Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2011-2012

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 vengono ideati tenendo conto dei 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*"**: 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.

Un parser deve essere in grado di *scoprire*, *diagnosticare* gli errori in maniera efficiente (ed efficace ai fini di correzione), e di *riprendere* l'analisi e scoprire nuovi errori.

- ✓ Scoperta degli errori
- ✓ Diagnosi dell'errore in termini chiari e accurati, preferibilmente con indicazione del numero di linea contenente l'errore e di un puntatore alla posizione nella linea
- ✓ Recupero e rapida ripresa del processo di analisi, con minimo *processing overhead*

Strategie di riparazione

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

Strategie di riparazione

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

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;

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;

error productions: viene fatto uso di produzioni che estendono la grammatica per generare gli errori più comuni. Metodo efficiente per la diagnostica.

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".

Strategie di riparazione

Not all are supported by all parser generators

Strategie di riparazione

Error Recovery: Panic Mode

- 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
- Consider the erroneous expression

  (1 + + 2) + 3
- Panic-mode recovery:
  - Skip ahead to next integer and then continue
- Bison: use the special terminal error to describe how much input to skip

  E → int | E + E | (E) | error int | (error)

Strategie di riparazione

Syntax Error Recovery: Error Productions

- 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

Strategie di riparazione

Error Recovery: **Local and Global Correction**

- 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

- **Analisi ascendente**
  - ✓ Riparazione locale
  - ✓ Produzione di errori
- **Analisi discendente**
  - ✓ Albero connesso
  - ✓ Predecessori e successori
  - ✓ Sincrotriple

Attenzione

L'analizzatore sintattico può accorgersi dell'errore con grande ritardo. Infatti l'errore può produrre un sintomo solo quando il prefisso analizzato non è più completabile in modo da derivare una frase sintatticamente corretta del linguaggio.

Esempio

$L = ab^*aa \cup cb^*cc$

Data la frase ab^ncc

Primo errore scoperto dopo ab^n

➔ **Correzione a distanza minima**

Gestione degli errori in parser predittivi

Un errore può verificarsi in un parser predittivo (LL(1))

- » Se il simbolo terminale in cima alla pila non corrisponde con il simbolo di ingresso
- » Se il simbolo in cima alla pila è un simbolo non terminale **A**, e il simbolo di ingresso è **a**, e **M[A,a]** è vuoto.

Cosa deve fare il parser quando si verifica un errore?

Il parser dovrebbe:

- » essere capace di fornire un messaggio di errore (più veritiero possibile).
- » recuperare l'errore ed essere capace di continuare il parsing della stringa d'ingresso.

Metodo dell'albero connesso

Scoperto un errore

- » cerca una ripresa ipotizzando la mancanza di una sottostringa nella pila.
- » se fallisce tenta una correzione saltando uno o più caratteri fino a quando l'analisi può riprendere.

```
void errore {
    do {
        do {
                if (la configurazione è valida)
                    { stampa dignostico
                        return}
                scarta il simbolo in cima alla pila
            } while (pila non è vuota)
        ripristina la pila
        avanza la testina di lettura
        } while la stringa è terminata
                }
```

Scelta dei token di sincronizzazione

Definire per ogni non terminale la **sincrotripla**:

(marca di apertura, non terminale, marca di chiusura)

1) Scartare tutti i caratteri fino a quando si trova un simbolo, specificatamente un simbolo finale di una stringa derivabile da A
2) Se c'è specificatamente il carattere iniziale di una stringa

Scelta delle sincrotriple

» Marca di apertura
First (A)

» Marca di chiusura
Follow(A)

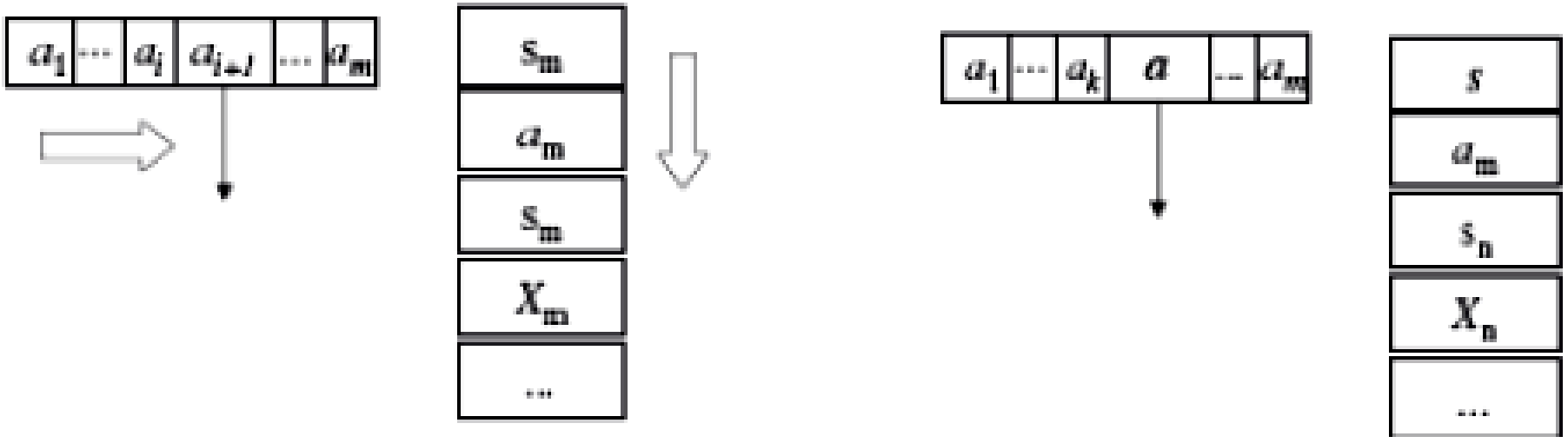
Scelta dei token di sincronizzazione

Tecniche "panic mode" per la scelta dei token di sincronizzazione:

- Per i non terminali *A:* FOLLOW(*A*) - si scartano tutti i terminali fino a trovarne uno in FOLLOW(*A*), quindi si fa pop(*A*)
- E' possibile considerare altri simboli, che terminano il costrutto corrente (es. '**;**' alla fine di uno statement di assegnazione).
- Altri esempi: per le espressioni le keyword delle istruzioni (**if** <expr> **then** …), per le istruzioni i blocchi, ecc.

Analisi LR -Trattamento degli errori – *panic mode*

Tecnica "panic mode": isolare la frase che contiene l'errore e riprendere l'analisi non appena possibile.
Quando scopre un errore l'automa percorre lo stack verso il basso fino a un non terminale sullo stack per cui goto[*A*,*a*]=*s*; pone *s* sullo stack e continua l'analisi.
Se si scorre tutta la pila, si passa al prossimo carattere di input, e così via



La scelta dei simboli può essere fatta in modo da isolare il tipo di frase (es. '**;**' per le espressioni)

Analisi LR -Trattamento degli errori – *phrase level*

Tecnica "phrase level": cercare di ripristinare l'analisi apportando correzioni "locali" alla frase errata.
Quando scopre un errore su un carattere ***a*** l'automa cerca di inserire un carattere ***b*** per cui è definita la "goto": la riparazione è accettata se l'analisi può riprendere da ***b***, altrimenti si ripristina la pila iniziale.
La tabella di parse "action[*A,a*]" può contenere esplicitamente il riferimento ad una funzione di gestione errore adatta al contesto.

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