

Coursework (Strand 2)

This coursework is worth 25% and is due on 11 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 a main developer of Isabelle and have used it for approximately the 14 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 =
  NULL
| EMPTY
| CHAR char
| SEQ rexp rexp
| ALT rexp rexp
| STAR rexp

```

The meaning of regular expressions is given as usual:

$L(\emptyset)$	$\stackrel{\text{def}}{=} \emptyset$	NULL
$L(\epsilon)$	$\stackrel{\text{def}}{=} \{\ [] \}$	EMPTY
$L(c)$	$\stackrel{\text{def}}{=} \{ [c] \}$	CHAR
$L(r_1 + r_2)$	$\stackrel{\text{def}}{=} L(r_1) \cup L(r_2)$	ALT
$L(r_1 \cdot r_2)$	$\stackrel{\text{def}}{=} L(r_1) @ L(r_2)$	SEQ
$L(r^*)$	$\stackrel{\text{def}}{=} (L(r))^*$	STAR

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

`L :: rexp \Rightarrow 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 ???. The theorem that needs to be proved is

```

theorem
  "matches r s  $\longleftrightarrow$  s  $\in$  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 ???.

```

1 fun
2   nullable :: "rexpr ⇒ bool"
3 where
4   "nullable NULL = False"
5 | "nullable EMPTY = True"
6 | "nullable (CHAR _) = False"
7 | "nullable (ALT r1 r2) = (nullable(r1) ∨ nullable(r2))"
8 | "nullable (SEQ r1 r2) = (nullable(r1) ∧ nullable(r2))"
9 | "nullable (STAR _) = True"
10
11 fun
12   der :: "char ⇒ rexp ⇒ rexp"
13 where
14   "der c NULL = NULL"
15 | "der c EMPTY = NULL"
16 | "der c (CHAR d) = (if c = d then EMPTY else NULL)"
17 | "der c (ALT r1 r2) = ALT (der c r1) (der c r2)"
18 | "der c (SEQ r1 r2) =
19     (if (nullable r1) then ALT (SEQ (der c r1) r2) (der c r2)
20      else SEQ (der c r1) r2)"
21 | "der c (STAR r) = SEQ (der c r) (STAR r)"
22
23 fun
24   ders :: "rexpr ⇒ string ⇒ rexp"
25 where
26   "ders r [] = r"
27 | "ders r (c # s) = ders (der c r) s"
28
29 fun
30   matches :: "rexpr ⇒ string ⇒ bool"
31 where
32   "matches r s = nullable (ders r s)"

```

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

```

1 fun
2   zeroable :: "rexpr ⇒ bool"
3 where
4   "zeroable NULL = True"
5 | "zeroable EMPTY = False"
6 | "zeroable (CHAR _) = False"
7 | "zeroable (ALT r1 r2) = (zeroable(r1) ∧ zeroable(r2))"
8 | "zeroable (SEQ r1 r2) = (zeroable(r1) ∨ zeroable(r2))"
9 | "zeroable (STAR _) = False"
10
11 lemma
12   "zeroable r ⟷ L r = {}"
13 proof (induct)
14   case (NULL)
15   have "zeroable NULL" "L NULL = {}" by simp_all
16   then show "zeroable NULL ⟷ (L NULL = {})" by simp
17 next
18   case (EMPTY)
19   have "¬ zeroable EMPTY" "L EMPTY = {[]}" by simp_all
20   then show "zeroable EMPTY ⟷ (L EMPTY = {})" by simp
21 next
22   case (CHAR c)
23   have "¬ zeroable (CHAR c)" "L (CHAR c) = {[c]}" by simp_all
24   then show "zeroable (CHAR c) ⟷ (L (CHAR c) = {})" by simp
25 next
26   case (ALT r1 r2)
27   have ih1: "zeroable r1 ⟷ L r1 = {}" by fact
28   have ih2: "zeroable r2 ⟷ L r2 = {}" by fact
29   show "zeroable (ALT r1 r2) ⟷ (L (ALT r1 r2) = {})"
30     using ih1 ih2 by simp
31 next
32   case (SEQ r1 r2)
33   have ih1: "zeroable r1 ⟷ L r1 = {}" by fact
34   have ih2: "zeroable r2 ⟷ L r2 = {}" by fact
35   show "zeroable (SEQ r1 r2) ⟷ (L (SEQ r1 r2) = {})"
36     using ih1 ih2 by (auto simp add: Conc_def)
37 next
38   case (STAR r)
39   have "¬ zeroable (STAR r)" "[ ] ∈ L (r) ^ 0" by simp_all
40   then show "zeroable (STAR r) ⟷ (L (STAR r) = {})"
41     by (simp (no_asm) add: Star_def) blast
42 qed

```

Figure 2: An Isabelle proof about the function zeroable.