1 I tipi nei linguaggi di programmazione

Un *sistema dei tipi* per un linguaggio di programmazione è un insieme di regole che consentono di dare un tipo ad espressioni, comandi ed altri costrutti del linguaggio. Un linguaggio si dice *tipato* se per esso è definito un tale sistema; altrimenti si dice *non tipato*. Il processo che porta alla determinazione di un tipo per i termini di un linguaggio si chiama *controllo dei tipi* (*type checking*).

Un linguaggio si dice *fortemente tipato* se il tipo di tutte le variabili è determinato a tempo di compilazione, altrimenti si dice *dinamicamente tipato* (se comunque esiste una nozione *non triviale* di tipo) o *non tipato*. Tra i inguaggi del primo genere ci sono il Fortran, l'ML, il Pascal, il Modula 2 e l'Ada; tra quelli del secondo lo Snobol e l'APL, mentre il LISP appartiene alla terza categoria. In un linguaggio fortemente tipato lo spazio di memoria richiesto per contenere il valore di ciascuna variabile durante l'esecuzione è completamente determinato a tempo di compilazione. In Pascal l'informazione sui tipi esiste solo a tempo di compilazione e scompare dai descrittori a tempo di esecuzione.

Esempio. In APL l'*esecuzione* dell'assegnamento

```
A <- 5
```

definisce `A` come una variabile intera (non vi sono dichiarazioni esplicite). Successivamente ad `A` potrebbe venir assegnato un array a due dimensioni. Dunque l'esecuzione di

```
A[2:3] <- 8
```

(assegnamento alla seconda riga, terza colonna di `A`) è corretta solo se *in quel momento* `A` è un tale array. Questo richiede controllo *dinamico* dei tipi. □

Lo scopo di un sistema dei tipi è quello di evitare che durante l'esecuzione di un programma occorrano errori quali, ad esempio, l'applicazione di funzioni ad argomenti inappropriati o il riferimento illegale della memoria. Tale obiettivo può comunque essere raggiunto in un linguaggio non tipato introducendo controlli dinamici (*dynamic checking*). Va notato che anche in un linguaggi con controlo statico dei tipi possono rendersi necessari controlli a tempo di esecuzione. L'esempio tipico a questo proposito è il controllo sugli indici di un array. Quando nessun programma genera errori a tempo di esecuzione il linguaggio si dice *corretto* (*sound*).

Distinguiamo due generi di errore: i cosiddetti errori *trapped*, ovvero quelli che provocano il fallimento della computazione, e gli errori *untrapped*, che non presentano sintomi immediatamente visibili, e sono perciò più insidiosi. Al primo tipo appartengono, ad esempio, la divisione per zero o un accesso proibito alla memoria; al secondo la lettura erronea di una porzione non significativa della memoria. Un linguaggio in cui nessun programma legale può generare errori untrapped si dice *sicuro* (*safe*).

Dato che errori quali il tentativo di accedere ad un array fuori dai suoi limiti non sono catturabili staticamente, spesso l'obiettivo della correttezza completa non sembra ragionevole. Alcuni sistemi dei tipi riescono dunque solo a garantire l'assenza di errori appartenenti ad un insieme "proibito", che include *tutti* gli errori untrapped ed *una parte* dei trapped. Un linguaggio in cui ogni programma legale è libero da errori proibiti si dice *strongly checked*. A questa categoria possono ovviamente appartenere anche linguaggi non tipati. Chiamando *Sound*, *Strongly-Checked* e *Safe* rispettivamente gli insiemi dei linguaggi sound, strongly checked e safe, abbiamo dunque le seguenti inclusioni:

$$Sound \subseteq Strongly\text{-}Checked \subseteq Safe.$$

Vi sono linguaggi che non garantiscono nemmeno la sicurezza. Linguaggi tipati ma non sicuri si dicono *weakly checked*, ovvero: a controllo debole. *Pascal* appartiene a questa categoria. Va notato che, dal punto di vista della mantenibilità del codice, un linguaggio tipato, anche se debolmente, è superiore ad uno non tipato, anche se sicuro. La seguente tabella fornisce esempi di linguaggi in varie combinazioni di tipaggio e sicurezza.

	Typed	*Untyped*
Safe	ML	LISP
Unsafe	C	Assembler

La definizione di una *teoria dei tipi*, ovvero un sistema dei tipi dato sotto forma di regole *formali* di deduzione, permette, ove sia definita anche una semantica formale, di dimostrare matematicamente proprietà dinamiche del linguaggio. Considerando "legali" solo quei programmi ai quali è possibile assegnare un tipo nella teoria, il cosiddetto *teorema di correttezza* (attenzione: correttezza del sistema dei tipi, non del linguaggio) afferma che nessun programma legale genera mai errori proibiti. Un altro risultato che mette in relazione la semantica statica con quella dinamica è quello che afferma che il tipo di un termine non cambia durante la valutazione: se t è di tipo T e $t \Rightarrow v$, allora anche v è di tipo T. In genere la dimostrazione avviene per induzione strutturale.