

Esercitazione lab. sul Parsing

F. d'Amore A. Marchetti Spaccamela

11 maggio 2021

Parsing a discesa ricorsiva

Parsing “brute-force” con *backtracking*

Parsing predittivo LL(1)

Parsing “semplice” (top-down)

- Data la grammatica

$$\begin{array}{lcl} S & \rightarrow & aS \mid bT \\ T & \rightarrow & cT \mid bac \end{array}$$

- Task 0:
Costruire un parser a discesa ricorsiva facendo riferimento al codice C distribuito.

Stringhe palindrome

- ▶ Linguaggio $L_1 \subset \{a, b\}^+$ così definito:
$$L_1 = \{x \mid x \in (a + b)^+ \wedge (x = w \cdot \tilde{w} \vee x = w \cdot a \cdot \tilde{w} \vee x = w \cdot b \cdot \tilde{w})\}.$$
 - ▶ L'operatore tilde ($\tilde{}$) “riflette” (inverte) il suo argomento.
- ▶ L_1 è il linguaggio delle stringhe palindrome di lunghezza pari o dispari; nel caso di lunghezza dispari il carattere centrale può egualmente essere a o b.
- ▶ Esempi di stringhe in L_1 : ababbbbaba, baabaaaaabaab;
esempio di stringa non appartenente a L_1 : baabaabbab.

Grammatica G_1 per L_1

- ▶ Grammatica G_1 (non LL(1)) tale che $\mathcal{L}(G_1) = L_1$:

$$(G_1.1) \quad S \rightarrow aSa$$

$$(G_1.2) \quad S \rightarrow bSb$$

$$(G_1.3) \quad S \rightarrow aa$$

$$(G_1.4) \quad S \rightarrow bb$$

$$(G_1.5) \quad S \rightarrow a$$

$$(G_1.6) \quad S \rightarrow b$$

- ▶ L_1 non ammette grammatiche LL(1) (dimostrazione complessa).

Task 1

- ▶ Scrivere un parser per L_1 , che opera a discesa ricorsiva e brute-force (attraverso backtracking).
- ▶ Si faccia riferimento al codice C fornito, e in particolare al file `brute-force-parser.c`
- ▶ Compilare con `make` sulla linea di comando. Uso del driver:
`./driver brute stringa1 stringa2 ...`, dove `stringa1`, `stringa2`, ecc., denotano le stringhe su cui fare il brute-force parsing per L_1 .
- ▶ (Avanzato) Tentare euristiche per gestire il problema del riconoscimento (precoce) di prefissi dell'input.

Possibili varianti al Task 1

- ▶ Scrivere un parser per il linguaggio $L'_1 = \{w \cdot 2 \cdot \tilde{w} \mid w \in \{0, 1\}^*\}$ che opera a discesa ricorsiva e brute-force (attraverso backtracking). L'_1 è costituito da stringhe palindrome di lunghezza dispari in cui esiste un separatore centrale (la cifra 2) che non compare nelle sottostringhe che si “specchiano”. Produzioni: $S \rightarrow 0S0 \mid 1S1 \mid 2$.
- ▶ Scrivere un parser per il linguaggio $\{a^n \mid n \geq 1\} \cup \{a^i b^i \mid i \geq 1\}$ che opera a discesa ricorsiva e brute-force (attraverso backtracking). Il linguaggio è l'unione di due semplicissimi linguaggi. Produzioni: $S \rightarrow S_1 \mid S_2$; $S_1 \rightarrow aS_1 \mid a$; $S_2 \rightarrow aS_2b \mid ab$.

Parentesi bilanciate

- ▶ Il linguaggio $L_2 \subset \{ (,) \}^+$ delle parentesi tonde bilanciate è (informalmente) formato da parole composte da parentesi (aperte, o sinistre, e chiuse, o destre) tale che ogni parentesi sinistra è chiusa da una corrispondente e successiva parentesi destra, e ogni parentesi destra corrisponde a una parentesi sinistra precedentemente aperta.
- ▶ Esempi di stringhe di L_2 : $((()))((()())())$, $((()((()))))$.
Esempi di stringhe non appartenenti: $((()())()$ (manca una parentesi destra), $((()))($ (una parentesi destra non ha una precedente e corrispondente parentesi sinistra; una parentesi sinistra non è successivamente chiusa da una corrispondente parentesi destra).

Grammatica G_2 per L_2

- ▶ Grammatica G_2 tale che $\mathcal{L}(G_2) = L_2$:

$$(G_2.1) \quad S \rightarrow (S)$$

$$(G_2.2) \quad S \rightarrow SS$$

$$(G_2.3) \quad S \rightarrow ()$$

- ▶ G_2 non è LL(1):
(per l'assenza di ε -produzioni i SELECT coincidono con i FIRST)

$$\text{FIRST}(S) = \{ (\}$$

$$\text{FIRST}(G_2.1) = \{ (\}$$

$$\text{FIRST}(G_2.2) = \{ (\}$$

$$\text{FIRST}(G_3.3) = \{ (\}$$

Tentativo di costruzione grammatica $LL(1)$ per L_2

- Risolviamo dapprima la ricorsione sinistra $S \rightarrow SS$ introducendo un nuovo non-terminale T , ottenendo

$$\begin{aligned} S &\rightarrow (S)T \\ S &\rightarrow ()T \\ T &\rightarrow ST \\ T &\rightarrow \varepsilon \end{aligned}$$

- Le prime due produzioni così ottenute hanno prefissi comuni, che risolviamo attraverso fattorizzazione, che introduce un nuovo non-terminale U , ottenendo così la grammatica G'_2 :

$$\begin{aligned} (G'_2.1) \quad S &\rightarrow (U \\ (G'_2.2) \quad U &\rightarrow S)T \\ (G'_2.3) \quad U &\rightarrow)T \\ (G'_2.4) \quad T &\rightarrow ST \\ (G'_2.5) \quad T &\rightarrow \varepsilon \end{aligned}$$

Neanche G'_2 è LL(1)

- È immediato verificare che

$$\text{SELECT}(T \rightarrow ST) \cap \text{SELECT}(T \rightarrow \varepsilon) = \{(\} \cap \{(\,, \$, \varepsilon\} = \{(\}$$

- Ciò significa che, quando il parser sta eseguendo la funzione associata a T e il prossimo carattere in input è $($, l'algoritmo non ha elementi per scegliere fra $T \rightarrow ST$ e $T \rightarrow \varepsilon$. Ai fini del parsing predittivo con lookahead 1 occorre perciò una grammatica differente.
- Proviamo a definirne una che sfrutta direttamente una ε -produzione.

Grammatica LL(1) per L_2

- Grammatica G_2'' con una ε -produzione tale che $\mathcal{L}(G_2'') = L_2$:

$$(G_2''.1) \quad S \rightarrow (T)T$$

$$(G_2''.2) \quad T \rightarrow (T)T$$

$$(G_2''.3) \quad T \rightarrow \varepsilon$$

(Scegliendo come assioma T il linguaggio generato sarebbe $L_2 \cup \{\varepsilon\}$)

- G_2'' è LL(1), come si riconosce immediatamente dalla seguente tabella

	FIRST	FOLLOW	SELECT
S	(\$	—
T	(, ε), \$	—
$(T)T$	(—	—
ε	ε	—	—
$S \rightarrow (T)T$	—	—	(
$T \rightarrow (T)T$	—	—	(
$T \rightarrow \varepsilon$	—	—	ε), \$

Task 2

- ▶ Scrivere un parser predittivo LL(1) per L_2 , che opera a discesa ricorsiva.
- ▶ Si faccia riferimento al codice C fornito, e in particolare al file `ll1-parser.c`
- ▶ Compilare con `make` sulla linea di comando. Uso del driver:
`./driver ll1 stringa1 stringa2 ...`, dove `stringa1`, `stringa2`, ecc., denotano le stringhe su cui fare il parsing predittivo per L_2 .
- ▶ N.B.: Sulla linea di comando del terminale proteggere le stringhe di parentesi con apici, altrimenti bash cercherà di interpretarle.

Possibili task aggiuntivi (per casa)

- ▶ Scrivere un parser a discesa ricorsiva brute-force basato su G_2 . (Non ci riesci? Nessuna sorpresa: perché?)
- ▶ Scrivere un parser a discesa ricorsiva brute-force basato su G'_2 .
- ▶ Modifica i parser scritti in modo che restituiscano il parse-tree, rappresentato come albero, in cui ogni nodo ha un puntatore al primo figlio (`firstChild`) e al prossimo fratello (`nextSibling`).