School of Computer Science, University of Nottingham

G52MAL Machines and their Languages, Spring 2012

Thorsten Altenkirch

Solutions to Exercises, Set 3

Friday 23rd March 2012

1. (a)

$$
\begin{array}{rll}
 & L(\mathbf{ab} + \mathbf{c}^*\emptyset + \epsilon\mathbf{c}) & \\
= & L(\mathbf{ab}) \cup L(\mathbf{c}^*\emptyset) \cup L(\epsilon\mathbf{c}) & \{L(E+F) = L(E) \cup L(F)\} \\
= & L(\mathbf{a})L(\mathbf{b}) \cup L(\mathbf{c}^*)L(\emptyset) \cup L(\epsilon)L(\mathbf{c}) & \{L(EF) = L(E)L(F)\} \\
= & L(\mathbf{a})L(\mathbf{b}) \cup L(\mathbf{c}^*)\emptyset \cup L(\epsilon)L(\mathbf{c}) & \{L(\emptyset) = \emptyset\} \\
= & L(\mathbf{a})L(\mathbf{b}) \cup \emptyset \cup L(\epsilon)L(\mathbf{c}) & \{S\emptyset = \emptyset\} \\
= & L(\mathbf{a})L(\mathbf{b}) \cup \emptyset \cup \{\epsilon\}L(\mathbf{c}) & \{L(\epsilon) = \{\epsilon\}\} \\
= & L(\mathbf{a})L(\mathbf{b}) \cup \emptyset \cup L(\mathbf{c}) & \{\{\epsilon\}S = S\} \\
= & \{a\}\{b\} \cup \emptyset \cup \{c\} & \{L(\mathbf{x}) = \{x\}\} \\
= & \{ab\} \cup \emptyset \cup \{c\} & \{\text{Concatenation of Languages}\} \\
= & \{ab, c\} & \{\text{Set Union}\}
\end{array}
$$

   (b)

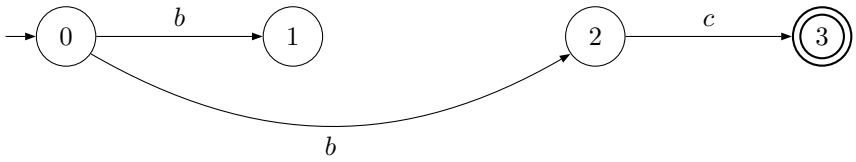
$$
\begin{array}{rll}
 & L(\mathbf{a}(\mathbf{b} + \mathbf{c})\mathbf{b} + (\emptyset + \mathbf{c})\epsilon) & \\
= & L(\mathbf{a}(\mathbf{b} + \mathbf{c})\mathbf{b}) \cup L((\emptyset + \mathbf{c})\epsilon) & \{L(E+F) = L(E) \cup L(F)\} \\
= & L(\mathbf{a})L(\mathbf{b} + \mathbf{c})L(\mathbf{b}) \cup L(\emptyset + \mathbf{c})L(\epsilon) & \{L(EF) = L(E)L(F)\} \\
= & L(\mathbf{a})L(\mathbf{b} + \mathbf{c})L(\mathbf{b}) \cup L(\emptyset + \mathbf{c})\{\epsilon\} & \{L(\epsilon) = \{\epsilon\}\} \\
= & L(\mathbf{a})L(\mathbf{b} + \mathbf{c})L(\mathbf{b}) \cup L(\emptyset + \mathbf{c}) & \{S\{\epsilon\} = S\} \\
= & L(\mathbf{a})(L(\mathbf{b}) \cup L(\mathbf{c}))L(\mathbf{b}) \cup L(\emptyset) \cup L(\mathbf{c}) & \{L(E+F) = L(E) \cup L(F)\} \\
= & L(\mathbf{a})(L(\mathbf{b}) \cup L(\mathbf{c}))L(\mathbf{b}) \cup \emptyset \cup L(\mathbf{c}) & \{L(\emptyset) = \emptyset\} \\
= & \{a\}(\{b\} \cup \{c\})\{b\} \cup \emptyset \cup \{c\} & \{L(\mathbf{x}) = \{x\}\} \\
= & \{a\}\{b, c\}\{b\} \cup \{c\} & \{\text{Set Union}\} \\
= & \{abb, acb\} \cup \{c\} & \{\text{Concatenation of Languages}\} \\
= & \{abb, acb, c\} & \{\text{Set Union}\}
\end{array}
$$

2. (a) $\epsilon + (\mathbf{a} + \mathbf{b})(\epsilon + \mathbf{c})$
   (b) $\mathbf{abbbb}^*\mathbf{c}$

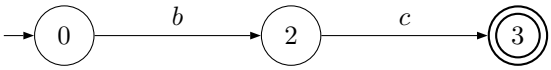
3. (a) $(\mathbf{a} + \mathbf{b} + \mathbf{c})^*$
   (b) $(\mathbf{b} + \mathbf{c})^*$
   (c) $(\mathbf{a} + \mathbf{b} + \mathbf{c})^*\mathbf{bbc}(\mathbf{a} + \mathbf{b} + \mathbf{c})^*$
   (d) $(\mathbf{b} + \mathbf{c})^*\mathbf{a}(\mathbf{b} + \mathbf{c})^*\mathbf{a}(\mathbf{a} + \mathbf{b} + \mathbf{c})^*$
   (e) $(\mathbf{a} + \mathbf{b})^*(\mathbf{b} + \mathbf{c})^*$
   (f) $(\mathbf{a} + \mathbf{c} + \mathbf{b}(\mathbf{a} + \mathbf{c})^*\mathbf{b})^*$
   (g) $(\epsilon + \mathbf{c})((\mathbf{a} + \mathbf{b})(\epsilon + \mathbf{c}))^*$
   (h) $(\epsilon + \mathbf{c} + \mathbf{cc})((\mathbf{a} + \mathbf{b})(\epsilon + \mathbf{c} + \mathbf{cc}))^*$

4. (a) Let us start by constructing NFAs for **b** and **c**. I have named the states to make it easier for you to see how they correspond to the final NFA, but normally you wouldn't name them until later.

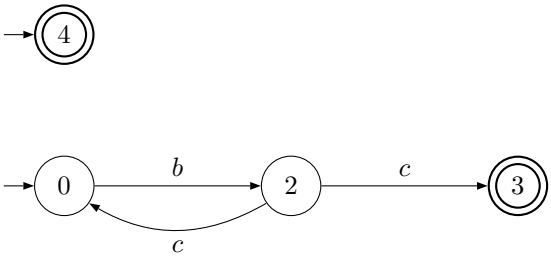
An NFA for **bc** is then constructed by composing the two NFAs sequentially:



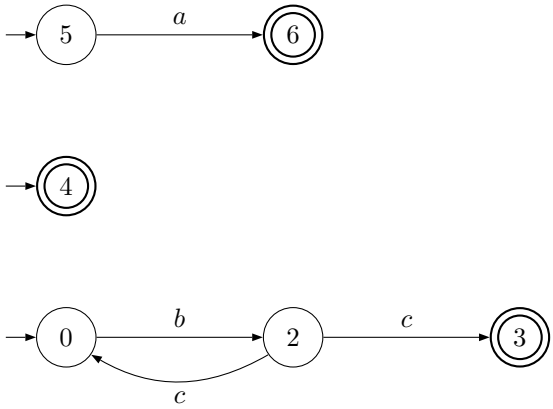
State 1 is now a dead-end state and can be removed:



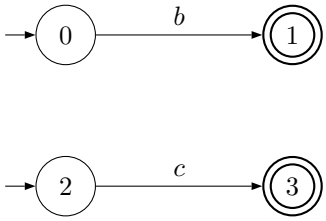
An NFA for $(\mathbf{bc})^*$ is then constructed by duplicating all arrows that point to an accepting state to point to all initial states, and then adding an initial accepting state:



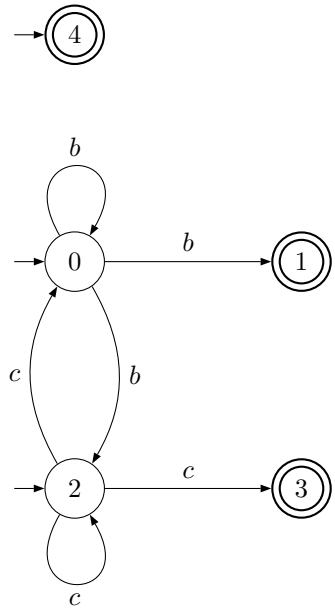
Finally, an NFA for $\mathbf{a} + (\mathbf{bc})^*$ is constructed by adding an NFA for $\mathbf{a}$ in parallel:



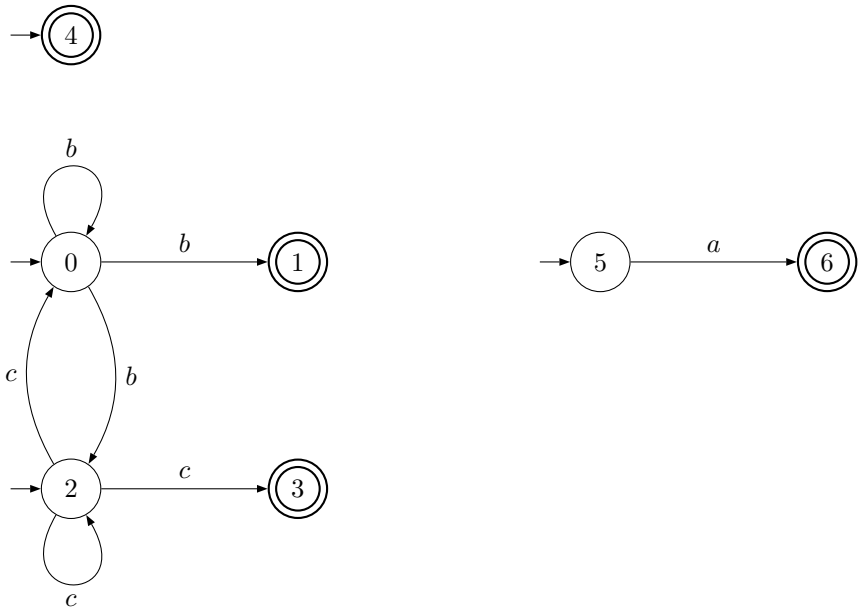
(b) Let us start with an NFA for $\mathbf{b} + \mathbf{c}$:



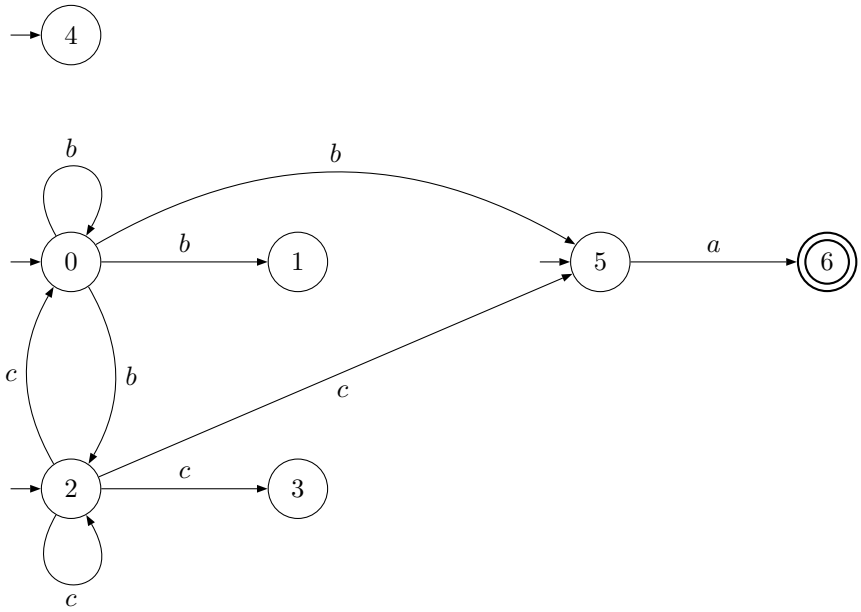
An NFA for $(\mathbf{b} + \mathbf{c})^*$ is then as follows:



To construct an NFA for $(\mathbf{b} + \mathbf{c})^*\mathbf{a}$, we first place the previous NFA side by side with an NFA for $\mathbf{a}$:

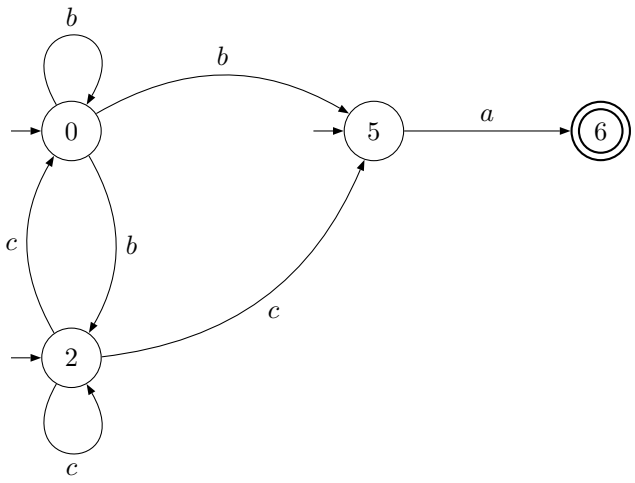


The two are then composed sequentially:

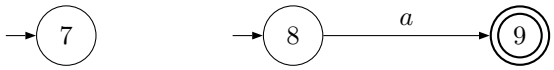


Note that the initial arrow was removed from State 5, then added again as the initial arrow for State 4 was duplicated to point to State 5.

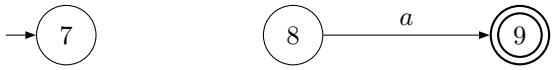
At this point, it's a good idea to remove dead-end states.



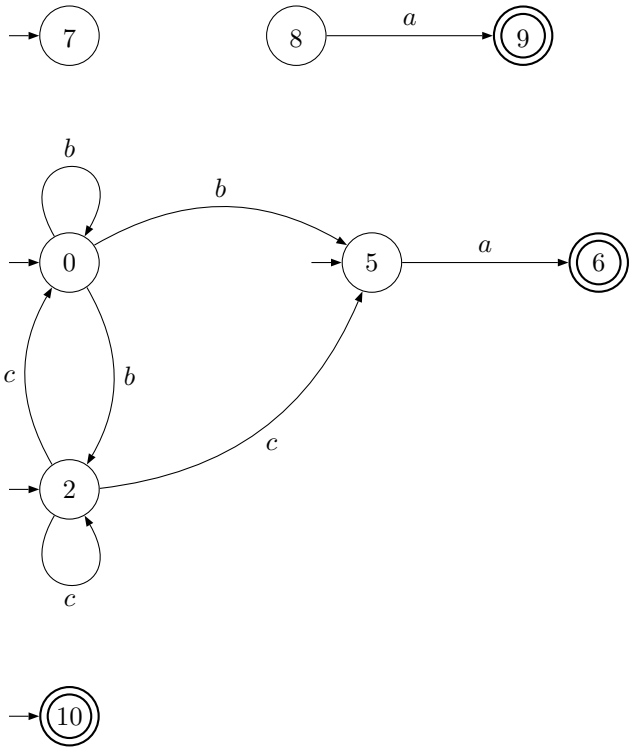
Next let's consider $\emptyset\mathbf{a}$. First we place the NFAs for $\emptyset$ and $\mathbf{a}$ side by side:



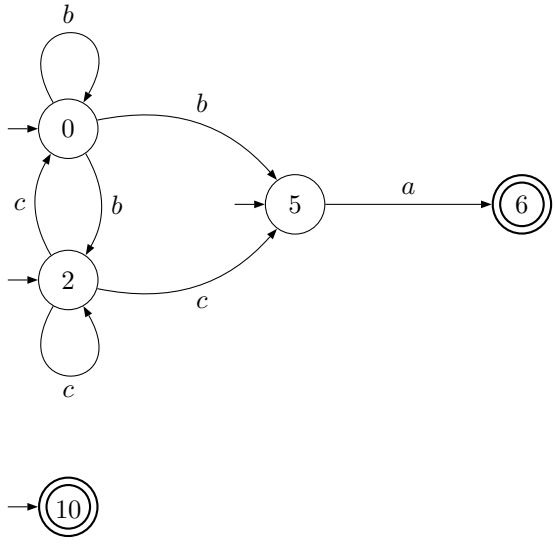
We now construct an NFA for $\emptyset\mathbf{a}$ by composing them in sequence. As $N(\emptyset)$ has no accepting states, this just requires setting all states from $N(\mathbf{a})$ to not be initial.



The NFA for $\emptyset\mathbf{a} + (\mathbf{b} + \mathbf{c})^*\mathbf{a} + \epsilon$ consists of $N(\emptyset\mathbf{a})$, $N((\mathbf{b} + \mathbf{c})^*\mathbf{a})$ and $N(\epsilon)$ in parallel.



Finally, State 7 can be eliminated as it is a dead-end state, and states 8 and 9 can be eliminated as they are unreachable.



5. (a) The empty word is just the empty list:

$$\varepsilon :: \mathit{Word}\ \sigma$$
$$\varepsilon\ =\ [\,]$$

(b) Concatenation for finite languages can be defined using list comprehension:

$$\mathit{langConcat} :: \mathit{Language}\ \sigma\ \to\ \mathit{Language}\ \sigma\ \to\ \mathit{Language}\ \sigma$$
$$\mathit{langConcat}\ l_1\ l_2\ =\ [w_1\ +\!\!+\ w_2\ \mid\ w_1\ \leftarrow\ l_1, w_2\ \leftarrow\ l_2]$$

It gets a bit trickier for infinite languages, as we want to ensure that all words get enumerated. For example, if l_2 is infinite, then, using the above definition, only the first word from l_1 will ever appear in the resultant language.

As each word in the resultant language is a pair of words (one from each argument language), the problem is to assign a countable ordering to all such pairs. This can be achieved by using the Cantor Pairing Function to assign an order of enumeration (see, for example, `http://en.wikipedia.org/wiki/Pairing_function`). However, for this problem, iterating through the pairs is all that is required, so defining an iterating function is sufficient (rather than directly defining the inverse pairing function).

```
nextpair :: (Int, Int) → (Int, Int)
nextpair (0, n)  = (n + 1, 0)
nextpair (m, n)  = (m − 1, n + 1)
```

Concatenation can now be defined recursively as follows:

```
langConcat :: Language σ → Language σ → Language σ
langConcat l1 l2 = lcAux (0, 0)
  where
    lcAux (m, n) | b1 ∧ b2   = ((l1 !! m) ++ (l2 !! n)) : lcAux (nextpair (m, n))
                 | b1 ∨ b2   = lcAux (nextpair (m, n))
                 | otherwise = []
                 where b1 = inBound m l1
                       b2 = inBound n l2
inBound :: Int → [a] → Bool
inBound n = not ∘ null ∘ drop n
```

Note that as the languages may be finite, we need to use *inBound* to check that we do not index a list (language in this case) out of bounds. If the bounds of both languages are exceeded, then all words have been enumerated and the recursion terminates.

Further note that this function could be made much more efficient. For example, specialised auxiliary functions could be introduced for the cases when one of the two languages is discovered to be finite.

(c) Exponentiation of languages can be defined by primitive recursion:

```
langExp :: Language σ → Int → Language σ
langExp l 0 = [ε]
langExp l n = langConcat l (langExp l (n − 1))
```

(d) The definition $L^* = \bigcup_{n=0}^{\infty} L^n$ can be encoded fairly directly:

```
kleeneStar :: Eq σ ⇒ Language σ → Language σ
kleeneStar l = unions [langExp l n | n ← [0 ..]]
```

(e) Finally, enumerating the language of a regular expression is a straightforward encoding of its semantics using the functions already defined:

$$
\begin{array}{lcl}
\mathit{re2lang} :: \mathit{Eq}\ \sigma\ \Rightarrow\ \mathit{RE}\ \sigma\ \rightarrow\ \mathit{Language}\ \sigma & & \\
\mathit{re2lang}\ \mathit{Empty} & = & [\,] \\
\mathit{re2lang}\ \mathit{Epsilon} & = & [\varepsilon] \\
\mathit{re2lang}\ (\mathit{Symbol}\ x) & = & [[x]] \\
\mathit{re2lang}\ (\mathit{Plus}\ \mathit{r1}\ \mathit{r2}) & = & \mathit{re2lang}\ \mathit{r1}\ \text{`}\mathit{union}\text{`}\ \mathit{re2lang}\ \mathit{r2} \\
\mathit{re2lang}\ (\mathit{Sequence}\ \mathit{r1}\ \mathit{r2}) & = & \mathit{langConcat}\ (\mathit{re2lang}\ \mathit{r1})\ (\mathit{re2lang}\ \mathit{r2}) \\
\mathit{re2lang}\ (\mathit{Star}\ r) & = & \mathit{kleeneStar}\ (\mathit{re2lang}\ r) \\
\mathit{re2lang}\ (\mathit{Paren}\ r) & = & \mathit{re2lang}\ r
\end{array}
$$