttt{Cif}\ n:\texttt{Num}}{nc:\texttt{Num}}$$

e alla segnatura



Il linguaggio definito è:

$$L_{\texttt{Cif}} = \{0,1,2,3,4,5,6,7,8,9\}$$
$$L_{\texttt{Num}} = L^{+}_{\texttt{Cif}}$$

con l'interpretazione algebrica delle operazioni

$$0^A = 0; 1^A = 1; 2^A = 2; 3^A = 3; 4^A = 4; 5^A = 5; 6^A = 6; 7^A = 7; 8^A = 8; 9^A = 9$$
$$op_1^A(c) = c; op_2^A(n,c) = n \cdot c$$

L'algebra della semantica in cui vogliamo interpretare il linguaggio è (ovviamente)

$$\texttt{Cif}^B = [\overline{0}, \overline{9}]$$
$$\texttt{Num}^B = \mathbb{N}$$
$$0^B = \overline{0}; 1^B = \overline{1}; 2^B = \overline{2}; 3^B = \overline{3}; 4^B = \overline{4}; 5^B = \overline{5}; 6^B = \overline{6}; 7^B = \overline{7}; 8^B = \overline{8}; 9^B = \overline{9}$$
$$op_1^B(x) = x; op_2^B(n,c) = n\boxed{*}10\boxed{+}c$$

dove la sopralineatura è stata aggiunta per distinguere i numeri dalle cifre e $\square$ è stato aggiunto per ricordare che si tratta delle funzioni sui numeri.[1]

[1] Si noti che se invece della metaregola $(\{n:\texttt{Num}, c:\texttt{Cif}\}, nc:\texttt{Num}, \emptyset)$ avessimo scelto la metaregola $(\{n:\texttt{Num}, c:\texttt{Cif}\}, cn:\texttt{Num}, \emptyset)$, che genera esattamente lo stesso linguaggio, per definire $op_2^B(c,n)$ avremmo avuto qualche difficoltà, dovendo moltiplicare c per 10 elevato alla "lunghezza" di n, cioè avremmo dovuto definire $op_2^B(n,c) = c * 10^{(n+1\,\mathbf{div}\,10)}$. Questo è solo un esempio del fatto che sistemi di inferenza, o strutture a regole, che definiscono lo stesso linguaggio non sono poi equivalenti nel senso che risultano più o meno comodi per definire funzioni il cui dominio è il linguaggio, o per fare prove per induzione.

L'interpretazione semantica, indicata $\underset{c}{\rightarrow}$ per le cifre e $\underset{\mathcal{N}}{\rightarrow}$ per i numeri, è definita, seguendo lo stile della definizione del suo dominio, dalle seguenti metaregole.

$$\frac{}{0\underset{c}{\rightarrow}\overline{0}} \quad \frac{}{1\underset{c}{\rightarrow}\overline{1}} \quad \frac{}{2\underset{c}{\rightarrow}\overline{2}} \quad \frac{}{3\underset{c}{\rightarrow}\overline{3}} \quad \frac{}{4\underset{c}{\rightarrow}\overline{4}} \quad \frac{}{5\underset{c}{\rightarrow}\overline{5}} \quad \frac{}{6\underset{c}{\rightarrow}\overline{6}}$$
$$\frac{}{7\underset{c}{\rightarrow}\overline{7}} \quad \frac{}{8\underset{c}{\rightarrow}\overline{8}} \quad \frac{}{9\underset{c}{\rightarrow}\overline{9}} \quad \frac{c\underset{c}{\rightarrow}x}{c\underset{\mathcal{N}}{\rightarrow}x} \quad \frac{c\underset{c}{\rightarrow}x\ \ n\underset{\mathcal{N}}{\rightarrow}y}{nc\underset{\mathcal{N}}{\rightarrow}x\boxed{*}10\boxed{+}y}$$

Il fatto che le regole siano le stesse e che il risultato dell'interpretazione semantica definita da una metaregola sia proprio la semantica dell'operazione algebrica associata alla metaregola stessa nell'algebra della semantica, garantisce che $h = \{\mathcal{C}, \mathcal{N}\}$, se è una famiglia di funzioni, sia anche un omomorfismo.

Inoltre, siccome il sistema induttivo che definisce il linguaggio è non ambiguo, anche il sistema che definisce l'interpretazione semantica è non ambiguo (perché sono in pratica lo stesso) e quindi h è una funzione.

Esempio 2.2 Supponiamo però di arricchire il linguaggio con il tipo espressioni intere e le operazioni di somma e prodotto (con parentesi per poter esprimere sia $3+2*4$ che $(3+2)*4$).

Quindi a tutto quanto fatto prima si aggiungono:

- alla struttura a regole
$$S = \{\ldots\texttt{EI}\}$$
$$T = \{\ldots +, *, (,)\}$$
$$\texttt{EI} ::= \texttt{Num}|(\texttt{EI})|\texttt{EI}+\texttt{EI}|\texttt{EI}*\texttt{EI}.$$

- al sistema induttivo
$$\frac{n:\texttt{Num}}{n:\texttt{EI}} \quad \frac{e:\texttt{EI}}{(e):\texttt{EI}} \quad \frac{e_1,e_2:\texttt{EI}}{e_1+e_2:\texttt{EI}} \quad \frac{e_1,e_2:\texttt{EI}}{e_1*e_2:\texttt{EI}}$$

- alla segnatura un tipo, EI, e quattro operazioni, op_3: $\texttt{Num} \rightarrow \texttt{EI}$, op_4: $\texttt{EI} \rightarrow \texttt{EI}$ e op_5, op_6: $\texttt{EI}^2 \rightarrow \texttt{EI}$

- Il linguaggio di tipo EI è l'insieme delle espressioni aritmetiche nell'usuale notazione;

- l'interpretazione algebrica delle nuove operazioni
$$op_3^A(n) = n; \qquad op_4^A(n) = (n); \qquad op_5^A(e_1,e_2) = e_1 \cdot + \cdot e_2; \qquad op_6^A(e_1,e_2) = e_1 \cdot * \cdot e_2$$

- nell'algebra della semantica il tipo EI è interpretato dai naturali e le operazioni sono interpretate nel modo ovvio, cioè
$$op_3^B(n) = n; \qquad op_4^B(x) = x; \qquad op_5^E(x_1,x_2) = x_1\boxed{+}x_2; \qquad op_6^B(e_1,e_2) = x_1\boxed{*}x_2$$

Allora seguendo lo stesso ragionamento l'interpretazioni semantica andrebbe arricchita da:

$$\frac{n\underset{\mathcal{N}}{\rightarrow}x}{n\underset{\varepsilon}{\rightarrow}x} \quad \frac{e\underset{\varepsilon}{\rightarrow}x}{(e)\underset{\varepsilon}{\rightarrow}x} \quad \frac{e_1\underset{\varepsilon}{\rightarrow}x_1\ \ e_2\underset{\varepsilon}{\rightarrow}x_2}{e_1+e_2\underset{\varepsilon}{\rightarrow}x_1\boxed{+}x_2} \quad \frac{e_1\underset{\varepsilon}{\rightarrow}x_1\ \ e_2\underset{\varepsilon}{\rightarrow}x_2}{e_1*e_2\underset{\varepsilon}{\rightarrow}x_1\boxed{*}x_2}$$

Ma in questo modo ad esempio $3+2*5$ può essere valutata (cioè avere interpretazione semantica) in due modi:

$$\dfrac{3+2*5\xrightarrow[\mathcal{E}]{}\overline{25}=\overline{5}\boxed{*}\overline{5}}{\dfrac{3+2\xrightarrow[\mathcal{E}]{}\overline{5}=\overline{3}\boxed{+}\overline{2}}{\dfrac{3\xrightarrow[\mathcal{E}]{}\overline{3}}{\dfrac{3\xrightarrow[\mathcal{N}]{}\overline{3}}{\underline{3\xrightarrow[\mathcal{C}]{}\overline{3}}}}\quad\dfrac{2\xrightarrow[\mathcal{E}]{}\overline{2}}{\dfrac{2\xrightarrow[\mathcal{N}]{}\overline{2}}{\underline{2\xrightarrow[\mathcal{C}]{}\overline{2}}}}}\qquad\dfrac{5\xrightarrow[\mathcal{E}]{}\overline{5}}{\dfrac{5\xrightarrow[\mathcal{N}]{}\overline{5}}{\underline{5\xrightarrow[\mathcal{C}]{}\overline{5}}}}}$$

oppure

$$\dfrac{3+2*5\xrightarrow[\mathcal{E}]{}\overline{13}=\overline{3}\boxed{+}\overline{10}}{\dfrac{3\xrightarrow[\mathcal{E}]{}\overline{3}}{\dfrac{3\xrightarrow[\mathcal{N}]{}\overline{3}}{\underline{3\xrightarrow[\mathcal{C}]{}\overline{3}}}}\qquad\dfrac{2*5\xrightarrow[\mathcal{E}]{}\overline{10}=\overline{2}\boxed{*}\overline{5}}{\dfrac{2\xrightarrow[\mathcal{E}]{}\overline{2}}{\dfrac{2\xrightarrow[\mathcal{N}]{}\overline{2}}{\underline{2\xrightarrow[\mathcal{C}]{}\overline{2}}}}\quad\dfrac{5\xrightarrow[\mathcal{E}]{}\overline{5}}{\dfrac{5\xrightarrow[\mathcal{N}]{}\overline{5}}{\underline{5\xrightarrow[\mathcal{C}]{}\overline{5}}}}}}$$

Il problema è che, essendo ambiguo il linguaggio che definisce le espressioni, il modo ovvio di definire l'interpretazione semantica è scorretto in quanto definisce una relazione che non è una funzione (ma se lo fosse sarebbe un omomorfismo).

Per risolvere questo problema ci sono due possibilità.

Complicare il linguaggio Si tratta di rendere non ambiguo il linguaggio in modo da essere nella stessa situazione vista per il linguaggio dei numeri. Per farlo ci sono sostanzialmente due modi. Il primo corrisponde, ad esempio, ad aggiungere parentesi (o qualunque altro meccanismo per dire "inizio" e "fine") attorno a somma e prodotto in modo che il linguaggio di tipo `EI` non sia ambiguo. Questa soluzione non è accettabile, perché complica la vita dell'utente del linguaggio, cioè del committente, abituato ad usare una sintassi ambigua la cui semantica è resa non ambigua mediante la regola di precedenza: si fanno prima i prodotti e poi le somme. Alternativamente si può complicare il linguaggio introducendo un nuovo tipo, i fattori, fare le somme fra espressioni, i prodotti fra fattori e si dice che ogni fattore è un'espressione, mentre per trasformare le espressioni in fattori bisogna incapsularle fra parentesi. Questa tecnica è sicuramente migliore della precedente, perché l'utente potrebbe non accorgersi neppure della modifica (se non guarda il manuale, ma prova semplicemente a scrivere le espressioni com'è abituato a fare), ma rende sicuramente la vita più scomoda a chi deve implementare un compilatore, perché si trova a dover fare l'analisi sintattica di un tipo in più, quindi, se si pensa che ogni tipo del linguaggio debba avere una funzione (mutuamente ricorsiva con le altre) che ne analizza la correttezza (ovvero l'appartenenza di una stringa al linguaggio di quel tipo) ogni nuovo tipo richiede una parte di lavoro in più.

Complicare la definizione della interpretazione semantica Questa soluzione corrisponde a rendere formale nella definizione di semantica la regola di precedenza nota all'utente finale ed è sicuramente la più conveniente, dal punto di vista del fatto che si complica il lavoro del progettatore "matematico" del linguaggio, che dovrebbe essere il più esperto e che, comunque, fa il lavoro una volta sola, al momento della definizione del linguaggio, mentre gli implementatori possono essere (e di solito sono) molto numerosi e gli utenti, sperabilmente, ancora di più.

Per adottare questa soluzione, l'intuizione è che il primo albero di valutazione deve essere inaccettabile, perché voglio fare prima i prodotti, quindi devo restringere l'applicazione della regola per la valutazione del prodotto in modo che si possa applicare solo quando non ci sono somme al livello più esterno (altrimenti siamo nel caso $x+y*z$ e vogliamo applicare la regola della somma a x e $y*z$). Perciò bisogna aggiungere una condizione a lato alla regola del prodotto che valga vero soltanto se nelle premesse non compaiono somme (così $x+y*z$ non può essere valutato come la valutazione di $x+y$ moltiplicata per la valutazione di z).

Esempio 2.3 Anzitutto, quindi, definiamo un predicato Q sul linguaggio di tipo `EI` che valga vero sulle espressioni che non contengono $+$ a livello esterno. Siccome un predicato è una particolare funzione a valori verità, possiamo definire anche Q mediante una tecnica induttiva ricalcata dalle regole che definiscono il linguaggio. Se ricalcassimo esattamente la definizione del linguaggio, nel caso particolare della regola che trasforma un numero in un'espressione, dovremmo estendere la definizione di Q in modo che lavorasse anche sui numeri. Però se un'espressione è un numero Q vale sicuramente vero (non ci sono $+$ a livello esterno in un numero), quindi possiamo omettere le premesse. Analogamente le regole si semplificano nel caso delle regole per somma (se sto aggiungendo un $+$ ovviamente c'e' un $+$ a livello esterno, quindi Q non vale) e per l'aggiunta di parentesi (se sto incapsulando l'espressione fra parentesi ogni eventuale $+$ verrà nascosto, quindi Q vale).

$$\dfrac{}{n\xrightarrow[Q]{}T}\ n\in L_{\mathtt{Num}}\qquad\dfrac{}{(e)\xrightarrow[Q]{}T}\qquad\dfrac{}{e_1+e_2\xrightarrow[Q]{}F}\qquad\dfrac{e_1\xrightarrow[Q]{}b_1\quad e_2\xrightarrow[Q]{}b_2}{e_1*e_2:b_1\wedge b_2}$$

Adesso si sostituisce la metaregola

$$\dfrac{e_1\xrightarrow[\mathcal{E}]{}x_1\ e_2\xrightarrow[\mathcal{E}]{}x_2}{e_1*e_2\xrightarrow[\mathcal{E}]{}x_1\boxed{*}x_2}$$

con la versione di applicazione ristretta

$$\dfrac{e_1\xrightarrow[\mathcal{E}]{}x_1\ e_2\xrightarrow[\mathcal{E}]{}x_2}{e_1*e_2\xrightarrow[\mathcal{E}]{}x_1\boxed{*}x_2}\quad Q(e_1)\wedge Q(e_2)$$

La funzione così definita, però, è un omomorfismo? nel caso generale no, perché ad esempio $h(op_6^A(op_5^A(3,2),5)) = h(3+2*5) = \overline{13}$, mentre $h(op_5^A(3,2)) = h(5) = \overline{5}$ e $op_6^B(\overline{5},\overline{5}) = \overline{25}$. Il punto cruciale è che la stringa $3+2*5$ la vogliamo vedere *solo* come $3+\overbrace{2*5}$ e non come $\overbrace{3+2}*5$, cioè vorremmo che op_6^A su $3+2$ e 5 fosse indefinita. Si tratta quindi di ridurre l'algebra della sintassi, che viene definita come un'algebra totale mediante la struttura a regole, ad un'algebra *parziale* in cui, cioè, l'interpretazione delle funzioni è parziale. Per fare questo la prima possibilità è descrivere la sintassi non mediante una struttura a regole, ma mediante un sistema induttivo generale (con condizioni a lato, esattamente le condizioni che si usano per definire la semantica). Questa scelta di solito *non* viene adottata, perché si preferisce mantenere la descrizione dei linguaggi quanto più semplice possibile, ma si preferisce interporre un secondo passo, di *semantica statica*, con cui si riduce l'algebra della sintassi ad un'algebra parziale e poi si dà la semantica come omomorfismo di algebre parziali. Nella sezione seguente si vedrà un esempio esteso.

Si noti che questo caso ben esemplifica la tecnica di definizione induttiva di funzioni:

1. si ricalca il sistema che definisce il dominio della funzione;

2. si semplificano le regole omettendo tutte le premesse che non sono necessarie al fine di calcolare il risultato (cioè, visivamente, tutte quelle del tipo $a\to b$ in cui la b non compare nelle conseguenze);

3. si complicano le regole in cui il risultato sulla conseguenza dipende dalla forma degli argomenti o dividendo in vari casi la regola o aggiungendo un predicato.

Ad esempio per definire induttivamente la funzione **mod**3 si parte dal sistema

$$\frac{}{0 \to 0} \qquad \frac{n \to x}{n+1 \to ?}$$

dove non so attribuire un valore nella conseguenza della seconda metaregola, perché dipende da se $x = 0$ (quindi $? = 1$) o $x = 1$ (quindi $? = 2$) 0 $x = 2$ (quindi $? = 0$). Perciò bisogna dividere il secondo caso in

$$\frac{n \to 0}{n+1 \to 1} \qquad \frac{n \to 1}{n+1 \to 2} \qquad \frac{n \to 2}{n+1 \to 0}$$

Riassunto delle puntate precedenti La fase di progettazione di un linguaggio consiste di

- Scelta di un oggetto concreto da descrivere, ovvero del fenomeno per "parlare" del quale il linguaggio deve fornire gli strumenti. Solitamente questo oggetto consiste di due parti: una famiglia di insiemi di elementi, divisi a seconda del loro tipo e una famiglia di funzioni (tipate) che manipolano questi elementi.
- Descrizione della sintassi; a sua volta articolata in
  - scelta del lessico o vocabolario (cioè dell'universo, usando la terminologia induttiva, o dei simboli terminali, usando la terminologia delle strutture a regole);
  - scelta della "grammatica" del linguaggio, cioè delle regole di costruzione del linguaggio, ovvero delle metaregole (o delle regole di struttura) che lo definiscono;

  La definizione di una struttura a regole implicitamente fissa anche una struttura algebrica a due livelli: la scelta dei non terminali e dei tipi di produzioni (cioè di voler costruire un oggetto di tipo s_{n+1} a partire da oggetti di tipo $s_1 \dots s_n$) rappresenta la scelta di una segnatura, mentre la scelta delle stringhe effettivamente usate nelle produzioni fissa la rappresentazione delle operazioni della segnatura nell'algebra della sintassi.
- Descrizione formale della semantica = definizione di un'algebra sulla segnatura introdotta mediante la sintassi
- Definizione *composizionale* della semantica (ovvero dell'interpretazione o valutazione) = omomorfismo dall'algebra della sintassi nell'algebra della semantica.

3 Semantica statica

Nella prassi la sintassi dei linguaggi viene data, per ragioni di semplicità, mediante una grammatica BNF (una struttura a regole); questo metodo, però, limita le capacità espressive del linguaggio descritto, nel senso che non tutti gli insiemi di stringhe che possono essere riconosciuti da un compilatore (cioè per cui si può scrivere un programma che legge una stringa e risponde "vero" se la stringa appartiene al linguaggio e "falso" altrimenti) possono essere descritti come linguaggi generati da una grammatica. Per di più anche nei casi in cui il linguaggio si può descrivere mediante una grammatica, la corrispondente struttura a regole è troppo complicata, a causa della necessità di introdurre nuovi tipi e operatori per descrivere il linguaggio (si provi ad esempio a descrivere l'insieme delle stringhe su $\{a, b\}$ che contengono tante a quante b mediante una grammatica e ci si renderà conto che non è possibile farlo senza utilizzare due tipi ausiliari, l'insieme delle stringhe che hanno un a in più e quello delle stringhe che hanno una b in più).

Per ovviare a questo, invece di descrivere la sintassi del linguaggio che si vuole progettare semplicemente mediante una BNF, si procede in due tempi: prima si definisce una grammatica che ritaglia, all'interno di tutte le stringhe un insieme A, poi si definisce (induttivamente) un predicato "isok" sull'insieme A il cui insieme di verità (cioè il sottoinsieme di A su cui isok vale) corrisponde al linguaggio che si voleva descrivere. Mentre la definizione della grammatica è guidata, usualmente, solo da criteri sintattici (nomi dei tipi e nomi delle operazioni che vogliamo rappresentare su di essi), la decisione di quali elementi del linguaggio vogliamo tenere e quali vogliamo buttare è guidata dall'intuizione di quali elementi rappresentano un valore e quali sono finiti nel linguaggio "per errore". Per questo motivo il secondo passo nella definizione della sintassi prende usualmente il nome di *semantica statica* pur essendo puramente sintattico.

Nell'esempio delle espressioni intere della sezione precedente la situazione è lievemente differente, perché l'algebra della sintassi viene resa parziale non riducendone i carriers, che rimangono uguali, ma riducendo l'interpretazione delle operazioni, cioè si passa da A a SA, dove i carriers e l'interpretazione di op_i per $i = 1, \dots, 5$ restano le stesse, ma l'interpretazione di op_6^{SA} è data da

$$op_6^{SA}(u, v) = \begin{cases} u \cdot * \cdot v & \text{se } Q(u) \wedge Q(v) \\ \text{indefinito} & \text{altrimenti} \end{cases}$$

cioè è descritta induttivamente da

$$\frac{}{<e_1, e_2> \to_{op_6^{SA}} e_1 * e_2} \quad Q(e_1) \wedge Q(e_2)$$

Questo tipo di problematica si ha ogniqualvolta si descrivono operatori con "precedenze". Provare per esercizio a descrivere le precedenze fra gli operatori booleani (**not** prevale su **and** che a sua volta prevale su **or**).

Come esempio rilevante di necessità di introduzione della semantica statica consideriamo la problematica dovuta all'introduzione degli identificatori. Si faccia attenzione al fatto che, proprio perché si tratta di un esempio "reale" e interessante, si verranno a sovrapporre problemi e dettagli specifici del caso in esame a tematiche relative al trattamento generale della semantica statica. Entrambi i livelli sono rilevanti ed interessanti, ma bisogna saperli distinguere.

3.1 Identificatori

Perché introdurre gli identificatori in un linguaggio? per leggibilità ed efficienza. Si consideri ad esempio l'espressione numerica

$$(3+2*5+7*9)*7+3+2*5+7*9+(3+2*5+7*9)*(3+2*5+7*9)*(3+2*5+7*9)$$

e la si confronti con una sua rappresentazione alternativa in uno stile matematico abbastanza usuale

$$\text{Sia } x = 3+2*5+7*9 \text{ in } x*7+x+x*x*x$$

I vantaggi immediati della seconda rappresentazione sono due: anzitutto la struttura della formula viene messa in evidenza e quindi è più facile leggere e più difficile sbagliare, ad esempio, nel ricopiare. Inoltre il calcolo $x = 3+2*5+7*9$ può essere eseguito una volta e utilizzato nelle 5 occorrenze della x.

Perciò la possibilità di dare un nome a pezzi di espressione non solo facilita la comprensione del calcolo, ma effettivamente sveltisce il calcolo. Supponiamo, allora di voler introdurre un meccanismo analogo nel linguaggio delle espressioni.

Il primo approccio "brutale" è aggiungere un tipo (cioè un non terminale) per rappresentare gli identificatori e una produzione che dice che una delle possibili forme di un'espressione è un identificatore. Come forma accettabile per gli identificatori usiamo (ma è una scelta arbitraria e irrilevante, se ne potevano fare altre) quella accettata da gran parte dei linguaggi imperativi, cioè sequenze alfanumeriche comincianti con una lettera (in modo da facilitare l'analisi sintattica ad un compilatore).

Si noti la differenza fra il simbolo terminale **if** e l'identificatore `if`; il primo è un simbolo atomico appartenente all'alfabeto dei terminali mentre il secondo è una stringa di lunghezza 2 sull'alfabeto dei terminali. Il fatto che vengano "disegnati" nello stesso modo è un semplice incidente, dovuto al fatto che i dati vengono introdotti in un calcolatore mediante una tastiera che offre solo tasti alfanumerici (o di controllo); si potrebbe immaginare di avere un calcolatore progettato apposta per il nostro linguaggio, con sulla tastiera tasti appositi per le parole chiave. In tal caso il simbolo terminale **if** verrebbe introdotto premendo il tasto contrassegnato **if** , mentre l'identificatore `if` verrebbe introdotto premendo in successione i tasti contrassegnati con `i` e con `f`.

Per abbreviare la descrizione, risulta conveniente anche introdurre il tipo delle "lettere" ma questo è dovuto alla scelta fatta sulla forma ammissibile per gli identificatori e quindi non è rilevante; si tratta solo di una semplificazione notazionale, senza controparte nel significato di ciò che vogliamo fare.

$$S = \{\ldots \texttt{Ide}, \texttt{Lettere}\}$$

$$T = \{\ldots a, b, c \ldots z\}$$

$$\texttt{Lettere} ::= a | \ldots | z.$$

$$\texttt{Ide} ::= \texttt{Lettere} | \texttt{Ide Lettere} | \texttt{Ide Cif}.$$

$$\texttt{EI} ::= \ldots | \texttt{Ide}.$$

In questo modo, però, ci troveremmo ad avere espressioni del tipo $3 + a$ a cui dover dare una semantica; per di più, avendo un meccanismo analogo per le espressioni booleane, anche espressioni del tipo **if** b **then** $b + 2$ **else** 6 apparterrebbero al linguaggio. Questi due esempi propongono due tipi di problematiche:

- espressioni intuitivamente corrette, o per lo meno che potrebbero essere corrette, necessiteranno di un meccanismo ausiliario per poter ottenere una semantica. Cioè ci vorrà un modo per sapere il valore rappresentato da a in $3 + a$ per poter fare la somma.

- alcune espressioni che risultano corrette rispetto ai meccanismi di definizione del linguaggio sono prive di senso e devono essere rifiutate. Inoltre per determinare se un'espressione è "sensata" ci basterebbe poter distinguere fra identificatori booleani e interi, cioè ci basterebbe sapere il tipo degli identificatori.

Lasciando per un momento da parte il primo problema, il secondo potrebbe ad esempio risolversi cambiando la forma degli identificatori in modo tale che basti guardare com'è fatto un identificatore per sapere se rappresenta un intero o un booleano. Ad esempio, potremmo stabilire che abbiamo non uno ma due tipi di identificatori come segue

$$S = \{\ldots \texttt{Ide}_I, \texttt{Ide}_B, \texttt{Lettere}\}$$

$$T = \{\ldots a, b, c \ldots z\}$$

$$\texttt{Lettere} ::= a | \ldots | z.$$

$$\texttt{Ide}_I ::= n | \texttt{Ide}_I \ \texttt{Lettere} | \texttt{Ide}_I \ \texttt{Cif}.$$

$$\texttt{EI} ::= \ldots | \texttt{Ide}_I.$$

$$\texttt{Ide}_B ::= b | \texttt{Ide}_B \ \texttt{Lettere} | \texttt{Ide}_B \ \texttt{Cif}.$$

$$\texttt{EB} ::= \ldots | \texttt{Ide}_B.$$

cioè gli identificatori interi iniziano con n e quelli booleani con b. A parte l'ovvia scomodità di una siffatta regola quando il linguaggio permetta di usare nomi significativi come identificatori, un meccanismo di questo tipo può essere effettivo solo finché i tipi degli identificatori sono una quantità finita (e si noti che se i tipi sono numerosi il meccanismo diventa molto farraginoso), ma diventa inapplicabile non appena ci si trovi a dover distinguere una quantità infinita di classi di identificatori, ad esempio non appena si ammettano identificatori come nomi di funzione. Pertanto questa soluzione non ha trionfato anzi è stata applicata, e solo in parte, esclusivamente agli albori dei linguaggi di programmazione.

Un'altra tecnica per sapere di che tipo sono gli identificatori è introdurre il concetto di *dichiarazione*. Ovvero si vuole arricchire il linguaggio con un meccanismo di creazione di *ambienti* locali, cioè di aree del programma in cui sono noti i tipi degli identificatori. In questo modo $3 + a$ sarà un'espressione corretta in un ambiente in cui a sia dichiarato di tipo intero, mentre sarà scorretto in un ambiente in cui a è dichiarato di tipo booleano o non è dichiarato affatto (o è un nome di funzione). Pertanto la correttezza di un'espressione, che fino a questo momento era un concetto assoluto, cioè una stringa rappresentava sempre un'espressione corretta o era sempre scorretta, diventa, a partire da questo momento, un concetto relativo al *contesto* in cui si incontra l'espressione, cioè all'ambiente in cui viene presa in considerazione.

Inoltre lo stesso meccanismo di introduzione degli identificatori mediante dichiarazione sarà anche utile al momento di attribuire una semantica alle espressioni. Infatti le dichiarazioni servono a creare un ambiente, cioè una specie di tabella di associazione fra nomi e valori, che permette di valutare un'espressione, sostituendo ad ogni simbolo di operazione la corrispondente operazione semantica e ad ogni identificatore il suo valore.

Quindi vorremo che la dichiarazione di un identificatore memorizzi in qualche modo 3 informazioni: il nome dell'identificatore che stiamo definendo, il suo tipo e il valore corrispondente. L'associazione nome-tipo servirà a stabilire un criterio di correttezza sintattica ed escludere espressioni tipo **if** b **then** $b + 2$ **else** 6, cioè a definire la *semantica statica* del linguaggio, mentre l'associazione nome-valore servirà al momento della valutazione semantica. La sintassi scelta è un miscuglio di notazioni presenti in vari linguaggi e non corrisponde esattamente a nessuno.

$S = \{\ldots \texttt{Lettere}, \texttt{Ide}, \texttt{Dec}\}$

$T = \{\ldots a, b, c \ldots z, ':', ';', '\}$

$\texttt{Lettere} ::= a | \ldots | z.$

$\texttt{Ide} ::= \texttt{Lettere} | \texttt{Ide Lettere} | \texttt{Ide Cif}.$

$\texttt{Dec} ::= \texttt{Ide}: \textbf{ int } = \texttt{EI} | \texttt{Ide}: \textbf{ bool } = \texttt{EB} | \texttt{Dec}; \texttt{Ide}: \textbf{ int } = \texttt{EI} | \texttt{Dec}; \texttt{Ide}: \textbf{ bool } = \texttt{EB}.$

$\texttt{EI} ::= \ldots | \texttt{Ide} | \textbf{ let } \texttt{Dec} \textbf{ in } \texttt{EI} \textbf{ endlet } \ldots$

$\texttt{EB} ::= \ldots | \texttt{Ide} | \textbf{ let } \texttt{Dec} \textbf{ in } \texttt{EB} \textbf{ endlet } \ldots$

quindi ad esempio $\textbf{let } \texttt{inutile}: \textbf{ int } = 3 \textbf{ in } 5 + \texttt{maidichiarato} \textbf{ endlet}$ è un'espressione intera, ma ovviamente non vogliamo (o meglio non possiamo) attribuirle alcuna semantica

3.1.1 Regole di semantica statica

Cominciamo (contrariamente a quanto visto a lezione) con il delimitare, mediante regole di semantica statica, a quale insieme di espressioni vorremo dare una semantica e quindi quali espressioni vogliamo tenere e quali vogliamo buttare dalla nostra sintassi.

L'idea è di fare una sorta di valutazione formale delle espressioni calcolandone non il valore ma il tipo (quindi restiamo nel mondo della sintassi), che è l'unica cosa che ci interessa ai fini di stabilire se la possiamo usare per costruire altre espressioni instanziando le produzioni. Quindi abbiamo bisogno del concetto di "ambiente semplificato", cioè di tabella di associazione identificatori-tipi, per poter interpretare ogni nuova dichiarazione come un aggiornamento dell'ambiente corrente, cioè come una funzione che, preso come argomento l'ambiente corrente, ne produce uno nuovo, aggiornato. Poi, utilizzando la nozione di ambiente, potremo stabilire la correttezza delle espressioni.

Si noti che questa descrizione del modo di procedere sembra suggerire due passi *indipendenti*, cioè prima la definizione degli ambienti a partire dalle dichiarazioni (fatto e finito) e poi la verifica di correttezza delle espressioni usando gli ambienti. Ma questo è vero solo analizzando una particolare espressione, mentre la definizione globale della costruzione degli ambienti e della correttezza statica sono mutuamente ricorsive, perché ci sono produzioni di tipo espressione che usano argomenti di tipo dichiarazione e viceversa; infatti per stabilire se $\textbf{let } a: \textbf{ int } = 3 + b \textbf{ in } 5 + b \textbf{ endlet}$ è corretta come espressione, devo già sapere se $a: \textbf{ int } = 3 + b$ genera un ambiente corretto, ma per questo devo sapere se l'espressione $3 + b$ è corretta e così via.

Anzitutto dobbiamo definire cos'è un ambiente per la semantica statica, cioè cos'è una tabella; se una tabella associa nomi a tipi sarà una funzione (parziale) dal linguaggio degli identificatori all'insieme dei tipi possibili il cui dominio (finito) è l'insieme dei nomi che compaiono nella tabella.

$$\mathrm{SE} = [L_{\texttt{Ide}} \xrightarrow[p]{} \texttt{Tipi}] \qquad \text{dove } \texttt{Tipi} = \{ \textbf{ int }, \textbf{ bool } \}$$

Note

- Se il nostro linguaggio avesse più tipi tutti i ragionamenti seguenti risulterebbero inalterati e l'unica variazione richiesta sarebbe l'aggiunta, in questo punto, dei nomi degli altri tipi all'insieme $\texttt{Tipi}$. Inoltre, ovviamente, ci sarebbero altre produzioni nella grammatica per permettere la dichiarazione di identificatori degli altri tipi e quindi bisognerebbe aggiungere regole analoghe a quelle che daremo nei casi di dichiarazione di interi e booleani.
- Se la forma ammessa per gli identificatori fosse diversa, cioè se $L_{\texttt{Ide}}$ fosse generato da un'altra grammatica, formalmente non cambierebbe nulla, cioè resterebbe scritto $L_{\texttt{Ide}}$ come dominio di un generico ambiente, ma l'insieme rappresentato sarebbe diverso. Il fatto che non ci sarebbe niente da cambiare sottolinea l'ininfluenza della scelta operata.

Fissato un ambiente, ogni espressione deve essere associata al suo tipo, se corretta, o ad un simbolo rappresentante l'errore. Analogamente ogni dichiarazione deve modificare l'ambiente, eventualmente sollevando un errore. Nel momento in cui si verifica un errore (sia sulle espressioni o sulle dichiarazioni) questo deve essere poi propagato da ogni successiva analisi, quindi ogni regola si dividerà in due casi: se l'ambiente costruito fino a quel momento è un vero ambiente (e in tal caso verrà arricchito o verrà prodotto un errore a seconda della dichiarazione) oppure se è un errore (nel qual caso, indipendentemente dalla dichiarazione o dalla espressione in esame, l'errore deve essere propagato).

Definiamo induttivamente le funzioni parziali $\mathcal{D}_s: L_{\texttt{Dec}} \times (\mathrm{SE} \cup \{\texttt{err}\}) \to \mathrm{SE} \cup \{\texttt{err}\}$ e $\mathcal{E}_s: (L_{\texttt{EI}} \cup L_{\texttt{EB}}) \times \mathrm{SE} \cup \{\texttt{err}\} \to \texttt{Tipi} \cup \{\texttt{err}\}$, dando solo alcune regole, da completare per esercizio.

Nel seguito usiamo le metavariabili $id \in L_{\texttt{Ide}}$, $e \in L_{\texttt{EI}} \cup L_{\texttt{EB}}$, $sr, sr' \in \mathrm{SE}$, $T \in \{ \textbf{ int }, \textbf{ bool } \}$ e $sre, sre', sre'' \in \mathrm{SE} \cup \{\texttt{err}\}$.

Regole di costruzione dell'ambiente

$\dfrac{e, sr \xrightarrow[\mathcal{E}_s]{} \textbf{int}}{id: \textbf{ int } = e, sr \xrightarrow[\mathcal{D}_s]{} sr[\textbf{ int } / id]}$	se l'espressione che si vuole associare ad id è un'espressione corretta di tipo intero rispetto all'ambiente sr, la dichiarazione in sr è corretta ed il suo effetto è arricchire sr con l'informazione sul tipo di id
$\dfrac{}{id: \textbf{ int } = e, \texttt{err} \xrightarrow[\mathcal{D}_s]{} \texttt{err}}$	se l'ambiente corrente è scorretto, la dichiarazione deve propagare l'errore
$\dfrac{e, sre \xrightarrow[\mathcal{E}_s]{} \texttt{err}}{id: \textbf{ int } = e, sre \xrightarrow[\mathcal{D}_s]{} \texttt{err}}$	se l'espressione è scorretta, la dichiarazione deve propagare l'errore
$\dfrac{e, sr \xrightarrow[\mathcal{E}_s]{} \textbf{bool}}{id: \textbf{ int } = e, sr \xrightarrow[\mathcal{D}_s]{} \texttt{err}}$	se l'espressione è di tipo booleano, la dichiarazione è inconsistente, quindi si deve generare un errore
...	analoghe regole per il caso $id: \textbf{ bool } = e$
$\dfrac{d, sre \xrightarrow[\mathcal{D}_s]{} sre'; id: T = e, sre' \xrightarrow[\mathcal{D}_s]{} sre''}{d; id: T = e, sre \xrightarrow[\mathcal{D}_s]{} sre''}$	cioè le modifiche complessive si ottengono effettuando le modifiche dovute a ciascun pezzo della dichiarazione in sequenza

Un identificatore è un'espressione corretta del tipo di cui è dichiarato; siccome le dichiarazioni servono a generare l'ambiente, dire che un identificatore è stato dichiarato è equivalente a dire che appartiene al dominio dell'ambiente.

Regole di introduzione degli identificatori

$\dfrac{}{id, sr \xrightarrow[\mathcal{E}_s]{} T} \quad sr(id) = T$	Se id è dichiarato di tipo T, allora è un'espressione di tipo T
$\dfrac{}{id, sr \xrightarrow[\mathcal{E}_s]{} \texttt{err}} \quad id \notin dom(sr)$	L'uso di un identificatore non dichiarato costituisce un errore
$\dfrac{}{id, \texttt{err} \xrightarrow[\mathcal{E}_s]{} \texttt{err}}$	Propagazione di errore

Per stabilire se un'espressione è corretta e di che tipo in un ambiente, si deve dare una regola di propagazione di correttezza e errore per ogni operatore del linguaggio, ad esempio qui vediamo il caso per la somma.

Regole di propagazione del tipaggio

$\dfrac{e_1, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{int} \quad e_2, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{int}}{e_1 + e_2, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{int}}$	la somma di espressioni intere è un'espressione intera
$\dfrac{e_1, sre \underset{\mathcal{E}_s}{\rightarrow} \texttt{err}}{e_1 + e_2, sre \underset{\mathcal{E}_s}{\rightarrow} \texttt{err}}$	propagazione di errore come primo argomento, bisognerà aggiungere anche una regola per la propagazione di errore come secondo argomento
$\dfrac{e_1, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{bool}}{e_1 + e_2, sr \underset{\mathcal{E}_s}{\rightarrow} \texttt{err}}$	non si applicare la somma ad un valore booleano, bisognerà aggiungere anche una regola analoga per il secondo argomento.

Se ci fossero altri tipi, bisognerebbe aggiungere una coppia di regole analoghe per ogni tipo diverso dagli interi.

Costrutto let

$\dfrac{d, sre \underset{\mathcal{D}_s}{\rightarrow} sre' \quad e, sre' \underset{\mathcal{E}_s}{\rightarrow} T}{\textbf{let } d \textbf{ in } e \textbf{ endlet }, sre \underset{\mathcal{E}_s}{\rightarrow} T}$	

Con ciò abbiamo classificato espressioni e dichiarazioni in "buoni" e "cattivi" relativamente ad un ambiente; quindi le espressioni possono essere di 3 tipi:

- quelle che in ogni possibile ambiente producono errore, ad esempio **if** x **then** x **else** $x+1$ **fi** .
- quelle che sono sempre corrette, indipendentemente dall'ambiente usato per analizzarle, perché contengono al loro interno le dichiarazioni di tutti gli identificatori che usano, ad esempio **let** x : **int** $=4; y$: **bool** $=$ **true in if** y **then** x **else** $x+1$ **fi endlet** ; queste si chiamano *staticamente corrette* e rappresentano un valore. Corrispondono all'idea di programma in un linguaggio funzionale.
- quelle la cui correttezza dipende dall'ambiente usato per analizzarle, cioè per le quali esistono due ambienti tali che in uno l'espressione sia corretta e nel'altro no; ad esempio **let** x : **int** $=3$ **in if** y **then** x **else** $x+1$ **fi endlet** è corretta in tutti e soli gli ambienti in cui y è di tipo booleano. È indispensabile, affinché si verifichi questo caso che vi compaia un identificatore globale. Queste espressioni, insieme a quelle al punto precedente, si chiamano *tipabili* e rappresentano funzioni (costanti nel caso di espressioni staticamente corrette) che associano un valore ad un ambiente.

Vogliamo escludere dall'algebra della sintassi le espressioni non tipabili, cioè quelle sempre scorrette; da questa restrizione si ha che l'interpretazione delle operazioni, quindi, diventa parziale ogniqualvolta il risultato è stato "buttato via".

Si potrebbe anche pensare di restringere il carrier alle sole espressioni staticamente corrette, ma, per come è stata definita la correttezza, ci saranno espressioni staticamente corrette le cui sottoespressioni non sono staticamente corrette; ad esempio **let** a : **int** $=3$ **in** $a+5$ **endlet** è corretta, ma la sua sottoespressione $a+5$ non lo è, perché usa un identificatore (che non è noto nell'ambiente vuoto), pur essendo ben tipata nell'ambiente generato dalla dichiarazione a : **int** $=3$. Questo corrisponde al fatto che, mentre le espressioni tipabili sono esprimibili in maniera induttiva con un sistema di inferenza che rispetta la struttura algebrica introdotta dalla grammatica, le espressioni staticamente corrette non sono generate da un siffatto sistema. Quindi ci troveremmo ad avere nella sintassi un gran numero di oggetti che non sono generati a partire da sotto-oggetti e quindi non potremmo più sfruttare una definizione induttiva di semantica.

$$\texttt{EI}^{SA} = \{e \mid e \in \texttt{EI}^{A} \land \exists sr \text{ t.c. } e, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{int}\}$$

$$\texttt{EB}^{SA} = \{e \mid e \in \texttt{EB}^{A} \land \exists sr \text{ t.c. } e, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{bool}\}$$

$s^{SA} = s^{A}$ per ogni altro tipo

$op^{SA}(x_1, \ldots, x_n) = op^{A}(x_1, \ldots, x_n)$ se $op^{A}(x_1, \ldots, x_n) \in s^{SA}$, indefinito altrimenti

3.1.2 Ridichiarazione di identificatori

A lezione questa parte è stata introdotta, ma non completata; in particolare non si è vista alcuna delle regole per distinguere fra ambiente locale e globale; pertanto si tratta di un argomento non indispensabile al superamento dell'esame, ma necessario per ottenere un voto medio-alto.

Nel seguito del corso, però, si assumerà staticamente scorretta una redichiarazione di un identificatore dichiarato alla stesso livello di annidamento.

Siamo quindi riusciti ad eliminare dalla nostra sintassi le espressioni in cui lo stesso identificatore compariva in diverse occorrenze allo stesso livello con tipi diversi; cioè espressioni del tipo **if** a **then** $a+5$ **else** $a+3$ **fi** risultano effettivamente scorrette, mentre espressioni del tipo **let** a : **int** $=3$ **in let** a : **bool** $=$ **true in if** a **then** 5 **else** 3 **fi endlet** $+\,a$ **endlet** sono corrette, perché è lecito ridichiarare all'interno di un ambiente locale un identificatore anche di tipo diverso. Questo corrisponde all'idea che si possono riutilizzare gli stessi nomi, in contesti differenti, per rappresentare valori diversi. Ma si consideri una espressione del tipo **let** a : **int** $=3; a$: **bool** $=$ **true in if** a **then** 5 **else** 3 **fi endlet** ; in base a quanto visto finora è corretta, benché a sia dichiarato due volte nella stessa dichiarazione con significati differenti. Formalmente questa espressione risulta corretta, perché la seconda dichiarazione vanifica la prima (verificare applicando le regole), ma sembra più probabile ipotizzare un errore di battitura da parte dell'utente piuttosto che una deliberata ridichiarazione; sarebbe quindi più opportuno intercettare anche questo tipo di espressioni, a livello sintattico, e considerarle errate.

Per distinguere fra la ridichiarazione di un identificatore globale (che è ammissibile) e la duplicazione di dichiarazione (che vogliamo eliminare), bisognerà distinguere fra "ambiente globale" e "ambiente locale" e portarsi dietro le informazioni separatamente. Nel seguito dati due ambienti sr e sr' useremo $sr[sr']$ per indicare $sr[v/x \mid (x,v) \in sr']$, cioè l'aggiornamento di sr usando le informazioni contenute in sr' e useremo $sr[[v/x]]$ per indicare che l'aggiornamento viene fatto solo se x non appartiene già al dominio di sr, ovvero si tratta effettivamente di un identificatore nuovo, altrimenti $sr[[v/x]] = \texttt{err}$; analogamente $sr[[sr']]$ sarà $sr[[v/x \mid (x,v) \in sr']]$. Quindi le due notazioni rappresentano rispettivamente l'aggiornamento (sr[...]) e l'aggiornamento *senza ridichiarazioni* (sr[[...]]) di un ambiente mediante un altro.

Riprendiamo rapidamente quanto fatto e vediamo dove si devono modificare le regole. Le regole che definiscono il tipaggio delle espressioni non vengono modificate, perché ai fini di verificare la correttezza di un'espressione poco importa distinguere fra identificatori globali e locali. Invece le regole che definiscono la correttezza (e il risultato) di una dichiarazione andranno modificate come segue. Anzitutto la funzione $\mathcal{D}_s$ deve cambiare tipo per lavorare su coppie "ambiente locale - ambiente globale"; quindi avremo $\mathcal{D}_s\colon L_{\texttt{Dec}} \times (\mathrm{SE} \cup \{\texttt{err}\})^2 \rightarrow (\mathrm{SE} \cup \{\texttt{err}\})^2$. Adottiamo la stessa convenzione sui nomi delle metavariabili, ma supponiamo che sr sia una coppia $<sr_g, sr_l>$ di ambienti, il primo "globale" e il secondo "locale" e usiamo $\bar{sr}$ per indicare $sr_g[sr_l]$.

$\dfrac{e, \bar{sr} \xrightarrow[\mathcal{E}_s]{} \mathbf{int}}{id : \mathbf{int} = e, sr \xrightarrow[\mathcal{D}_s]{} <sr_g, sr_l[[\,\mathbf{int}\,/id]]>}$	se id già appartiene all'ambiente locale $sr_l[[\,\mathbf{int}\,/id]] = \mathtt{err}$, quindi non si possono duplicare le dichiarazioni *allo stesso livello*
$\dfrac{}{id : \mathbf{int} = e, <\mathtt{err}, sre_l> \xrightarrow[\mathcal{D}_s]{} <\mathtt{err}, sre_l>}$	propagazione dell'errore globale
$\dfrac{}{id : \mathbf{int} = e, <sre_g, \mathtt{err}> \xrightarrow[\mathcal{D}_s]{} <sre_g, \mathtt{err}>}$	propagazione dell'errore locale
$\dfrac{e, \bar{sre} \xrightarrow[\mathcal{E}_s]{} \mathtt{err}}{id : \mathbf{int} = e, sre \xrightarrow[\mathcal{D}_s]{} <sre_g, \mathtt{err}>}$	se l'espressione è scorretta, la dichiarazione deve propagare l'errore a livello locale
$\dfrac{e, \bar{sr} \xrightarrow[\mathcal{E}_s]{} \mathbf{bool}}{id : \mathbf{int} = e, sr \xrightarrow[\mathcal{D}_s]{} <sr_g, \mathtt{err}>}$	se l'espressione è di tipo booleano, la dichiarazione è inconsistente, quindi si deve generare un errore
$\ldots$	analoghe regole per il caso $id : \mathbf{bool} = e$
$\dfrac{d, sre \xrightarrow[\mathcal{D}_s]{} sre'; id : T = e, sre' \xrightarrow[\mathcal{D}_s]{} sre''}{d; id : \mathbf{bool} = e, sre \xrightarrow[\mathcal{D}_s]{} sre''}$	cioè le modifiche complessive si ottengono effettuando le modifiche dovute a ciascun pezzo della dichiarazione in sequenza

Costrutto let Un'ulteriore e fondamentale modifica si avrà anche nel costrutto **let**

$\dfrac{d, <sre, \emptyset> \xrightarrow[\mathcal{D}_s]{} <sre'_g, sre'_g> \; e, \overline{sre'} \xrightarrow[\mathcal{E}_s]{} T}{\mathbf{let}\ d\ \mathbf{in}\ e\ \mathbf{endlet}\ , sre \xrightarrow[\mathcal{E}_s]{} T}$	quando iniziamo a valutare d l'ambiente locale è vuoto; la valutazione di d genera un ambiente locale che viene usato per aggiornare quello globale e valutare e; si noti che in generale $sre'_g = sre$

3.1.3 Semantica delle espressioni con identificatori

Avendo ristretto la sintassi, vogliamo adesso attribuire, in modo induttivo/composizionale/modulare, una semantica alle espressioni, cioè vogliamo definire un'algebra della semantica B ed un omomorfismo da SA in B.

La definizione completa di B e dell'omomorfismo di valutazione è lasciata per esercizio al lettore, seguendo lo schema qui proposto, in cui sono riportati solo i tipi e le operazioni maggiormente rilevanti.

Carriers Tralasciando i carriers dei tipi di scarso interesse, come cifre e lettere, la cui interpretazione semantica non richiede particolare fantasia, discutiamo solo i tipi di maggior interesse.

identificatori Data l'estrema semplicità del linguaggio degli identificatori, ed essendo la scelta degli identificatori ben lontana dal fuoco della nostra attenzione, la semantica scelta è la più banale possibile, ovvero il carrier semantico sarà il linguaggio stesso (e anche le interpretazioni delle operazioni saranno le stesse del caso sintattico e l'omomorfismo di valutazione sarà la funzione identica). Se si fosse scelto un linguaggio degli identificatori più complesso, ad esempio dotato di operazioni sugli identificatori, lo si sarebbe potuto porre in relazione con un dominio semantico maggiormente significativo.

dichiarazioni Come detto, le dichiarazioni servono a creare un ambiente in cui valutare gli identificatori quindi sarà necessario avere un tipo ausiliario E che è un potenziamento di SE nel senso che oltre ad associare ad un identificatore il suo tipo, gli associa anche il suo valore. Quindi definiamo $\mathrm{E} = [L_{\mathtt{Ide}} \xrightarrow[p]{} \mathtt{Tipi} \times \mathtt{Val}]$, dove $\mathtt{Val} = \mathbb{N} \cup \mathbb{B}$; se si aggiungessero altri tipi al linguaggio bisognerebbe ampliare la definizione di `Val` aggiungendo gli elementi dei nuovi tipi.

Siccome abbiamo solo due tipi, i cui valori sono disgiunti, dal valore attribuito ad un identificatore si potrebbe desumere il suo tipo e quindi si potrebbe semplificare $\mathrm{E} = [L_{\mathtt{Ide}} \xrightarrow[p]{} \mathtt{Val}]$. Questa semplificazione è ammissibile solo grazie alla estrema semplificazione sui meccanismi di tipo scelti per il linguaggio didattico (ad esempio non sarebbe più sensato se avessimo il tipo "reali" o "razionali" perché da $x = 3$ non si potrebbe più sapere il tipo di x); inoltre la semplificazione non aiuta molto, quindi non verrà adottata.

Siccome le funzioni usate nell'algebra semantica sono parziali (ad esempio $3 - 7$ non è definito), la valutazione di un'espressione sintatticamente corretta, può risultare indefinita e quindi anche dichiarazioni staticamente corrette possono non essere in grado di aggiornare un ambiente. Ad esempio la dichiarazione $a : \mathbf{int} = y - 10$ è sintatticamente corretta in ogni ambiente in cui y sia dichiarato di tipo intero, ma è in grado di aggiornare solo ambienti in cui y sia associato ad un valore minore o uguale a 10, altrimenti non si sa quale valore attribuire ad a.

Pertanto si ha $[\![\mathtt{Dec}]\!] = [\mathrm{E} \xrightarrow[p]{} \mathrm{E}]$

espressioni Avendo introdotto gli identificatori nel nostro linguaggio, la semantica di un'espressione sarà dipendente dal valore attribuito agli identificatori, quindi un'espressione sarà una funzione che, preso un ambiente per valutare gli identificatori che vi compaiono, produce un valore del tipo opportuno.

$$[\![\mathtt{EI}]\!] = [\mathrm{E} \xrightarrow[p]{} \mathbb{N}] \qquad\qquad [\![\mathtt{EB}]\!] = [\mathrm{E} \xrightarrow[p]{} \mathbb{B}]$$

L'interpretazione degli operatori aritmetici del linguaggio sarà quella ovvia, cioè ad esempio se op fosse l'operazione corrispondente alla produzione $\mathtt{EI} ::= \mathtt{EI} + \mathtt{EI}$, la sua interpretazione in B sarebbe definita da $op^B(e_1, e_2) = f$, dove $f(r) = e_1(r) \boxed{+} e_2(r)$ e $\boxed{+}$ è la somma fra numeri.

Più interessante è l'interpretazione delle operazioni legate al tipo dichiarazione e all'uso di identificatori; concentriamoci su di esse.

Operazioni con risultato di tipo `Dec` Siccome il risultato dell'interpretazione di una di queste operazioni deve essere un elemento di $[\![\mathtt{Dec}]\!] = [\mathrm{E} \xrightarrow[p]{} \mathrm{E}]$, per definire il risultato è sufficiente definirne l'applicazione ad un generico ambiente r.

$op^B_{\mathtt{Dec::=Ide:}\,\mathbf{int}\,\mathtt{=EI}}(id, e)r = r[e(r)/id]$	se $e(r)$ non è definito, il risultato dell'operazione è indefinito
$op^B_{\mathtt{Dec::=Dec;Ide:}\,\mathbf{int}\,\mathtt{=EI}}(d, id, e)r = d(r)[e(d(r))/id]$	cioè è la composizione

Operazioni con risultato di tipo `EI` Siccome il risultato dell'interpretazione di una di queste operazioni deve essere un elemento di $[\![\mathtt{EI}]\!] = [\mathrm{E} \xrightarrow[p]{} \int]$, per definire il risultato è sufficiente definirne l'applicazione ad un generico ambiente r.

$op^B_{\mathtt{EI::=Ide}}(id)r = r(id)$	cioè corrisponde alla valutazione dell'identificatore
$op^B_{\mathtt{EI::=}\,\mathbf{let}\,\mathtt{Dec}\,\mathbf{in}\,\mathtt{EI}\,\mathbf{endlet}}(d, e)r = e(d(r))$	cioè si usa d per aggiornare l'ambiente corrente e poi vi si valuta e

L'omomorfismo di valutazione semantica è dato per induzione seguendo lo schema del sistema che definisce la sintassi, e questo garantisce la composizionalità. Le regole relative agli operatori aritmetici sono la traduzione di quelle viste prima dell'introduzione degli identificatori, lievemente aggiustate in modo da far quadrare i conti, dato che adesso i valori semantici sono funzioni e non numeri. Ad esempio per il prodotto avremo:

$$\frac{e_1 \xrightarrow[\mathcal{E}]{} f_1 \;\; e_2 \xrightarrow[\mathcal{E}]{} f_2}{e_1 * e_2 \xrightarrow[\mathcal{E}]{} g} \quad Q(e_1) \wedge Q(e_2) \qquad g(r) = f_1(r) \boxed{*} f_2(r)$$

Per le operazioni che manipolano gli identificatori avremo, ad esempio:

$$\frac{e \xrightarrow[\mathcal{E}]{} f}{id: \ \mathbf{int} \ = e \xrightarrow[\mathcal{D}]{} \delta} \quad \delta(r) = r[f(r)/id]$$

$$\frac{e \xrightarrow[\mathcal{E}]{} f \;\; d \xrightarrow[\mathcal{D}]{} \delta}{d; id: \ \mathbf{int} \ = e \xrightarrow[\mathcal{D}]{} \delta'} \quad \delta'(r) = \delta(r)[f(\delta(r))/id]$$

$$\frac{}{id \xrightarrow[\mathcal{E}]{} f} \quad f(r) = r(id)$$

$$\frac{d \xrightarrow[\mathcal{D}]{} \delta \;\; e \xrightarrow[\mathcal{E}]{} f}{\mathbf{let}\ d\ \mathbf{in}\ e\ \mathbf{endlet} \xrightarrow[\mathcal{E}]{} g} \quad g(r) = f(\delta(r))$$

Si noti la stretta corrispondenza fra le regole che definiscono l'interpretazione semantica delle operazioni algebriche e quelle che definiscono l'omomorfismo di valutazione semantica. Infatti ogni qualvolta si vuole attribuire semantica ad un linguaggio sono equivalenti:

- definire l'interpretazione degli operatori nell'algebra della semantica (cioè dotare i valori semantici di una struttura algebrica) e poi ricavarne che l'omomorfismo è definito induttivamente nell'unico modo possibile, cioè dalle regole del tipo

  $$\frac{\{u_i \to x_i \mid i \in [1,n]\}}{op^A(u_1, \ldots, u_n) \to op^B(x_1, \ldots, x_n)}$$

  dove, siccome op^A è esattamente la conseguenza della produzione associata ad op, le regole risultano quelle della struttura a regole.

- definire induttivamente una funzione di valutazione semantica mediante regole del tipo

  $$\frac{\{u_i \to x_i \mid i \in [1,n]\}}{op^A(u_1, \ldots, u_n) \to w}$$

  dove w è un'espressione che usa le x_i e poi dare ai valori semantici una struttura algebrica definendo $op^B(x_1, \ldots, x_n) = w$.

Avendo seguito entrambi gli approcci ad un tempo, naturalmente si è avuto un po' di ridondanza.

3.2 Dichiarazioni di funzione

Supponiamo di voler ulteriormente arricchire il linguaggio delle espressioni permettendo di definire funzioni. Questo vuol dire che si dovranno arricchire sia le dichiarazioni, aggiungendo un nuovo schema di dichiarazione, sia le espressioni, ammettendo di formare espressioni mediante chiamate di funzioni su argomenti corretti. Le problematiche sono le stesse già viste per il caso degli identificatori di valori, ma l'esposizione risulta complicata dal fatto che è necessario gestire funzioni.

Si noti che mentre utilizziamo le funzioni all'interno, non aggiungiamo un nuovo tipo al linguaggio per rappresentare le funzioni; questo da un lato facilita le caso, ma dall'altro restringe le possibilità del linguaggio, perché non c'è modo di usare funzioni come parametri di altre funzioni.

Invece per permettere un numero arbitrario di parametri, abbiamo bisogno dei tipi ausiliarii "liste di parametri", dove un parametro è semplicemente una coppia nome-tipo, e liste di argomenti (per applicare la funzione).

$S = \{\ldots, \texttt{ParList}\}$

$T = \{\ldots, \ \mathbf{function}\ \}$

$\texttt{ParList} ::= \texttt{Ide}: \ \mathbf{int} \ | \texttt{Ide}: \ \mathbf{bool} \ | \texttt{ParList}; \texttt{Ide}: \ \mathbf{int} \ | \texttt{ParList}; \texttt{Ide}: \ \mathbf{bool}\ .$

$\texttt{ArgList} ::= \texttt{EI} | \texttt{EB} | \texttt{ArgList}; \texttt{EI} | \texttt{ArgList}; \texttt{EB}.$

$\texttt{Dec} ::= \ldots | \ \mathbf{function}\ \texttt{Ide}: \ \mathbf{int} \ = \texttt{EI} | \ \mathbf{function}\ \texttt{Ide}: \ \mathbf{bool} \ = \texttt{EB} |$

$\ \mathbf{function}\ \texttt{Ide}(\texttt{ParList}): \ \mathbf{int} \ = \texttt{EI} | \ \mathbf{function}\ \texttt{Ide}(\texttt{ParList}): \ \mathbf{bool} \ = \texttt{EB}$

$\texttt{EI} ::= \ldots | \texttt{Ide}(\texttt{ArgList}).$

$\texttt{EB} ::= \ldots | \texttt{Ide}(\texttt{ArgList}).$

Le condizioni di correttezza statica sono abbastanza pesanti, perché dobbiamo richiedere che ogni funzione abbia il giusto numero e tipo di parametri, oltre che sia dichiarata.

Anzitutto dobbiamo estendere la definizione di SE per far sì che agli identificatori di funzione si possa associare il tipo della funzione; per definizione di SE è sufficiente modificare la definizione di $\texttt{Tipi}$ e porre $\texttt{Tipi}_B = \{ \ \mathbf{int}\ , \ \mathbf{bool}\ \}$ e $\texttt{Tipi} = \texttt{Tipi}_B \cup (\texttt{Tipi}_B^* \times \texttt{Tipi}_B)$. Dobbiamo inoltre introdurre il simbolo per la funzione che verifica la correttezza statica di liste di parametri, PL. Siccome la lista dei parametri di una funzione serve a rendere disponibili i nomi dei parametri per descrivere il significato (o corpo) della funzione (oltre che a permettere in fase di valutazione di applicare la funzione ai suoi argomenti), la funzione PL interpreterà ogni lista corretta come l'ambiente locale generato dalla lista dei parametri, quindi $PL\colon L_{\texttt{ParList}} \to SE$.

Nel seguito, oltre alle convenzioni notazionali adottate nella sottosezione sulla semantica statica, useremo $k\colon L_{\texttt{ParList}} \to \texttt{Tipi}_B^+$, definita induttivamente da $k(id: T) = T$ e $k(pl; id: T) = k(pl) \cdot T$, cioè k memorizza i tipi dei parametri che compaiono in pl in una stringa di tipi. Analogamente useremo $h\colon L_{\texttt{ArgList}} \to \texttt{Tipi}_B^+$, definita induttivamente da $h(e) = T$ e $h(le; e) = h(le) \cdot T$ dove e è un'espressione di tipo T.

$\dfrac{}{id : T \underset{\mathcal{PL}}{\rightarrow} \{(id, T)\}}$	un parametro viene rappresentato dall'ambiente che contiene una sola associazione, cioè il nome del parametro con il suo tipo
$\dfrac{pl \underset{\mathcal{PL}}{\rightarrow} sr}{pl; id : T \underset{\mathcal{PL}}{\rightarrow} sr[[T/id]]}$	uno stesso identificatore non può comparire due volte nella lista dei parametri; si noti che se $id \in dom(sr)$, allora $sr[[T/id]] = \mathtt{err}$
$\dfrac{pl \underset{\mathcal{PL}}{\rightarrow} \mathtt{err}}{pl; id : T \underset{\mathcal{PL}}{\rightarrow} \mathtt{err}}$	propagazione d'errore
$\dfrac{e, sr \underset{\mathcal{E}_s}{\rightarrow} T}{\textbf{function } id : T = e, sr \underset{\mathcal{D}}{\rightarrow} sr[(\lambda, T)/id]}$	in più ci vorranno le solite regole di introduzione d'errore se il tipo di e è diverso dal tipo della funzione o se il tipo di e è un errore
$\dfrac{pl \underset{\mathcal{PL}}{\rightarrow} sr' \quad e, sr[sr'] \underset{\mathcal{E}_s}{\rightarrow} T}{\textbf{function } id(pl) : T = e, sr \underset{\mathcal{D}}{\rightarrow} sr[(k(pl), T)/id]}$	se la lista dei parametri non contiene duplicazioni e il corpo è un'espressione ben formata usando l'ambiente corrente e i parametri, la dichiarazione è corretta
$\dfrac{pl \underset{\mathcal{PL}}{\rightarrow} \mathtt{err}}{\textbf{function } id(pl) : T = e, sre \underset{\mathcal{D}}{\rightarrow} \mathtt{err}}$	se la lista dei parametri è sbagliata la dichiarazione è sbagliata; inoltre bisognerebbe aggiungere le regole per gli altri casi d'errore
$\dfrac{}{id, sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{ int}} \quad sr(id) = (\lambda, \textbf{ int })$	il nome di una funzione costante è un'espressione
$\dfrac{\{e_i, sr \underset{\mathcal{E}_s}{\rightarrow} T_i \mid i \in [1, n]\}}{id(e_1 \dots e_n), sr \underset{\mathcal{E}_s}{\rightarrow} \textbf{ int}} \quad sr(id) = (T_1 \dots T_n, \textbf{ int })$	se gli argomenti sono giusti la chiamata è corretta
$\dots$	altrimenti ci vogliono regole di propagazione di errore.

La semantica della dichiarazione di funzione è l'arricchimento dell'ambiente corrente con una nuova associazione, che è il nome della funzione con una funzione con lo stesso numero e tipo di argomenti ed il cui valore si ottiene valutando il corpo della funzione. Quindi, ad esempio, la dichiarazione di una funzione costante di tipo intero associa al nome della funzione un numero, che è la valutazione dell'espressione.

$$\frac{e, rs \underset{\mathcal{E}}{\rightarrow} x}{[\![\textbf{ function } id : \textbf{ int } = e]\!]r = r[x/id]}$$

mentre la dichiarazione di una funzione a valori interi con un solo argomento di tipo booleano sarà valutata in

$$\frac{}{[\![\textbf{ function } id(par : \textbf{ bool }) : \textbf{ int } = e]\!]r = r[f/id]} \qquad \text{dove } f \colon \mathbb{B} \rightarrow \int, f(b) = x \text{ se } e, r[b/par] \underset{\mathcal{E}}{\rightarrow} x$$

A Algebre eterogenee

Una segnatura è semplicemente un modo di impacchettare un insieme di nomi per indicare insiemi e di nomi per indicare funzioni fra quegli insiemi, ricordando i (nomi dei) tipi degli argomenti e del risultato.

Def. A.1 Una segnatura Σ consiste di un insieme di *sort* o *tipi* e un insieme F di *simboli di operazione*, dotati di un'operazione di *arità* che associa ad ogni simbolo in F una coppia in $S^* \times S$. Per indicare che l'arità di op è $(s_1 \dots s_n, s)$, useremo la notazione $op \colon s_1 \times \dots \times s_n \rightarrow s$ e se $n = 0$, ovvero $s_1 \dots s_n = \lambda$, abbrevieremo in $op \colon \rightarrow s$. $\square$

Un'algebra su una segnatura è un'interpretazione dei nomi che compaiono nella segnatura con oggetti corrispondenti, quindi ogni sort viene associata ad un insieme e ogni simbolo di operazione con una funzione del tipo corretto.

Def. A.2 Data una segnatura $\Sigma = (S, F)$, un'*algebra eterogenea su* Σ, A, consiste di una famiglia di insiemi $\{s^A\}_{s \in S}$, detti *carrier* o *supporti*, e per ciascun simbolo di operazione $op \colon s_1 \times \dots \times s_n \rightarrow s$ in F, di una funzione $op^A \colon s_1^A \times \dots \times s_n^A \rightarrow s^A$ (se $n = 0$, $op^A \colon \rightarrow s^A$ è una funzione costante, cioè un elemento di s^A). Se le funzioni sono tutte totali, l'algebra si dice *totale*; se una o più sono funzioni parziali, l'algebra si dice *parziale*. $\square$

Date due algebre sulla stessa segnatura siamo interessati a quelle funzioni fra di esse che preservano la struttura, ovvero a quelle traslazioni, dette *omomorfismi*, che possono essere effettuate indifferentemente prima o dopo l'applicazione di una delle funzioni dell'algebra.

Def. A.3 Data una segnatura $\Sigma = (S, F)$, e due algebre su Σ, A e B, un *omomorfismo* da A in B, $h \colon A \rightarrow B$, è una famiglia di funzioni $\{h_s \colon s^A \rightarrow s^B\}_{s \in S}$ tali che per ogni $op \colon s_1 \times \dots \times s_n \rightarrow s$ in F si abbia

$$h_s(op^A(a_1, \dots, a_n)) = op^B(h_{s_1}(a_1), \dots, h_{s_1}(a_n))$$

Nel caso in cui op^A o op^B siano parziali, il significato dell'ultima uguaglianza è che o entrambi i lati sono indefiniti o entrambi i lati sono lo stesso elemento del carrier. $\square$