G52MAL Machines and their Languages, Spring 2012

Thorsten Altenkirch

Exercises, Set 3

Friday 24st February 2012

Deadline: Wednesday 14th March 2012, in your tutorial

(extended deadline)

Let $\Sigma = \{a, b, c\}$ for questions 1–4.

1. Explicitly compute the languages denoted by the following regular expressions:

   (a) $\mathbf{ab} + \mathbf{c}^*\emptyset + \epsilon\mathbf{c}$

   (b) $\mathbf{a}(\mathbf{b} + \mathbf{c})\mathbf{b} + (\emptyset + \mathbf{c})\epsilon$

2. Give regular expressions denoting the following languages:

   (a) $\{\varepsilon, a, b, ac, bc\}$

   (b) $\{a\, b^n c \mid n \in \mathbb{N}, n > 2\}$

3. Give regular expressions defining the following languages:

   (a) All words.

   (b) All words that do not contain any as.

   (c) All words that contain the sequence bbc.

   (d) All words that contain at least two as.

   (e) All words such that all as appear before all cs.

   (f) All words such that the total number of bs is even.

   (g) All words that do not contain the sequence cc.

   (h) All words that do not contain the sequence ccc.

4. For each of the following regular expressions, construct an equivalent NFA following the graphical construction given in the lectures (and lecture notes). You may eliminate unreachable and "dead-end" (those from which no accepting state can be reached) states, but you should not perform any other reductions.

   (a) $\mathbf{a} + (\mathbf{bc})^*$

   (b) $\emptyset\mathbf{a} + (\mathbf{b} + \mathbf{c})^*\mathbf{a} + \epsilon$

5. **Bonus Exercise**

   Consider the following data type encoding regular expressions:

$$
\begin{aligned}
\textbf{data}\ \mathit{RE}\ \sigma\ =\ & \mathit{Empty} \\
\mid\ & \mathit{Epsilon} \\
\mid\ & \mathit{Symbol}\ \sigma \\
\mid\ & \mathit{Plus}\ (\mathit{RE}\ \sigma)\ (\mathit{RE}\ \sigma) \\
\mid\ & \mathit{Sequence}\ (\mathit{RE}\ \sigma)\ (\mathit{RE}\ \sigma) \\
\mid\ & \mathit{Star}\ (\mathit{RE}\ \sigma) \\
\mid\ & \mathit{Paren}\ (\mathit{RE}\ \sigma) \\
& \textbf{deriving}\ (\mathit{Eq}, \mathit{Show})
\end{aligned}
$$

   The type parameter σ is the underlying alphabet.

For example, some regular expressions over the alphabets of characters and integers are as follows:

```
--  ϵ + abc
re1 :: RE Char
re1 = Epsilon `Plus` ((Symbol 'a' `Sequence` Symbol 'b') `Sequence` Symbol 'c')
--  (01)*
re2 :: RE Char
re2 = Star (Paren (Symbol '0' `Plus` Symbol '1'))
--  1*
re3 :: RE Int
re3 = Star (Symbol 1)
```

Consider also the following encoding of words and languages:

```
type Word σ = [σ]
type Language σ = [Word σ]
```

(a) Define the empty word for any alphabet:

```
ε :: Word σ
```

(b) Define a function that concatenates two languages.

```
langConcat :: Language σ → Language σ → Language σ
```

Note that this is substantially more challenging for infinite languages than for finite languages. I suggest that you first define *langConcat* for finite languages, and then only attempt to extend it to infinite languages if you are feeling particularly adventurous.

(c) Define a function that raises a language to an integer power (you can ignore negative integers).

```
langExp :: Language σ → Int → Language σ
```

(d) Define a function that applies the Kleene Star operation to a language.

```
kleeneStar :: Eq σ ⇒ Language σ → Language σ
```

Note that while this function will not be terminating, it should be *productive*. That is, it should enumerate all words in the (infinite) resultant language, rather than hanging. Thus, for example, *take n* (*kleeneStar l*) should terminate for any language *l* and positive integer *n*.

(e) Define a function that enumerates the language of a regular expression.

```
re2lang :: Eq σ ⇒ RE σ → Language σ
```

Hint: You may find the following functions helpful:

```
import Data.List (union)
unions :: Eq a ⇒ [[a]] → [a]
unions = foldr union []
```

Note that *unions* has been defined using *foldr* rather than *foldl*. If you have a working solution, try using *foldl* instead and see if it makes a difference.