15-150
Fall 2024

Dilsun Kaynar

LECTURE 14

Regular Expressions

Motivating examples

Validate URL:

www.<*either* cs *or* ece>.<*either* cmu *or* pitt>.edu

Find each line that contains only letters and single spaces:

`grep "[A-Za-z ]*"` *filename*

Today

- Regular expressions
- Regular languages
- Matcher
- Correctness
  - Proof-directed debugging
  - Termination
  - Soundness and completeness

Hierarchy of Computer Languages

Class of Languages	Recognizer	Applications
Unrestricted	Turing machines	General computational questions
Context-sensitive	Linear-bounded automata	Some simple type-checking
Context-free	Non-deterministic automata with one stack	Syntax checking
Regular	Finite automata	Tokenization

Hierarchy of Computer Languages

Class of Languages	Recognizer	Applications
Unrestricted	Turing machines	General computational questions
Context-sensitive	Linear-bounded automata	Some simple type-checking
Context-free	Non-deterministic automata with one stack	Syntax checking
Regular	Finite automata	Tokenization

Excursions from my office

"c"means going to the coffee machine and coming back

"p" means going to the printer and coming back

"m" means going to a meeting and coming back

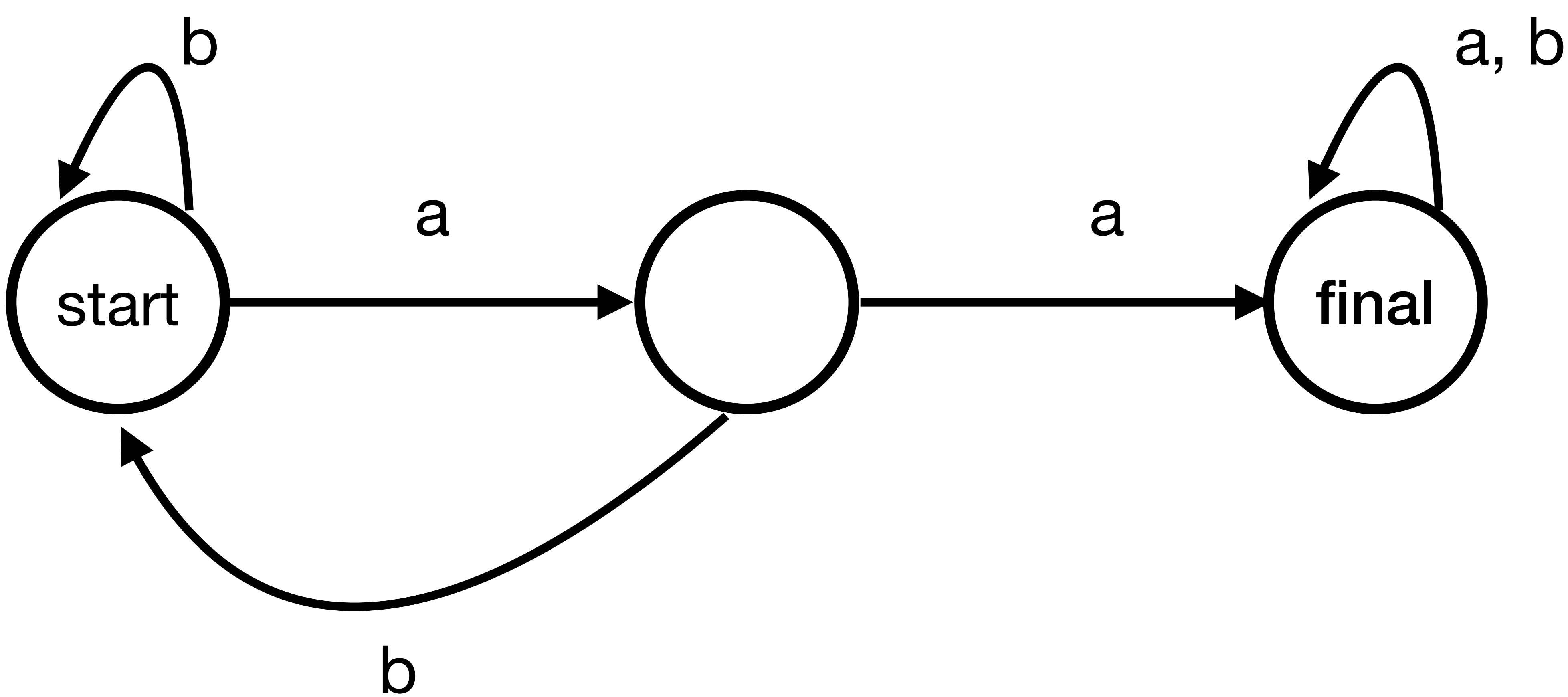
{cpmc, cccc, mcmmm, …}

Succinct way to describe my excursions in a given day?

c^* Arbitrary number of trips to coffee machine

$(c+p)^*\ m$ Arbitrary number of trips to coffee machine or printer, followed by a meeting

A finite automaton



This automaton accepts all strings over the alphabet {a,b} that contain at least two consecutive “a”s.

Notation and Definitions

Σ is an alphabet of characters.

Example: $\Sigma = \{a,b\}$
(In SML, #"a" : char)

Σ^* means the set of all finite-length strings over alphabet Σ.

Example: aabba in $\{a,b\}^*$
(In SML, "aabba": string)

ε is the empty string, containing no characters.

(In SML, "": string)

Notation and Definitions

Σ is an alphabet of characters.

A language over Σ is a subset of Σ^*.

Regular expressions

A regular expression over an alphabet Σ is one of the following:

$$a \mid 0 \mid 1 \mid r_1\, r_2 \mid r_1 + r_2 \mid r^*$$

letter in Σ | special symbols | concatenation | alternation | Kleene star

We use parantheses without regarding them as a part of the language.

$L(r)$: Language of a regular expression

$L(a) = \{a\}$

$L(0) = \{\}$

$L(1) = \{\varepsilon\}$

$L(r_1 r_2) = \{s_1 s_2 \mid s_1 \in L(r_1) \text{ and } s_2 \in L(r_2)\}$

$L(r_1 + r_2) = \{s \mid s \in L(r_1) \text{ or } s \in L(r_2)\}$

$L(r^*) = \{s_1 \ldots s_n \mid n \geq 0 \text{ with } s_i \in L(r) \text{ for } 0 \leq i \leq n\}$

includes ε for $n = 0$

Alternatively,

$L(r^*) = \{\varepsilon\} \cup \{s_1 s_2 \mid s_1 \in L(r) \text{ and } s_2 \in L(r^*)\}$

A language L is regular if $L = L(r)$ for some regular expression r.

Examples

$L(a) = \{a\}$
$L(0) = \{\}$
$L(1) = \{\varepsilon\}$
$L(r_1\ r_2) = \{s_1\ s_2 \mid s_1 \in L(r_1) \text{ and } s_2 \in L(r_2)\}$
$L(r_1 + r_2) = \{s \mid s \in L(r_1) \text{ or } s \in L(r_2)\}$
$L(r^*) = \{s_1 \dots s_n \mid n \geq 0 \text{ with } s_i \in L(r) \text{ for } 0 \leq i \leq n\}$
Alternatively,
$L(r^*) = \{\varepsilon\} \cup \{s_1 s_2 \mid s_1 \in L(r) \text{ and } s_2 \in L(r^*)\}$

Assume $\Sigma = \{a,b\}$
What is the language for each of the following regular expressions?

a
aa
$(a+b)^*$
$(a+b)^*aa(a+b)^*$
$(a+1)(b+ba)^*$

$L(a) = \{a\}$
$L(aa) = \{aa\}$
$L((a+b)^*) = \Sigma^*$ (set of all strings over Σ)
$L((a+b)^*aa(a+b)^*)$ = set of strings with at least two consecutive “a”s.
$L((a+1)(b+ba)^*)$ = set of strings without two consecutive “a”s.

Examples

Assume $\Sigma = \{a,b\}$
All of the regular expressions below generate the same language:

L(ab+b*ab)
L((1+b*)ab
L((1+bb*)ab)
L(b*ab)
L(b*ab+0)

All strings Σ^* consisting of 0 or more “b”s followed by ab (and nothing thereafter)

Representing regular expressions

$a \mid 0 \mid 1 \mid r_1\, r_2 \mid r_1 + r_2 \mid r^*$

```
datatype regexp = Char of char
                | Zero
                | One
                | Times of regexp * regexp
                | Plus of regexp * regexp
                | Star of regexp
```

accept and match

```
(* accept : regexp -> string -> bool

   REQUIRES:  true
   ENSURES:   (accept r s)  returns true, if s ∈ L(r);
              (accept r s)  returns false, otherwise.
*)
```

```
(* accept : regexp -> string -> bool

   REQUIRES:  true
   ENSURES:   (accept r s)  ≅ true, if s ∈ L(r);
              (accept r s)  ≅ false, otherwise.
*)
```

Consider **regular expression** **r = (a + ab) (a + b)**

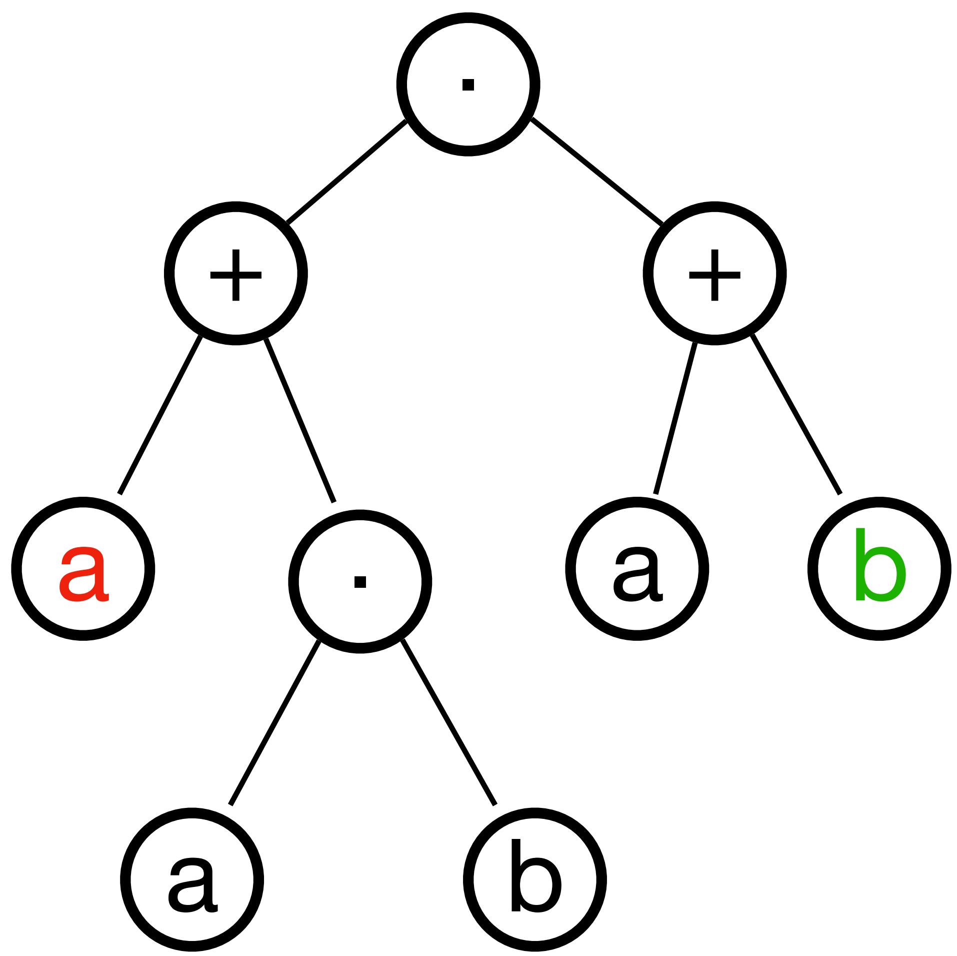
What is the language of r, i.e., what is L(r)? {aa,ab,aba,abb}

What does accept return when we apply it to r and "aba" ?

How do we split "aba"?

aba

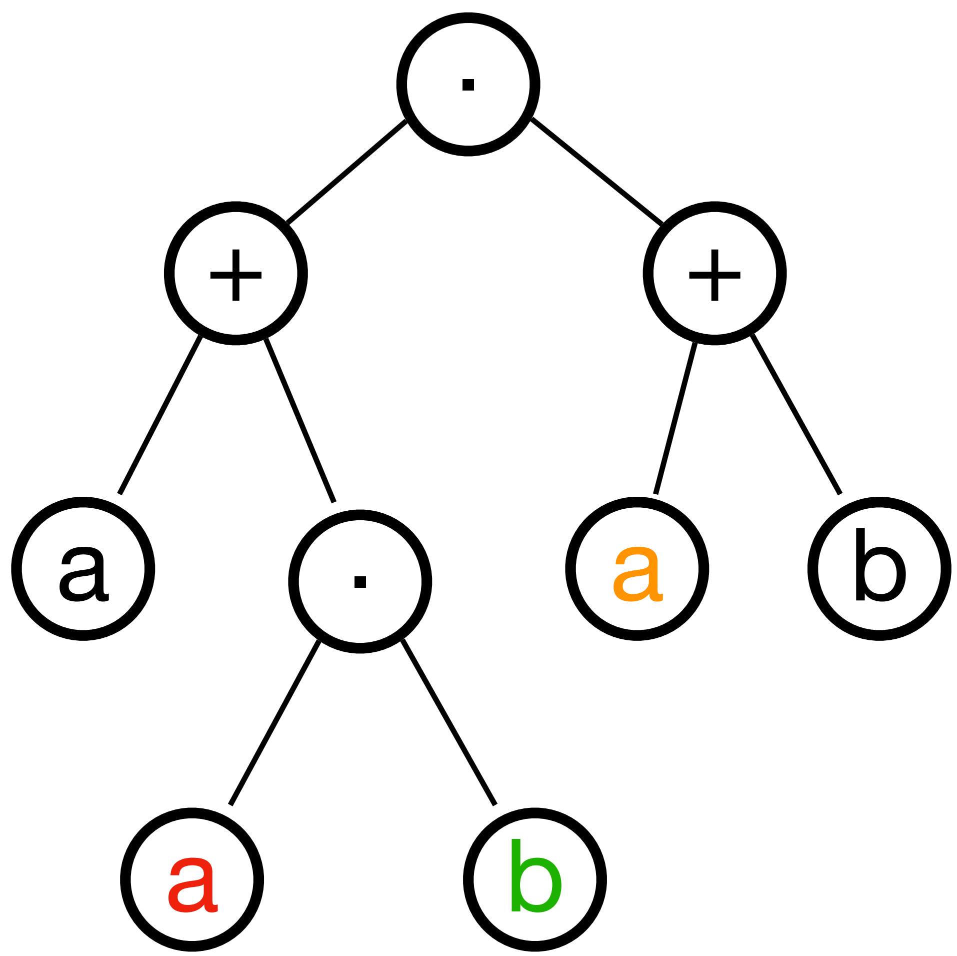
(a + ab) (a + b)



need to backtrack

aba

$(a + ab)\,(a + b)$



accept and match

```
(* accept : regexp -> string -> bool

   REQUIRES:  true
   ENSURES:   (accept r s)  ≅ true, if s ∈ L(r);
              (accept r s)  ≅ false, otherwise.
*)


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

   REQUIRES: k is total.
   ENSURES:  (match r cs k)  ≅ 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)  ≅ false, otherwise.
*)

fun accept r s =   match r (String.explode s) List.null
```

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

$L(a) = \{a\}$
$L(0) = \{\}$
$L(1) = \{\varepsilon\}$
$L(r_1\ r_2) = \{s_1\ s_2 \mid s_1 \in L(r_1) \text{ and } s_2 \in L(r_2)\}$
$L(r_1 + r_2) = \{s \mid s \in L(r_1) \text{ or } s \in L(r_2)\}$
$L(r^*) = \{s_1 \ldots s_n \mid n \geq 0 \text{ with } s_i \in L(r) \text{ for } 0 \leq i \leq n\}$
Alternatively,
$L(r^*) = \{\varepsilon\} \cup \{s_1 s_2 \mid s_1 \in L(r) \text{ and } s_2 \in L(r^*)\}$

```
fun match (Char(a)) cs k =  (case cs of
                               [ ] => false
                             | (c::cs') => (a=c) andalso k(cs'))
  | match (Zero) _ _ =   false
  | match (One) cs k =  k(cs)
  | match (Times (r1,r2)) cs k =  match r1 cs (fn cs' => match r2 cs' k)
  | match (Plus (r1,r2)) cs k =   match r1 cs k orelse match r2 cs k
  | match (Star(r)) cs k =   k(cs) orelse match r cs (fn cs' => match Star(r) cs' k)
```

(match r cs k) ≅ true, if cs can be split as cs ≅ p@s
with p representing a string in L(r) and k(s) evaluating to true

Termination

Theorem :

For all values r, cs, k (of the correct type), with k total, match r cs k reduces to a value.

Think about structural induction on r and the case

match (Star(r)) cs k = k(cs) **orelse** match r cs (**fn** cs' => match Star(r) cs' k)

Termination

Theorem :

For all values r, cs, k (of the correct type), with k total, match r cs k reduces to a value.

Think about structural induction on r and the case

match (Star(r)) cs k = k(cs) **orelse** match r cs (**fn** cs' => match Star(r) cs' k)

In IH, we may assume match r cs (fn cs' => match Star(r) cs' k) reduces to a value when (fn cs' => match Star(r) cs' k) is total. But do we know that it is total?

Circular argument!

$L(a) = \{a\}$
$L(0) = \{\}$
$L(1) = \{\varepsilon\}$
$L(r_1\ r_2) = \{s_1\ s_2 \mid s_1 \in L(r_1) \text{ and } s_2 \in L(r_2)\}$
$L(r_1 + r_2) = \{s \mid s \in L(r_1) \text{ or } s \in L(r_2)\}$
$L(r^*) = \{s_1 \ldots s_n \mid n \geq 0 \text{ with } s_i \in L(r) \text{ for } 0 \leq i \leq n\}$
Alternatively,
$L(r^*) = \{\varepsilon\} \cup \{s_1 s_2 \mid s_1 \in L(r) \text{ and } s_2 \in L(r^*)\}$

```
fun match (Char(a)) cs k =  (case cs of
                                  [ ] => false
                                | (c::cs') => (a=c) andalso k(cs'))
  | match (Zero) _ _ =   false
  | match (One) cs k =  k(cs)
  | match (Times (r1,r2)) cs k =  match r1 cs (fn cs' => match r2 cs' k)
  | match (Plus (r1,r2)) cs k =   match r1 cs k orelse match r2 cs k
  | match (Star(r)) cs k =   k(cs) orelse match r cs (fn cs' => match Star(r) cs' k)
```

may lead to an infinite loop

Example: match(Star(One)) ["#a"] List.null

List.null ["#a"] is false and match One cs k' will pass cs to k'

Two ways to fix the problem

- Change code
- Change specification to require that the input regular expression be in *standard form*
    - If Star(r) appears in the regular expression then ε is not in the language of r.

match function

```
fun match (Char(a)) cs k =  (case cs of
                                 [ ] => false
                               | (c::cs') => (a=c) andalso k(cs'))
    | match (Zero) _ k =  false
    | match (One) cs k = k(cs)
    | match (Times (r1,r2)) cs k =  match r1 cs (fn cs' => match r2 cs' k)
    | match (Plus (r1,r2)) cs k =  match r1 cs k orelse match r2 cs k
    | match (Star(r)) cs k = k(cs) orelse match r cs  (fn cs' => match Star(r) cs' k)
```

Or we could require that r be in standard form

A regular expression r is in *standard form* if and only if for any subexpression Star(r') of r, L(r') does not contain the empty string.

Sketch of a Proof of Correctness

- Prove termination: show that `(match r cs k)` returns a value for all arguments `r, cs, k` satisfying REQUIRES (We will assume this).

- Prove soundness and completeness (We will do this assuming termination and write out one case).

Soundness and Completenes (assuming termination)

ENSURES: (match r cs k) ≅ true, if cs ≅ p@s,
with p ∈ L(r) and k(s) ≅ true;
(match r cs k) ≅ false, otherwise

Given termination, we can rephrase the spec as follows:

ENSURES: (match r cs k) ≅ true if and only if there exist p, s such that
cs ≅ p@s, p ∈ L(r) and k(s) ≅ true

Theorem:

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

(match r cs k) $\cong$ true

if and only if

there exist p, s such that

cs $\cong$ p@s, $p \in L(r)$ and k(s) $\cong$ true

We are assuming termination as a lemma.

Proof: By structural induction on r

Base cases: Zero, One, Char (a) for every a: char

Inductive cases: Plus (r_1, r_2), Times (r_1,r_2), Star (r)

Theorem:

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

`(match r cs k)` $\cong$ `true`

if and only if

there exist `p`, `s` such that

`cs` $\cong$ `p@s`, `p` $\in$ `L(r)` and `k(s)` $\cong$ `true`

We are assuming termination as a lemma.

Inductive case: `r = Plus` (r_1, r_2) for some r_1 and r_2

IH: For i = 1,2, for all values `cs: char list, k: char list -> bool`, with `k` total `(match` r_i `cs k)` $\cong$ `true` if and only if there exist `p`, `s` such that `cs` $\cong$ `p@s`, `p` $\in$ `L(`r_i`)` and `k(s)` $\cong$ `true`

NTS: For all values `cs: char list, k: char list -> bool`, with `k` total `(match (Plus` (r_1, r_2)`) cs k)` $\cong$ `true` if and only if there exist `p`, `s` such that `cs` $\cong$ `p@s`, `p` $\in$ `L(Plus` (r_1, r_2)`)` and `k(s)` $\cong$ `true`.

Soundness

Inductive case: $r = \text{Plus}\ (r_1, r_2)$ for some r_1 and r_2

IH: For i = 1,2, for all values cs: char list, k: char list -> bool, with k total $(\text{match}\ r_i\ \text{cs k}) \cong \text{true}$ if and only if there exist p, s such that $\text{cs} \cong \text{p@s}$, $p \in L(r_i)$ and $k(s) \cong \text{true}$

NTS: For all values cs: char list, k: char list -> bool, with k total $(\text{match}\ (\text{Plus}\ (r_1, r_2))\ \text{cs k})) \cong \text{true}$ if and only if there exist p, s such that $\text{cs} \cong \text{p@s}$, $p \in L(\text{Plus}\ (r_1, r_2))$ and $k(s) \cong \text{true}$.

(Part 1): Suppose $(\text{match}\ (\text{Plus}\ (r_1, r_2))\ \text{cs k}) \cong \text{true}$

NTS: There exist p, s such that
such that $\text{cs} \cong \text{p@s}$, $p \in L(\text{Plus}\ (r_1, r_2))$ and $k(s) \cong \text{true}$.

$\text{true} \cong (\text{match}\ (\text{Plus}\ (r_1, r_2))\ \text{cs k})$ [Assumption]

$\cong (\text{match}\ r_1\ \text{cs k})$ **orelse** $(\text{match}\ r_2\ \text{cs k})$ [Plus]

One or both arguments to **orelse** must be true. Let's suppose the first one.

By IH for r_1 there exist p, s such that $\text{cs} \cong \text{p@s}$, $p \in L(r_1)$ and $k(s) \cong \text{true}$.

$p \in L(\text{Plus}\ (r_1, r_2))$ by language definition for Plus.

Completeness

Inductive case: $r = \text{Plus}(r_1, r_2)$ for some r_1 and r_2

IH: For i = 1,2, for all values cs: char list, k: char list -> bool, with k total (match r_i cs k) $\cong$ true if and only if there exist p, s such that cs $\cong$ p@s, p $\in L(r_i)$ and k(s) $\cong$ true

NTS: For all values cs: char list, k: char list -> bool, with k total (match (Plus (r_1, r_2)) cs k) $\cong$ true if and only if there exist p, s such that cs $\cong$ p@s, p $\in$ L(Plus (r_1, r_2)) and k(s) $\cong$ true.

(Part 2): Suppose cs $\cong$ p@s, p $\in$ L(Plus (r_1, r_2)) and k(s) $\cong$ true.

NTS: (match (Plus (r_1, r_2)) cs k) $\cong$ true

(match (Plus (r_1, r_2)) cs k)

$\cong$ (match r_1 cs k) **orelse** (match r_2 cs k) [Plus]

By supposition, there exist p, s such that cs $\cong$ p@s, p $\in$ L(Plus (r_1, r_2)) and k(s) $\cong$ true. By language definition for Plus, p $\in L(r_1)$ and/or p $\in L(r_2)$.
If p $\in L(r_1)$, then (match r_1 cs k) $\cong$ true, by IH for r_1.
Otherwise, (match r_1 cs k) $\cong$ false by termination, p $\in L(r_2)$,
and (match r_2 cs k) $\cong$ true by IH for r_2.