# 15–150: Principles of Functional Programming
## *Proving Correctness of the Regular Expression Matcher*

Michael Erdmann
Spring 2025

## 1 Introduction

The notes on regular expression matching[1] outline one proof technique; these notes discuss another.

## 2 Correctness as Termination, Soundness, and Completeness

We would like to prove that `match` satisfies its specs. Recall those are:

```
match : regexp -> char list -> (char list -> bool) -> bool

REQUIRES: k is total.

ENSURES:  match r cs k  returns true,
                        if cs can be split as cs ≅ p@s,
                        with p representing a string in L(r)
                        and k(s) evaluating to true;
          match r cs k  returns false, otherwise.
```

• One way to prove correctness is to prove directly that `match` satisfies the specs:
One must show that `(match r cs k)` returns `true` under the conditions specified and `false` otherwise.

• Another way to prove correctness is to first prove *termination* of `match`, assuming `k` is total. By termination we mean that evaluation of `(match r cs k)` always returns some value, i.e., does not loop forever or raise an exception. That proof is surprisingly difficult, and we will omit it.

Instead, let us assume termination is given. Then it is enough to prove the following:

**Theorem:** For all values `r : regexp`, `cs : char list`, `k : char list -> bool`, with `k` total,

$$\texttt{match r cs k} \cong \texttt{true}$$

if and only if

there exist `p` and `s` such that $\texttt{cs} \cong \texttt{p @ s}$, $\texttt{p} \in L(\texttt{r})$, and $\texttt{(k s)} \cong \texttt{true}$.

The "only if" direction of this theorem is frequently called "soundness". Soundness means that if the matcher returns `true`, then the character list really does split as specified.

The "if" direction of the theorem is frequently called "completeness". Completeness means that if the character list splits as described, then the matcher really does return `true`.

Notice that the theorem never discusses any situations in which the matcher returns `false`. It does not need to do so, since we have assumed termination: We know that the matcher always returns some value (assuming `k` is total), so the theorem covers all possibilities.

[1] See `regexp.pdf` linked via this lecture's web page, originally authored by Bob Harper.

## 3 Proof of Soundness and Completeness

The proof of Theorem is by structural induction on `r`.

For the Base Cases, one has three cases, requiring that one prove Theorem for each of `One`, `Zero`, and `(Char a)`, with `a` any `char`. We omit those simple proofs in this writeup.

For the Inductive Cases, one may assume that Theorem holds for any regular subexpression, and then must establish Theorem for each of the recursive cases in the definition of `regexp`. The notes by Bob Harper discuss the constructors `Times` and `Star` (with a different proof technique), so we focus merely on the constructor `Plus` here.

The Induction Hypothesis means we get to assume Theorem for two subexpressions `r1` and `r2`.

We then need to show Theorem for `Plus(r1,r2)`, i.e., we need to establish:

For all values `cs : char list`, `k : char list -> bool`, with `k` total,
`(match (Plus(r1,r2)) cs k)` $\cong$ `true` if and only if
there exist `p` and `s` such that `cs` $\cong$ `p @ s`, `p` $\in L($`Plus(r1,r2)`$)$, and `(k s)` $\cong$ `true`.

We split that proof into the two directions of the "if and only if":

**Soundness:** We need to prove: If `(match (Plus(r1,r2)) cs k)` $\cong$ `true`,
then there exist `p` and `s` such that `cs` $\cong$ `p @ s`, `p` $\in L($`Plus(r1,r2)`$)$, and `(k s)` $\cong$ `true`.

Showing:

| | | |
|---|---|---|
| | `true` | |
| $\cong$ | `(match (Plus(r1,r2)) cs k)` | [by assumption] |
| $\cong$ | `(match r1 cs k) orelse (match r2 cs k)` | [by the `Plus` clause in `match`] |

One or both of the arguments to the `orelse` must therefore evaluate to `true`. Let us suppose it is the first argument. (If the first argument does not evaluate to `true` then it must evaluate to `false`, since the overall `orelse` reduces to `true`. In this case, the second argument must evaluate to `true`, after which the proof is similar.)

So `(match r1 cs k)` $\cong$ `true`. By the Induction Hypothesis for `r1`, we know that there exist `p` and `s` such that `cs` $\cong$ `p @ s`, `p` $\in L($`r1`$)$, and `(k s)` $\cong$ `true`. By the language definition for `Plus` (see again the notes by Bob Harper), we therefore see that `p` $\in L($`Plus(r1,r2)`$)$, completing the soundness part of the proof.

**Completeness:** We need to prove: If there exist `p` and `s` such that `cs` $\cong$ `p @ s`, `p` $\in L($`Plus(r1,r2)`$)$, and `(k s)` $\cong$ `true`, then `(match (Plus(r1,r2)) cs k)` $\cong$ `true`.

Showing: The language definition for `Plus` tells us that either `p` $\in L($`r1`$)$ or `p` $\in L($`r2`$)$ or both. Let us suppose `p` $\in L($`r2`$)$. (The proof is similar for `r1`.)

By the Induction Hypothesis for `r2`, `(match r2 cs k)` $\cong$ `true`.

Therefore:

| | | |
|---|---|---|
| | `(match (Plus(r1,r2)) cs k)` | |
| $\cong$ | `(match r1 cs k) orelse (match r2 cs k)` | [by the `Plus` clause in `match`] |
| $\cong$ | `true` | [termination of `match`, IH as stated above] |

That completes the completeness part of the proof, and thus the proof for `Plus`.