Graduate Programming Languages
Homework Assignment

Wes Weimer

Exercise 1: Bookkeeping. How long did this take you? Was it easy? Tell me something about yourself that I do not already know. All non-empty answers receive full credit. The usual submission mechanism applies: submit your textual answers in a PDF file named after your UVA ID (e.g., `mst3k.pdf`) as well as `mst3k-hw3.ml` and `test-mst3k.input` (see below).

Exercise 2: *Regular Expressions* are commonly used as abstractions for string matching. Here is an abstract grammar for regular expressions:

e	::=	"`x`"	singleton — matches the character `x`
	|	empty	skip — matches the empty string
	|	$e_1\ e_2$	concatenation — matches e_1 followed by e_2
	|	$e_1 \mid e_2$	or — matches e_1 or e_2
	|	$e*$	Kleene star — matches 0 or more occurrence of e
	|	.	matches any single character
	|	["`x`" − "`y`"]	matches any character between `x` and `y` inclusive
	|	$e+$	matches 1 or more occurrences of e
	|	$e?$	matches 0 or 1 occurrence of e

We will call the first five cases the *primary* forms of regular expressions. The last four cases can be defined in terms of the first five. We also give an abstract grammar for strings (modeled as lists of characters):

s	::=	nil	empty string
	|	"`x`" :: s	string with first character `x` and other characters s

We write "`bye`" as shorthand for "`b`" :: "`y`" :: "`e`" :: nil. This exercise requires you to give large-step operational semantics rules of inference related to regular expressions matching strings. We introduce a judgment:

$$\vdash e \text{ matches } s \text{ leaving } s'$$

The interpretation of the judgment is that the regular expression e matches some prefix of the string s, leaving the suffix s' unmatched. If $s' = \text{nil}$ then

r matched s exactly. Examples:

$$\vdash \text{"h"}(\text{"e"}+) \text{ matches "hello" leaving "llo"}$$

Note that this operational semantics may be considered *non-deterministic* because we expect to be able to derive all three of the following:

$$\begin{array}{l} \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "ello"} \\ \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "hello"} \\ \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "llo"} \end{array}$$

Here are two rules of inference:

$$\frac{s = \text{"x"} :: s'}{\vdash \text{"x" matches } s \text{ leaving } s'} \qquad \frac{}{\vdash \text{empty matches } s \text{ leaving } s}$$

Give large-step operational semantics rules of inference for the other three primal regular expressions.

Exercise 3: We want to update our operational semantics for regular expressions to capture multiple suffices. We want our new operational semantics to be deterministic — it return the set of all possible answers from the single-answer operational semantics above. We introduce a new judgment:

$$\vdash e \text{ matches } s \text{ leaving } S$$

And use rules of inference like the following:

$$\frac{}{\vdash \text{"x" matches } s \text{ leaving } \{s' \mid s = \text{"x"} :: s'\}} \qquad \frac{}{\vdash \text{empty matches } s \text{ leaving } \{s\}}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \quad \vdash e_2 \text{ matches } s \text{ leaving } S'}{\vdash e_1 \mid e_2 \text{ matches } s \text{ leaving } S \cup S'}$$

You must do one of the following:

- *either* give operational semantics rules of inference for $e*$ and e_1e_2. You may *not* place a derivation inside a set constructor, as in: $\{x \mid \exists y.\ \vdash e \text{ matches } x \text{ leaving } y\}$. Each inference rules must have a finite and fixed set of hypotheses.
- *or* argue in one or two sentences that it cannot be done correctly in the given framework. Back up your argument by presenting two attempted but "wrong" rules of inference and show that each one is either unsound or incomplete with respect to our intuitive notion of regular expression matching.

Part of doing research is getting stuck. When you get stuck, you must be able to recognize whether "you are just missing something" or "the problem is actually impossible".

Exercise 4: Optional. In the class notes (marked as "optional material" in the lecture slides) we defined an equivalence relation $c_1 \sim c_2$ for IMP commands. Computing equivalence turned out to be undecideable: $c \sim c$ iff c halts. We can define a similar equivalence relation for regular expressions: $e_1 \sim e_2$ iff $\forall s \in S.\ \vdash e_1 \text{ matches } s \text{ leaving } S_1 \ \wedge\ \vdash e_2 \text{ matches } s \text{ leaving } S_2 \Rightarrow S_1 = S_2$ (note that we are using an "updated" operational semantics that returns the set of all possible matched suffices, as in the previous problem). You must *either* claim that $e_1 \sim e_2$ is undecideable by reducing it to the halting problem *or* explain in two or three sentences how to compute it *or* write "I choose not to do this problem" (you will receive full credit). You may assume that I am familiar with the relevant literature.

Exercise 5: Download the Homework 3 code pack from the course web page. Update the skeletal SMT solver so that it correctly integrates the given DPLL-style CNF SAT solver with the given theory of bounded arithmetic. In particular, you must update only the Main.solve function. Your updated solver must be correct. This notably implies that it must correctly handle all of the included test cases — we use `diff` for some testing, but if you change only the listed method you should end up with the same answers as the reference.

Why do the last two included tests take such a comparatively long time? Impress me with your knowledge of DPLL(T) — feel free to use information from the assigned reading or related papers, not just from the lecture slides. I am looking for a reasonably detailed answer. Include a discussion of which single module you would rewrite first to improve performance, as well as how you would change that module.

Exercise 6: Submit `mst3k-hw3.ml` and `test-mst3k.input` files according to the instructions in the code pack.