Esercitazione lab. sul Parsing

F. d'Amore A. Marchetti Spaccamela

28 maggio 2019

Parsing a discesa ricorsiva

- Parsing "brute-force" con *backtracking*
- Parsing predittivo LL(1)
- Uso del tool JFlap

Stringhe palindrome

- Linguaggio $L_1 \subset \{\mathtt{a}, \mathtt{b}\}^+$ così definito:
  $L_1 = \{x \mid x \in (\mathtt{a} + \mathtt{b})^+ \wedge (x = w \cdot \tilde{w} \vee x = w \cdot \mathtt{a} \cdot \tilde{w} \vee x = w \cdot \mathtt{b} \cdot \tilde{w})\}$.
  - 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 $\mathscr{L}(G_1) = L_1$:

$$
\begin{array}{lcll}
(G_1.1) & S & \rightarrow & \mathtt{a}S\mathtt{a} \\
(G_1.2) & S & \rightarrow & \mathtt{b}S\mathtt{b} \\
(G_1.3) & S & \rightarrow & \mathtt{aa} \\
(G_1.4) & S & \rightarrow & \mathtt{bb} \\
(G_1.5) & S & \rightarrow & \mathtt{a} \\
(G_1.6) & S & \rightarrow & \mathtt{b}
\end{array}
$$

- 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.

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 $\mathscr{L}(G_2) = L_2$:

$$\begin{array}{lccl} (G_2.1) & S & \to & (S) \\ (G_2.2) & S & \to & SS \\ (G_2.3) & S & \to & () \end{array}$$

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

$$\begin{array}{lcl} \text{FIRST}(S) & = & \{(\} \\ \text{FIRST}(G_2.1) & = & \{(\} \\ \text{FIRST}(G_2.2) & = & \{(\} \\ \text{FIRST}(G_3.3) & = & \{(\} \end{array}$$

Tentativo di costruzione grammatica LL(1) per L_2

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

$$\begin{array}{lcl} S & \to & (S)\,T \\ S & \to & ()\,T \\ T & \to & ST \\ T & \to & \varepsilon \end{array}$$

- 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':

$$\begin{array}{lcl} (G_2'.1) \quad S & \to & (U \\ (G_2'.2) \quad U & \to & S)\,T \\ (G_2'.3) \quad U & \to &)\,T \\ (G_2'.4) \quad T & \to & ST \\ (G_2'.5) \quad T & \to & \varepsilon \end{array}$$

Neanche G_2' è LL(1)

- È immediato verificare che

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

- 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 $T \to ST$ e $T \to \varepsilon$. 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 $\mathscr{L}(G_2'') = L_2$:

$$\begin{array}{lccl} (G_2''.1) & S & \to & (\,T\,)\,T \\ (G_2''.2) & T & \to & (\,T\,)\,T \\ (G_2''.3) & T & \to & \varepsilon \end{array}$$

(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	$(, \varepsilon$	$), \$$	—
$(\,T\,)\,T$	(	—	—
ε	ε	—	—
$S \to (\,T\,)\,T$	—	—	(
$T \to (\,T\,)\,T$	—	—	(
$T \to \varepsilon$	—	—	$\varepsilon,), \$$

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.

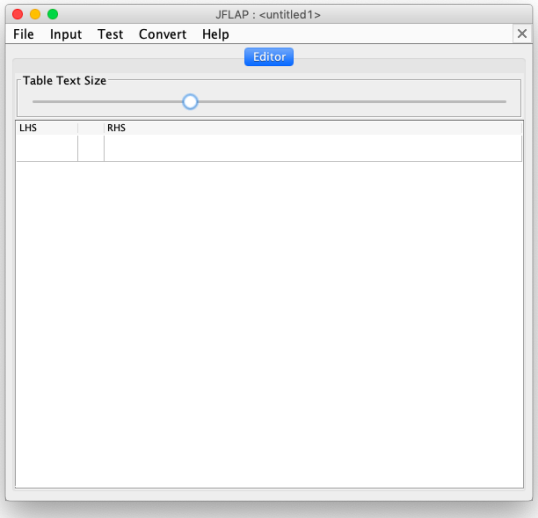
Possibili task addizionali (per casa)

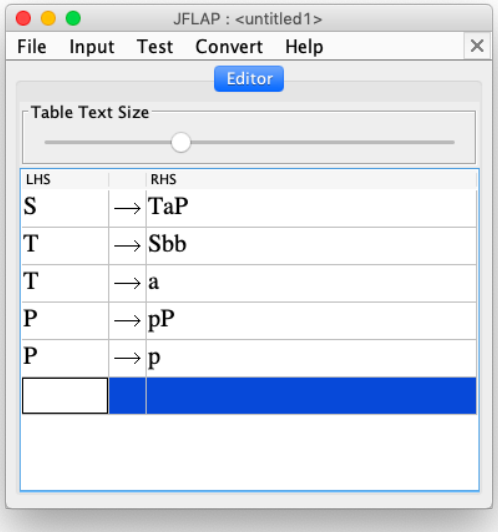
- Scrivere un parser a discesa ricorsiva brute-force basato su G_2. (Non ci riesci? Nessuna sorpresa: perché?)
- Scrivere un parser a discesa ricorsiva brute-force basato su G_2'.
- Modifica i parser scritti in modo che restituiscano il parse-tree, rappresentato come albero, in cui ogni nodo ha un puntatore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

Tool JFlap

- JFLAP è un software freeware per lo studio di automi e linguaggi sviluppato e manutenuto dalla Duke University.
- Disponibile come file jar
  - JFlap 7.1 (usa Java 8)
    `http://www.jflap.org/jflaptmp/july27-18/JFLAP7.1.jar`
  - JFlap 7.0 (usa Java 6)
    `http://www.jflap.org/jflaptmp/may15-2011/withoutSource/JFLAP.jar`
- Contiene sezione dedicata al parsing: all'avvio pigiare il pulsante "Grammar"

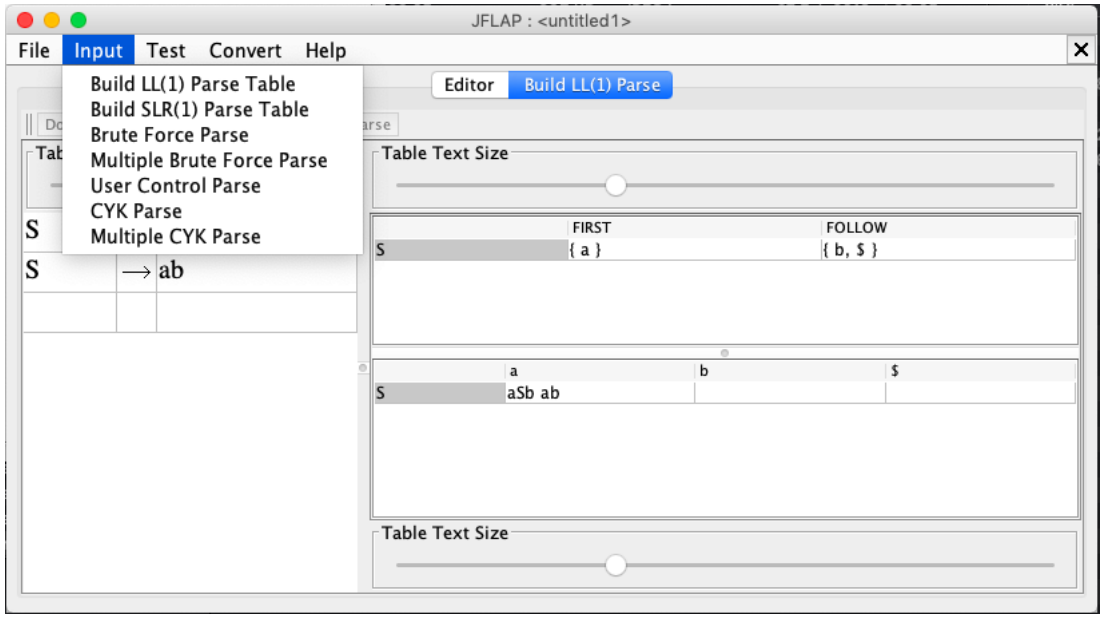
JFlap 1





- Inserire le produzioni della grammatica: nel riquadro sx (LHS = left-hand side) il non-terminale parte sx della produzione; nel riquadro dx (RHS = right-hand side) la parte destra della produzione.
- Usare il default: lettere maiuscole per i non-terminali; minuscole/cifre per i terminali.
- Per una ε-produzione il tool usa il simbolo di default λ piuttosto che ε.
- Usare i comandi del menu "Input" per eseguire il parsing (Brute Force Parse, o altri)

JFlap 2



Il menu “Input” consente di svolgere operazioni interessanti (SLR e CYK parsing non sono stati studiati).