CSC148, Assignment #2
Regular expressions
due November 5th, 2013, 11:59 p.m.

October 30, 2013

introduction

Regular expressions (abbreviated to *regex*, the pronunciation of which gives rise to endless flame wars...) are used in various programming languages and utilities to match entire classes of strings. This assignment will give you experience modelling a regular expression as a tree, and detecting which strings match a given regular expression.

We won't favour any particular language or utility, such as Python, Java, or Linux's grep, but use a stripped-down simplified regular expression form that contains all the essential principles. In particular, you are **not** permitted to import Python's regular expression module `re` into any file you submit.

We'll only be dealing with the binary alphabet, i.e., $\{0, 1\}$. Generalizing to arbitrary alphabet is straight forward, but doesn't buy us much for the purpose of this assignment. The elementary regex symbols we'll use can easily typed in Python source code. A regex (over binary alphabet $0, 1$) is a nonempty string made up of the following symbols.

`'0'` called zero

`'1'` one

`'e'` pronounced "ee" or "epsilon"

`'|'` bar

`'.'` dot

`'*'` star

`'('` left parenthesis, or left

`')'` right parenthesis, or right

There are several rules determining that a string made up of these symbols is a valid regular expression:

1. There are 3 regexes of length one. They are:
   - `'0'`
   - `'1'`
   - `'e'`

2. If r is a regular expression, then so is r + '*', where the plus symbol '+' means string concatenation, as in Python.

3. If r_1 and r_2 are regexes, then so are:

   - '(' + r_1 + '|' + r_2 + ')'
   - '(' + r_1 + '.' + r_2 + ')'

Here are some examples of regexes:

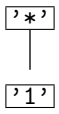
- '0'
- '1'
- 'e'
- '0*'
- '1*'
- 'e*'
- '(0|1)'
- '(1.0)'
- '(e|0)'
- '(1.e)'
- '(0*|1*)'
- '((0.1).0)'
- '((1.(0|1)*).0)'

tree representation of regexes

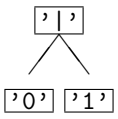
Every regex can be (uniquely) represented as a tree. Each leaf node contains exactly one of '0', '1', or 'e'. Each internal node contains exactly one of '.', '—', or '*'.

A regex of length one is represented by a tree of one node containing the symbol in the regex. For example, the regex '0' is represented by the tree whose root is the leaf node containing '0'.

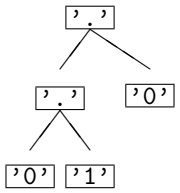
A regex of the form r + '*' is represented by a tree whose root node contains '*', and that node has one child which is the tree that represents the regex r. E.g., the regex '1*' is represented by the following tree:



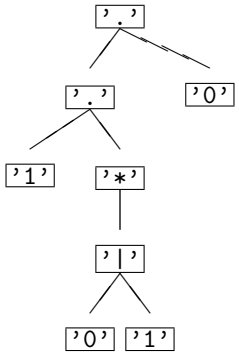
A regex of the form '(' + r_1 + '|' + r_2 + ')'x1 is represented by a tree whose root node contains '|', and that node has left and right children which are the trees that represent the regexes r_1 and r_2 respectively. For example, the regex '(0|1)' is represented by the following tree:



A regex of the form '(' + r_1 + '.' + r_2 + ')' is represented just as a regex of the form '(' + r_1 + '|' + r_2 + ')', except the root now contains '.' rather than '|'. For example, the regex '((0.1).0)' is represented by the following tree:



Here's an example that combines all concepts from above. The regex '((1.(0|1)*).0)' is represented by the following tree:



matching strings with regexes

A binary string is a string (possibly empty) that contains only the symbols '0' and '1'. For a regex r and a binary string s, we define below what it means for r to match s. (Equivalently we may also say that s matches r, or that r and s match).

1. A regex of length one matches exactly one string. Specifically:
   - the regex '0' matches the string '0'
   - the regex '1' matches the string '1'
   - the regex 'e' matches the string '' (i.e., 'e' matches the empty string)
2. A regex of the form r + '*' matches string s if and only if either
   (a) s equals '' (empty string), or
   (b) s has the form $s_1 + s_2 + \cdots + s_k$ where $k > 0$ and r matches every s_i

   For example, the regex '0*' matches any string (possibly empty) that contains no other symbols than the symbol '0'
3. A regex of the form '(' + r_1 + '|' + r_2 + ')' matches string s if and only if:

(a) r_1 matches s, or

(b) r_2 matches s, or

(c) both of the above

For, example, the regex `'(1|0*)'` matches the string `'1'` as well as any string that contains only the symbol `'0'`

4. A regex of the form `'('` + r_1 + `'.'` + r_2 + `')'` matches a string s if and only if there are two strings s_1 and s_2 (each possibly empty) such that

   (a) s is the concatenation of s_1 and s_2 (i.e., s equals $s_1 + s_2$),

   (b) r_1 matches s_1, and

   (c) r_2 matches s_2.

   For example, the regex `'(1*.0)'` matches any string that contains exactly one `'0'`, and that `'0'` occurs at the very end.

Here's an example that combines all concepts from above. The regex `'((1.(0|1)*).0)'` matches any string that starts with `'1'` and ends with `'0'`.

your job

You may work with up to two other students currently in CSC148 — in other words, a group of 1 to 3 people. Your tasks are:

1. Complete the initializer of `RegexTree` in regextree.py. In the same directory you will need regextreenode.py in order to import the various regex node classes. Currently the initializer sets `self.root` to `None`, but when you're done it will be the root node of the regular expression tree that corresponds to the argument `regex`. Remember that you are **not** allowed to import Python regular expression modules such as **re**.

2. Also in regextree.py, create a module-level function `regex_match(r, s)` that returns `True` if and only if string `s` matches regular expression tree `r`.

3. Add your name(s) to the license at the top of regextree.py, indicating that you have added intellectual value to it. You may only distribute the file, or modified versions of it, with the same license

4. Submit `regextree.py` to MarkUs