

Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2014/2015

L'input per il parser

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Nel contesto della compilazione l'input per il parser è costituito da una stringa di token (che indicheremo genericamente con θ) generata dall'analizzatore lessicale.
- ▶ Per semplicità di linguaggio ignoreremo la presenza dello scanner e supporremo che il parser legga direttamente θ dallo stream di input.
- ▶ Per individuare la “fine” della stringa di input, negli algoritmi di parsing supporremo che la stringa stessa sia terminata da uno speciale simbolo, non presente fra i token del linguaggio, ad esempio \$.

L'input per il parser

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Se S è l'assioma iniziale della grammatica, per tenere conto del simbolo di terminazione introduciamo “formalmente” un nuovo assioma S' , con la sola produzione $S' \rightarrow S\$$.
- ▶ In questo modo, $S \Rightarrow^* \theta$ se e solo se $S' \Rightarrow^* \theta\$$.
- ▶ Nel seguito lasceremo implicita questa “aggiunta” alla grammatica, a meno che non risulti importante considerarla per comprendere meglio qualche altro concetto.

Tipi di parser

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Una prima classificazione suddivide il parsing in accordo all'ordine di costruzione del parse tree per θ .
- ▶ Nel parsing *top-down* l'albero viene costruito a partire dalla radice, corrispondente all'assioma iniziale.
- ▶ Equivalentemente, possiamo dire che nel parsing top-down si cerca una derivazione canonica sinistra per θ partendo da S' .
- ▶ Si noti che la costruzione top-down dell'albero di derivazione corrisponde in modo naturale al riconoscimento di categorie sintattiche (es, un comando o una espressione) in termini delle parti costituenti.

Tipi di parser (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Nel parsing *bottom-up* il parse tree per θ viene costruito procedendo dalle foglie verso la radice.
- ▶ Equivalentemente, possiamo dire che nel parsing bottom-up si cerca una derivazione canonica (destra) per la stringa $\theta\$$ applicando *riduzioni* successive.
- ▶ Una riduzione non è nient'altro che l'applicazione "in senso opposto" di una produzione della grammatica.
- ▶ Si noti che la costruzione bottom-up dell'albero di derivazione corrisponde in modo naturale al riconoscimento di singole porzioni di un programma e alla loro composizione in parti più complesse.

Tipi di parser (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ I tipi di parser più diffusi includono:
 - ▶ parser a *discesa ricorsiva* con backtracking,
 - ▶ parser a discesa ricorsiva senza backtracking (parsing *predittivi*),
 - ▶ parser di tipo *shift-reduce*.
- ▶ I parser a discesa ricorsiva sono di tipo top-down, mentre i parser shift-reduce sono di tipo bottom-up.
- ▶ Considereremo sottoinsiemi di grammatiche libere per cui si possono costruire parser efficienti a discesa ricorsiva (grammatiche $LL(1)$) o di tipo shift-reduce (grammatiche $LR(1)$)

Scelta della produzione da usare al generico passo

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ In entrambi i tipi di parser (top-down o bottom-up) la scelta della produzione da utilizzare (rispettivamente, in avanti o all'indietro) viene effettuata in funzione dello *stato* interno del parser e della prossima *porzione di input* (1 o più caratteri).
- ▶ Come vedremo, lo stato interno del parser sarà tipicamente espresso dall'informazione memorizzata nella cima di una struttura dati stack.
- ▶ Il numero di caratteri di input considerati per prendere la decisione è noto invece come *lookahead*.
- ▶ Noi saremo interessati di norma ad un lookahead di un carattere.

Algoritmo generale

- ▶ Un parser a discesa ricorsiva (d.r.) costruisce il parse tree (eventualmente non in modo esplicito) a partire dall'assioma ed esaminando progressivamente l'input.
- ▶ Al generico passo, l'algoritmo è idealmente "posizionato" su un nodo x dell'albero.
- ▶ Se il nodo è una foglia etichettata con un simbolo terminale a , l'algoritmo controlla se il prossimo simbolo in input coincide con a .
- ▶ In caso affermativo fa avanzare il puntatore di input, altrimenti (nel caso più semplice) dichiara errore.
- ▶ Se invece il nodo è un simbolo non terminale A , sceglie una produzione $A \rightarrow X_1 X_2 \dots X_k$, crea i nodi (figli di A) etichettandoli con X_1, X_2, \dots, X_k , e passa ricorsivamente ad esaminare tali nodi, da sinistra verso destra.

Algoritmo generale (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Da un punto di vista implementativo, un parser a d.r. può essere realizzato come una collezione di procedure mutuamente ricorsive, una per ogni simbolo non terminale della grammatica.
- ▶ Il problema fondamentale consiste nella “scelta” della produzione da applicare, nel caso in cui (per un dato non terminale) ne esista più d’una.
- ▶ Lo pseudocodice nella diapositiva seguente lascia aperto questo problema, che andremo successivamente a risolvere in almeno due modi diversi.

Procedura per il generico non terminale A

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- 1: Scegli “opportunamente” una produzione
 $A \rightarrow X_1 X_2 \dots X_k$ ($X_j \in \mathcal{V}$)
- 2: **for** $j = 1, \dots, k$ **do**
- 3: **if** $X_j \in \mathcal{N}$ **then**
- 4: $X_j()$
- 5: **else**
- 6: $x \leftarrow \text{READ}()$
- 7: **if** $X_j \neq x$ **then**
- 8: $\text{ERROR}()$ {Include il caso $x = \text{EOF}$ }

Esempio

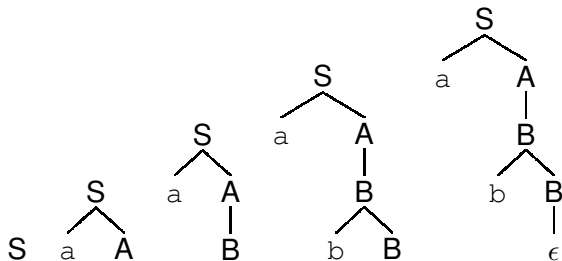
- ▶ Si consideri la seguente grammatica, che genera il “solito” linguaggio $\{a^n b^m : n > 0, m \geq 0\}$:

$$S \rightarrow aA$$

$$A \rightarrow aA \mid B$$

$$B \rightarrow bB \mid \epsilon$$

- ▶ Su input ab un parser a d.r. nondeterministico produce la seguente derivazione:



Implementazione concreta

- ▶ Per eliminare il non determinismo insito nel codice precedente, una prima soluzione consiste nell'esplorare tutte le possibili produzioni relative al generico non terminale A prima eventualmente di dichiarare errore.
- ▶ Se una particolare produzione fallisce, ma prima del fallimento sono stati letti simboli di input, è necessario operare un *backtracking* sull'input stesso.
- ▶ Per i parser a discesa ricorsiva ciò può essere sufficientemente agevole (dal punto di vista dell'implementatore), anche se computazionalmente pesante.
- ▶ Questa prima variante è mostrata nella diapositiva successiva.

Procedura per il generico non terminale A , con backtracking

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

```
1: SAVEINPUTPOINTER()
2: for all produzione  $A \rightarrow X_1 X_2 \dots X_k$  ( $X_j \in \mathcal{V}$ ) do
3:   for  $j = 1, \dots, k$  do
4:     if  $X_j \in \mathcal{N}$  then
5:        $X_j()$ 
6:     else
7:        $x \leftarrow \text{READ}()$ 
8:       if  $X_j \neq x$  then
9:         RESTOREINPUTPOINTER()
10:      break
      {Solo nel codice per l'assioma iniziale}
11: if not EOF() then
12:   ERROR()
```

Esempio

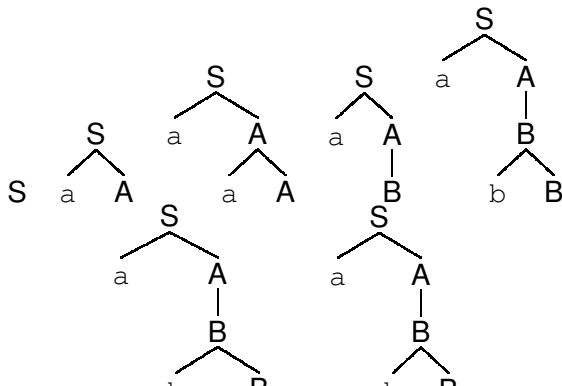
- Riconsideriamo la grammatica

$$S \rightarrow aA$$

$$A \rightarrow aA \mid B$$

$$B \rightarrow bB \mid \epsilon$$

- Su input ab un parser a d.r. con backtracking potrebbe produrre la seguente derivazione:



Parsing a discesa ricorsiva (continua)

- ▶ Se analizziamo attentamente il codice del parser a d.r., comprendiamo perché una grammatica con ricorsioni a sinistra sia inadatta al parsing a discesa ricorsiva.
- ▶ Supponiamo, infatti, che ad un determinato passo il parser sia “posizionato” su un nodo (etichettato con il non terminale) A dell'albero.
- ▶ Supponiamo inoltre che la prima produzione che viene (tentativamente) applicata abbia una ricorsione a sinistra, sia cioè del tipo $A \rightarrow A\alpha$ (dove α è una qualunque stringa di terminali e/o non terminali).
- ▶ Accade allora che il codice relativo al non terminale A :
 - ▶ consideri il primo simbolo della parte sinistra della produzione, che è ancora A ;
 - ▶ chiami ricorsivamente la procedura per A , innescando così un ciclo infinito.

Esempio

- ▶ Per la grammatica con precedenza di operatori (che abbiamo già incontrato):

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \mathbf{id} \mid (E)$$

le procedure per i non terminali E e T innescano potenzialmente un ciclo infinito.

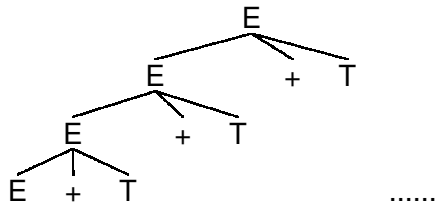
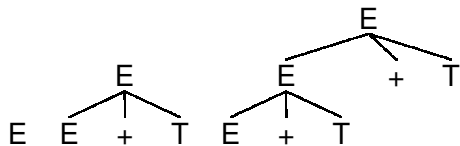
- ▶ Ad esempio, su input $\mathbf{id} + \mathbf{id}$, la produzione corretta da applicare inizialmente è $E \rightarrow E + E$ (se si applica $E \rightarrow \mathbf{id}$ la derivazione fallisce e bisogna operare backtracking sull'input), ma questa innesca un ciclo infinito.

Esempio (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva



Parsing a discesa ricorsiva (continua)

- ▶ Una soluzione migliore rispetto all'esplorazione esaustiva delle possibili derivazioni consiste nell'usare un certo numero di caratteri di lookahead per decidere la prossima produzione da utilizzare.
- ▶ Naturalmente, se questa strada possa essere percorsa con successo dipende dalla grammatica.
- ▶ Consideriamo la semplice grammatica:

$$A \rightarrow aA \mid bB$$

$$B \rightarrow \epsilon \mid bB$$

- ▶ Per entrambi i non terminali, la scelta della produzione da usare può essere fatta guardando solo il prossimo simbolo x in input.
 - ▶ Per A : se $x = a$, usa la produzione $A \rightarrow aA$; se $x = b$, usa la produzione $A \rightarrow bB$.
 - ▶ Per B : se $x = \$$ (end of input), usa la produzione $B \rightarrow \epsilon$; se $x = b$, usa la produzione $B \rightarrow bB$;

Grammatiche $LL(1)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Un parser predittivo può essere realizzato agevolmente nel caso di grammatiche cosiddette $LL(1)$.
- ▶ La doppia L sta per Left-Left, ad indicare che l'input è letto da sinistra verso destra e che la derivazione prodotta è canonica sinistra.
- ▶ Il “parametro” 1 indica che un carattere di lookahead è sufficiente per decidere correttamente la produzione da utilizzare.
- ▶ Più in generale, si possono considerare grammatiche $LL(k)$, dove sono sufficienti (e, in generale, necessari) k caratteri di lookahead.

Grammatiche “non” $LL(k)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Nessuna grammatica con ricorsioni a sinistra può essere $LL(k)$, per nessun k .
- ▶ Anche nel caso in cui esistano produzioni con prefissi comuni la quantità di lookahead necessaria non è limitabile a priori.
- ▶ Un esempio grammatica con prefissi comuni è data dal più volte esaminato caso del “dangling else”.

$$S \rightarrow I \mid A$$
$$I \rightarrow \mathbf{if} \ B \ \mathbf{then} \ S \mid \mathbf{if} \ B \ \mathbf{then} \ S \ \mathbf{else} \ S$$
$$A \rightarrow \mathbf{a}, \quad B \rightarrow \mathbf{b}$$

- ▶ Infatti, la quantità di codice che può essere presente fra le keyword **then** e **else** non è limitabile a priori.

Grammatiche “non” $LL(k)$ (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Un altro esempio è dato dalla grammatica:

$$A \rightarrow aB \mid aC$$

$$B \rightarrow aB \mid b$$

$$C \rightarrow aC \mid c$$

- ▶ Su input $a^n b$ è necessario un lookahead di $n + 1$ caratteri per decidere inizialmente che la produzione corretta è $A \rightarrow aB$.

Grammatiche $LL(1)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Analizziamo ora in dettaglio le caratteristiche che deve avere una grammatica G per potersi qualificare $LL(1)$.
- ▶ Consideriamo un generico non terminale per il quale esistano almeno due produzioni, poniamo

$$A \rightarrow \alpha \mid \beta$$

- ▶ Se vale simultaneamente che $\alpha \xRightarrow{*} a\alpha'$ e $\beta \xRightarrow{*} a\beta'$, cioè se da α e da β si possono derivare stringhe che iniziano con lo stesso simbolo non terminale, allora G non è $LL(1)$.

Esempi

- ▶ Il caso più semplice è quando α e β iniziano con lo stesso simbolo terminale.
- ▶ Ad esempio, se la grammatica contiene le produzioni $S \rightarrow aA \mid aB$ non può essere $LL(1)$.
- ▶ Tuttavia il problema può non essere di così immediata evidenza.
- ▶ Ad esempio, la grammatica

$$S \rightarrow aA \mid B$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow b \mid C$$

$$C \rightarrow aB \mid c$$

non è $LL(1)$ perché $B \Rightarrow C \Rightarrow aB$ e dunque la procedura per il non terminale S , su input $a\dots$ non può decidere quale produzione usare ($S \rightarrow aA$ oppure $S \rightarrow B$) guardando solo il primo simbolo di input.

Grammatiche $LL(1)$ (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ La situazione appena descritta non è l'unica che deve essere evitata affinché la grammatica sia $LL(1)$.
- ▶ Un secondo vincolo è che, sempre considerando una coppia di produzioni $A \rightarrow \alpha \mid \beta$, se risulta $\alpha \xRightarrow{*} \epsilon$ allora non può accadere che $\beta \xRightarrow{*} \epsilon$ (e viceversa).
- ▶ Infatti, in caso contrario, qualunque sia il prossimo simbolo di input entrambe le produzioni potrebbero teoricamente essere valide.
- ▶ Anche questo non basta ancora; è infatti necessario che sia verificata un'ultima condizione.
- ▶ Se $\alpha \xRightarrow{*} \epsilon$ e $\beta \xRightarrow{*} a\beta'$, allora non deve accadere che $\mathcal{S} \xRightarrow{*} \gamma_1 A a \gamma_2$, dove $\beta', \gamma_1, \gamma_2 \in \mathcal{V}^*$.
- ▶ In altri termini a non deve apparire subito dopo A in alcuna forma di frase.

Esempio

- ▶ Si consideri la grammatica

$$S \rightarrow A \mid B$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow Aab$$

- ▶ Su input la stringa “a” la procedura per il non terminale S non può decidere correttamente la produzione da usare.
- ▶ Risulta infatti:

$$S \Rightarrow A \Rightarrow aA \Rightarrow a$$

$$S \Rightarrow A \Rightarrow aA \Rightarrow a$$

e

$$S \Rightarrow B \Rightarrow Aab \Rightarrow ab$$

FIRST e FOLLOW

- ▶ Per esprimere in modo compatto le condizioni che qualificano come $LL(1)$ una data grammatica, introduciamo le funzioni *FIRST* e *FOLLOW*.
- ▶ Data $G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ e data una stringa $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$, si definisce $FIRST(\alpha)$ l'insieme dei simboli terminali con cui può iniziare una frase derivabile da α , più eventualmente ϵ se $\alpha \Rightarrow^* \epsilon$:

$$FIRST(\alpha) = \{x \in \mathcal{T} \mid \alpha \Rightarrow^* x\beta, \beta \in \mathcal{T}^*\} \cup \{\epsilon\}, \text{ se } \alpha \Rightarrow^* \epsilon.$$

- ▶ Per ogni non terminale $A \in \mathcal{N}$, $FOLLOW(A)$ è l'insieme dei terminali che si possono trovare immediatamente alla destra di A in una forma di frase di una qualche derivazione canonica (destra o sinistra):
 $x \in FOLLOW(A)$ se $S \Rightarrow^* \alpha Ax\beta$, con $\alpha, \beta \in \mathcal{V}^*$.

Calcolo di $FIRST(\alpha)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Definiamo innanzitutto come si calcola $FIRST(\alpha)$ nel caso in cui α sia un singolo simbolo della grammatica, cioè $\alpha = X$ con $X \in \mathcal{N} \cup \mathcal{T}$.
 - ▶ Se X è un terminale, si pone naturalmente $FIRST(X) = \{X\}$;
 - ▶ se X è un non terminale il calcolo procede per passi, con l'inizializzazione $FIRST(X) = \{\}$.
 - ▶ Se esiste la produzione $X \rightarrow X_1 \dots X_n$, e risulta $\epsilon \in FIRST(X_j), j = 1, \dots, k - 1$, poniamo $FIRST(X) = FIRST(X) \cup \{x\}$ per ogni simbolo $x \in FIRST(X_k)$.
 - ▶ Infine, se esiste la produzione $X \rightarrow \epsilon$ oppure $\epsilon \in FIRST(X_j), j = 1, \dots, k$, poniamo $FIRST(X) = FIRST(X) \cup \{\epsilon\}$.

- ▶ Si riconsideri la grammatica per le espressioni che “forza” la precedenza di operatori:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \mathbf{id} \mid (E)$$

- ▶ Per questa grammatica risulta
 - ▶ $FIRST(F) = \{ (, \mathbf{id} \}$;
 - ▶ $FIRST(T) = FIRST(F)$;
 - ▶ $FIRST(E) = FIRST(T)$.

Esempi

- ▶ La seguente grammatica genera lo stesso linguaggio della precedente

$$E \rightarrow (E)E' \mid \mathbf{id} E'$$

$$E' \rightarrow +EE' \mid \times EE' \mid \epsilon$$

- ▶ Risulta

- ▶ $FIRST(E) = \{(, \mathbf{id}\}$
- ▶ $FIRST(E') = \{+, \times, \epsilon\}$

- ▶ Si consideri ora la grammatica per $a^n b^m c^k$

$$A \rightarrow aA \mid BC \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

- ▶ Per questa grammatica risulta

- ▶ $FIRST(C) = \{c, \epsilon\}$
- ▶ $FIRST(B) = \{b, \epsilon\}$
- ▶ $FIRST(A) = \{a, b, c, \epsilon\}$

Calcolo di $FIRST(\alpha)$ (continua)

- ▶ Il calcolo di $FIRST(\alpha)$, dove $\alpha = X_1 \dots X_n$ è una generica stringa di terminali e nonterminali, può ora essere svolto nel modo seguente
- ▶ $a \in FIRST(\alpha)$ se e solo se, per qualche indice $k \in 1, \dots, n$, risulta $a \in FIRST(X_k)$ e $\epsilon \in FIRST(X_j)$, $j = 1, \dots, k - 1$ (si suppone sempre $\epsilon \in FIRST(X_0)$).
- ▶ Se $\epsilon \in FIRST(X_j)$, $j = 1, \dots, n$, allora $\epsilon \in FIRST(\alpha)$.
- ▶ Ad esempio, nel caso della seconda grammatica del lucido precedente

$$A \rightarrow aA \mid BC \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

risulta: $FIRST(aA) = \{a\}$ e $FIRST(BC) = \{b, c, \epsilon\}$.

Calcolo di $FOLLOW(A)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Il calcolo di $FOLLOW(A)$, per un generico non terminale A , può essere svolto in questo modo.
- ▶ Se esiste la produzione $B \rightarrow \alpha A \beta$, tutti i terminali in $FIRST(\beta)$ si inseriscono in $FOLLOW(A)$.
- ▶ In particolare, poiché (almeno implicitamente) esiste sempre la produzione $S' \rightarrow S \$$, il simbolo speciale $\$$ sta sempre nel $FOLLOW(S)$.

Calcolo di $FOLLOW(A)$

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Se esiste la produzione $B \rightarrow \alpha A$, tutti i terminali che stanno in $FOLLOW(B)$ si inseriscono in $FOLLOW(A)$.
- ▶ Infatti, se esiste una derivazione $S \Rightarrow^* \beta B \gamma$, allora l'applicazione della produzione $B \rightarrow \alpha A$ rende evidente che esiste anche la derivazione $S \Rightarrow^* \beta \alpha A \gamma$.
- ▶ Dunque ciò che segue B in una forma di frase (cioè il $FIRST(\gamma)$) può anche seguire A .
- ▶ Si arriva alla stessa conclusione anche nel caso in cui $B \rightarrow \alpha A \beta$ e $\epsilon \in FIRST(\beta)$.

Esempio

- ▶ Consideriamo ancora la grammatica

$$E \rightarrow (E)E' \mid \mathbf{id} E'$$

$$E' \rightarrow +EE' \mid \times EE' \mid \epsilon$$

- ▶ Possiamo subito stabilire che $FOLLOW(E)$ include il simbolo \$ e il simbolo); inoltre contiene i simboli in $FIRST(E')$ (eccetto ϵ) e cioè + e \times .
- ▶ $FOLLOW(E')$ include $FOLLOW(E)$, a causa (ad esempio) della produzione $E \rightarrow \mathbf{id}E'$.
- ▶ La produzione $E' \rightarrow +EE'$, unitamente a $E' \rightarrow \epsilon$, stabilisce che vale anche il contrario, e cioè che $FOLLOW(E)$ include $FOLLOW(E')$.
- ▶ Mettendo tutto insieme si ottiene $FOLLOW(E) = FOLLOW(E') = \{\$,), +, \times\}$.

- ▶ Per la grammatica

$$A \rightarrow aA \mid BC \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

risulta $FOLLOW(A) = FOLLOW(C) = \{\$, \}$ e
 $FOLLOW(B) = \{c, \}$.

- ▶ Per la grammatica con precedenza di operatori:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \mathbf{id} \mid (E)$$

risulta:

- ▶ $FOLLOW(E) = \{\$, +,)\}$;
- ▶ $FOLLOW(T) = FOLLOW(E) \cup \{\times\}$;
- ▶ $FOLLOW(F) = FOLLOW(T)$.

Grammatiche $LL(1)$ (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ Possiamo ora rivedere in modo più compatto la nozione di grammatica $LL(1)$.
- ▶ Una grammatica è $LL(1)$ se, qualora esistano due produzioni $A \rightarrow \alpha$ e $A \rightarrow \beta$, deve risultare:

$$FIRST(\alpha) \cap FIRST(\beta) = \{\},$$

- ▶ Inoltre, se $\alpha \xRightarrow{*} \epsilon$, deve valere $FOLLOW(A) \cap FIRST(\beta) = \{\}$.
- ▶ Analogamente, se $\beta \xRightarrow{*} \epsilon$, deve valere $FOLLOW(A) \cap FIRST(\alpha) = \{\}$.

Esempio

- ▶ Si consideri la grammatica

$$A \rightarrow BE$$

$$B \rightarrow C \mid D$$

$$C \rightarrow \epsilon \mid cc$$

$$D \rightarrow \epsilon \mid dd$$

$$E \rightarrow c \mid d$$

- ▶ Poiché risulta $FIRST(C) \cap FIRST(D) = \{\epsilon\}$, la grammatica non è $LL(1)$.
- ▶ Infatti, supponiamo che la stringa in input inizi con c . Dopo la riscrittura dell'assioma il parser verrebbe a trovarsi con la forma di frase BE e il carattere c in input.
- ▶ A questo punto potrebbe essere corretto derivare tanto CE (se l'input fosse, ad esempio, ccd) quanto DE (se l'input fosse c).

Esempio

- ▶ Si modifichi la precedente grammatica nel modo seguente

$$A \rightarrow BE$$

$$B \rightarrow C \mid D$$

$$C \rightarrow \epsilon \mid cc$$

$$D \rightarrow dd$$

$$E \rightarrow c \mid d$$

(cancellando cioè la produzione $D \rightarrow \epsilon$).

- ▶ Poiché risulta $FIRST(D) \cap FOLLOW(B) = \{d\}$, la grammatica non è $LL(1)$.
- ▶ Il problema si verifica con input che inizia con d , perché potrebbe essere corretto (dopo la derivazione iniziale) derivare tanto CE (se l'input fosse d) quanto DE (se l'input fosse, ad esempio ddc).

Esempio

Parsing

Generalità sul parsing
Parser a discesa ricorsiva

- ▶ Consideriamo ancora la grammatica con precedenza di operatori:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \mathbf{id} \mid (E)$$

- ▶ Sappiamo già che tale grammatica non è $LL(1)$ perché contiene ricorsioni a sinistra.
- ▶ Possiamo anche verificare, ad esempio, che $FIRST(E) \cap FIRST(T) \supseteq \{\mathbf{id}\}$.
- ▶ Eliminando la left-recursion si può però ottenere una grammatica equivalente che è $LL(1)$

Esempio (continua)

- ▶ Per la grammatica modificata

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow \times FT' \mid \epsilon$$

$$F \rightarrow \mathbf{id} \mid (E)$$

dobbiamo solo verificare che $FOLLOW(E')$ non contenga il simbolo $+$ e che $FOLLOW(T')$ non contenga il simbolo \times .

- ▶ È facile verificare che risulta:
 - ▶ $FOLLOW(E) = FOLLOW(E') = \{\$, \})\}$;
 - ▶ $FIRST(E') = \{+, \epsilon\}$;
 - ▶ $FOLLOW(T) = (FIRST(E') \setminus \{\epsilon\}) \cup FOLLOW(E') = \{+, \$, \})\}$;
 - ▶ $FOLLOW(T') = FOLLOW(T) = \{+, \$, \})\}$;

Tabella di parsing $LL(1)$

- ▶ Per una grammatica $LL(1)$ è possibile costruire (utilizzando le funzioni *FIRST* e *FOLLOW*) una tabella, detta *tabella di parsing*, che, per ogni non terminale A e ogni terminale x , prescrive il comportamento del parser.
- ▶ Indicheremo con M_G la tabella di parsing relativa alla grammatica G (o semplicemente con M , se la grammatica è evidente).
- ▶ La generica entry $M_G[A, x]$ della tabella può contenere una produzione $A \rightarrow \alpha$ di G , oppure essere vuota, ad indicare che si è verificato un errore.
- ▶ Disponendo di M_G , la prima riga dell'algoritmo a discesa ricorsiva (cioè la scelta della produzione) viene sostituita da un lookup alla tabella $M_G[A, x]$.

Costruzione della tabella di parsing

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- ▶ L'algoritmo di costruzione della parsing table è molto semplice ed è formato da un ciclo principale nel quale si prendono in considerazione tutte le produzioni.
- ▶ Per ogni produzione $A \rightarrow \alpha$:
 - ▶ per ogni simbolo x in $FIRST(\alpha)$ si pone $M[A, x] = 'A \rightarrow \alpha'$;
 - ▶ se $\epsilon \in FIRST(\alpha)$, per ogni simbolo y in $FOLLOW(A)$ si pone $M[A, y] = 'A \rightarrow \alpha'$.
- ▶ Tutte le altre entry della tabella vengono lasciate vuote (ad indicare l'occorrenza di un errore).
- ▶ Se la grammatica è $LL(1)$, nessuna entry della tabella viene riempita con più di una produzione.

Esempio

- ▶ Consideriamo ancora la grammatica

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow \times FT' \mid \epsilon$$

$$F \rightarrow \mathbf{id} \mid (E)$$

- ▶ Il calcolo completo degli insiemi *FIRST* e *FOLLOW* produce:

- ▶ $FIRST(F) = FIRST(T) = FIRST(E) = \{\mathbf{id}, (\};$
- ▶ $FIRST(E') = \{+, \epsilon\}, FIRST(T') = \{\times, \epsilon\};$
- ▶ $FOLLOW(E) = FOLLOW(E') = \{\$, \});$
- ▶ $FOLLOW(T) = (FIRST(E') \setminus \{\epsilon\}) \cup FOLLOW(E') = \{+, \$, \});$
- ▶ $FOLLOW(T') = FOLLOW(T) = \{+, \$, \});$
- ▶ $FOLLOW(F) = (FIRST(T') \setminus \{\epsilon\}) \cup FOLLOW(T') = \{\times, +, \$, \}).$

Esempio (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- L'algoritmo prima delineato produce quindi la seguente tabella di parsing

N.T.	Simbolo di input					
	id	()	+	×	\$
E	$E \rightarrow TE'$	$E \rightarrow TE'$				
E'			$E' \rightarrow \epsilon$	$E' \rightarrow +TE'$		$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$	$T \rightarrow FT'$				
T'			$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	$T' \rightarrow \times FT'$	$T' \rightarrow \epsilon$
F	$F \rightarrow \text{id}$	$F \rightarrow (E)$				

in cui le entry vuote corrispondono ad una situazione di errore.

Esempio

- Se tentassimo di produrre la tabella di parsing per la grammatica

$$E \rightarrow (E)E' \mid \mathbf{id} E'$$

$$E' \rightarrow +EE' \mid \times EE' \mid \epsilon$$

otterremmo

N.T.	Simbolo di input					
	id	()	+	×	\$
E	$E \rightarrow \mathbf{id}E'$	$E \rightarrow (E)E'$				
E'			$E' \rightarrow \epsilon$	$E' \rightarrow +EE'$ $E' \rightarrow \epsilon$	$E' \rightarrow \times EE'$ $E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$

- Con in input il carattere + o il carattere ×, il parser non saprebbe quindi come procedere.
- Il non soddisfacimento delle proprietà $LL(1)$ è in questo caso una conseguenza dell'ambiguità della grammatica.

Esempio (continua)

- ▶ A volte è possibile risolvere il conflitto presente in una entry della tabella scegliendo opportunamente la produzione da applicare (fra quelle in conflitto).
- ▶ Naturalmente la scelta non deve compromettere la capacità di riconoscere il linguaggio generato dalla grammatica.
- ▶ Nel caso dell'esempio, si deve optare in favore delle produzioni $E' \rightarrow +EE'$ e $E' \rightarrow \times EE'$, anziché $E' \rightarrow \epsilon$ (si provi, ad esempio, a riconoscere la stringa **id + id**).
- ▶ Si noti tuttavia che, pur avendo risolto l'ambiguità, l'interpretazione delle espressioni che deriva dall'albero di parsing non è quella "naturale" (non viene soddisfatta la precedenza naturale degli operatori).
- ▶ Al riguardo, si provi a costruire l'albero di parsing per la stringa **id × id + id**.

Esempio

- ▶ Consideriamo la seguente grammatica (che esprime, usando simbologia diversa, il problema del *dangling else* che abbiamo già esaminato in precedenza):

$$S \rightarrow \mathbf{iEtSeS} \mid \mathbf{iEtS} \mid \mathbf{a}$$

$$E \rightarrow \mathbf{b}$$

- ▶ Tale grammatica presenta produzioni con prefisso comune e dunque non è idonea al parsing a d.r.
- ▶ È possibile eliminare i prefissi comuni, ottenendo:

$$S \rightarrow \mathbf{iEtSS'} \mid \mathbf{a}$$

$$S' \rightarrow \mathbf{eS} \mid \epsilon$$

$$E \rightarrow \mathbf{b}$$

e risulta

- ▶ $FIRST(S) = \{\mathbf{i, a}\}$; $FIRST(S') = \{\mathbf{e, \epsilon}\}$;
- ▶ $FIRST(E) = \{\mathbf{b}\}$;
- ▶ $FOLLOW(S) = FOLLOW(S') = \{\$, \mathbf{e}\}$.

Esempio (continua)

- ▶ La grammatica modificata non è ancora $LL(1)$ in quanto $FIRST(\mathbf{e}S) \cap FOLLOW(S') = \{\mathbf{e}\}$.
- ▶ Ciò si riflette in una definizione multipla per una entry della tabella di parsing:

N.T.	Simbolo di input					
	i	t	e	a	b	\$
S	$S \rightarrow \mathbf{iEtSS'}$			$S \rightarrow \mathbf{a}$		
S'			$S' \rightarrow \mathbf{eS}$ $S' \rightarrow \epsilon$			$S' \rightarrow \epsilon$
E					$E \rightarrow \mathbf{b}$	

- ▶ Tuttavia se, con in input il carattere **e**, il parser venisse “istruito” a scegliere sempre la produzione $S' \rightarrow \mathbf{eS}$, l’ambiguità si risolverebbe (e pure con l’intepretazione “naturale”, che associa ogni **else** al **then** più vicino).

Implementazione non ricorsiva

- ▶ È possibile dare un'implementazione non ricorsiva di un parser predittivo (cioè che non richiede backtracking) utilizzando esplicitamente una pila.
- ▶ La pila serve per memorizzare i simboli della parte destra della produzione scelta ad ogni passo.
- ▶ Tali simboli verranno poi “confrontati” con l'input (se terminali) o ulteriormente riscritti (se non terminali).
- ▶ Il comportamento del parser è illustrato nella seguente diapositiva.

Implementazione non ricorsiva (continua)

- ▶ Inizialmente, sullo stack sono contenuti (partendo dal fondo) i simboli \$ ed \mathcal{S} , mentre la variabile z punta al primo carattere di input.
- ▶ Al generico passo, il parser controlla il simbolo X sulla testa dello stack;
 - ▶ se X è un non terminale e $M[X, z] = 'X \rightarrow X_1 \dots X_k'$, esegue una pop dallo stack (rimuove cioè X) seguita da k push dei simboli X_k, \dots, X_1 , nell'ordine;
 - ▶ se X è un non terminale e $M[X, z] = 'error'$, segnala una condizione di errore;
 - ▶ se X è un terminale e $X = z$, esegue una pop e fa avanzare z ;
 - ▶ se X è un terminale e $X \neq z$, segnala una condizione di errore.
- ▶ Le operazioni terminano quando $X = \$$ e la stringa è accettata se $z = \$$.

Esempio

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

- Consideriamo nuovamente la grammatica delle espressioni con precedenza di operatore, della quale ricordiamo la tabella di parsing:

N.T.	Simbolo di input					
	id	()	+	×	\$
E	$E \rightarrow TE'$	$E \rightarrow TE'$				
E'			$E' \rightarrow \epsilon$	$E' \rightarrow +TE'$		$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$	$T \rightarrow FT'$				
T'			$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	$T' \rightarrow \times FT'$	$T' \rightarrow \epsilon$
F	$F \rightarrow \text{id}$	$F \rightarrow (E)$				

- Supponendo di avere la stringa **id + id** in input, la seguente tabella illustra il progressivo contenuto dello stack e dell'input.

Esempio (continua)

Parsing

Generalità sul parsing

Parser a discesa ricorsiva

Stack	Input
$\$E$	id + id\$
$\$E'T$	id + id\$
$\$E'T'F$	id + id\$
$\$E'T'id$	id + id\$
$\$E'T'$	+id\$
$\$E'$	+id\$
$\$E'T+$	+id\$
$\$E'T$	id\$
$\$E'T'F$	id\$
$\$E'T'id$	id\$
$\$E'T'$	\$
$\$E'$	\$
\$	\$

Esercizi proposti

Parsing

Generalità sul parsing
Parser a discesa ricorsiva

- ▶ Si calcolino gli insiemi *FIRST* e *FOLLOW* per la seguente grammatica:

$$S \rightarrow \mathbf{c} \mid AS \mid BS$$

$$A \rightarrow \mathbf{a}B \mid \epsilon$$

$$B \rightarrow \mathbf{b}A \mid \epsilon$$

e si costruisca la relativa tabella di parsing per un parser a discesa ricorsiva.

- ▶ Si calcolino gli insiemi *FIRST* e *FOLLOW* per la grammatica G_2 , che descrive le espressioni regolari su $\{0, 1\}$, dopo averla modificata in modo da eliminare i prefissi comuni. Se ne costruisca quindi la tabella di parsing per un parser a discesa ricorsiva.