EECS 510 Formal Language Project: Description

This language is the language of arithmetic expressions. Its alphabet consists of the basic numerals: 0 through 9, the standard English alphabet: a through z, parentheses: both left and right, and six operators: plus, minus, times, divide, modulo, and exponent. The language consists of the strings that can be assembled from combinations of the alphabet characters such that the resulting string is in a format that is consistent with standard arithmetic principles. Each operator represents an arithmetic operation: plus -> addition, minus -> subtraction, times -> multiplication, divide -> division, modulo -> modulo (remainder division), exponent -> exponentiation. The basic form of the language is an expression, which consists of either: an operator and two ‘sub’-expressions, one on each side of the operator; a sequence of numerals, with a length of at least one, to represent an integer; or a single letter to represent a variable. Expressions can also be surrounded by parentheses, where there must be both a left and right parenthesis, such that the right parenthesis exists to the right of the left parenthesis and are separated by an expression. The minus operator can also exist as a unary operator, representing a negative sign. In this case, it can be placed before a number, a variable, or parentheses surrounding an expression. A description of the alphabet and more complete language rules are presented below.

This language was inspired by a calculator software engineering project. The calculator took in an arithmetic expression in the form of a string, tokenized it, then calculated a result from the expression. This language aims to establish a formal definition of the strings that could be evaluated by the calculator. The grammar for this language can produce strings that could be solved by the calculator, such that the produced strings could be used as test cases for the calculator. The grammar design incorporates the order of operations into the derivation of the strings. However, the order of operations is only observable in a derivation tree, which this project is not concerned with. As such, there is some ambiguity in the grammar, as some strings can be derived in different ways, with different ‘meanings’. As an example, the equations and both produce the same string from the grammar, . The automaton for this language determines if a string is part of the language, such that the automata could be inserted into the calculator and used to determine the validity of the input string before any further processing is done. The automaton does not perform any actual calculations, its only purpose is to determine if a given string is part of the language. As such, it is not concerned with, and does not recognize, the order of operations that is encoded from the grammar. From the example above, the automaton only sees the string , and only evaluates whether or not the string belongs to the language, regardless of how the string was derived or its original meaning.

Arithmetic expressions:

Alphabet: {#, α, +, -, \*, /, %, ^, (, )}

#: [0-9] - Numerals

α: [a-z] – Variables

Operators

‘+’: Plus - binary

‘-‘: Minus – unary or binary

‘\*’: Times – binary

‘/’: Divide – binary

‘%’: Modulo – binary

‘^’: Exponent – binary

‘(‘: Left parenthesis – must be followed by matching right parenthesis

‘)’: Right parenthesis – must be preceded by matching left parenthesis

Rules:

1. Expressions:
   1. Contain operator and two sub-expressions
   2. Can be contained within set of parentheses
   3. Sub-expressions can be:
      1. Another expression
      2. Sequence of any amount of #’s
      3. Single α
      4. May be preceded by ‘-‘ if:
         1. # sequence
         2. α
         3. Contained within parentheses
2. Parentheses
   1. Every left (open) parenthesis – ‘(‘ must be followed -at some point- by a right (close) parenthesis – ‘)’
   2. Contain an expression within them
   3. Set may be preceded by ‘-‘
   4. Cannot be directly preceded/followed by #, α or another parentheses set
   5. Implemented via a stack in the PDA
3. Order of operations
   1. Parentheses
   2. Exponentiation
      1. Right-hand side expression (exponent) precedes the exponentiation
   3. Multiplication, division, modulo
   4. Addition, subtraction
   5. Left to right, within each level
4. Terminal symbol characteristics
   1. Numerals
      1. Can go anywhere. No special rules
      2. Sequences of any length (greater than zero)
      3. Cannot be directly preceded or followed by variables
   2. Variables (alphabetic characters)
      1. Go anywhere numerals go
      2. Single character sequences only
      3. Cannot directly precede or follow numerals
   3. Left parenthesis – ‘(‘
      1. –First character, or not preceded by number, variable, right parenthesis, and followed by number, variable, minus, left parenthesis
      2. If present, is start of expression
         1. Must be at start of string or preceded by an operator or left parenthesis
         2. Cannot be preceded by right parenthesis, a numeral, or a variable
      3. Cannot be followed directly by a right parenthesis, or an operator (except unary minus)
      4. Must be followed by a left parenthesis, a numeral, a variable or a unary minus
      5. Must be followed eventually by a matching right parenthesis, at the other end of the expression
      6. Cannot be last character
   4. Right parenthesis – ‘)’
      1. –Last character, or preceded by number, variable, right parenthesis and not followed by number, variable, left parenthesis
      2. If present, is end of expression
         1. Must be end of string or followed by an operator or right parenthesis
         2. Cannot be followed by left parenthesis, numeral, variable
      3. Cannot be preceded directly by a left parenthesis, or an operator
      4. Must be preceded by right parenthesis, number, variable
      5. Must be preceded (not directly) by matching left parenthesis at beginning of expression
      6. Cannot be first character
   5. Minus – ‘-‘
      1. If first character, following character must be number, variable, left parenthesis
      2. If preceding character is not number, variable, right parenthesis, following character must be number, variable, left parenthesis
      3. Otherwise, following character must be number, variable, minus, left parenthesis
      4. Cannot be end of expression or end of string
   6. Other operators – ‘+’, ’\*’, ‘/’, ‘%’, ‘^’ (plus, times, divide, modulo, exponent)
      1. Must be preceded by number, variable, right parenthesis
      2. Must be followed by number, variable, minus, left parenthesis
      3. Cannot start or end expression or string
      4. Must be preceded and followed by expression