Esercitazione guidata 1

Sfruttando le tecniche viste in casi più semplici, si dia la semantica statica del linguaggio imperativo (didattico) descritto dalla seguente grammatica LW:

Prog ::= Decs **main**() Stats.
Decs ::= Dec | Dec Decs.
Dec ::= VDec | FDec.
VDec ::= PDec | Type Id = Exp;
PDec ::= Type Id;
FDec ::= RType Id(PDecs) {Stats [**result** Exp]}
PDecs ::= PDec | PDec PDecs.
Rtype ::= Type | **void**.
Type ::= **int** | **bool**.
Stats ::= Stat | Stat Stats.
Stat ::= Exp; | **while** (Exp) {Stats} | **if** (Exp) {Stats} **else** {Stats}.
Exp ::= Id | Id = Exp | Id(Exps) | Exp + Exp | …
Exps ::= Exp | Exp Exps.

Questo linguaggio sarà usato frequentemente per le domande sulla semantica statica (all'orale)

Vincoli statici (informali)

Nel corpo del programma possono essere usati solo identificatori dichiarati precedentemente e devono essere usati in modo consistente col tipo con cui sono dichiarati. Inoltre non si ammettono doppie dichiarazioni.

Nelle dichiarazioni con inizializzazione il tipo deve corrispondere al tipo dell'espressione usata per inizializzare

Una funzione ha la parte **result** Exp se e solo se il suo tipo risultante *non* è **void**

Se una funzione ha la parte **result** Exp, il suo tipo risultante deve essere lo stesso tipo di Exp

Le espressioni che compaiono in **while** e **if** devono essere di tipo booleano

Un'espressione corretta usa solo identificatori dichiarati e in modo consistente col loro tipo

Un'assegnazione è corretta solo se il tipo dell'espressione è uguale a quello dell'identificatore

Una chiamata di funzione è corretta se il numero e il tipo dei parametri attuali coincide con quelli dei parametri formali

Problematiche (e soluzioni) 1

Dichiarazioni e Tipi: rispetto a quanto visto finora ci sono due novità:

- Il tipo dell'identificatore è dichiarato esplicitamente (Type Id) per cui non c'è bisogno di dedurlo
- Bisogna verificare che nelle dichiarazioni con inizializzazione (Type Id = Exp;) il tipo scelto e il tipo dell'espressione usata per inizializzare coincidano

La problematica connessa ad ambiente locale e globale si ritrova nella dichiarazione di funzione: RType Id(PDecs) {Stats [**result** Exp]}, nel cui corpo si possono usare variabili e funzioni globali mentre i parametri costruiscono un ambiente locale (ed ovviamente i nomi dei parametri devono essere tutti distinti)

Problematiche (e soluzioni) 2

Dichiarazioni di funzione.
Per verificare che la parte **result** Exp ci sia se e solo se il suo tipo risultante non è **void**, la cosa più facile è dare regole separate per i due casi.
Questo, in effetti corrisponde a espandere la notazione compatta nelle corrispondenti produzioni:

FDec ::= Type Id(PDecs) {Stats **result** Exp} |
void Id(PDecs) {Stats}

e dare una regola per ciascuna produzione

L'insieme dei tipi associati agli identificatori andrà ampliato per trattare i tipi funzionali: Types = BTypes $\cup$ (BTypes$^+$ $\times$ RTypes), dove BTypes = {int, bool} e RTypes = BTypes $\cup$ {void}

Come si trovano le regole della semantica statica
Euristica

Prima di tutto va chiarito che quanto segue è una metodologia che supporta il lavoro di definizione della semantica statica, non un algoritmo che seguito ciecamente porta ad una buona definizione.

Concentriamoci sul dare le regole per i casi corretti.
Principio 1. Si parte dando una regola per ogni produzione della grammatica

Principio 2. Per dare ciascuna regola si lavora top down:

a) Si parte dalla conseguenza, lasciando in bianco il risultato
b) Si mette una premessa per ogni sottostringa della stringa di cui si vuol dare la semantica statica, introducendo le identificazioni fra le metavariabili suggerite dai vincoli
c) Se necessario si catturano eventuali altri vincoli con condizioni a lato
d) Si usano le metavariabili introdotte nelle premesse per esprimere il risultato nelle conseguenze

Come si trovano le regole della semantica statica Esempio applicato ad una regola

Consideriamo il caso della dichiarazione con inizializzazione Type Id = Exp; una stringa del linguaggio "generata" da questa produzione ha la forma t x = e, dove $t \in L_{Type}(LW)$, $x \in L_{Id}(LW)$, ed $e \in L_{Exp}(LW)$

b) Nelle premesse metto una clausola per ogni componente della stringa
A priori dovrebbe usare un t', ma dai vincoli informali so che deve essere uguale a t

Queste due sono superflue, perché identificatori e tipi sono sempre corretti

c) Non si ammettono doppie dichiarazioni

$$\frac{\cancel{\rho_g, \rho_l \vdash t} \quad \cancel{\rho_g, \rho_l \vdash x} \qquad \rho_g[\rho_l] \vdash e: t}{\rho_g, \rho_l \vdash t\ x = e \rightarrow \rho_l[t/x]} \quad x \notin Dom(\rho_l)$$

d) Se tutto è andato bene il risultato è l'aggiornamento

a) Scrivo le conseguenze, lasciando in bianco il risultato.
Siccome è una dichiarazione avrà questa forma

Esempio 2

Consideriamo il caso della applicazione di funzione Id(Exps); una stringa del linguaggio "generata" da questa produzione ha la forma x(el), dove $x \in L_{Id}(LW)$, ed $el \in L_{Exps}(LW)$

Se la correttezza di una singola espressione produce il suo tipo, quella di una sequenza di espressioni produrrà una lista di tipi

Devo verificare che x sia una funzione e qual è il suo tipo

$$\frac{\rho \vdash el: tl}{\rho \vdash x(el): rt} \quad x \in Dom(\rho),\ \rho(x) = \langle tl, rt \rangle$$

Vediamo anche le regole per Exps. $_\vdash_:_ \subseteq Env^s \times L_{Exps}(LW) \times Types^+$
Nella grammatica abbiamo due produzioni Exps ::= Exp | Exp Exps.
Quindi avremo due regole

$$\frac{\rho \vdash e: t}{\rho \vdash e: t} \quad e \in L_{Exp}(LW)$$

Caso base: la sequenza ha lunghezza 1

$$\frac{\rho \vdash e: t \quad \rho \vdash el: tl}{\rho \vdash e\ el: t\ tl}$$

Passo induttivo