Dispensa 5

5.1 Analisi semantica e type checking

Nei compilatori la fase immediatamente successiva al parsing è l'*analisi semantica*. Mentre il parser verifica la correttezza sintattica del programma sorgente, l'analizzatore semantico effettua tutta una serie di controlli per verificare che ogni costrutto del programma sorgente è usato consistentemente nel contesto in cui si trova. Restringeremo il nostro studio alla funzione più importante svolta dall'analizzatore semantico che è il *type checking* (la presenza del type checker nei compilatori presume evidentemente che il linguaggio sorgente sia un linguaggio di programmazione tipato).
Informalmente, il ruolo del type checker è quello di verificare che il tipo di dato di un costrutto "fa match" con il tipo atteso dal contesto, ossia quello permesso dalle convenzioni stabilite nel linguaggio sorgente. Per esempio, se una certa operazione `OP` è definita sugli interi, il type checker segnalerà un errore di tipo all'analisi della stringa sorgente `8 OP ciao` in quanto verificherà che uno degli operandi è di tipo stringa e non integer.

Di seguito introduciamo brevemente alcune problematiche generali del type checking (*overloading*, *conversione di tipo* e *type checking statico/dinamico*) prima di dare la definizione formale di type checker.

Overloading

Ci sono delle operazioni che hanno un significato diverso a seconda del contesto, e quindi possono avere domini di tipi diversi. Un semplice caso è l'operatore di addizione + che può essere applicato a interi, reali, complessi, matrici, etc. Cioè, ad esempio:
- la stringa `A + B` è corretta se `A` e `B` sono entrambi interi o entrambi matrici, etc.
- la stringa `A + B` è scorretta se, per esempio, `A` è intero e `B` è matrice.

In ogni caso è il contesto a disambiguare, e a far segnalare o meno l'errore da parte del type checker. A questa situazione si dà il nome di "overloading", e, ad esempio, si dice che l'operatore + è "overloaded" (sovraccaricato).

Conversione di tipo e coercizione

Non sempre un'operazione applicata a operandi di tipo diverso va considerata scorretta. Ad esempio, si considerino le seguenti due stringhe:

```
8 + ciao
8 + 4.0
```

E' evidente che la prima stringa è illegale perché i tipi degli operandi (integer e stringa) non sono compatibili; nel secondo caso invece pur essendo i tipi degli operandi diversi (integer e real) la stringa ha senso e quindi non dovrebbe essere segnalata come scorretta dal compilatore. Quello che si fa è una conversione di tipo, e un semplice esempio è proprio quando si somma un intero a un reale: l'intero viene convertito in reale (cioè trasformato nella stessa rappresentazione interna, a livello di pattern di bit) e l'operazione è consentita.

Nota importante: se la conversione di tipo è fatta automaticamente dal compilatore (come nel caso delle conversioni da interi a reali in Pascal per esempio) essa è chiamata *coercizione*. Le conversioni di tipo sono invece dette *esplicite* se fatte via software dal programmatore.

Type checking statico e dinamico

Il type checking si dice *statico* se fatto in fase di compilazione (noi stiamo studiando questo caso) oppure *dinamico* se fatto in fase di esecuzione, ossia quando viene eseguito il programma target.
In generale, sarebbe più conveniente avere un type checking di tipo statico in quanto a tempo di esecuzione non è necessario mantenere informazioni di tipo in apposite strutture dati, né sono necessari controlli di tipo a run-time. Ne consegue un sostanziale vantaggio (rispetto al type checking dinamico) sia in termini di occupazione di memoria sia in termini di velocità di esecuzione. Tuttavia, nella pratica ci sono situazioni in cui il controllo può essere fatto solo dinamicamente, cioè a run-time. Come semplice esempio, si consideri il seguente frammento di programma dove si è dichiarato:

```
A : array[0...100] of char;
i : integer;
```

Ora, se all'interno del programma da qualche parte viene computato `A[i]` (per esempio se c'è un'istruzione di assegnamento `A[i] := x;`) il compilatore non può staticamente garantire che durante l'esecuzione il valore di i si manterrà nel range `[0...100]`.

La nozione di type checking che abbiamo introdotto informalmente all'inizio può essere formalizzata attraverso la seguente catena di definizioni:

Definizione (type checker):
Un type checker implementa un sistema di tipi

Definizione (sistema di tipi):
Un sistema di tipi è una collezione di regole per assegnare espressioni di tipo alle varie parti di un programma

Definizione (espressione di tipo):
Un'espressione di tipo è o un tipo base oppure è ottenuta dall'applicazione di un operatore (chiamato costruttore di tipo) a un'altra espressione di tipo

Nei linguaggi di programmazione comuni tipi base sono ad esempio *boolean*, *char*, *integer* e *real*, mentre comuni costruttori di tipo sono gli *array*, i *record*, etc.

5.2 Esempi di type checking mediante traduzione guidata dalla sintassi

Un modo comune per specificare il modulo type checker all'interno di un compilatore è quello di scrivere uno schema di traslazione le cui produzioni descrivono il linguaggio sorgente e le cui azioni semantiche (che verranno eseguite durante il parsing) implementano il type checking. Vediamo nel seguito una serie di esempi:

Esempio1: type checking di una generica operazione

Anzitutto si noti che fare il type checking di un'operatore attraverso un'azione semantica significa sostanzialmente verificare la correttezza di tipo degli operandi e assegnare un'espressione di tipo al risultato. Ne consegue che il frammento di schema di traslazione (produzione – azione semantica)

per il type checking di una generica operazione (per es. binaria) OP che si applica ad operandi rispettivamente di tipo T_1 e T_2 e con risultato di tipo T_3 è:

$E \rightarrow E_1$ OP E_2 { E.type := (if E_1.type = T_1 and E_2.type = T_2) then T_3 else type_error}

Nell'azione semantica 'type' è un attributo sintetizzato del non-terminale E che contiene come valore l'espressione di tipo di E. Il valore 'type_error' è un tipo base speciale usato per segnalare un errore durante il type checking. L'azione semantica semplicemente controlla che i tipi degli operandi siano corretti, e di conseguenza assegna l'espressione di tipo al risultato oppure ne segnala l'errore.

Esempio2: type checking di statements

Vediamo il frammento di schema di traslazione (produzione – azione semantica) per il type checking di quattro semplici casi di statements:

- *Assegnamento*

$S \rightarrow$ id := E { S.type := (if id.type = E.type) then void else type_error}

Nell'azione semantica void è un altro tipo base speciale che denota l'assenza di valore di tipo, e permette agli statements di essere controllati. L'azione semantica controlla come condizione di type checking corretto che i tipi dell'espressione E e dell'identificatore id coinvolti nell'assegnamento siano compatibili (nell'esempio proprio uguali per semplicità), in tal caso assegna allo statement risultante S il valore fittizio void (per propagare la correttezza del controllo) altrimenti type_error.

- *Condizionale*

$S \rightarrow$ if E then S_1 { S.type := (if E.type = boolean) then S_1.type else type_error}

In questo caso l'azione semantica come condizione di type checking corretto controlla semplicemente che l'espressione condizionale E sia di tipo booleano, in tal caso il valore dello statement S_1 presente nella parte destra della produzione (corretto o scorretto che sia) viene propagato allo statement risultante S altrimenti si assegna type_error.

- *While loop*

$S \rightarrow$ while E do S_1 { S.type := (if E.type = boolean) then S_1.type else type_error}

L'azione semantica dello statement *while* è praticamente equivalente a quella precedente dell'*if-then* infatti anche in questo caso come condizione di type checking corretto essa deve semplicemente controllare che l'espressione condizionale E sia di tipo booleano, in tal caso il valore dello statement S_1 presente nella parte destra della produzione (corretto o scorretto che sia) viene propagato allo statement risultante S altrimenti si assegna type_error.

- *Sequenza di statements*

$S \rightarrow S_1 ; S_2$ { S.type := (if S_1.type = void and S_2.type = void) then void else type_error}

L'azione semantica controlla che gli statements S_1 e S_2 nella sequenza siano di tipo legale void e in questo caso ne propaga il valore allo statement risultante S, altrimenti si assegna type_error.

Esempio3: type checking per la coercizione da interi a reali

Come semplice caso di studio, si consideri il costrutto dell'addizione sul dominio di tipi integer e real. Il frammento di schema di traslazione (produzione – azione semantica) per il type checking dell'addizione che implementa la coercizione da interi a reali è il seguente:

$E \rightarrow E_1 + E_2$ { E.type := (if E_1.type = integer and E_2.type = integer) then integer
(else if E_1.type = integer and E_2.type = real) then real
(else if E_1.type = real and E_2.type = integer) then real
(else if E_1.type = real and E_2.type = real) then real }

L'azione semantica sostanzialmente controlla caso per caso il tipo degli argomenti assegnando di conseguenza il tipo al risultato. Nel caso in cui il tipo di uno dei due argomenti è integer e l'altro è real essa implementa la coercizione settando automaticamente a real il tipo del risultato.