Regexp Module

`smlhelp.github.io` – Auxiliary Library

Jacob Neumann

July 2021

Contents

Note 1

Throughout, `Sigma` is assumed to be some equality type.

Note 2

This document assumes the definitions and declarations of https://github.com/smlhelp/aux-library/blob/main/documentation/language.pdf, and assumes that the `Language` structure has been opened.

In particular, we have

Defn. 3

aux-library: Language.sml

```
type 'S language = 'S list -> bool
```

A total function `D : Sigma list -> bool` is said to **compute** a set L of values of type `Sigma list` if

$$\texttt{D cs} \Longrightarrow \texttt{true} \qquad \text{iff} \qquad \texttt{cs} \in L$$

i.e. we can run `D` on any value `cs : Sigma list` and get back either `true` or `false`, which correctly indicates that either `cs` $\in L$ or `cs` $\notin L$, respectively.

1 Preliminary Declarations & Main Definitions

Declaration 4

aux-library: Regexp.sml

```
datatype ''S regexp =
    Zero
    | One
    | Const of ''S
    | Plus of ''S regexp * ''S regexp
    | Times of ''S regexp * ''S regexp
    | Star of ''S regexp
```

Defn. 5 (Language)

Given a value `R : Sigma regexp`, define the **language of R** to be the set of values $\mathcal{L}(\texttt{R}) \subseteq \textsf{Values}(\texttt{Sigma list})$ given recursively by:

$$
\begin{aligned}
\mathcal{L}(\texttt{Zero}) &= \emptyset \\
\mathcal{L}(\texttt{One}) &= \{\texttt{[]}\} \\
\mathcal{L}(\texttt{Const(c)}) &= \{\texttt{[c]}\} \\
\mathcal{L}(\texttt{Plus(r1,r2)}) &= \mathcal{L}(\texttt{r1}) \cup \mathcal{L}(\texttt{r2}) \\
\mathcal{L}(\texttt{Times(r1,r2)}) &= \{\texttt{v}_1\texttt{@v}_2 \mid \texttt{v}_1 \in \mathcal{L}(\texttt{r1}) \text{ and } \texttt{v}_2 \in \mathcal{L}(\texttt{r2})\} \\
\mathcal{L}(\texttt{Star(r)}) &= \{\texttt{v}_1\texttt{@v}_2\texttt{@ ... @v}_n \mid n \in \mathbb{N},\ \texttt{v}_1, \texttt{v}_2, \ldots, \texttt{v}_n \in \mathcal{L}(\texttt{r})\}
\end{aligned}
$$

Defn. 6

A total function `D : Sigma regexp -> Sigma language` is said to **compute the regular expression decision problem** (over the alphabet `Sigma`) if

$$\texttt{D R cs} \Longrightarrow \texttt{true} \quad \text{if and only if} \quad \texttt{cs} \in \mathcal{L}(\texttt{R})$$

for all `R : Sigma regexp` and all `cs : Sigma list`.

Defn. 7

A pair `(p,s) : Sigma list * Sigma list` is said to be a **splitting** of `cs : Sigma list` if

$$\texttt{cs} \cong \texttt{p @ s}.$$

In such a splitting, `p` is called the **prefix** and `s` the **suffix**.

Declaration 8

aux-library: Regexp.sml

```
exception NoMatch
```

Defn. 9

For any type `t`, we'll say that a function

`k : Sigma list * Sigma list -> t`

is **almost total** if, for all `(p,s) : Sigma list * Sigma list`, either

`k(p,s)` $\hookrightarrow$ `v` for some value `v` or `k(p,s)` raises `NoMatch`.

If `k(p,s)` $\hookrightarrow$ `v`, then we say that `k` **accepts** `(p,s)` **with result** `v`.

2 Matching Specification

Declaration 10 (Depth)

Define the **depth** of `R : Sigma regexp` recursively by

aux-library: Regexp.sml

```
fun depth Zero = 0
  | depth One = 0
  | depth (Const(_)) = 0
  | depth (Plus(R1,R2)) =
      1 + Int.max(depth R1,depth R2)
  | depth (Times(R1,R2)) =
      1 + Int.max(depth R1,depth R2)
  | depth (Star(R)) =
      1 + depth R
```

Value Spec

```
match : ''a regexp
        -> ''a list
        -> (''a list * ''a list -> 'b)
        -> 'b
```

REQUIRES: `k` is almost total (Defn. 9)

ENSURES:

$$\texttt{match r cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{r}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Value Spec

`LL : ''S regexp -> ''S language`

ENSURES: `LL` computes the regular expression decision problem: for all `R : Sigma regexp`, `(LL R) : Sigma list -> bool` is a total function such that

$$\texttt{LL R cs} \Longrightarrow \texttt{true} \quad \text{iff} \quad \texttt{cs} \in \mathcal{L}(\texttt{R})$$

3 Matching Implementation

aux-library: Regexp.sml

```
datatype ''S regexp =
    Zero
    | One
    | Const of ''S
    | Plus of ''S regexp * ''S regexp
    | Times of ''S regexp * ''S regexp
    | Star of ''S regexp

fun depth Zero = 0
  | depth One = 0
  | depth (Const(_)) = 0
  | depth (Plus(R1,R2)) =
      1 + Int.max(depth R1,depth R2)
  | depth (Times(R1,R2)) =
      1 + Int.max(depth R1,depth R2)
  | depth (Star(R)) =
      1 + depth R

exception NoMatch
```

aux-library: Regexp.sml

```
fun match Zero _ _ = raise NoMatch
  | match One cs k = k([],cs)
  | match (Const(c)) [] k = raise NoMatch
  | match (Const(c)) (c'::cs') k =
      if c=c'
      then k([c'], cs')
      else raise NoMatch
  | match (Plus(R1,R2)) cs k =
        (match R1 cs k
          handle NoMatch => match R2 cs k)
  | match (Times(R1,R2)) cs k =
      match R1 cs (fn (res',cs') =>
        match R2 cs' (fn (res'',cs'') =>
          k (res'@res'',cs'')))
  | match (Star(R)) cs k =
      k([],cs)
        handle NoMatch =>
          match R cs (fn (res',cs') =>
            if (cs = cs')
            then raise NoMatch
            else
              match (Star(R)) cs' (fn (res'',cs'') =>
                k(res'@res'',cs'')))

val LL = fn R => fn s =>
  match R s (fn (_,[]) => true | _ => raise NoMatch)
  handle NoMatch => false
```

4 Zero, One, Const Correctness

Lemma 11

For any almost total `k` and any `cs : Sigma list`,

$$\texttt{match Zero cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Zero}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Proof. Since $\mathcal{L}(\texttt{Zero}) = \emptyset$, there cannot be any splitting of any `cs` whose prefix is in $\mathcal{L}(\texttt{Zero})$. Therefore, this spec says that `match Zero cs k` always ought to raise `NoMatch`, which it does. **Spec Satisfied!**

Cor. 12

For any almost total `k`, `(fn cs => match Zero cs k)` is almost total.

Lemma 13

For any almost total `k` and any `cs : Sigma list`,

$$\texttt{match One cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{One}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Proof. Since $\mathcal{L}(\texttt{One}) = \{\texttt{[]}\}$, the only splitting of `cs` which we need to consider is `([],cs)`. But notice that

$$\texttt{k([],cs)} \cong \begin{cases} \texttt{v} & \text{if } \texttt{k} \text{ accepts } \texttt{([],cs)} \text{ with result } \texttt{v} \\ \texttt{raise NoMatch} & \text{if } \texttt{k} \text{ rejects } \texttt{([],cs)} \end{cases}$$

so we have **Spec Satisfied!** by putting `match One cs k = k([],cs)`. □

Cor. 14

For any almost total `k`, `(fn cs => match One cs k)` is almost total.

Lemma 15

For any almost total `k`, any `cs : Sigma list`, and any value `c : Sigma`

$$\texttt{match (Const c) cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Const c}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Proof. Begin by recalling that $\mathcal{L}(\texttt{Const c}) = \{\texttt{[c]}\}$. Break into two cases: `cs=[]` and `cs=c'::cs'`.

If `cs=[]`, then there is no prefix of `cs` in $\mathcal{L}(\texttt{Const c})$, so `match (Const c) [] k` should raise `NoMatch`. **Spec Satisfied!**

If `cs=c'::cs'`, then break into two further cases: either `c=c'` or `c<>c'`. In the latter case, there is no prefix of `cs` in $\mathcal{L}(\texttt{Const c})$, so again `NoMatch` should be raised. **Spec Satisfied!** However, if `c=c'`, then `[c']` $\in \mathcal{L}(\texttt{Const c})$, so whatever behavior `k ([c'],cs')` has is what behavior `match (Const c) cs k` should have. **Spec Satisfied!**

Cor. 16

For any almost total `k` and any value `c:Sigma`, `(fn cs => match (Const c) cs k)` is almost total.

5 Plus Correctness

Inductive Assumption (Section 5)

For some `R1,R2 : Sigma regexp`,

$$\texttt{match R1 cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{R1}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

$$\texttt{match R2 cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{R2}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

for all values `cs : Sigma list` and all almost total `k`.

In particular: if `k` is almost total, then so too are `(fn cs => match R1 cs k)` and `(fn cs => match R2 cs k)`.

Lemma 17

For any almost total `k`, any `cs : Sigma list`,

$$\texttt{match (Plus(R1,R2)) cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Plus(R1,R2)}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Cor. 18

For any almost total `k`, `(fn cs => match (Plus(R1,R2)) cs k)` is almost total.

6 Times Correctness

Inductive Assumption (Section 6)

For some `R1,R2 : Sigma regexp`,

$$\texttt{match R1 cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{R1}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

$$\texttt{match R2 cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{R2}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

for all values `cs : Sigma list` and all almost total `k`.

In particular: if `k` is almost total, then so too are `(fn cs => match R1 cs k)` and `(fn cs => match R2 cs k)`.

Lemma 19

For any almost total `k`, any `cs : Sigma list`,

$$\texttt{match (Times(R1,R2)) cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Times(R1,R2)}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Cor. 20

For any almost total `k`, `(fn cs => match (Times(R1,R2)) cs k)` is almost total.

7 Star Correctness

Inductive Assumption (Section 7)

For some `R : Sigma regexp` and some `cs : Sigma list`

$$\texttt{match R cs g} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{R}) \text{ and } \texttt{g} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

$$\texttt{match (Star(R)) cs' g} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs'} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Star(R)}) \text{ and } \texttt{g} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

for all values `cs' : Sigma list` *of length less than* `cs`, and all almost total `g`.

In particular: if `g` is almost total, then so too is `(fn cs => match R cs g)`. And if `g` is almost total, `match (Star(R)) cs' g` for `cs'` of length less than `cs`.

Lemma 21

For any almost total `k`,

$$\texttt{match (Star(R)) cs k} \cong \begin{cases} \texttt{v} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \text{ such that } \texttt{p} \in \mathcal{L}(\texttt{Star(R)}) \text{ and } \texttt{k} \text{ accepts } \texttt{(p,s)} \text{ with result } \texttt{v}. \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Cor. 22

For any almost total `k`, `(fn cs => match (Star(R)) cs k)` is almost total.

8 match and LL Correctness

9 Reduction, Representation, and Printing Specification

Value Spec

`represent : (''S -> string) -> ''S regexp -> string`

REQUIRES: `toStr` is total, `R` is non-`Zero`

ENSURES: `represent toStr R` converts the regular expression `R` into POSIX-like regular expression syntax, using `toStr` to convert `Sigma` values into strings. This uses the convention of representing `One` as `(.{0,0})`

Defn. 23

A value `R : Sigma regexp` is said to be **`Zero`-reduced** if either:

- `R = Zero`
- `R` does not contain any `Zero`s

Value Spec

`reduce : ''S regexp -> ''s regexp`

ENSURES: `(reduce R)` $\Longrightarrow$ `R'` such that $\mathcal{L}(\texttt{R}) = \mathcal{L}(\texttt{R'})$ and `R'` is `Zero`-reduced (and also `R'` may have some superfluous `One`s removed)

Value Spec

`printRep : (''S -> string) -> ''S regexp -> unit`

REQUIRES: `toStr` is total, `(reduce R)` $\not\Longrightarrow$ `Zero`

ENSURES: `printRep toStr R` $\Longrightarrow$ `()`

EFFECT: Prints the value of `(represent toStr (reduce R))` into the REPL.