CS314 Spring 2017

Assignment 1

Due Friday, February 3, 11:59pm

submission: pdf file through sakai.rutgers.edu

1 Problem — Three simple rewrite systems

Remember our "rewrite game" in the second lecture. We represent arithmetic values > 0 as sequences of "|" symbols. For example, | represents value 1, and ||||| represents value 5. The input to your rewrite system is either a single value representation, or two value representations surrounded by a begin ($) and end (#) marker, and separated by a & marker. For example, the single input value 3 is represented by $|||#, and the input pair 2,5 is represented by $||&|||||#. The normal forms produced by the rewrite systems do not contain any markers.

Give rules of rewrite systems that implement different arithmetic operations on our chosen representation. A rewrite system consists of a set of rewrite rules of the form $X \Rightarrow Y$ as discussed in class.

1. **Successor function**: f(x) = x + 1, x>0
   Example: $|||# will be rewritten to ||||
   Show the rewrite sequence of your rewrite system for the example input.

2. **Double function**: f(x) = 2 * x, x>0
   Example: $|||# will be rewritten to |||||| Show the rewrite sequence of your rewrite system for the example input.

3. **Addition function**: f(x,y) = x + y, x>0 and y>0
   Example: $||&|||# will be rewritten to ||||| Show the rewrite sequence of your rewrite system for the example input.

2 Problem — A rewrite system for modulo 3 addition

An *interpreter* for a language L maps programs written in L to their answers. Remember that a language is a set of words. Let us define our language $L_{add-mod3}$ inductively as follows:

1. The words **0**, **1**, and **2** are in $L_{add-mod3}$.

2. Assume that both A and B stand for words in the language $L_{add-mod3}$. Then

   (a) **(**A+B**)** are also in $L_{add-mod3}$.

Examples of add-mod3 expressions are: $((\mathbf{1} + \mathbf{2}) + \mathbf{0})$ and $(\mathbf{1} + (\mathbf{2} + \mathbf{2}))$. However, $\mathbf{1} + \mathbf{1}$ is not in the language (parenthesis are missing).

Give a rewrite system that "evaluates" or "computes" the value of expressions in $L_{add-mod3}$. The operators $+$ corresponds to the standard modulo 3 addition functions given below:

x	y	x $+_{mod3}$ y
0	0	0
0	1	1
1	0	1
0	2	2
2	0	2
1	1	2
1	2	0
2	1	0
2	2	1

1. Define a rewrite system for modulo 3 expressions in $L_{add-mod3}$ that produces the final value of the expression. A final value is represented by either $\mathbf{0}$, $\mathbf{1}$ or $\mathbf{2}$. Your rewrite system is basically an interpeter for $L_{add-mod3}$.

   For example, our two expressions $((\mathbf{1} + \mathbf{2}) + \mathbf{0})$ and $(\mathbf{1} + (\mathbf{2} + \mathbf{2}))$ should be rewritten to $\mathbf{0}$ and $\mathbf{2}$, respectively. You can assume that your rewrite system will only be presented with correct $L_{add-mod3}$ expressions, so don't worry about error messages.

2. Show your rewrite system steps that are performed for our two example expressions given above. For each step clearly show the left-hand side of the rule in the current expression that you are rewriting.

3. Is the choice of your next rewrite rule and its left-hand side always unique in your rewrite system? If not, show an example.

3 Problem — Regular expressions

We defined our regular expression language on page 12, lecture 2. The language contains the operators * and $^+$. We can eliminate r^* from our language without reducing its expressiveness since r^* has the same semantics as $(r^+|\epsilon)$.

1. Can we eliminate r^+ from our language without reducing its expressiveness by using a semantically equivalent regular expression that uses r^*? If yes, give the semantically equivalent regular expression.

2. Can we eliminate both, r^* and r^+, from the language without reducing the expressiveness? If yes, give the semantically equivalent regular expressions for the eliminated operators.

4 Problem — Regular expressions

Write a regular expression for floating point numbers that you want to use in your new programming language. You should allow numbers of the form 0.34, 221.5E20, and 1.0E-5, but **not** .12, 33., 21.5E, or E30. Since this is not a total specification of all possible number patterns, make reasonable assumptions for "filling in the holes". Remember: you are a language designer here.

5 Problem — Regular expressions

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. ((ϵ | a) b*)*
2. 1(0|1)*0(0|1)1

6 Problem — Regular expressions

Write a regular expression for the following languages. Make the expression as compact (short) as possible.

1. All strings of "a"s, "b"s, "c"s, and "d"s that contain no "a"s or "d"s immediately following any "b"s. "Not Immediately following" means here that the symbol immediately to the right of a "b" cannot be an "a" or "d". However, "a"s or "d"s may occur later in the string.
2. All strings of "a"s, "b"s, and "c"s that do not contain more than 1 "b" and no more than 4 "c"s.