Regular Expressions – Specification and Correctness Proof

15-150 Principles of Functional Programming (M21)

Jacob Neumann

June 15, 2021

1 Specifications

aux-library: Regexp.sml

```
signature REGEXP =
sig
    datatype ''a regexp =
          Const of ''a
        | One
        | Zero
        | Times of ''a regexp * ''a regexp
        | Plus of ''a regexp * ''a regexp
        | Star of ''a regexp

    val depth : ''a regexp -> int

    exception NoMatch

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

    val LL : char regexp -> Language.language

end
```

$$
\begin{array}{lll}
\mathcal{L}(\texttt{Const(c)}) & = & \{\texttt{[c]}\} \\
\mathcal{L}(\texttt{Zero}) & = & \emptyset \\
\mathcal{L}(\texttt{One}) & = & \{\texttt{[]}\} \\
\mathcal{L}(\texttt{Times(r1,r2)}) & = & \{\texttt{s1@s2} \mid \texttt{s1} \in \mathcal{L}(\texttt{r1}) \text{ and } \texttt{s2} \in \mathcal{L}(\texttt{r2})\} \\
\mathcal{L}(\texttt{Plus(r1,r2)}) & = & \mathcal{L}(\texttt{r1}) \cup \mathcal{L}(\texttt{r2}) \\
\mathcal{L}(\texttt{Star(r)}) & = & \{\texttt{s1 @ s2 @ ... @ sn} \mid n \in \mathbb{N},\ \texttt{s1,s2,..., sn} \in \mathcal{L}(r)\}
\end{array}
$$

Defn. For any type `t`, a pair `(p,s) : t list * t list` is said to be a **splitting** of `L : t list` if

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

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

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

REQUIRES: for all `(p,s)`, `k(p,s)` either reduces to a value or raises `NoMatch`

ENSURES:

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

2 (Almost) Totality

Prop. For any equality type `T`, any type `t`, any value `r : T regexp`, any value `cs : T list`, and any value `k : T list * T list -> t` satisfying the REQUIRES clause for `match`, either

`match r cs k` $\hookrightarrow$ `v` for some `v` or `match r cs k` raises `NoMatch`

Proof. Let `k` be an arbitrary value satisfying the REQUIRES. We proceed by structural induction on `r`.

Base Case: `Const` `r=Const(a)` for some `a:T`. If `cs=[]` or `cs=c::cs'` for some `c<>a`, then observe

$$\texttt{match r cs k} \cong \texttt{raise NoMatch}.$$

Otherwise, if `cs` $\cong$ `a::cs'`, then

$$\texttt{match r cs k} \hookrightarrow \texttt{k([a],cs')}$$

and, by hypothesis, `k` either evaluates to a value or raises `NoMatch`.

Base Case: `Zero` `r=Zero`. Observe `match Zero cs k` always raises `NoMatch`, satisfying the claim.

Base Case: `One` `r=One`.

$$\texttt{match One cs k} \hookrightarrow \texttt{k([],cs)}$$

and, by hypothesis, `k` either evaluates to a value or raises `NoMatch`.

Inductive Hypothesis For all `g` satisfying the REQUIRES, and all `cs`,

`match r1 cs g` $\hookrightarrow$ `v` for some `v` or `match r1 cs g` raises `NoMatch`

`match r2 cs g` $\hookrightarrow$ `v` for some `v` or `match r2 cs g` raises `NoMatch`

Inductive Step: `Times` `r=Times(r1,r2)`. Notice that since `k` satisfies the REQUIRES, the function `k''` given by

```
fn (res'',cs'') => k(res'@res'',cs'')
```

also satisfies the REQUIRES (assuming `res'` is some value). We can use this fact, plus our inductive hypothesis for `r2`, to get that the function `k'` given by

```
fn (res',cs') => match r2 cs' k''
```

always either evaluates to a value or raises `NoMatch` when applied to some value `(res', cs')`. Thus `k'` satisfies the REQUIRES. So, by the inductive hypothesis for `r1`,

`match r1 cs k'` $\hookrightarrow$ `v` for some `v` or `match r1 cs k'` raises `NoMatch`

and since `match r cs k` $\Longrightarrow$ `match r1 cs k'`, we have the claim.

Inductive Step: `Plus` `r=Plus(r1,r2)`. By the inductive hypothesis for `r1`, we have that `match r1 cs k` either evaluates to a value or raises `NoMatch`. If it raises `NoMatch`, then

`match r cs k` $\Longrightarrow$ `match r2 cs k`

and, by the inductive hypothesis for `r2`, this either evaluates to a value or raises `NoMatch`.

Inductive Step: `Star` `r=Star(r1)`. By the assumption that `k` satisfies the REQUIRES, the evaluation of `k([],cs)` either evaluates to a value or raises `NoMatch`. If it evaluates to a value `v`, then

`match (Star(r1)) cs k` $\hookrightarrow$ `v`

too. If it raises `NoMatch`, then

```
match r cs k  ==>  match r cs (fn (res',cs') =>
                   if (cs = cs')
                   then raise NoMatch
                   else
                   match (Star(r)) cs' (fn (res'',
                   cs'') => k(res'@res'',cs'')))
```

Now observe that the continuation

```
(fn (res',cs') =>
    if (cs = cs')
    then raise NoMatch
    else match (Star(r)) cs' (fn (res'',cs'') =>
        k(res'@res'',cs'')))
```

either evaluates to a value or raises `NoMatch` for any `(res',cs')`: if `cs=cs'`, then this raises `NoMatch`. Otherwise, it steps to

```
match (Star(r)) cs' (fn (res'',cs'') => k(res'@res'',cs'')).
```

Since `k` evaluates to a value or raises `NoMatch`, so does `(fn (res'',cs'') => k(res'@res'',cs''))`, and so t □

3 Correctness

Defn. Given `r : T regexp` and `k` satisfying the REQUIRES of `match`, a pair of `T list` values `(p,s)` is said to **satisfy** `r` and `k` if

$$\texttt{p} \in \mathcal{L}(\texttt{r}) \qquad \text{and} \qquad \texttt{k(p,s)} \hookrightarrow \texttt{v} \text{ for some value } \texttt{v}$$

Thm. For any equality type `T`, any type `t`, any value `cs : T list`, and any value `k : T list * T list -> t` satisfying the REQUIRES clause for `match`,

$$\texttt{match r cs k} \cong \begin{cases} \texttt{k(p,s)} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \\ & \text{that satisfies } \texttt{r} \text{ and } \texttt{k} \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Proof. Proceed by structural induction on `r`.

Base Case: `Const` `r=Const(a)` for some `a:T`. Pick arbitrary `k` satisfying the REQUIRES. We split into two cases, `cs=[]` and `cs=c::cs'`. If `cs=[]`, then the only possible prefix of `cs` is `[]`. However,

$$\texttt{[]} \notin \mathcal{L}(\texttt{Const(a)}),$$

so no splitting of `cs` can satisfy `r` and `k`. Accordingly,

$$\texttt{match (Const(a)) [] k} \quad \cong \quad \texttt{raise NoMatch}. \qquad \textbf{Spec satisfied!}$$

If `cs=c::cs'`, then either `c=a` or `c<>a`. If `c=a`, then `[c]` is the only prefix of `cs` which is in $\mathcal{L}(\texttt{Const(a)})$. Since

$$\texttt{match (Const(a)) (c::cs') k} \quad \Longrightarrow \quad \texttt{k([c],cs')} \qquad (\text{when } \texttt{c=a})$$

we get that `match r cs k` will evaluate to whatever value `k(p,s)` evaluates to if `([c],cs')` satisfies `r` and `k`, or raise `NoMatch` if `k([c],cs')` does. **Spec satisfied!** If `c<>a`, then no prefix of `cs` is in $\mathcal{L}(\texttt{r})$, so no splitting of `cs` satisfies `r` and `k`. So `match r cs k` raises `NoMatch`. **Spec satisfied!**

Base Case: `Zero` `r=Zero`. Pick arbitrary `cs` and arbitrary `k` satisfying the REQUIRES. There is no prefix of `cs` in $\mathcal{L}(\texttt{r}) = \emptyset$, so no splitting of `cs` satisfies `r` and `k`. Therefore,

$$\texttt{match Zero cs k} \quad \cong \quad \texttt{raise NoMatch}. \qquad \textbf{Spec satisfied!}$$

Base Case: `One` `r=One`. For any `cs`, the prefix `[]` is in $\mathcal{L}(\texttt{One})$, and this is the only prefix of `cs` in $\mathcal{L}(\texttt{One})$. Therefore, the only splitting of `cs` that could possibly satisfy `r` and `k` is `([],cs)`. So

$$\texttt{match One cs k} \quad \Longrightarrow \quad \texttt{k([],cs)}$$

so `match r cs k` evaluates to the same value as `k([],cs)` if `([],cs)` satisfies `r` and `k`, and raises `NoMatch` if not. **Spec satisfied!**

Inductive Hypothesis For all `k` satisfying the REQUIRES, and all `cs`,

$$\texttt{match r1 cs k} \cong \begin{cases} \texttt{k(p,s)} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \\ & \text{that satisfies } \texttt{r1} \text{ and } \texttt{k} \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

$$\texttt{match r2 cs k} \cong \begin{cases} \texttt{k(p,s)} & \text{where } \texttt{(p,s)} \text{ is a splitting of } \texttt{cs} \\ & \text{that satisfies } \texttt{r2} \text{ and } \texttt{k} \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p,s)} \end{cases}$$

Inductive Step: `Times` `r=Times(r1,r2)`.

Let `k` be any function satisfying the REQUIRES.

Suppose there is a splitting `(p,s)` of `cs` satisfying `Times(r1,r2)` and `k`. Then, by definition, there must be `p1` $\in \mathcal{L}($`r1`$)$ and `p2` $\in \mathcal{L}($`r2`$)$ such that `p` $\cong$ `p1@p2` and `k(p,s)` evaluates to some value. So then observe

`(p1,p2@s)` satisfies `r1` and `k'`.

To see this, note that `p1` $\in \mathcal{L}($`r1`$)$ and that

$$\begin{aligned} & \texttt{k'(p1,p2@s)} \\ \Longrightarrow\ & \texttt{match r2 cs' k''} && (\texttt{p2@s} \hookrightarrow \texttt{cs'}) \\ \cong\ & \begin{cases} \texttt{k''(p'',s'')} & \text{where } \texttt{(p'',s'')} \text{ is a splitting} \\ & \text{of } \texttt{cs'} \text{ that satisfies } \texttt{r2} \text{ and } \texttt{k''} \\ \texttt{raise NoMatch} & \text{if there is no such } \texttt{(p'',s'')} \end{cases} \end{aligned}$$

but there is such a `(p'',s'')`, namely `(p2,s)`. This is because we specified that `p2@s` $\cong$ `cs'`, we assumed that `p2` $\in \mathcal{L}($`r2`$)$ and we know `k''(p2,s)` evaluates to a value:

$$\begin{aligned} \texttt{k''(p2,s)} & \Longrightarrow \texttt{k(p1@p2,s)} \\ & \cong \texttt{k(p,s)} && (\texttt{p1@p2} \cong \texttt{p}\text{, above}) \\ & \hookrightarrow \text{ some value} && \text{(assumed above)} \end{aligned}$$

So, in this case, we get that

$$\texttt{match r cs k} \quad \cong \quad \texttt{k(p,s)}.$$

Spec satisfied!

Now assume there is no such splitting of `cs` which satisfies `Times(r1,r2)` and `k`. This could be the case because there is no prefix of `cs` in $\mathcal{L}($`r1`$)$. In this case, there cannot be any `(p',s')` satisfying `r1` and `k`, so we'll get

$$\texttt{match r cs k} \quad \cong \quad \texttt{raise NoMatch}.$$ **Spec satisfied!**

So suppose `p1` is some prefix of `cs` which is in $\mathcal{L}($`r1`$)$.

Inductive Step: `Plus`

Inductive Step: `Star` □