Linguaggi Formali e Traduttori

4.1 Parsing top-down e grammatiche LL(1)

Sommario

Problema

- Data una grammatica $G = (V, T, P, S)$ e una stringa $w \in T^*$, determinare se

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w$$

  o, equivalentemente, se esiste un albero sintattico di G con radice S e prodotto w.

- La costruzione dell'automa corrispondente a G produce un PDA non deterministico.
- Per alcune G sappiamo che non è possibile trovare un DPDA.

In questa lezione

- Identifichiamo una famiglia di grammatiche libere per le quali è possibile costruire riconoscitori (parser) deterministici, cioè che non fanno uso di backtracking.
- Questi parser sono detti **top-down** perché costruiscono l'albero sintattico di w dalla radice (top) verso le foglie (down) o, equivalentemente, cercano una derivazione sinistra per w.

Strategia per il parsing top-down

Data una grammatica $G = (V, T, P, S)$ e una stringa $w \in T^*$, il parser cerca di ottenere una derivazione a sinistra $S \Rightarrow_{lm}^* w$ in cui, al passo i, il parser sa che

$$S \Rightarrow_{lm}^* uA\beta$$

e deve stabilire se

$$uA\beta \Rightarrow_{lm}^* w$$

Ci sono due casi da considerare:

- Se u **non è** prefisso di w, allora il parser **rifiuta** w.
- Se $w = uav$, allora il parser deve **scegliere** una produzione per riscrivere A

  $$A \to \alpha_1 \mid \cdots \mid \alpha_n$$

  e per farlo può usare a come "guida", a patto che tale simbolo identifichi univocamente l'α_i tale che $\alpha_i\beta \Rightarrow_{lm}^* av$.

Per ogni produzione $A \to \alpha_i$ occorre saper calcolare gli insiemi di simboli terminali che possono **iniziare** le stringhe derivate da $\alpha_i\beta$ e richiedere che tali insiemi siano disgiunti.

Stringhe annullabili (NULL)

Definizione

Data una grammatica $G = (V, T, P, S)$, diciamo che $\alpha \in (V \cup T)^*$ è **annullabile**, e scriviamo $\texttt{NULL}(\alpha)$, se e solo se $\alpha \Rightarrow_G^* \varepsilon$, ovvero se α può essere riscritta nella stringa vuota.

Come determinare se una stringa è annullabile

(1) Se $\texttt{NULL}(X_1), \dots, \texttt{NULL}(X_n)$, allora $\texttt{NULL}(X_1 \cdots X_n)$.
(2) Se esiste una produzione $A \to \alpha \in P$ e $\texttt{NULL}(\alpha)$, allora $\texttt{NULL}(A)$.

Note

- Come caso particolare di (1) quando $n = 0$ abbiamo $\texttt{NULL}(\varepsilon)$.
- Combinando (1) e (2) abbiamo che $A \to \varepsilon \in P$ implica $\texttt{NULL}(A)$.
- Una stringa che contiene simboli terminali non è mai annullabile.

Esempi di stringhe annullabili

$$
\begin{aligned}
A &\rightarrow a \mid Bc \\
B &\rightarrow \varepsilon \mid bB \\
C &\rightarrow d \mid Cc \mid BB
\end{aligned}
$$

Esempi di stringhe annullabili

$$\begin{aligned} A &\to a \mid Bc \\ B &\to \varepsilon \mid bB \\ C &\to d \mid Cc \mid BB \end{aligned}$$

- Da $\textsc{null}(\varepsilon)$ e dalla produzione $B \to \varepsilon$ deduciamo $\textsc{null}(B)$.
- Da $\textsc{null}(B)$ e dalla produzione $C \to BB$ deduciamo $\textsc{null}(C)$.
- Da $\textsc{null}(B)$ e $\textsc{null}(C)$ deduciamo $\textsc{null}(BC)$.
- Da $\neg\textsc{null}(a)$ e $\neg\textsc{null}(Bc)$ deduciamo $\neg\textsc{null}(A)$.

Inizi di una stringa (FIRST)

Definizione

Data una grammatica $G = (V, T, P, S)$ e una stringa $\alpha \in (V \cup T)^*$, indichiamo con $\textsc{first}(\alpha)$ gli **inizi** di α, ovvero l'insieme dei simboli terminali che possono trovarsi all'inizio delle stringhe derivate da α. Formalmente:

$$\textsc{first}(\alpha) \stackrel{\text{def}}{=} \{a \in T \mid \alpha \Rightarrow_G^* a\beta\}$$

Attenzione

Il libro di testo usa un'unica funzione $\textsc{first}_{libro}$ che combina $\textsc{null}$ e $\textsc{first}$ così:

$$\textsc{first}_{libro}(\alpha) = \begin{cases} \textsc{first}(\alpha) \cup \{\varepsilon\} & \textbf{se } \textsc{null}(\alpha) \\ \textsc{first}(\alpha) & \textbf{altrimenti} \end{cases}$$

In pratica, l'approccio seguito dal libro ammette il simbolo speciale ε tra gli inizi di α per indicare il fatto che α è annullabile. Noi abbiamo definito un predicato $\textsc{null}(\alpha)$ apposito mentre $\textsc{first}(\alpha)$ contiene solo simboli terminali.

Come calcolare FIRST

È possibile calcolare $\textsc{first}(\alpha)$ per induzione su α, usando le seguenti regole:

$$
\begin{aligned}
\textsc{first}(\varepsilon) &= \emptyset \\
\textsc{first}(a) &= \{a\} \\
\textsc{first}(A) &= \bigcup_{A \to \alpha} \textsc{first}(\alpha) \\
\textsc{first}(X\alpha) &= \begin{cases} \textsc{first}(X) \cup \textsc{first}(\alpha) & \text{se } \textsc{null}(X) \\ \textsc{first}(X) & \text{altrimenti} \end{cases}
\end{aligned}
$$

Attenzione

Applicando le regole qui sopra, può capitare di arrivare a equazioni della forma

$$\textsc{first}(A) = \textsc{first}(A) \cup \mathcal{S}$$

dove $\mathcal{S}$ è un insieme di terminali. Questa equazione si può semplificare a

$$\textsc{first}(A) = \mathcal{S}$$

in quanto siamo interessati a ottenere il più piccolo insieme di terminali con la proprietà descritta nella slide precedente.

Esempi di calcolo di FIRST

$$
\begin{array}{lcl}
S & \to & Ac \mid Ba \\
A & \to & \varepsilon \mid a \\
B & \to & b \\
C & \to & a \mid Cb \\
D & \to & \varepsilon \mid d \mid Db
\end{array}
$$

Esempi di calcolo di FIRST

$$
\begin{aligned}
S &\to Ac \mid Ba \\
A &\to \varepsilon \mid a \\
B &\to b \\
C &\to a \mid Cb \\
D &\to \varepsilon \mid d \mid Db
\end{aligned}
$$

Variabili annullabili

- $\textsc{null}(A)$
- $\textsc{null}(D)$

Calcolo di FIRST di tutte le variabili

- $\textsc{first}(B) = \textsc{first}(b) = \{b\}$
- $\textsc{first}(A) = \textsc{first}(\varepsilon) \cup \textsc{first}(a) = \{a\}$
- $\textsc{first}(S) = \textsc{first}(Ac) \cup \textsc{first}(Ba) = \textsc{first}(A) \cup \textsc{first}(c) \cup \textsc{first}(B) = \{a, b, c\}$
- $\textsc{first}(C) = \textsc{first}(a) \cup \textsc{first}(Cb) = \{a\} \cup \textsc{first}(C) = \{a\}$
- $\textsc{first}(D) = \textsc{first}(\varepsilon) \cup \textsc{first}(d) \cup \textsc{first}(Db) = \{d\} \cup \textsc{first}(D) \cup \textsc{first}(b) = \{b, d\}$

Seguiti di una variabile (FOLLOW)

Definizione

Data una grammatica $G = (V, T, P, S)$ e una variabile $A \in V$, indichiamo con $\textsc{follow}(A)$ i **seguiti** di A, ovvero l'insieme dei simboli terminali che possono seguire A in una forma sentenziale. Formalmente:

$$\textsc{follow}(A) \stackrel{\text{def}}{=} \{a \in T \mid S \Rightarrow_G^* \alpha A a \beta\}$$

Attenzione

- Per convenzione aggiungeremo una sentinella **$** ai seguiti del simbolo iniziale S.
- In questo modo il parser può capire quando è arrivato alla fine della stringa da riconoscere.

Come calcolare FOLLOW

Il calcolo di $\textsc{follow}$ si effettua in due fasi.

Fase 1

In questa fase si annotano relazioni di appartenenza ed inclusione insiemistica secondo il seguente algoritmo:

- Annotare $\$ \in \textsc{follow}(S)$.
- Ripetere i passi seguenti per ogni produzione e per ogni variabile nel corpo di queste:
  1. Se $A \to \alpha B \beta$, allora annotare $\textsc{first}(\beta) \subseteq \textsc{follow}(B)$.
  2. Se $A \to \alpha B \beta$ e $\textsc{null}(\beta)$, allora annotare $\textsc{follow}(A) \subseteq \textsc{follow}(B)$.

Caso particolare di (2): se $A \to \alpha B$, allora annotare $\textsc{follow}(A) \subseteq \textsc{follow}(B)$.

Fase 2

Si determinano i seguiti propagando i simboli terminali (e $\$$) rispettando l'ordine delle inclusioni insiemistiche $\subseteq$ che sono state annotate.

Per grammatiche complesse può essere utile fare una tabella con due colonne, l'elenco di tutte le variabili nella prima ed i seguiti corrispondenti alle variabili nella seconda.

Esempi di calcolo di FOLLOW

$$
\begin{array}{lcl}
S & \to & Ac \mid Ba \\
A & \to & \varepsilon \mid a \\
B & \to & b \\
C & \to & a \mid Cb \\
D & \to & \varepsilon \mid d \mid Db
\end{array}
$$

Esempi di calcolo di FOLLOW

$$
\begin{aligned}
S &\to Ac \mid Ba \\
A &\to \varepsilon \mid a \\
B &\to b \\
C &\to a \mid Cb \\
D &\to \varepsilon \mid d \mid Db
\end{aligned}
$$

Fase 1

- $\$ \in \textsc{follow}(S)$
- $\textsc{first}(c) \subseteq \textsc{follow}(A)$
- $\textsc{first}(a) \subseteq \textsc{follow}(B)$
- $\textsc{first}(b) \subseteq \textsc{follow}(C)$
- $\textsc{first}(b) \subseteq \textsc{follow}(D)$

Fase 2

X	$\textsc{follow}(X)$
S	$\{\$\}$
A	$\{c\}$
B	$\{a\}$
C	$\{b\}$
D	$\{b\}$

Insiemi guida

Definizione

Data una grammatica $G = (V, T, P, S)$ e una produzione $A \to \alpha$, indichiamo con $\textsc{guida}(A \to \alpha)$ l'**insieme guida** di $A \to \alpha$, ovvero l'insieme

$$\textsc{guida}(A \to \alpha) \stackrel{\text{def}}{=} \begin{cases} \textsc{first}(\alpha) \cup \textsc{follow}(A) & \textbf{se } \textsc{null}(\alpha) \\ \textsc{first}(\alpha) & \textbf{altrimenti} \end{cases}$$

Intuizione

Un parser predittivo che sceglie di riscrivere la variabile A usando la produzione $A \to \alpha$ si aspetta di leggere nella stringa di input uno dei simboli nell'insieme guida di $A \to \alpha$.

Sono due i casi da considerare:

1. Il simbolo è uno degli inizi di α, oppure
2. α è annullabile ed il simbolo è uno dei seguiti di A.

Grammatiche LL(1)

Definizione

Diciamo che una grammatica $G = (V, T, P, S)$ è **LL(1)** se, per ogni coppia di produzioni distinte $A \to \alpha$ e $A \to \beta$ in P, abbiamo che

$$\textsc{guida}(A \to \alpha) \cap \textsc{guida}(A \to \beta) = \emptyset$$

Intuizione

Noto il simbolo da riscrivere A, note le produzioni $A \to \beta_1 \mid \cdots \mid \beta_n$ e noto il prossimo simbolo terminale a nella stringa da riconoscere, in una grammatica LL(1) esiste al massimo una produzione "giusta" tale che $a \in \textsc{guida}(A \to \beta_i)$ dunque il parser predittivo identifica univocamente la produzione $A \to \beta_i$ a partire da a.

Cosa c'è nel nome LL(1)

- L $\to$ la stringa in input viene analizzata da sinistra (left) a destra;
- L $\to$ il parser cerca di costruire una derivazione canonica sinistra (leftmost);
- 1 $\to$ il parser usa un solo simbolo terminale della stringa per scegliere la produzione.

Esempio: espressioni aritmetiche

$$
\begin{array}{rcll}
E & \to & TE' & \\
E' & \to & \texttt{+}TE' \mid \varepsilon & \textsc{null}(E') \\
T & \to & FT' & \\
T' & \to & \texttt{*}FT' \mid \varepsilon & \textsc{null}(T') \\
F & \to & \texttt{(}E\texttt{)} \mid \texttt{id} &
\end{array}
$$

- $\texttt{\$} \in \textsc{follow}(E)$
- $\{\texttt{+}\} = \textsc{first}(E') \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(E')$
- $\textsc{follow}(E') \subseteq \textsc{follow}(T)$
- $\{\texttt{*}\} = \textsc{first}(T') \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(T')$
- $\textsc{follow}(T') \subseteq \textsc{follow}(F)$
- $\{\texttt{)}\} = \textsc{first}(\texttt{)}) \subseteq \textsc{follow}(E)$

Esempio: espressioni aritmetiche

$$
\begin{array}{lcll}
E & \to & TE' & \\
E' & \to & \texttt{+}TE' \mid \varepsilon & \textsc{null}(E') \\
T & \to & FT' & \\
T' & \to & \texttt{*}FT' \mid \varepsilon & \textsc{null}(T') \\
F & \to & \texttt{(}E\texttt{)} \mid \texttt{id} &
\end{array}
$$

$$
\begin{aligned}
&\textsc{first}(E) = \textsc{first}(T) = \{\texttt{(}, \texttt{id}\} \\
&\textsc{first}(E') = \{\texttt{+}\} \\
&\textsc{first}(T) = \textsc{first}(F) = \{\texttt{(}, \texttt{id}\} \\
&\textsc{first}(T') = \{\texttt{*}\} \\
&\textsc{first}(F) = \{\texttt{(}, \texttt{id}\}
\end{aligned}
$$

- $\texttt{\$} \in \textsc{follow}(E)$
- $\{\texttt{+}\} = \textsc{first}(E') \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(E')$
- $\textsc{follow}(E') \subseteq \textsc{follow}(T)$
- $\{\texttt{*}\} = \textsc{first}(T') \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(T')$
- $\textsc{follow}(T') \subseteq \textsc{follow}(F)$
- $\{\texttt{)}\} = \textsc{first}(\texttt{)}) \subseteq \textsc{follow}(E)$

Esempio: espressioni aritmetiche

$$
\begin{array}{lcll}
E & \to & TE' & \\
E' & \to & \texttt{+}TE' \mid \varepsilon & \textsc{null}(E') \\
T & \to & FT' & \\
T' & \to & \texttt{*}FT' \mid \varepsilon & \textsc{null}(T') \\
F & \to & \texttt{(}E\texttt{)} \mid \texttt{id} &
\end{array}
$$

$$\textsc{first}(E) = \textsc{first}(T) = \{\texttt{(}, \texttt{id}\}$$
$$\textsc{first}(E') = \{\texttt{+}\}$$
$$\textsc{first}(T) = \textsc{first}(F) = \{\texttt{(}, \texttt{id}\}$$
$$\textsc{first}(T') = \{\texttt{*}\}$$
$$\textsc{first}(F) = \{\texttt{(}, \texttt{id}\}$$

- $\texttt{\$} \in \textsc{follow}(E)$
- $\{\texttt{+}\} = \textsc{first}(E') \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(T)$
- $\textsc{follow}(E) \subseteq \textsc{follow}(E')$
- $\textsc{follow}(E') \subseteq \textsc{follow}(T)$
- $\{\texttt{*}\} = \textsc{first}(T') \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(F)$
- $\textsc{follow}(T) \subseteq \textsc{follow}(T')$
- $\textsc{follow}(T') \subseteq \textsc{follow}(F)$
- $\{\texttt{)}\} = \textsc{first}(\texttt{)}) \subseteq \textsc{follow}(E)$

X	$\textsc{follow}(X)$
E	$\texttt{\$}, \texttt{)}$
E'	$\texttt{\$}, \texttt{)}$
T	$\texttt{\$}, \texttt{)}, \texttt{+}$
T'	$\texttt{\$}, \texttt{)}, \texttt{+}$
F	$\texttt{\$}, \texttt{)}, \texttt{+}, \texttt{*}$

Esercizi

1. Calcolare gli insiemi guida della grammatica nella slide 14. La grammatica è LL(1)?
2. Calcolare gli insiemi guida della seguente grammatica e determinare se è LL(1).

$$\begin{aligned} A &\to BC \mid D \\ B &\to \varepsilon \mid a \\ C &\to b \mid cCc \\ D &\to \varepsilon \mid CD \end{aligned}$$

3. Ripetere l'esercizio precedente per la grammatica

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ SS'\ \texttt{fi} \mid \texttt{skip} \\ S' &\to \texttt{else}\ S \mid \varepsilon \\ E &\to \texttt{true} \mid \texttt{false} \end{aligned}$$

   in cui S, S' ed E sono variabili e **`if`**, **`then`**, ... sono terminali.

4. Ripetere l'esercizio precedente dopo aver rimosso il terminale **`fi`** dalla grammatica.

Linguaggi Formali e Traduttori

4.2 Parsing ricorsivo discendente

Sommario

Problema

- Realizzazione pratica di un parser top-down.

In questa lezione

- Studiamo una tecnica basata sulla **ricorsione** per la realizzazione pratica di un parser top-down.

Struttura del parser ricorsivo

Idea

Usare la pila del linguaggio di programmazione per "ricordare" il suffisso della forma sentenziale sinistra da riconoscere.

Elementi chiave

- Il parser ha una procedura per ogni variabile della grammatica.
- La procedura A nel parser riconosce le stringhe generate da A nella grammatica.
- La procedura A usa il simbolo corrente e gli insiemi guida, per scegliere la produzione $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ da usare per riscrivere A.
- Per ogni simbolo X trovato nel corpo della produzione scelta:
  - Se X è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio X. In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
  - Se X è una variabile, il metodo invoca la procedura X.

Algoritmo di parsing ricorsivo

```
var w : string                          // w è la stringa da riconoscere con $ in fondo
var i : int                             // i è l'indice del prossimo simbolo di w da leggere

procedure match(a : symbol)
  if w[i] = a then i ← i + 1 else error

procedure parse(v : string)             // v è la stringa da riconoscere
  w ← v$
  i ← 0
  S()                                   // S è il simbolo iniziale della grammatica
  match($)                              // controlla di aver letto tutta la stringa

procedure A()                           // A → α1 | ··· | αn sono le produzioni per A
  if w[i] ∈ GUIDA(A → α1) then
  ⋮
  else if w[i] ∈ GUIDA(A → αk) then
    for X ∈ αk do
      if X è un terminale then match(X) else X()
  ⋮
  else error                            // w[i] non è nell'insieme guida di nessuna produzione per A
```

Implementazione Java del parser (classe base)

```
public abstract class Parser {
  private String w;                 // stringa da riconoscere
  private int    i;                 // indice del prossimo simbolo

  protected char peek()             // legge il simbolo corrente
  { return w.charAt(i); }

  protected void match(char a)      // controlla il simbolo corrente
  { if (peek() == a) i++; else throw error(); }

  public void parse(String v) {     // avvia il parsing di v
    w = v + "$";
    i = 0;
    S();
    match('$');
  }

  protected abstract void S();      // simbolo iniziale della grammatica

  protected SyntaxError error() { … }   // emette errore e interrompe
}
```

- Per semplicità assumiamo che i simboli siano caratteri.

Esempio: parser per il linguaggio $a^n b^n$

Grammatica

$$S \to aSb \mid \varepsilon$$

Insiemi guida

- $\textsc{guida}(A \to aSb) = \{a\}$
- $\textsc{guida}(A \to \varepsilon) = \{b, \$\}$

Codice del parser

```
public class AnBn extends Parser {
  protected void S() {
    switch (peek()) {

    case 'a': // S → aSb
      match('a');
      S();
      match('b');
      break;

    case 'b': // S → ε
    case '$':
      break;

    default:
      throw error();
    }
  }
}
```

Esercizi

Implementazione di parser

Implementare il parser ricorsivo discendente per le seguenti grammatiche:

1. La grammatica delle stringhe della forma wcw^R dove $w \in \{0,1\}^*$:
   - $S \to c \mid 0S0 \mid 1S1$
2. La grammatica delle stringhe di parentesi quadre bilanciate:
   - $S \to \varepsilon \mid [S]S$
3. La grammatica delle stringhe della forma $a^n b^n c^m$:
   - $S \to XC$
   - $X \to \varepsilon \mid aXb$
   - $C \to \varepsilon \mid cC$
4. La grammatica delle espressioni aritmetiche in forma prefissa:
   - $E \to 0 \mid 1 \mid \cdots \mid 9 \mid +EE \mid *EE$
5. La grammatica delle espressioni aritmetiche in forma infissa.

Linguaggi Formali e Traduttori

4.3 Grammatiche fattorizzabili e ricorsive a sinistra

Sommario

Problema

- Molte grammatiche utili per descrivere linguaggi di programmazione non sono LL(1).
  1. Presenza di **produzioni fattorizzabili**

$$A \to \alpha\beta_1 \mid \alpha\beta_2$$

  2. Presenza di **produzioni ricorsive a sinistra**

$$A \to A\alpha \mid \beta$$

  3. Presenza di **ambiguità**

In questa lezione

- Studiamo alcune tecniche per modificare produzioni fattorizzabili e ricorsive a sinistra senza cambiare il linguaggio generato dalla grammatica in modo da renderla – spesso, ma non sempre – LL(1).

Fattorizzazione

Problema

Data una grammatica con le produzioni

$$A \to \alpha\beta_1 \mid \alpha\beta_2$$

abbiamo

$$\textsc{guida}(A \to \alpha\beta_1) \supseteq \textsc{first}(\alpha) \qquad \textsc{guida}(A \to \alpha\beta_2) \supseteq \textsc{first}(\alpha)$$

dunque

$$\textsc{guida}(A \to \alpha\beta_1) \cap \textsc{guida}(A \to \alpha\beta_2) \neq \emptyset$$

tranne nel caso degenere in cui α genera solo ε.

Soluzione

Fattorizzare il prefisso comune α introducendo una nuova variabile A':

$$A \to \alpha A' \qquad A' \to \beta_1 \mid \beta_2$$

Esempio di fattorizzazione

La grammatica

- $S \to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S\ \texttt{fi}$
- $S \to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi}$
- $S \to a$
- $E \to b$

non è LL(1) infatti, dovendo espandere la variabile S quando il prossimo token nella stringa da riconoscere è **`if`**, il parser non saprebbe quale delle produzioni per S usare.

La grammatica è fattorizzabile nel modo seguente:

- $S \to \texttt{if}\ E\ \texttt{then}\ S\ S'$
- $S' \to \texttt{else}\ S\ \texttt{fi} \mid \texttt{fi}$
- $S \to a$
- $E \to b$

In particolare, gli insiemi guida delle produzioni della grammatica modificata sono ora disgiunti due a due.

Ricorsione immediata a sinistra

Problema

Una grammatica con le produzioni

$$A \to A\alpha \mid \beta$$

è detta **immediatamente ricorsiva a sinistra** in quanto la produzione $A \to A\alpha$ ha A sia in testa che come primo simbolo del suo corpo. La grammatica non è LL(1):

$$\textsc{guida}(A \to A\alpha) \supseteq \textsc{first}(A) \supseteq \textsc{first}(\beta) \qquad \textsc{guida}(A \to \beta) \supseteq \textsc{first}(\beta)$$

Osservazione

La grammatica genera stringhe $\beta\alpha\alpha\cdots\alpha$ composte da una β seguita da zero o più α.

Soluzione

Introdurre una nuova variabile per spostare la ricorsione da sinistra a destra:

$$A \to \beta A' \qquad A' \to \varepsilon \mid \alpha A'$$

Esempio di eliminazione della ricorsione

La grammatica

- $E \to E + T \mid T$
- $T \to T * F \mid F$
- $F \to n \mid (E)$

è immediatamente ricorsiva a sinistra nelle produzioni per E e per T. Ad esempio:

$$\begin{aligned} \textsc{guida}(E \to E + T) &= \textsc{first}(E) = \{n, (\} \\ \textsc{guida}(E \to T) &= \textsc{first}(T) = \{n, (\} \end{aligned}$$

Esempio di eliminazione della ricorsione

La grammatica

- $E \to E + T \mid T$
- $T \to T * F \mid F$
- $F \to n \mid (E)$

è immediatamente ricorsiva a sinistra nelle produzioni per E e per T. Ad esempio:

$$\begin{aligned} \textsc{guida}(E \to E + T) &= \textsc{first}(E) = \{n, (\} \\ \textsc{guida}(E \to T) &= \textsc{first}(T) = \{n, (\} \end{aligned}$$

Eliminando la ricorsione immediata a sinistra otteniamo la grammatica:

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to *FT' \mid \varepsilon$
- $F \to n \mid (E)$

Ricorsione a sinistra: caso generale

Una grammatica con le produzioni

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

in cui nessun β_i inizia con A, genera stringhe della forma

$$\beta_i \alpha_{k_1} \alpha_{k_2} \cdots \alpha_{k_l}$$

L'eliminazione della ricorsione immediata a sinistra porta alla grammatica

$$A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$
$$A' \to \varepsilon \mid \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A'$$

Osservazioni

- In generale, l'eliminazione della ricorsione a sinistra non garantisce che la grammatica risultante sia LL(1).
- Ad esempio, nella grammatica qui sopra basta che uno degli α_i sia annullabile per avere insiemi guida delle produzioni per A' non disgiunti.

Ricorsione indiretta a sinistra

In alcune grammatiche alcune ricorsioni a sinistra sono "indirette":

$$\begin{aligned} S &\to Aa \mid b \\ A &\to Ac \mid Sd \mid \varepsilon \end{aligned}$$

Tentando di eliminare la ricorsione a sinistra per le produzioni di A otteniamo

$$\begin{aligned} S &\to Aa \mid b \\ A &\to SdA' \mid A' \\ A' &\to \varepsilon \mid cA' \end{aligned}$$

ma la grammatica non è LL(1), infatti:

- $\textsc{guida}(A \to SdA') \supseteq \textsc{first}(S) \supseteq \textsc{first}(A) \supseteq \textsc{first}(A') \ni c$
- $\textsc{guida}(A \to A') \supseteq \textsc{first}(A') \ni c$

C'è una **ricorsione indiretta** a sinistra che riguarda la variabile A

$$A \Rightarrow Sd \Rightarrow Aad$$

Eliminazione della ricorsione indiretta

Idea

Si può esporre la ricorsione indiretta facendo opportune riscritture di variabili.

Algoritmo

1. Si impone un ordine (arbitrario) alle variabili della grammatica.
2. Considerando ogni variabile secondo l'ordine imposto, si elimina la ricorsione immediata per quella variabile e si riscrivono le occorrenze di quella variabile che compaiono nei corpi delle produzioni delle variabili seguenti.

Esempio

$$
\begin{array}{lcl@{\quad}c@{\quad}lcl@{\quad}c@{\quad}lcl}
S & \to & Aa \mid b & \Rightarrow & S & \to & Aa \mid b & \Rightarrow & S & \to & Aa \mid b \\
A & \to & Ac \mid Sd \mid \varepsilon & \Rightarrow & A & \to & Ac \mid Aad \mid bd \mid \varepsilon & \Rightarrow & A & \to & bdA' \mid A' \\
 & & & & & & & & A' & \to & \varepsilon \mid cA' \mid adA'
\end{array}
$$

Esercizi

1. Applicare le trasformazioni studiate in questa lezione alla grammatica non ambigua delle formule booleane per farla diventare LL(1).
2. Implementare il parser top-down per la grammatica ottenuta nell'esercizio precedente così ottenuta. Scegliere caratteri ASCII "normali" per rappresentare i connettivi logici, ad esempio & per $\wedge$, | per $\vee$ e ~ per $\neg$.