Gestione degli errori

Dispensa del corso di Linguaggi e Traduttori

A.A. 2005-2006

Giovanni Semeraro

La maggior parte dei compilatori dedica gran parte di tempo e spazio alla gestione degli errori.

- Generazione dell'errore;
- Individuazione dell'errore;
- Report dell'errore;
- Recovery dell'errore;
- Riparazione dell'errore.

Per molti utenti piu' importanti della velocita' del compilatore e del codice prodotto.

Generazione dell'errore

La particolare progettazione di un linguaggio puo' facilitare alcuni tipi di errori rispetto ad altri.

Report dell'errore

Dovrebbe informare l'utente su dove e' l'errore e, se possibile, di quale errore si tratti.

In **line 2, reserved word** is mispelled

Ci sono alcuni schemi che poi vengono opportunamente riempiti con parametri attuali relativi al particolare programma.

Un altro problema e' decidere se fare un report dell'errore solo la prima volta che si presenta, oppure tutte le volte.

Individuazione e Recovery dell'errore

Individuazione e recovery sono combinati fra di loro.

Trovato l'errore, il recovery cerca di eliminarlo o ignorarlo per poter continuare il parsing.

II recovery si avvale di tecniche euristiche in quanto e' impossibile garantire che il recovery non generi invece degli ulteriori errori "spuri" o ignori altri errori che erano invece presenti.

Fortunatamente gran parte degli errori si riducono a semplici errori quali dimenticare i terminatori dei blocchi (BEGIN END in Pascal), delle istruzioni (";" in Pascal) od errori di battitura.

In questi casi, si puo' semplicemente saltare una parte del programma sorgente fino ad arrivare ad un nuovo delimitatore od all'inizio di un nuovo costrutto.

Ovviamente si possono in questo modo non scoprire altri errori.

Parsing LL

Abbiamo un input ed uno stack.
Grammatica:
S → aAS
S → bA
A → cA
A → d

Input
ab...

Top-down parser:

Stack	Input
$ S	ab..
$ SAa	ab..
$ SA	b..

A questo punto poiche' A e' al top dello stack e poiche' A puo' solo derivare stringhe con c o d si trova un errore.

II report puo' essere:

UNEXPECTED b

oppure

EXPECTING c OR d

Per il recovery dobbiamo aggiungere o cancellare qualcosa dallo stack o dall'input a seconda dei casi.

In questo caso possiamo rimpiazzare b con c o d.

Parsing ricorsivo discendente

L'identificazione e' abbastanza semplice poiche' e' gia' nel codice.

II recovery dell'errore e' piu' complicato poiche' il parser puo' gia' essere di molti livelli all'interno della ricorsione.

II metodo qui descritto e' di Wirth (1976). Si consideri una procedura ricorsiva discendente generica che implementa la produzione:
$N \rightarrow \alpha$

PROCEDURE N
BEGIN
.....
codice per α
.....
END

II primo token restituito dall'analisi lessicale su α dovrebbe essere in FIRST(α).
Quindi, all'inizio della procedura N, si dovrebbe fare un test per controllare cio'. In caso di fallimento, si potrebbero saltare dei tokens fino a trovare un token che sia in FIRST(α).

Analogamente, all'uscita della procedura, il prossimo token dovrebbe essere in FOLLOW(N). In caso di fallimento, si potrebbero saltare dei tokens fino a trovare un token che sia in FOLLOW(N) o un token speciale che funga da delimitatore.

```
PROCEDURE N
BEGIN
Test(FIRST(α))
.......
codice per α
.......
Test(FOLLOW(N))

END

Test (ValidSet)
IF NextToken NOT IN Validset
THEN error
WHILE NexToken NOT IN Validset DO
NextToken:= GetToken
ENDWHILE
ENDIF
```

Questa organizzazione puo' essere migliorata. Un problema e' infatti se FIRST(α) e' stato completamente omesso.

II parser puo' ancora continuare se permettiamo questa omissione e cerchiamo un FOLLOW(N). Se lo troviamo ritorniamo alla procedura che ha chiamato N pretendendo di averlo trovato.

```
PROCEDURE N
BEGIN
Test(FIRST(α) ∪ FOLLOW(N))
IF NextToken IN FIRST(α) THEN
.....
codice per α
.....
Test(FOLLOW(N))
ENDIF
END
```

Possiamo ancora essere piu' precisi. L'errore infatti potrebbe essere che anche il FOLLOW(N) e' stato omesso.

Possiamo comunque continuare il parsing se troviamo un FOLLOWer del chiamante di N o un simbolo speciale che consenta al chiamante di continuare.

In questo caso dobbiamo passare l'insieme FOLLOW del chiamante come parametro.

```
PROCEDURE N (FollowSet: SetOfTokens)
BEGIN
Test(FIRST(α) ∪ FollowSet)
IF NextToken IN FIRST(α) THEN
.....
codice per α
.....
Test(FollowSet)
ENDIF
END
```

II chiamante di N ha adesso la responsabilita' di passare un insieme FOLLOW appropriato per N.

Esempio:
La procedura Fattore e' chiamata da Termine.
L'insieme FOLLOW per Fattore e' aggiornato con il suo terminatore *.

```
PROCEDURE Term(FollowSet: SetOfTokens)
.......
Factor(FollowSet ∪ {*})
.......
END
```

Riparazione dell'errore

L'unico scopo del recovery e' arrivare ad una nuova situazione per continuare il parsing, senza oramai produrre piu' il codice (l'unico obiettivo e' individuare eventualmente altri errori).

Altre volte, per casi molto semplici, il compilatore puo' provvedere alla correzione dell'errore vera e propria, per poi produrre il codice oggetto.

Ambienti di programmazione

Ambienti integrati: l'editor (spesso **diretto dalla sintassi)** puo' essere chiamato dal compilatore e viceversa (Turbo Pascal, Lisp). In questo caso e' molto più semplice la riparazione degli errori.

II compilatore si sospende, attende che l'utente esegua la modifica opportuna con l'editor e poi riparte da dove si era sospeso.

Produzioni con errori

Alcuni errori sono tanto comuni che i compilatori possono prevederli.

Ad esempio, in Pascal, dimenticare END, mettere ";" prima di ELSE ecc.

E' semplice aggiungere queste situazioni direttamente alle produzioni.

Esempio:

Program → BEGIN Statement END.
| BEGIN Statement {Error Production}

Statement → IF Condition THEN Statement
ELSE Statement
| IF Condition THEN Statement;
ELSE Statement {Error Production}