COSC252: Programming Languages:

Formal Languages

Jeremy Bolton, PhD
Asst Teaching Professor

GEORGETOWN UNIVERSITY

Outline

I. ***Formal*** Perspective: review of languages and grammar
   I. Regular Languages
      I. Regular Expressions (Regular Grammars)
      II. Finite State Machines
   II. Context-Free Languages
      I. BNF Productions (Regular Grammars)
      II. Push Down Automata

Languages

- A language L is a set of sentences.
- A sentence is a sequence of characters from some input alphabet Σ

FSM

- A finite state machine is a 5-tuple:
  - (Q, Σ, δ, q_0, F)

  - Q: finite set of all states
  - Σ : alphabet (finite set of characters)
  - δ: state transition function, $\delta: Qx\Sigma \rightarrow \mathrm{Q}$
  - $q_0 \in Q$: start state
  - F$\subset Q$: set of accepting state(s)

GEORGETOWN UNIVERSITY

RegEx

- R is a regular expression on input alphabet Σ , if R is …
  1. $a \in \Sigma$, is a regular expression
  2. The empty string ϵ is a regular expression.
  3. The regular expression that represents the empty language θ is a regular expression.
  4. If R_1 and R_2 are regular expressions, then $R_1 \mid R_2$ is a regular expression
     - selection
  5. If R_1 and R_2 are regular expressions, then $R_1 R_2$ is a regular expression
     - concatenation
  6. If R_1 is a regular expression, then R_1^* is a regular expression
     - repetition

Regular Languages

- A language L is a regular Language iff there exists a regular expression generator. A language L is a regular Language iff there exists a finite state machine recognizer.

  – Note: for each Regular Expression, that generates a regular language L, there exists a FSM that recognizes L

  – Note: for each FSM, that recognizes a regular language L, there exists a RegEx that generates L

  – Regular Language Examples on alphabet $\Sigma = \{0,1\}$ (Can you find the corresponding regex and fsm?):
    - L = {s| for all sentences s that have exactly one 1}
    - L = {s| the length of s is a multiple of 3}
    - L = {s| s starts and ends with the same symbol}

CFG /BNF Production Set

- A context free grammar on an input alphabet Σ is a 4-tuple: (N, Σ, R, S)
    1. N: a set of non-terminals (variables representing abstractions)
    2. Σ: input alphabet (a set of terminals)
    3. R: a finite set of rules consisting of a nonterminal production (non-terminal followed by its production rule: a sequence of terminals and non-terminals)
    4. $S \in N$: start symbol

Pushdown Automaton

- A Pushdown Automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Q: set of states
  - Σ : input alphabet
  - Γ : stack alphabet (and operation)
  - $\delta: Q x \Sigma x \Gamma \rightarrow Q x \Gamma$, Transition function
  - $q_0 \in Q$: start state
  - $F \subset Q$: accept state(s)

GEORGETOWN UNIVERSITY

CFL

- A language L is a Context Free Language iff there exists a context free grammar (BNF) generator. A language L is a Context Free Language iff there exists a pushdown automaton recognizer.

  – Note: for each CFG, that generates a CFL L, there exists a PDA that recognizes L

  – Note: for each PDA, that recognizes a CFL L, there exists a CFG that generates L

  – CFL Examples on alphabet $\Sigma = \{0,1\}$ (Can you find the corresponding CFG ~~and PDA~~?):
    - L = {s| for all sentences s that have exactly one 1}
    - L = {s| n zeros followed by n ones}
    - L = {s| n zeros followed by 2n ones}

Language Hierarchy

- Venn Diagram
- The set of all context free languages is a super set of the set of all regular languages.
  - A CFG can generate anything a RegEx can generate … and more

GEORGETOWN UNIVERSITY

LR and LL grammars

- Languages can be categorized by their recognizers (parsers)
  - LL grammars generate languages that can be recognized by a Top Down Parser
  - LR grammars generate languages that can be recognized by a Bottom Up Parser
  - We can further specify a these grammars by how many lookaheads are needed to recognize the language correctly. This extra information also indicates the “complexity” of the parse.
    - LL(k) : Language can be recognized by a Top Down parser with k lookaheads
    - LR(k) : Language can be recognized by a Bottom Up parser with k lookaheads.
  - Note: The set of languages generated by LR(k) grammars is a super set of languages generated by an LL(k) grammar, for all k.

Grammars Categorized by "Parse-ability"

- Find the LL(k) and LR(k) grammar classification for the following grammars. That is, given G generates L , find the smallest k_1 and $k_1$such that, $L \in LL(k_1)$ and $L \in LR(k_2)$

- G_1:

$$E \rightarrow T + E \mid T \ - E \mid T$$
$$T \rightarrow id$$

- G_2:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' | -TE' | \epsilon$$
$$T \rightarrow id$$

GEORGETOWN UNIVERSITY

Grammars Categorized by Parse-ability

- Find the LL(k) and LR(k) grammar classification for the following grammars. That is, given G generates L , find the smallest k_1 and $k_1$such that, $L \in LL(k_1)$ and $L \in LR(k_2)$

- G_3:
  $A \rightarrow aB$
  $B \rightarrow bC$
  $C \rightarrow b$

- G_4:
  $A \rightarrow aB$
  $B \rightarrow C$
  $C \rightarrow b$ | c

- G_5:
  $\mathrm{E} \rightarrow E - T \mid T$
  $T \rightarrow (F)T \mid id \mid (E)$
  $\mathrm{F} \rightarrow id$

Example: Parsing c-style casts

<exp> → <exp> '-' <sub_exp>
    | <sub_exp>

<sub_exp> → '(' <type_name> ')' <sub_exp>
    | <id>
    | <literal>
    | '(' <exp> ')'

<type_name> → id
    | … <other_type_descriptions>

The problem is that the first <id> in "(<id>) <id>" is a <type_name>, but in "(<id>) - <id>" it is an <exp>, and the two must be reduced differently when the ")" is seen but before the "-" or second <id> has been seen by an LR(1) parser.

Example: Parameter Lists

- Example Usage
  - void foo(int a, int b, float c, float d);
  - void foo (int a, b, float c, d);

<header> → <type_name> <id> '(' <params> ')‘ ‘;’
| <type_name> <id> '(' ')‘ ‘;’

<type_name> → <id>
| … <other_descriptions>

<params > → <param>
| <params> ',' <param>

<param> → <type_name> <ids>

<ids> → <id>
| <ids> ',' <id>

Notice that after a “<ids> ," the next symbols can be "a b" (a is a type_name, b is a parameter name of type a) or "a ," or "a)" (a is a parameter name of the current type), but an LR(1) parser can't see far enough ahead to decide whether the "," is part of a "params" (in which case the preceding “<ids>" must be reduced to a "param"), or part of a bigger "ids".

Appendix

GEORGETOWN UNIVERSITY