Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2013-2014

Linguaggi Formali e Compilatori

Trattamento degli errori

Giacomo PISCITELLI

Trattamento degli errori

Comunemente gli errori di programmazione possono essere di varia natura:

- ✓ **lessicali**, quando derivano da cattiva ortografia di identificatori, parole chiave, operatori o stringhe erroneamente non delimitate da apici;
- ✓ **sintattici**, quando sono determinati da simboli **;** mal posizionati, parentesi tonde o graffe non bilanciate, comparsa di token (**case**) che non sono regolarmente preceduti da altri token (**switch**);
- ✓ **semantici**, quando sono dovuti a non corrispondenza tra operatori e operandi oppure di numero o di tipo tra parametri;
- ✓ **logici**, quando sono dovuti a bachi dell'algoritmo risolutore o a cattiva conoscenza o uso del linguaggio, con improprio uso di operatori.

Error kind	Example	Detected by ...
Lexical	... $ ...	Lexer
Syntax	... x *% ...	Parser
Semantic	... int x; y = x(3); ...	Type checker
Correctness	your favorite program	Tester/User

Trattamento degli errori

Un motivo per trattare l'argomento dell'*error recovery* assieme all'analisi sintattica è che molti errori, qualunque sia la loro causa, appaiono essere di natura sintattica e vengono *evidenziati appena l'analisi sintattica di una frase non può procedere*.

Anche se pochi linguaggi tengono conto di tutti i possibili errori e delle possibili reazioni al loro verificarsi, *la precisione dei metodi d'analisi dei linguaggi di programmazione permette di rivelare gli errori sintattici con notevole efficienza*.

Diversi metodi di parsing, quali LL e LR, rilevano gli errori quasi immediatamente, non appena il flusso di token dallo scanner non è più congruente con la grammatica.

Più precisamente i parser hanno solitamente la proprietà ***"viable prefix"*** (prefisso ammissibile): sono in grado, cioè, di rilevare un errore non appena si presenta nell'input un prefisso non valido al fine di costruire una corretta stringa del linguaggio.

Trattamento degli errori

Un parser deve essere in grado di *scoprire*, *diagnosticare* gli errori in maniera efficiente (ed efficace ai fini di correzione), *riprendere* l'analisi per scoprire nuovi errori. Obiettivo apparentemente semplice, ma molto complesso da perseguire.

La Diagnosi dell'errore deve essere chiara e accurata, meglio se con indicazione del numero di linea contenente l'errore e di un puntatore alla posizione nella linea.

Il Recupero deve prevedere la ripresa del processo di analisi, con minimo appesantimento dell'elaborazione (*processing overhead*).

Trattamento degli errori

Good error handling is not easy to achieve

Dopo aver rivelato un errore, come fare a recuperare e riprendere l'analisi?

Sebbene nessuna strategia si sia rivelata accettabile in generale, un certo numero di metodi si sono dimostrati di buona applicabilità.

L'approccio più semplice è quello di evidenziare un messaggio informativo appena rivelato l'errore, senza segnalare ulteriori errori prima di aver ripristinato la capacità del parser di riprendere l'analisi.

Infatti, se gli errori si accumulano oltre un certo limite, è meglio rinunciare a segnalare un'inutile e incomprensibile valanga di inesattezze.

Trattamento degli errori

Per quanto attiene l'error recovery, le strategie più frequentemente adoperate sono 4:

1) panic mode: scoperto l'errore, il parser riprende l'analisi in corrispondenza di alcuni token selezionati, detti *token sincronizzanti* (es.: delimitatori `begin end ; }` che hanno un ruolo chiaro e non ambiguo nel sorgente) *scartando* alcuni caratteri.

Svantaggi: può essere *scartato* molto input, ma ha il vantaggio della semplicità e di evitare di produrre un *loop* infinito;

- Simplest, most popular method
- When an error is detected:
  - Discard tokens until one with a clear role is found
  - Continue from there
- Such tokens are called synchronizing tokens
  - Typically the statement or expression terminators
- Panic-mode recovery:
  - Skip ahead to next integer and then continue
- Bison: use the special terminal error to describe how much input to skip

Trattamento degli errori

2) phrase level: scoperto l'errore, il parser può apportare correzioni locali sul resto dell'input inserendo/ modificando/ cancellando alcuni terminali per poter riprendere l'analisi (es.: scambiando '**,**' con '**;**', cancellando o inserendo '**;**')

Svantaggi: possibili *loop* infiniti, difficoltà a trattare situazioni in cui l'errore è avvenuto prima del punto di rivelazione;

Trattamento degli errori

3) error productions: viene fatto uso di produzioni che estendono la grammatica per generare gli errori più comuni.

Metodo efficiente per la diagnostica.

- Idea: specify in the grammar known common mistakes
- Essentially promotes common errors to alternative syntax
- Example:
  - Write 5 x instead of 5 * x
  - Add the production E → ... | E E
- Disadvantage
  - Complicates the grammar

Trattamento degli errori

4) global correction: si cerca di "calcolare" la migliore correzione possibile alla derivazione errata (minimo costo di interventi per inserzioni/cancellazioni/).

Metodo d'interesse teorico ma poco usato in pratica, se non per attuare strategia "phrase level".

- Idea: find a correct "nearby" program
  - Try token insertions and deletions
  - Exhaustive search

- Disadvantages:
  - Hard to implement
  - Slows down parsing of correct programs
  - "Nearby" is not necessarily "the intended" program
  - Not all tools support it

Trattamento degli errori

Not all are supported by all parser generators

Error Recovery in BISON

Un mixing di Error productions e Panic Mode

Il simbolo terminale **error** è sempre definito: non c'è bisogno di dichiararlo ed è riservato per la gestione degli errori.

Il parser Bison genera un **error** token ogni volta che viene incontrato un errore sintattico. Se è stata prevista una regola che riconosca il token nel contesto corrente, il parser può continuare nell'analisi.

Per esempio:

```
stmnts:    /* empty string */
         | stmnts '\n'
         | stmnts exp '\n'
         | stmnts error '\n'
```

La quarta ed ultima regola dell'esempio prevede che un token **error** seguito da un newline rappresenta una valida alternativa per riconoscere il non-terminale **stmnts**.

Cosa succede se un errore sintattico avviene nel mezzo di uno statement (per esempio di una `exp`: per una spiegazione del modo di funzionare dell'esempio e, in generale, delle modalità di ripresa del processo di analisi in BISON, cfr. Capitolo 6 Error Recovery del manuale.

BISON: Error Reporting Function yyerror

Il parser generato da BISON rivela un errore sintattico (o parse error) ogni volta che legge un token che non può soddisfare alcuna regola sintattica. Un'azione nella grammatica può anche esplicitamente "produrre" una rivelazione d'errore usando la macro **YYERROR**.

In tal caso il parser generato da BISON si aspetta di riportare l'errore chiamando la specifica funzione di nome **yyerror**, che deve essere fornita normalmente nell'Epilogo.

Tale funzione viene chiamata da **yyparse** ogni volta che viene rivelato un errore e riceve da yyparse un argomento. Per un errore sintattico, la stringa passata è normalmente "**syntax error**".

Per ulteriori dettagli sulle modalità di reporting di un errore si rimanda al par. 4.7 The Error Reporting Function yyerror del manuale.

Trattamento degli errori – *passato e presente*

Syntax Error Recovery: Past and Present

- Past
  - Slow recompilation cycle (even once a day)
  - Find as many errors in one cycle as possible
  - Researchers could not let go of the topic

- Present
  - Quick recompilation cycle
  - Users tend to correct one error/cycle
  - Complex error recovery is less compelling
  - Panic-mode seems enough