

CS314 Spring 2023

Homework 1

Due Tuesday, January 31, 11:59pm

submission: pdf file through canvas

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. You do not have to worry about incorrect input.

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. **Triple function:** $f(x) = 3 * x, x > 0$

Example: \$|||# will be rewritten to ||||| Show the rewrite sequence of your rewrite system for the example input.

3. **Subtraction function:** $f(x,y) = x - y, x > 0, y > 0, \text{ and } x > y$

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: $((1 + 2) + 0)$ and $(1 + (2 + 2))$. However, $1 + 1$ 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 **0**, **1** or **2**. Your rewrite system is basically an interpreter for $L_{add-mod3}$.

For example, our two expressions $((1 + 2) + 0)$ and $(1 + (2 + 2))$ should be rewritten to **0** and **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

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. $((\epsilon \mid 1) 0^*)^*$
2. $0(0|1)^*1(0|1)1$

4 Problem — Regular expressions

Write a regular expression for the following language. **You must use the regular language definition introduced in class (see lecture 3).** Make the expression as compact (short) as possible.

1. All strings of “a”s, “b”s, and “c”s that contain exactly 2 “a”s
2. All strings of “a”s, “b”s, and “c”s that contain at least 1 “b” or at least 3 “c”s.
3. All strings of “a”s, “b”s, and “c”s that do not contain more than 1 “b” and no more than 3 “c”s.

5 Problem — Regular expressions and finite state machines

You are designing a new language with fixed-point numbers. Every fixed-point number should have a unique representation. This means, no leading or trailing 0’s are allowed, and every number must have a “point”. Examples:

- Allowed: 0.0, 10.0, 45000.007, 0.888
- Not allowed: 0, 10, 10., 10.00, 045000.007, .888

1. Write a regular expression for fixed-point numbers for your language.
2. Give a DFA in the form of a state transition graph that recognizes your language. Note: No need to introduce error states; your DFA can reject the input if it gets “stuck”. Keep your DFA as small as possible.

6 Problem — Regular expressions and finite state machines

Use the discussed “translation” strategy for constructing an ϵ -NFA from a regular expression as discussed in lecture 3 for the regular expression.

letter (letter | digit)*

Show the ϵ -NFA for the above regular expression.