Coursework (Strand 2)

This coursework is worth 20% and is due on 13 December at 16:00. You are asked to prove the correctness of the regular expression matcher from the lectures using the Isabelle theorem prover. You need to submit a theory file containing this proof. The Isabelle theorem prover is available from

`http://isabelle.in.tum.de`

This is an interactive theorem prover, meaning that you can make definitions and state properties, and then help the system with proving these properties. Sometimes the proofs are also automatic. There is a shortish user guide for Isabelle, called "Programming and Proving in Isabelle/HOL" at

`http://isabelle.in.tum.de/documentation.html`

and also a longer (free) book at

`http://www.concrete-semantics.org`

The Isabelle theorem prover is operated through the jEdit IDE, which might not be an editor that is widely known. JEdit is documented in

`http://isabelle.in.tum.de/dist/Isabelle2014/doc/jedit.pdf`

If you need more help or you are stuck somewhere, please feel free to contact me (christian.urban@kcl.ac.uk). I am one of the main developers of Isabelle and have used it for approximately the 16 years. One of the success stories of Isabelle is the recent verification of a microkernel operating system by an Australian group, see `http://sel4.systems`. Their operating system is the only one that has been proved correct according to its specification and is used for application where high assurance, security and reliability is needed.

The Task

In this coursework you are asked to prove the correctness of the regular expression matcher from the lectures in Isabelle. For this you need to first specify what the matcher is supposed to do and then to implement the algorithm. Finally you need to prove that the algorithm meets the specification. The first two parts are relatively easy, because the definitions in Isabelle will look very similar to the mathematical definitions from the lectures or the Scala code that is supplied at KEATS. For example very similar to Scala, regular expressions are defined in Isabelle as an inductive datatype:

```
datatype rexp =
  ZERO
| ONE
| CHAR char
| SEQ rexp rexp
| ALT rexp rexp
| STAR rexp
```

The meaning of regular expressions is given as usual:

$$\begin{array}{rcll}
L(\mathbf{0}) & \stackrel{\text{def}}{=} & \varnothing & \texttt{ZERO} \\
L(\mathbf{1}) & \stackrel{\text{def}}{=} & \{[]\} & \texttt{ONE} \\
L(c) & \stackrel{\text{def}}{=} & \{[c]\} & \texttt{CHAR} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) & \texttt{ALT} \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & L(r_1) \,@\, L(r_2) & \texttt{SEQ} \\
L(r^*) & \stackrel{\text{def}}{=} & (L(r))^* & \texttt{STAR}
\end{array}$$

You would need to implement this function in order to state the theorem about the correctness of the algorithm. The function L should in Isabelle take a `rexp` as input and return a set of strings. Its type is therefore

$$\texttt{L :: rexp} \Rightarrow \texttt{string set}$$

Isabelle treats strings as an abbreviation for lists of characters. This means you can pattern-match strings like lists. The union operation on sets (for the `ALT`-case) is a standard definition in Isabelle, but not the concatenation operation on sets and also not the star-operation. You would have to supply these definitions. The concatenation operation can be defined in terms of the append function, written `_ @ _` in Isabelle, for lists. The star-operation can be defined as a "big-union" of powers, like in the lectures, or directly as an inductive set.

The functions for the matcher are shown in Figure 1. The theorem that needs to be proved is

```
theorem
  "matches r s ⟷ s ∈ L r"
```

which states that the function *matches* is true if and only if the string is in the language of the regular expression. A proof for this lemma will need side-lemmas about `nullable` and `der`. An example proof in Isabelle that will not be relevant for the theorem above is given in Figure 2.

```
fun
  nullable :: "rexp ⇒ bool"
where
  "nullable ZERO = False"
| "nullable ONE = True"
| "nullable (CHAR _) = False"
| "nullable (ALT r1 r2) = (nullable(r1) ∨ nullable(r2))"
| "nullable (SEQ r1 r2) = (nullable(r1) ∧ nullable(r2))"
| "nullable (STAR _) = True"

fun
  der :: "char ⇒ rexp ⇒ rexp"
where
  "der c ZERO = ZERO"
| "der c ONE = ZERO"
| "der c (CHAR d) = (if c = d then ONE else ZERO)"
| "der c (ALT r1 r2) = ALT (der c r1) (der c r2)"
| "der c (SEQ r1 r2) =
     (if (nullable r1) then ALT (SEQ (der c r1) r2) (der c r2)
                      else SEQ (der c r1) r2)"
| "der c (STAR r) = SEQ (der c r) (STAR r)"

fun
  ders :: "rexp ⇒ string ⇒ rexp"
where
  "ders r [] = r"
| "ders r (c # s) = ders (der c r) s"

fun
  matches :: "rexp ⇒ string ⇒ bool"
where
  "matches r s = nullable (ders r s)"
```

Figure 1: The definition of the matcher algorithm in Isabelle.

```
fun
  zeroable :: "rexp ⇒ bool"
where
  "zeroable ZERO = True"
| "zeroable ONE = False"
| "zeroable (CHAR _) = False"
| "zeroable (ALT r1 r2) = (zeroable(r1) ∧ zeroable(r2))"
| "zeroable (SEQ r1 r2) = (zeroable(r1) ∨ zeroable(r2))"
| "zeroable (STAR _) = False"

lemma
  "zeroable r ⟷ L r = {}"
proof (induct)
  case (ZERO)
  have "zeroable ZERO" "L ZERO = {}" by simp_all
  then show "zeroable ZERO ⟷ (L ZERO = {})" by simp
next
  case (ONE)
  have "¬ zeroable ONE" "L ONE = {[]}" by simp_all
  then show "zeroable ONE ⟷ (L ONE = {})" by simp
next
  case (CHAR c)
  have "¬ zeroable (CHAR c)" "L (CHAR c) = {[c]}" by simp_all
  then show "zeroable (CHAR c) ⟷ (L (CHAR c) = {})" by simp
next
  case (ALT r1 r2)
  have ih1: "zeroable r1 ⟷ L r1 = {}" by fact
  have ih2: "zeroable r2 ⟷ L r2 = {}" by fact
  show "zeroable (ALT r1 r2) ⟷ (L (ALT r1 r2) = {})"
    using ih1 ih2 by simp
next
  case (SEQ r1 r2)
  have ih1: "zeroable r1 ⟷ L r1 = {}" by fact
  have ih2: "zeroable r2 ⟷ L r2 = {}" by fact
  show "zeroable (SEQ r1 r2) ⟷ (L (SEQ r1 r2) = {})"
    using ih1 ih2 by (auto simp add: Conc_def)
next
  case (STAR r)
  have "¬ zeroable (STAR r)" "[] ∈ L (r) ^ 0" by simp_all
  then show "zeroable (STAR r) ⟷ (L (STAR r) = {})"
    by (simp (no_asm) add: Star_def) blast
qed
```

Figure 2: An Isabelle proof about the function zeroable.