Given the same input, i.e., a sequence of characters starting with $ and ending with #, and any combination of 0s and 1s in-between, specify a set of rewrite rules that determine whether the string contains the same number of 0s and 1s.

Here is some sample “output”:
$0011#$ should be rewritten as $# $1001#$ should be rewritten as $# $110110#$ should be rewritten as $11#$ $0001100#$ should be rewritten as $000#$

In other words, the $# indicates that the input string has the same number of 0s and 1s. If the string does not contain the same number of 0s and 1s, the resulting string shows how many more 0s or 1s there are in the input string.

$01\Rightarrow \varepsilon, 10 \Rightarrow \varepsilon$

or

$01 \Rightarrow T, 10 \Rightarrow T, T# \Rightarrow #, T1 \Rightarrow 1, 1T \Rightarrow 1, T0 \Rightarrow 0, 0T \Rightarrow 0$
HW1

…RE for real numbers…

\[(\pm \mid \pm \varepsilon \, \text{digit}^+) \times (\varepsilon \mid . \, \text{digit}^+ (\varepsilon \mid E (\varepsilon \mid -) \, \text{digit}^+ ))\]
Describe the formal languages denoted by the following regular expressions using the English language (e.g.: All strings over the alphabet . . . that have the property . . .):

1. \((\varepsilon \lor 0) \, 1^*\)*
2. \((0|1)^*0(0|1)(0|1)\)
3. \((00|11)^* \, ((01|10)(00|11)^*(01|10)(00|11)^*)^*\)

1. 1) could be an empty string; 2) \((\varepsilon \lor 0) \, 1^*\) could be eps, 0, 1;
   any combination of 1’s and 0’s

2. 1) \((0|1)^*\) == any combination of 0’s and 1’s; 2) \((0 | 1)\) could be 1 or 0; 3) \(0(0|1)(0|1)\) could be 000, 001, 010, 011
   All binary strings of length \(\geq 3\) with 0 as the third to last digit

3. 1) could be empty string; 2) should be with even # of digits.
   01010010101001001000100101001100
   010100000001110100101010
   0101000111
   even # of 01’s or 10’s == even # of 1’s and even # of 0’s
Write a regular expression for the following languages.

1. All strings of a’s, b’s, and c’s that contain no b’s following any c’s.
2. All strings of a’s, b’s, and c’s that do not contain more than 1 b and 3 a’s.

1. no b’s directly following any c’s

   1) before the first “c”, could be any combination of a’s and b’s
      starts with \((a \mid b)^*\)

   2) after the first “c”, could be any combination of a+b’s and c’s
      \((a \mid b)^*(a+b^* \mid c)^*\)

   3) could be simplified as
      \(a^*(a+b^* \mid c)^*\)
Write a regular expression for the following languages.

1. All strings of a’s, b’s, and c’s that contain no b’s following any c’s.
2. All strings of a’s, b’s, and c’s that do not contain more than 1 b and 3 a’s.

2. 1) strings containing no more than 1 “b”
   
   (a | c)* (b | eps) (a | c)*

2) strings containing no more than 3 “a”

   2.1) no “a” at all, (c)* (b | eps) (c)*
   2.2) 1 “a”, a (c)* (b | eps) (c)* | c*a (b | eps) (c)* | c* (b | eps) c*a
   2.3) 2 “a”, ........use OR to represent all the possibilities
   2.4) 3 “a”, ..... 

3) OR links everything!

......
1. Specify a DFA using a transition diagram and a formal FSA specification \(<S, s, F, T>\) (see lecture 2) that recognizes the following language: “All strings of 0’s and 1’s that, when interpreted as a binary number, are divisible by 4. In other words, value(binary number) modulo 4 = 0.”

2. Specify a DFA using a transition diagram and a formal FSA specification \(<S, s, F, T>\) (see lecture 2) that recognizes the following language: “All strings of 0’s and 1’s that, when interpreted as a binary number, are divisible by 3. In other words, value(binary number) modulo 3 = 0.”

   a) 
   \[
   \begin{align*}
   &X \mod N = i \\
   &Y \mod N = j \\
   \end{align*}
   \]

   then,

   \[
   \begin{align*}
   &(X+Y) \mod N = (i + j) \mod N \\
   &(X*Y) \mod N = (i * j) \mod N \\
   \end{align*}
   \]

   b) 
   binary string \(X\), eg. \(X = 101 = 5(D)\)

   after putting a “1” to the end, \(X1 = 1011 = 2*X + 1 = 11(D)\)

   after putting a “0” to the end, \(X0 = 1010 = 2*X = 10(D)\)
1. Specify a DFA using a transition diagram and a formal FSA specification \(<S, s, F, T>\) (see lecture 2) that recognizes the following language: “All strings of 0’s and 1’s that, when interpreted as a binary number, are divisible by 4. In other words, value(binary number) modulo 4 = 0.”

2. Specify a DFA using a transition diagram and a formal FSA specification \(<S, s, F, T>\) (see lecture 2) that recognizes the following language: “All strings of 0’s and 1’s that, when interpreted as a binary number, are divisible by 3. In other words, value(binary number) modulo 3 = 0.”

a) $X \mod N = i$
$Y \mod N = j$

then,
$(X+Y) \mod N = (i + j) \mod N$

$(X*Y) \mod N = (i * j) \mod N$

b) binary string $X$ , eg. $X= 101 = 5(D)$

after putting a “1” to the end, $X1 = 1011 = 2*X + 1 = 11(D)$

after putting a “0” to the end, $X0 = 1010 = 2*X = 10(D)$
Review 2

• Grammar: Parse Trees
  • Root: start, should be non-terminal
  • Leaves: terminals
  • Internals: non-terminals
  • For each internal N, N + N’s children=a Production
Parse Tree

Derivation: \(<\text{start}\> \Rightarrow \ldots \Rightarrow \text{terminal string}\)

Parse: \(\text{terminal string} \Rightarrow \ldots \Rightarrow <\text{start}>\)
Grammar G1:

\[ \text{<Stmt>} \rightarrow \text{<A>} \mid \text{<A>} \text{<B>} \]

\[ \text{<A>} \rightarrow a \mid \text{ala<A>} \]

\[ \text{<B>} \rightarrow b \mid \text{<B>} \text{b} \]

Leftmost derivation of aabb in G1:

\[ \text{<Stmt>} \rightarrow \text{<A>} \text{<B>} \]

\[ \rightarrow \text{a<A>} \text{<B>} \]

\[ \rightarrow \text{aa <B>} \]

\[ \rightarrow \text{aa <B>}b \]

\[ \rightarrow \text{aa bb} \]
Grammar G2:

<stmt> ::= <if-stmt> | <assgn> | ...

<if-stmt> ::= if <expr> then <stmt> |
             if <expr> then <stmt> else <stmt>

<assgn> ::= <id> := <digit>

<expr> ::= <id> = 0

<id> ::= a | b | c | ... | x | y | z

<digit> ::= 0|1|2|3|4|5|6|7|8|9
if x = 0 then if y = 0 then z := 1 else w := 2
if x = 0 then if y = 0 then z := 1 else w := 2

Q: which tree is correct?
Parse Tree

Grammar G2*:

<stmt> ::= <stmt1> | <stmt2>
<stmt1> ::= if <expr> then <stmt1> else <stmt> | <assign> | ...
<stmt2> ::= if <expr> then <stmt>

<assign> ::= <id> := <digit>
<expr> ::= <id> = 0
{id} ::= a | b | c | ... | x | y | z
<digit> ::= 0|1|2|3|4|5|6|7|8|9

Grammar G2:

<stmt> ::= <if-stmt> | <assign> | ...
<if-stmt> ::= if <expr> then <stmt> | if <expr> then <stmt> else <stmt>
<assign> ::= ...

*if there were another if-stmt before a “else”, it must have its own “else” ==
if there were compound “if-stmt”, every “else” is assigned to the nearest “if”.
Parse Tree

Grammar G2*:

\[ \text{<stmt>} ::= \text{<stmt1>} | \text{<stmt2>} \]
\[ \text{<stmt1>} ::= \text{if <expr> then <stmt1> else <stmt>} | \text{<assgn>} | ... \]
\[ \text{<stmt2>} ::= \text{if <expr> then <stmt>} \]
\[ \text{<assgn>} ::= \text{id} := \text{<digit>} \]
\[ \text{<expr>} ::= \text{id} = 0 \]
\[ \text{id} ::= a | b | c | ... | x | y | z \]
\[ \text{<digit>} ::= 0|1|2|3|4|5|6|7|8|9 \]

*several options are offered when parsing a NT

LL(1) => scan from the **Left**, Leftmost derivation, depends on the next 1 symbol(terminal).
Parse Tree

• Grammar Ambiguity

\[ \text{<expr>} ::= \text{<expr> } + \text{<term>} \mid \text{<expr> } - \text{<term>} \mid \text{<term>} \]

\[ \text{<term>} ::= \text{<term> } * \text{<factor>} \mid \text{<term> } / \text{<factor>} \mid \text{<factor>} \]

\[ \text{<factor>} ::= \text{<var>} \mid \text{<num>} \mid ( \text{<expr>} ) \]

Precedence.
Associativity.
Dangling-else Problem

• Grammar: Dangling else Problem
  • Dangling else Problem: Which “if” does a “else” belong to.

if x == 0 then if y == 0 then z := 1 else w := 2
Review 2

Dangling else Problem--- if x == 0 then if y == 0 then z := 1 else w := 2

◆ <stmt> ::= <if-stmt> | <assgn> | ...
◆ <if-stmt> ::= if <expr> then <stmt> |
  if <expr> then <stmt> else <stmt>

✓ <stmt> ::= <stmt1> | <stmt2>
✓ <stmt1> ::= if <expr> then <stmt1> else <stmt> | <assgn> | ...
✓ <stmt2> ::= if <expr> then <stmt>