Semantica dinamica

Vogliamo definire una funzione che associ ad ogni termine corretto del mio linguaggio di programmazione un valore.
Questa associazione deve essere descritta in modo composizionale, cioè il valore corrispondente ad ogni termine deve essere determinato sulla base dei valori corrispondenti ai suoi sottotermini (e del costruttore esterno)

Per poter fare questo, ogni termine deve essere scomponibile in maniera unica. Ci sono varie scelte possibili per ottenere questo:
Usare grammatiche non ambigue
Usare non i termini ma una rappresentazione non ambigua, ad esempio l'algebra dei termini (algebra iniziale, modello assolutamente libero…)

In realtà l'unicità della scomposizione è una proprietà sufficiente ma non necessaria, basta infatti usare grammatiche semanticamente non ambigue (ad esempio Exp::= Num | Exp + Exp va bene,
ma Exp::= Num | Exp + Exp |Exp *Exp no)

Grammatiche ed algebre
(riassunto da LP)

Data una grammatica G = (T,N,Pr), le si può associare una segnatura

$\Sigma_G = (S, Op)$

S= N è l'insieme dei non terminali di G

Op consiste di un'operazione per ogni produzione di G:

l'operazione corrispondente alla produzione α ::= str, dove

str $= w_0\, \alpha_1\, w_1\, \alpha_2 \ldots \alpha_n\, w_n$ con $\alpha_i \in N$ e $w_i \in T^*$

avrà arità $\alpha_1 \times \alpha_2 \times \ldots \times \alpha_n \rightarrow \alpha$ e sarà indicata dal simbolo $f_{\alpha ::= str}$

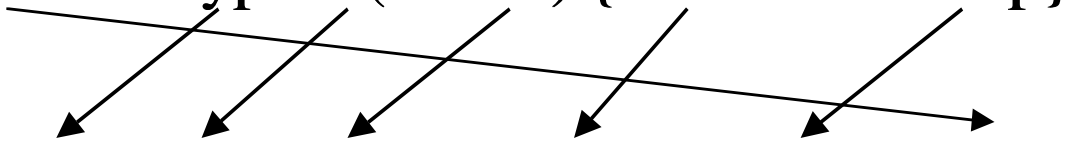
Siccome a noi interessa solo la struttura algebrica e non i nomi usati per tipi e operazioni, useremo la sintassi astratta, cioè la classe di isomorfismo di Σ_G

Metodo pratico per determinare Σ_G

Il problema è trovare le operazioni (i tipi sono banalmente i non terminali)

Supponiamo di dover trovare il tipaggio dell'operazione f corrispondente alla produzione

Fdec ::= Type Id(**PDecs**) {**Stats result** Exp}



$$f : Type \times Id \times PDecs \times Stats \times Exp \rightarrow Fdec$$

Il tipo restituito dall'operazione sarà quello a sinistra della produzione

Si evidenziano i non terminali presenti nella parte destra della produzione

Si riportano tali non terminali come argomenti (separati da $\times$)

Algebre su Σ_G

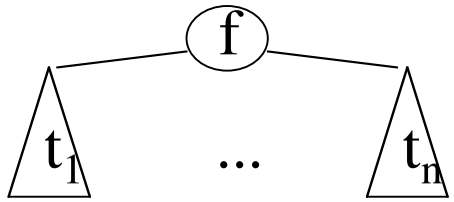
Algebre iniziali (o dei termini, o assolutamente libere)
Per noi sono interessanti perché ogni loro elemento è ottenuto in modo unico per applicazione successiva delle operazioni della segnatura.
Sono quindi candidati ideali per la definizione della semantica, perché non presentano problemi di ambiguità.
Per ciascuna algebra A sulla stessa segnatura esiste un unico omomorfismo !A dall'algebra iniziale in A. Sono quindi tutte isomorfe fra loro.

Un rappresentante canonico è la seguente algebra T_{Σ_G}

I suoi carrier sono definiti induttivamente dalle seguenti regole, una per ciascuna operazione $f : s_1 \times ... \times s_n \rightarrow s$

$$\frac{t_1 : s_1 \ \ldots \ t_n : s_n}{f(t_1, \ldots, t_n) : s}$$

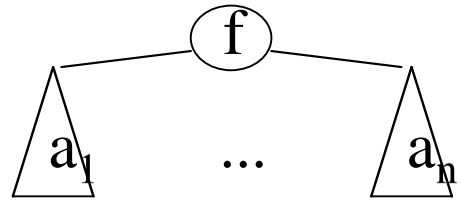
Albero →



La base è data dalle operazioni costanti che a loro volta derivano da produzioni la cui parte destra non contiene non terminali

A ciascuna operazione $f : s_1 \times ... \times s_n \rightarrow s$ corrisponde una funzione
$f^{T_{\Sigma_G}} : {s_1}^{T_{\Sigma_G}} \times ... \times {s_n}^{T_{\Sigma_G}} \rightarrow {s}^{T_{\Sigma_G}}$
definita da

$f^{T_{\Sigma_G}}(a_1, \ldots, a_n) =$



Algebre su Σ_G

Algebre sintattiche hanno come carriers il linguaggio definito dalla grammatica.

L'algebra L_G è definita da

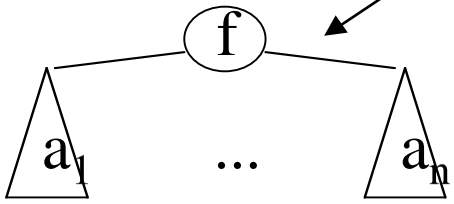
Il suo carrier di tipo s è proprio il linguaggio di tipo s, cioè $s^{L_G} = L_s(G)$

A ciascuna operazione $f : s_1 \times \ldots \times s_n \rightarrow s$, relativa alla produzione $s ::= w_0\, s_1\, w_1\, s_2 \ldots s_n\, w_n$ corrisponde la funzione $f^{L_G} : s_1{}^{L_G} \times \ldots \times s_n{}^{L_G} \rightarrow s^{L_G}$ definita da $f^{L_G}(a_1, \ldots, a_n) = w_0\, a_1\, w_1\, a_2 \ldots a_n\, w_n$

L_G è ben definita?

Com'è definito $!\, L_G : T_{\Sigma_G} \rightarrow L_G$ l'unico omomorfismo?

Generico elemento di $s^{T_{\Sigma_G}}$

$!\, L_{Gs}($ 

$)$

$= f^{L_G}(!\, L_{Gs1}(a_1), \ldots, !\, L_{Gsn}(a_n))$

$f^{T_{\Sigma_G}}(a_1, \ldots, a_n)$

Unica possibilità se vogliamo che $!\, L_G = \{!\, L_{Gs} : s^{T_{\Sigma_G}} \rightarrow s^{L_G}\}$ sia un omomorfismo

Esercizio induttivo

Provare che data una grammatica G, avente una produzione pr
$s ::= w_0\, s_1\, w_1\, s_2 \ldots s_n\, w_n$ se $a_i \in L_{si}(G)$, allora $w_0\, a_1\, w_1\, a_2 \ldots a_n\, w_n \in L_s(G)$

Proviamo per induzione aritmetica su $i \in N$ che $s_j \rightarrow^* a_j$ per $j \in [1,i]$ implica
$s \rightarrow^* w_0\, a_1\, w_1\, a_2 \ldots w_{i-1}\, a_i\, w_i\, s_{i+1} \ldots s_n\, w_n$

Base i=0. Per definizione di riscrittura ad un passo $s \rightarrow w_0\, s_1\, w_1\, s_2 \ldots s_n\, w_n$ da cui
$s \rightarrow^* w_0\, s_1\, w_1\, s_2 \ldots s_n\, w_n$

Passo induttivo. Sia vero che $s_j \rightarrow^* a_j$ per $j \in [1,i+1]$, allora $s_j \rightarrow^* a_j$ per $j \in [1,i]$ e quindi per per ipotesi induttiva che $s \rightarrow^* \underbrace{w_0\, a_1\, w_1\, a_2 \ldots w_{i-1}\, a_i\, w_i}_{u}\, s_{i+1} \underbrace{\ldots s_n\, w_n}_{v}$

Sappiamo quindi che $s \rightarrow^* u\, s_{i+1}\, v$ basta provare che $s_{i+1} \rightarrow^* a_{i+1}$ implica
$u\, s_{i+1}\, v \rightarrow^* u\, a_{i+1}\, v$ e per transitività otterremo $s \rightarrow^* u\, a_{i+1}\, v$, che è la tesi

Anche questo lemma si prova per induzione aritmetica, sul numero k di passi necessari per provare che $s_{i+1} \rightarrow^* a_{i+1}$

Base k=0. Cioè $a_{i+1} = s_{i+1}$; per riflessività, $u\, s_{i+1}\, v \rightarrow^* u\, s_{i+1}\, v$

Passo induttivo. Sia vero che $s_{i+1} \rightarrow^* a_{i+1}$ in k passi implica $u\, s_{i+1}\, v \rightarrow^* u\, a_{i+1}\, v$ e supponiamo che $s_{i+1} \rightarrow^* a'_{i+1}$ in k+1 passi.

Allora avremo $s_{i+1} \rightarrow^* a_{i+1}$ in k passi e $a_{i+1} \rightarrow a'_{i+1}$. Quindi $u\, s_{i+1}\, v \rightarrow^* u\, a_{i+1}\, v$.

Per definizione di riscrittura ad un passo, $a_{i+1} \rightarrow a'_{i+1}$ implica che esistono $\alpha,\beta \in (T \cup N)^*$ ed una produzione $s' ::= w$ t.c. $a_{i+1} = \alpha\, s'\, \beta$ e $a'_{i+1} = \alpha\, w\, \beta$.

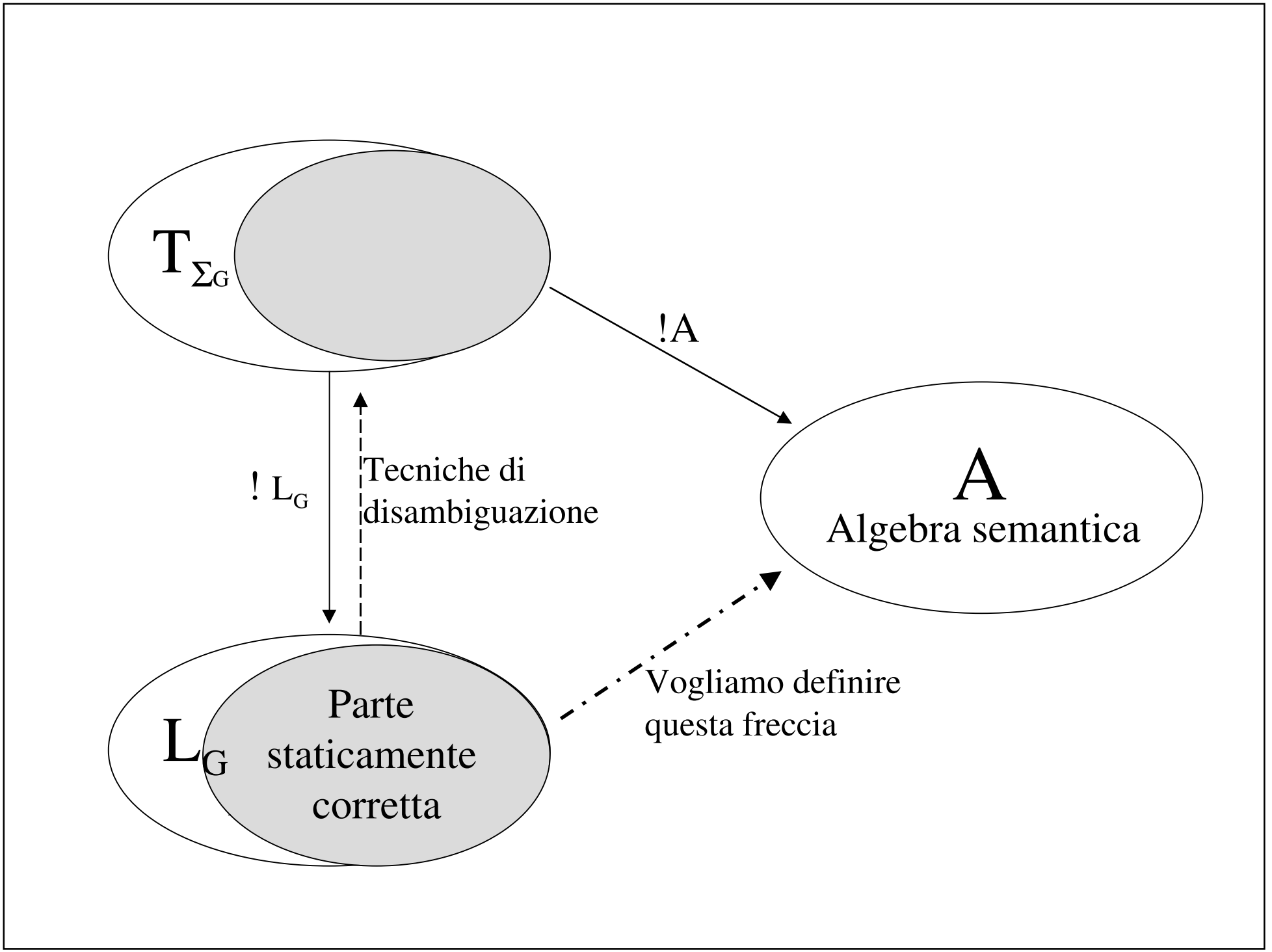
Allora $u\, \alpha\, s'\, \beta\, v = u\, a_{i+1}\, v \rightarrow u\, a'_{i+1}\, v = u\, \alpha\, w\, \beta\, v$ da cui per transitività si conclude

Semantica

In questo contesto, dare la semantica di un linguaggio diventa fissare un'algebra (quella semantica) ed avere come semantica l'unico omomorfismo dall'algebra iniziale in quella semantica.

Quasi. In realtà ci sono 3 punti non immediati

- L'algebra della semantica per i linguaggi di programmazione è spesso *parziale,* cioè l'interpretazione delle operazioni sono funzioni parziali (questo problema lo aggireremo)
- Noi vogliamo associare un valore ad una stringa del linguaggio, quindi ci servirà una funzione che mappi il linguaggio sull'algebra iniziale
- Siccome siamo più interessati alla funzione semantica che all'algebra, spesso si descrivono direttamente le clausole della semantica e si deducono carrier ed operazioni dell'algebra semantica in modo che la funzione definita sia un omomorfismo


T_{Σ_G}
!A
! L_G
Tecniche di disambiguazione
A
Algebra semantica
L_G
Parte staticamente corretta
Vogliamo definire questa freccia