Assignment 4

CSCI 3136: Principles of Programming Languages

Due Mar 12, 2018

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

We use regular expressions to describe regular languages. Regular expressions are themselves strings over an alphabet including regular characters and special characters such as parentheses, "`.`", "`|`", and "`*`". The string "`.*(1.1|1..1).*`" is a valid regular expression while the string "`((10|)(`" is not. This language turns out not be regular (because it includes strings with arbitrarily deeply nested sets of parentheses). Let us assume we have already constructed a scanner that identifies the tokens regular expressions are composed of.[1] The tokens we are allowed to build regular expressions from are

`(` `)` `|` `.` `*` `+` `?` *char*

where *char* denotes any valid character of the underlying alphabet. You don't have to worry about recognizing valid characters; we assume the scanner takes care of this for us.

(a) Provide a context-free grammar that defines the language of regular expressions. As is customary, assume that each input string is terminated by a special end-of-string marker `$`. Make sure the grammar represents the semantic structure of regular expressions. For example, the partial grammar

$$E \to char$$
$$E \to EE$$
$$E \to E|E$$

would allow us to parse the regular expression "`ab|c`" to be composed of two regular expressions "`a`" and "`b|c`", which is semantically incorrect. The correct parse is a regular expression composed of two regular expressions "`ab`" (which in turn is composed of smaller regular expressions) and "`c`". Review how we ensured arithmetic expressions are parsed in a way that reflects operator precedence to get an idea how you can ensure that the parse tree corresponding to a regular expression correctly reflects its structure.

(b) Provide the parse tree used to generate the string "`.*(1.1|1..1).*`" using your grammar.

(c) Verify that your grammar is LL(1). (If it is not, modify the grammar to make it LL(1).) In particular, provide FIRST, FOLLOW, and PREDICT sets for characters and productions and explain why they prove that the language is LL(1).

(d) Provide a deterministic push-down automaton based on your grammar that decides the language of regular expressions.

[1] This is something you'd probably not do in practice because the tokens of regular expressions are individual characters or escaped characters, so the work of the scanner is easy to integrate into the parser in this case.