Strings and Regular Expressions

Stephen A. Edwards

Columbia University

Spring 2023

[^]*?@[^]*?\.[^]*

Alphabets, Strings, and the Empty String

An *alphabet* Σ is a finite set of symbols.
Strings over Σ are members of Σ^*, defined by

$$\frac{}{\text{``''} \in \Sigma^*}\text{ epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\text{ char}$$

Judgments: $c \in \Sigma$ "character in Σ" $s \in \Sigma^*$ "sequence of zero or more characters"
Variables: c "character" s "sequence"
Symbols: " " "start and end of a string"

If $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}\}$,

"" The empty string

$$\frac{}{\text{``''} \in \Sigma^*}\text{ epsilon}$$

Alphabets, Strings, and the Empty String

An *alphabet* Σ is a finite set of symbols.
Strings over Σ are members of Σ^*, defined by

$$\frac{}{\text{“”} \in \Sigma^*}\,\text{epsilon} \qquad\qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\,\text{char}$$

Judgments: $c \in \Sigma$ “character in Σ” $s \in \Sigma^*$ “sequence of zero or more characters”
Variables: c “character” s “sequence”
Symbols: “ ” “start and end of a string”

If $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}\}$,

“”	The empty string
“a”	The string consisting of just “a”

$$\frac{\mathsf{a} \in \Sigma \quad \dfrac{}{\text{“”} \in \Sigma^*}\,\text{epsilon}}{\text{“a”} \in \Sigma^*}\,\text{char}$$

← Choose s to be the empty sequence

Alphabets, Strings, and the Empty String

An *alphabet* Σ is a finite set of symbols.

Strings over Σ are members of Σ^*, defined by

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad\qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

Judgments: $c \in \Sigma$ “character in Σ” $s \in \Sigma^*$ “sequence of zero or more characters”

Variables: c “character” s “sequence”

Symbols: “ ” “start and end of a string”

If $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}\}$,

“”	The empty string
“a”	The string consisting of just “a”
“ba”	The string consisting of “b” followed by “a”

$$\frac{\mathsf{b} \in \Sigma \qquad \dfrac{\mathsf{a} \in \Sigma \qquad \dfrac{}{\text{“”} \in \Sigma^*}\ \text{epsilon}}{\text{“a”} \in \Sigma^*}\ \text{char}}{\text{“ba”} \in \Sigma^*}\ \text{char}$$

← Prepend characters from right to left

Alphabets, Strings, and the Empty String

An *alphabet* Σ is a finite set of symbols.

Strings over Σ are members of Σ^*, defined by

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

Judgments: $c \in \Sigma$ "character in Σ" $s \in \Sigma^*$ "sequence of zero or more characters"

Variables: c "character" s "sequence"

Symbols: " " "start and end of a string"

If $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$,

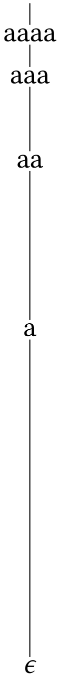
""	The empty string
"a"	The string consisting of just "a"
"ba"	The string consisting of "b" followed by "a"
"aba"	The string "a" followed by "b" followed by "a"
"abcd"	The four-letter string "abcd"
"sphinxofblackquartzjudgemyvow"	A pangram with only a, o, and u repeated

Alphabets, Strings, and the Empty String

An *alphabet* Σ is a finite set of symbols.
Strings over Σ are members of Σ^*, defined by

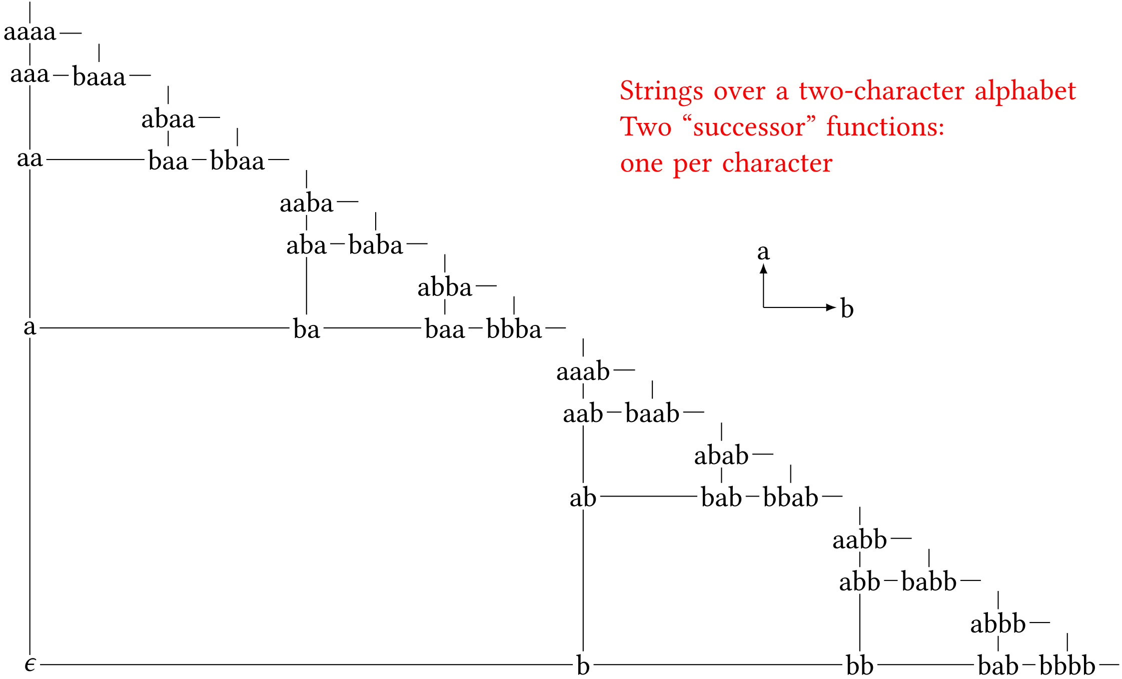
$$\frac{}{\text{``''} \in \Sigma^*}\text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\text{char}$$

```
infixr 5  :                  -- Not legal Haskell    a : b : c = a : (b : c)
data [a] = []                -- [] is the empty list   "a" is a type variable
         | a : [a]           -- : is the list "cons" or prepend operator

type String = [Char]         -- In Haskell, strings are lists of characters

"Hello"                      -- shorthand for
'H' : 'e' : 'l' : 'l' : 'o' : []
```



A visualization of the Peano encoding of natural numbers $\epsilon = 0$ a = successor

Strings over a two-character alphabet
Two “successor” functions:
one per character



Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ab”?

“ab” = “ab”

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ab”?

$$\frac{\mathsf{a} \in \Sigma \qquad \text{“b”} = \text{“b”}}{\text{“ab”} = \text{“ab”}}\ \text{equal}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ab”?

$$\frac{\mathsf{a} \in \Sigma \qquad \dfrac{\mathsf{b} \in \Sigma \quad \text{“”} = \text{“”}}{\text{“b”} = \text{“b”}}\ \text{equal}}{\text{“ab”} = \text{“ab”}}\ \text{equal}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{``''} = \text{``''}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} = \text{``}s_2\text{''}}{\text{``}cs_1\text{''} = \text{``}cs_2\text{''}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”
Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ab”?

$$\frac{\mathsf{a} \in \Sigma \qquad \dfrac{\mathsf{b} \in \Sigma \qquad \dfrac{}{\text{``''} = \text{``''}}\ \text{equal-epsilon}}{\text{``b''} = \text{``b''}}\ \text{equal}}{\text{``ab''} = \text{``ab''}}\ \text{equal}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{``''} = \text{``''}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} = \text{``}s_2\text{''}}{\text{``}cs_1\text{''} = \text{``}cs_2\text{''}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ac”?

“ab” = “ac”

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ac”?

$$\frac{\mathsf{a} \in \Sigma \quad \text{“b”} = \text{“c”}}{\text{“ab”} = \text{“ac”}}\ \text{equal}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Is “ab” = “ac”?

$$\frac{\mathsf{a} \in \Sigma \quad \dfrac{?}{\text{“b”} = \text{“c”}}\,?}{\text{“ab”} = \text{“ac”}}\ \text{equal}$$

← We are stuck: the *equal* rule requires identical initial characters

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{“”} = \text{“”}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} = \text{“}s_2\text{”}}{\text{“}cs_1\text{”} = \text{“}cs_2\text{”}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

Additional Theorems

Reflexive: For any $s \in \Sigma^*$, $s = s$

Symmetric: For any $s_1, s_2 \in \Sigma^*$ with $s_1 = s_2$, $s_2 = s_1$.

Transitive: For any $s_1, s_2, s_3 \in \Sigma^*$ with $s_1 = s_2$ and $s_2 = s_3$, $s_1 = s_3$.

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Equality

$$\frac{}{\text{``''} = \text{``''}}\ \text{equal-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} = \text{``}s_2\text{''}}{\text{``}cs_1\text{''} = \text{``}cs_2\text{''}}\ \text{equal}$$

Judgments: “s_1” = “s_2” “Strings “s_1” and “s_2” are equal”

Variables: s_1, s_2 “character sequence” c “character”

```
(==)                 :: [Char] -> [Char] -> Bool
     [] == []        = True                    -- equal-epsilon
c1 : s1 == c2 : s2   = c1 == c2 && s1 == s2    -- equal
      _ == _         = False                   -- default case

data  [a]  =  [] | a : [a]  deriving (Eq, Ord) -- Default implementation of Eq
```

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} +\!+ \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} +\!+ \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} +\!+ \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

Judgments: “s_1” ++ “s_2” = “s_3” “Concatenating strings “s_1” and “s_2” gives string “s_3””

Variables: s, s_1, s_2, s_3 “character sequence” c “character”

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} +\!+ \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} +\!+ \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} +\!+ \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

Is “ab” ++ “cde” = “abcde”?

“ab” ++ “cde” = “abcde”

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{``}s\text{''} \in \Sigma^*}{\text{``''} +\!\!+ \text{``}s\text{''} = \text{``}s\text{''}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} +\!\!+ \text{``}s_2\text{''} = \text{``}s_3\text{''}}{\text{``}cs_1\text{''} +\!\!+ \text{``}s_2\text{''} = \text{``}cs_3\text{''}}\ \text{concat}$$

Is “ab” ++ “cde” = “abcde”?

$$\frac{\text{a} \in \Sigma \qquad \text{``b''} +\!\!+ \text{``cde''} = \text{``bcde''}}{\text{``ab''} +\!\!+ \text{``cde''} = \text{``abcde''}}\ \text{concat}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} \mathbin{++} \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} \mathbin{++} \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} \mathbin{++} \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

Is “ab” ++ “cde” = “abcde”?

$$\frac{a \in \Sigma \qquad \dfrac{b \in \Sigma \qquad \text{“”} \mathbin{++} \text{“cde”} = \text{“cde”}}{\text{“b”} \mathbin{++} \text{“cde”} = \text{“bcde”}}\ \text{concat}}{\text{“ab”} \mathbin{++} \text{“cde”} = \text{“abcde”}}\ \text{concat}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{``}s\text{''} \in \Sigma^*}{\text{``''} \mathbin{++} \text{``}s\text{''} = \text{``}s\text{''}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}s_3\text{''}}{\text{``}cs_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}cs_3\text{''}}\ \text{concat}$$

Is “ab” ++ “cde” = “abcde”?

$$\dfrac{\mathsf{a} \in \Sigma \quad \dfrac{\mathsf{b} \in \Sigma \quad \dfrac{\text{``cde''} \in \Sigma^*}{\text{``''} \mathbin{++} \text{``cde''} = \text{``cde''}}\ \text{concat-epsilon}}{\text{``b''} \mathbin{++} \text{``cde''} = \text{``bcde''}}\ \text{concat}}{\text{``ab''} \mathbin{++} \text{``cde''} = \text{``abcde''}}\ \text{concat}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} \mathbin{++} \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} \mathbin{++} \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} \mathbin{++} \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

Is “ab” ++ “cde” = “abcde”?

$$\dfrac{\mathsf{a} \in \Sigma \quad \dfrac{\mathsf{b} \in \Sigma \quad \dfrac{\dfrac{\mathsf{c} \in \Sigma \quad \dfrac{\mathsf{d} \in \Sigma \quad \dfrac{\mathsf{e} \in \Sigma \quad \dfrac{}{\text{“”} \in \Sigma^*}\ \text{epsilon}}{\text{“e”} \in \Sigma^*}\ \text{char}}{\text{“de”} \in \Sigma^*}\ \text{char}}{\text{“cde”} \in \Sigma^*}\ \text{char}}{\text{“”} \mathbin{++} \text{“cde”} = \text{“cde”}}\ \text{concat-epsilon}}{\text{“b”} \mathbin{++} \text{“cde”} = \text{“bcde”}}\ \text{concat}}{\text{“ab”} \mathbin{++} \text{“cde”} = \text{“abcde”}}\ \text{concat}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\,\text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\,\text{char}$$

String Concatenation

$$\frac{\text{``}s\text{''} \in \Sigma^*}{\text{``''} \mathbin{++} \text{``}s\text{''} = \text{``}s\text{''}}\,\text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}s_3\text{''}}{\text{``}cs_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}cs_3\text{''}}\,\text{concat}$$

Theorem: “” is also a right identity for concatenation

Assume “s” $\in \Sigma^*$. From the definition of Σ^*, s must be of the form $c_1 c_2 \cdots c_n$ where $c_i \in \Sigma$.

$$\dfrac{c_1 \in \Sigma \qquad \begin{matrix} \dfrac{c_n \in \Sigma \qquad \dfrac{\dfrac{}{\text{``''} \in \Sigma^*}\,\text{epsilon}}{\text{``''} \mathbin{++} \text{``''} = \text{``''}}\,\text{concat-epsilon}}{\text{``}c_n\text{''} \mathbin{++} \text{``''} = \text{``}c_n\text{''}}\,\text{concat} \\ \vdots \\ \text{``}c_2 \cdots c_n\text{''} \mathbin{++} \text{``''} = \text{``}c_2 \cdots c_n\text{''} \end{matrix}}{\text{``}c_1 c_2 \cdots c_n\text{''} \mathbin{++} \text{``''} = \text{``}c_1 c_2 \cdots c_n\text{''}}\,\text{concat}$$

Alphabets, Strings, and the Empty String

$$\frac{}{\text{``''} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s\text{''} \in \Sigma^*}{\text{``}cs\text{''} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{``}s\text{''} \in \Sigma^*}{\text{``''} \mathbin{++} \text{``}s\text{''} = \text{``}s\text{''}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{``}s_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}s_3\text{''}}{\text{``}cs_1\text{''} \mathbin{++} \text{``}s_2\text{''} = \text{``}cs_3\text{''}}\ \text{concat}$$

Theorem: String concatenation is a function

If "s_1" ++ "s_2" = "s_3" and "s_1" ++ "s_2" = "s_4" then "s_3" = "s_4".

Alphabets, Strings, and the Empty String

$$\frac{}{\text{“”} \in \Sigma^*}\ \text{epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s\text{”} \in \Sigma^*}{\text{“}cs\text{”} \in \Sigma^*}\ \text{char}$$

String Concatenation

$$\frac{\text{“}s\text{”} \in \Sigma^*}{\text{“”} +\!\!+ \text{“}s\text{”} = \text{“}s\text{”}}\ \text{concat-epsilon} \qquad \frac{c \in \Sigma \quad \text{“}s_1\text{”} +\!\!+ \text{“}s_2\text{”} = \text{“}s_3\text{”}}{\text{“}cs_1\text{”} +\!\!+ \text{“}s_2\text{”} = \text{“}cs_3\text{”}}\ \text{concat}$$

```
infixr 5  :                   -- Not legal Haskell    a : b : c = a : (b : c)
data [a] = [] | a : [a]       -- Not legal Haskell   [] is empty list   : is cons
type String = [Char]

infixr 5  ++                      -- a ++ b ++ c = a ++ (b ++ c)
(++) :: [a] -> [a] -> [a]         -- Concatenate two lists
(++) []      s  = s               -- concat-epsilon
(++) (c:s1) s2 = c : s1 ++ s2     -- concat     c : (s1 ++ s2)
```

Regular Expressions

A character matches itself	"a" ∼ a "x" ∼ x
Juxtaposition matches a sequence	"abc" ∼ abc
| indicates a choice	"ab" ∼ ab|bc "bc" ∼ ab|bc
* means "zero or more"	"", "a", "aa", "aaa", "aaaa", "aaaaa", ... ∼ a*

Regular Expressions

A character matches itself	“a” $\sim$ a	“x” $\sim$ x
Juxtaposition matches a sequence	“abc” $\sim$ abc	

$$\frac{}{\text{“”} \sim \epsilon}\,\text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\,\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\,\text{seq}$$

Judgments: $s \sim r$ “string s matches regular expression r”

Variables: c character r regular expression s string

Symbols: ϵ “ ” | * a b c d…

Regular Expressions

A character matches itself "a" $\sim$ a "x" $\sim$ x

Juxtaposition matches a sequence "abc" $\sim$ abc

$$\frac{}{\text{“”} \sim \epsilon}\,\text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\,\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\; s_2 = s_3}{s_3 \sim r_1 r_2}\,\text{seq}$$

"abc" $\sim$ abc?

"abc" $\sim$ abc

Regular Expressions

A character matches itself $\text{"a"} \sim \mathtt{a} \quad \text{"x"} \sim \mathtt{x}$

Juxtaposition matches a sequence $\text{"abc"} \sim \mathtt{abc}$

$$\frac{}{\text{""} \sim \epsilon}\,\text{epsilon} \qquad \frac{c \in \Sigma}{\text{"}c\text{"} \sim c}\,\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\, s_2 = s_3}{s_3 \sim r_1 r_2}\,\text{seq}$$

"abc" ∼ abc?

$$\frac{\text{"a"} \sim \mathtt{a} \qquad \text{"bc"} \sim \mathtt{bc} \qquad \text{"a"} +\!\!+ \text{"bc"} = \text{"abc"}}{\text{"abc"} \sim \mathtt{abc}}\,\text{seq}$$

Regular Expressions

A character matches itself $\text{“a”} \sim \mathsf{a}$ $\quad \text{“x”} \sim \mathsf{x}$

Juxtaposition matches a sequence $\text{“abc”} \sim \mathsf{abc}$

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

“abc” ∼ abc?

$$\frac{\dfrac{a \in \Sigma}{\text{“a”} \sim \mathsf{a}}\ \text{char} \quad \dfrac{\text{“b”} \sim \mathsf{b} \quad \text{“c”} \sim \mathsf{c} \quad \text{“b”} \mathbin{++} \text{“c”} = \text{“bc”}}{\text{“bc”} \sim \mathsf{bc}}\ \text{seq} \quad \dfrac{\vdots}{\text{“a”} \mathbin{++} \text{“bc”} = \text{“abc”}}\ \text{concat}}{\text{“abc”} \sim \mathsf{abc}}\ \text{seq}$$

Regular Expressions

A character matches itself $\text{“a”} \sim \mathsf{a} \quad \text{“x”} \sim \mathsf{x}$

Juxtaposition matches a sequence $\text{“abc”} \sim \mathsf{abc}$

$$\frac{}{\text{“”} \sim \epsilon}\text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\text{seq}$$

“abc” ∼ abc?

$$\frac{\dfrac{a \in \Sigma}{\text{“a”} \sim \mathsf{a}}\text{char} \quad \dfrac{\dfrac{b \in \Sigma}{\text{“b”} \sim \mathsf{b}}\text{char} \quad \dfrac{c \in \Sigma}{\text{“c”} \sim \mathsf{c}}\text{char} \quad \dfrac{\vdots}{\text{“b”} \mathbin{++} \text{“c”} = \text{“bc”}}\text{concat}}{\text{“bc”} \sim \mathsf{bc}}\text{seq} \quad \dfrac{\vdots}{\text{“a”} \mathbin{++} \text{“bc”} = \text{“abc”}}\text{concat}}{\text{“abc”} \sim \mathsf{abc}}\text{seq}$$

Regular Expressions

A character matches itself   “a” ∼ a   “x” ∼ x

Juxtaposition matches a sequence   “abc” ∼ abc

$$\frac{}{\text{“”} \sim \epsilon}\,\text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\,\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\,\text{seq}$$

“abc” ∼ abc?   What should we do about ambiguity?

$$\frac{\dfrac{a \in \Sigma}{\text{“a”} \sim \mathsf{a}}\,\text{char} \quad \dfrac{\dfrac{b \in \Sigma}{\text{“b”} \sim \mathsf{b}}\,\text{char} \quad \dfrac{c \in \Sigma}{\text{“c”} \sim \mathsf{c}}\,\text{char} \quad \dfrac{\vdots}{\text{“b”} \mathbin{++} \text{“c”} = \text{“bc”}}\,\text{concat}}{\text{“bc”} \sim \mathsf{bc}}\,\text{seq} \quad \dfrac{\vdots}{\text{“a”} \mathbin{++} \text{“bc”} = \text{“abc”}}\,\text{concat}}{\text{“abc”} \sim \mathsf{abc}}\,\text{seq}$$

$$\frac{\dfrac{\dfrac{a \in \Sigma}{\text{“a”} \sim \mathsf{a}}\,\text{char} \quad \dfrac{b \in \Sigma}{\text{“b”} \sim \mathsf{b}}\,\text{char} \quad \dfrac{\vdots}{\text{“a”} \mathbin{++} \text{“b”} = \text{“ab”}}\,\text{concat}}{\text{“ab”} \sim \mathsf{ab}}\,\text{seq} \quad \dfrac{c \in \Sigma}{\text{“c”} \sim \mathsf{c}}\,\text{char} \quad \dfrac{\vdots}{\text{“ab”} \mathbin{++} \text{“c”} = \text{“abc”}}\,\text{concat}}{\text{“abc”} \sim \mathsf{abc}}\,\text{seq}$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq} \qquad \frac{s \sim r}{s \sim (r)}\ \text{paren}$$

Judgments: $s \sim r$ “string s matches regular expression r”

Variables: c character r regular expression s string

Symbols: ϵ “ ” | * ( )

“abc” ∼ a(bc)?

$$\dfrac{\dfrac{a \in \Sigma}{\text{“a”} \sim \text{a}}\ \text{char} \qquad \text{“bc”} \sim \text{(bc)} \qquad \dfrac{\vdots}{\text{“a”} +\!\!+ \text{“bc”} = \text{“abc”}}\ \text{concat}}{\text{“abc”} \sim \text{a(bc)}}\ \text{seq}$$

Regular Expressions

$$\frac{}{\text{``''} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{``}c\text{''} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq} \qquad \frac{s \sim r}{s \sim (r)}\ \text{paren}$$

Judgments: $s \sim r$ "string s matches regular expression r"

Variables: c character r regular expression s string

Symbols: ϵ " " | * ( )

"abc" ∼ a(bc)?

$$\dfrac{\dfrac{a \in \Sigma}{\text{``a''} \sim \mathtt{a}}\ \text{char} \qquad \dfrac{\text{``bc''} \sim \mathtt{bc}}{\text{``bc''} \sim \mathtt{(bc)}}\ \text{paren} \qquad \dfrac{\vdots}{\text{``a''} \mathbin{++} \text{``bc''} = \text{``abc''}}\ \text{concat}}{\text{``abc''} \sim \mathtt{a(bc)}}\ \text{seq}$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq} \qquad \frac{s \sim r}{s \sim (r)}\ \text{paren}$$

Judgments: $s \sim r$ “string s matches regular expression r”

Variables: c character r regular expression s string

Symbols: ϵ “ ” | * ( )

“abc” ∼ a(bc)?

$$\dfrac{\dfrac{a \in \Sigma}{\text{“a”} \sim \texttt{a}}\ \text{char} \qquad \dfrac{\dfrac{\dfrac{b \in \Sigma}{\text{“b”} \sim \texttt{b}}\ \text{char} \quad \dfrac{c \in \Sigma}{\text{“c”} \sim \texttt{c}}\ \text{char} \quad \dfrac{\vdots}{\text{“b”} \mathbin{++} \text{“c”} = \text{“bc”}}\ \text{concat}}{\text{“bc”} \sim \texttt{bc}}\ \text{seq}}{\text{“bc”} \sim \texttt{(bc)}}\ \text{paren} \qquad \dfrac{\vdots}{\text{“a”} \mathbin{++} \text{“bc”} = \text{“abc”}}\ \text{concat}}{\text{“abc”} \sim \texttt{a(bc)}}\ \text{seq}$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

```haskell
-- An algebraic data type resolves ambiguity in RE structure (parentheses unneeded)
data RE = Epsilon    -- Epsilon/empty string
        | Ch Char    -- Single Character
        | Seq RE RE  -- Sequence, e.g., r1 r2

infix 5 ~~          -- Regular expression match operator (Haskell already uses ~)
(~~)                :: String -> RE -> Bool
""   ~~ Epsilon     = True      -- epsilon
[c1] ~~ Ch c2       = c1 == c2  -- char
s3   ~~ Seq r1 r2   =           -- What to do for seq? How do we choose s1, s2?
_    ~~ _           = False     -- default
```

Regular Expressions

$$\frac{}{\text{``''} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{``}c\text{''} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

```
ghci> import Data.List (inits, tails)
ghci> inits "abc"          -- All prefixes, shortest first
["","a","ab","abc"]

ghci> tails "abc"          -- All suffixes, longest first
["abc","bc","c",""]

ghci> :t zipWith           -- zipWith f [a1, a2, ...] [b1, b2, ...] = [f a1 b1, f a2 b2, ...]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]

ghci> :t or                -- Logical OR of a list of Booleans
or :: Foldable t => t Bool -> Bool
```

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\,\text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\,\text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\,\text{seq}$$

```haskell
import Data.List (inits, tails)
data RE = Epsilon | Ch Char | Seq RE RE

infix 5 ~~
(~~)                :: String -> RE -> Bool
""    ~~ Epsilon    = True         -- epsilon
[c1] ~~ Ch c2       = c1 == c2     -- char
s3    ~~ Seq r1 r2  = or $ zipWith testSplit (inits s3) (tails s3) -- seq
                      where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
_     ~~ _          = False        -- default
```

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1 | r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1 | r_2}\ \text{alt-r}$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1 | r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1 | r_2}\ \text{alt-r}$$

```haskell
data RE = Epsilon | Ch Char | Seq RE RE | Alt RE RE

(~~)                  :: String -> RE -> Bool
""   ~~ Epsilon       = True                              -- epsilon
[c1] ~~ Ch c2         = c1 == c2                          -- char
s3   ~~ Seq r1 r2     = or $ zipWith testSplit (inits s3) (tails s3) -- seq
                        where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
s    ~~ Alt r1 r2     = s ~~ r1 || s ~~ r2                -- alt-l and alt-r
_    ~~ _             = False                             -- default
```

Regular Expressions

$$\frac{}{\text{``''} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{``}c\text{''} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1 | r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1 | r_2}\ \text{alt-r} \qquad \frac{}{\text{``''} \sim r\star}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r\star \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r\star}\ \text{star-1}$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1|r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1|r_2}\ \text{alt-r} \qquad \frac{}{\text{“”} \sim r*}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r* \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r*}\ \text{star-1}$$

```haskell
data RE = Epsilon | Ch Char | Seq RE RE | Alt RE RE | Star RE
(~~)                 :: String -> RE -> Bool    -- HANGS TESTING “b” ~ ϵ*
""   ~~ Epsilon      = True                     -- epsilon
[c1] ~~ Ch c2        = c1 == c2                 -- char
s3   ~~ Seq r1 r2    = or $ zipWith testSplit (inits s3) (tails s3)
                       where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
s    ~~ Alt r1 r2    = s ~~ r1 || s ~~ r2       -- alt-l and alt-r
""   ~~ Star _       = True                     -- star-0
s3   ~~ Star r       = or $ zipWith testSplit (inits s3) (tails s3)
                       where testSplit s1 s2 = s1 ~~ r && s2 ~~ (Star r)
_    ~~ _            = False                    -- default
```

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1|r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1|r_2}\ \text{alt-r} \qquad \frac{}{\text{“”} \sim r*}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r* \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r*}\ \text{star-1}$$

```
data RE = Epsilon | Ch Char | Seq RE RE | Alt RE RE | Star RE
(~~)                :: String -> RE -> Bool    -- STILL HANGS ON “b” ∼ ϵ*
""   ~~ Epsilon     = True                     -- epsilon
[c1] ~~ Ch c2       = c1 == c2                 -- char
s3   ~~ Seq r1 r2   = or $ zipWith testSplit (inits s3) (tails s3)
                      where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
s    ~~ Alt r1 r2   = s ~~ r1 || s ~~ r2       -- alt-l and alt-r
""   ~~ Star _      = True                     -- star-0
s3   ~~ Star r      = or $ zipWith testSplit (inits s3) (tails s3)
                      where testSplit s1 s2 = s2 ~~ (Star r) && s1 ~~ r
_    ~~ _           = False                    -- default
```

Regular Expressions

$$\frac{}{\text{``''} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{``}c\text{''} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1 | r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1 | r_2}\ \text{alt-r} \qquad \frac{}{\text{``''} \sim r\star}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r\star \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r\star}\ \text{star-1}$$

“b” $\sim \epsilon\star$?

$$\text{``b''} \sim \epsilon\star$$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1 | r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1 | r_2}\ \text{alt-r} \qquad \frac{}{\text{“”} \sim r\star}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r\star \quad s_1 \mathbin{++} s_2 = s_3}{s_3 \sim r\star}\ \text{star-1}$$

“b” $\sim \epsilon\star$?

$$\frac{\text{“”} \sim \epsilon \qquad \text{“b”} \sim \epsilon\star \qquad \text{“”} \mathbin{++} \text{“b”} = \text{“b”}}{\text{“b”} \sim \epsilon\star}\ \text{star-1}$$

$s_1 = \text{“”}$ $\quad s_2 = \text{“b”}$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1|r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1|r_2}\ \text{alt-r} \qquad \frac{}{\text{“”} \sim r\star}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r\star \quad s_1 +\!+\ s_2 = s_3}{s_3 \sim r\star}\ \text{star-1}$$

“b” $\sim \epsilon\star$?

$$\frac{\dfrac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \text{“b”} \sim \epsilon\star \qquad \text{“”} +\!+\ \text{“b”} = \text{“b”}}{\text{“b”} \sim \epsilon\star}\ \text{star-1}$$

$s_1 = \text{“”}$ $\quad s_2 = \text{“b”}$

Regular Expressions

$$\frac{}{``" \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{``c" \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 ++ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1|r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1|r_2}\ \text{alt-r} \qquad \frac{}{``" \sim r\star}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r\star \quad s_1 ++ s_2 = s_3}{s_3 \sim r\star}\ \text{star-1}$$

"b" $\sim \epsilon\star$?

$$\frac{\frac{}{``" \sim \epsilon}\ \text{epsilon} \qquad \frac{\vdots}{``\text{b}" \sim \epsilon\star}\ \text{star-1} \qquad ``" ++ ``\text{b}" = ``\text{b}"}{``\text{b}" \sim \epsilon\star}\ \text{star-1}$$

$s_1 = ``"$ $\quad s_2 = ``\text{b}"$

Regular Expressions

$$\frac{}{\text{“”} \sim \epsilon}\ \text{epsilon} \qquad \frac{c \in \Sigma}{\text{“}c\text{”} \sim c}\ \text{char} \qquad \frac{s_1 \sim r_1 \quad s_2 \sim r_2 \quad s_1 +\!\!+\ s_2 = s_3}{s_3 \sim r_1 r_2}\ \text{seq}$$

$$\frac{s \sim r_1}{s \sim r_1|r_2}\ \text{alt-l} \qquad \frac{s \sim r_2}{s \sim r_1|r_2}\ \text{alt-r} \qquad \frac{}{\text{“”} \sim r{\star}}\ \text{star-0} \qquad \frac{s_1 \sim r \quad s_2 \sim r{\star} \quad s_1 +\!\!+\ s_2 = s_3 \quad s_1 \neq \text{“”}}{s_3 \sim r{\star}}\ \text{star-1}$$

```
(~~)                   :: String -> RE -> Bool
""    ~~ Epsilon       = True                              -- epsilon
[c1] ~~ Ch c2          = c1 == c2                          -- char
s3    ~~ Seq r1 r2     = or $ zipWith testSplit (inits s3) (tails s3)
                         where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
s     ~~ Alt r1 r2     = s ~~ r1 || s ~~ r2                -- alt-l and alt-r
""    ~~ Star _        = True                              -- star-0
s3    ~~ Star r        = or $ zipWith testSplit (inits s3) (tails s3)
                   where testSplit [] _  = False
                         testSplit s1 s2 = s1 ~~ r && s2 ~~ (Star r)
_     ~~ _             = False                             -- default
```

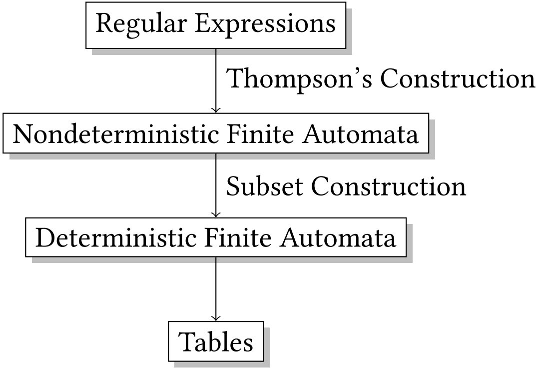
Backtracking sucks

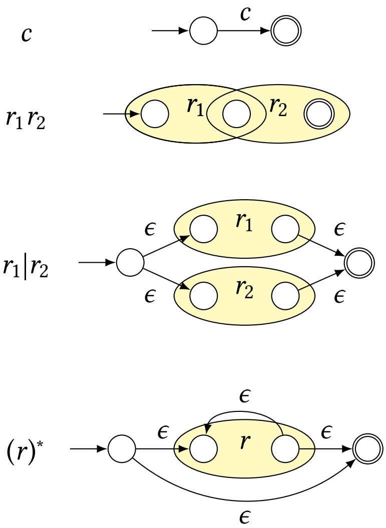
This is a backtracking algorithm that tries everything until it works

What does this do on "aaaaaaabb" ∼ (aaa)*b*?

```
(~~)                    :: String -> RE -> Bool
""    ~~ Epsilon        = True                           -- epsilon
[c1] ~~ Ch c2           = c1 == c2                       -- char
s3    ~~ Seq r1 r2      = or $ zipWith testSplit (inits s3) (tails s3)
                          where testSplit s1 s2 = s1 ~~ r1 && s2 ~~ r2
s     ~~ Alt r1 r2      = s ~~ r1 || s ~~ r2             -- alt-l and alt-r
""    ~~ Star _         = True                           -- star-0
s3    ~~ Star r         = or $ zipWith testSplit (inits s3) (tails s3)
                    where testSplit [] _  = False
                          testSplit s1 s2 = s1 ~~ r && s2 ~~ (Star r)
_     ~~ _              = False                          -- default
```

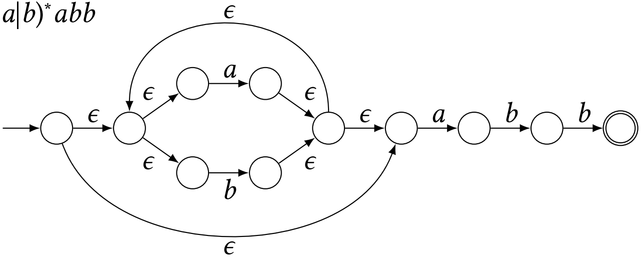
A Better Way: Thompson's Algorithm



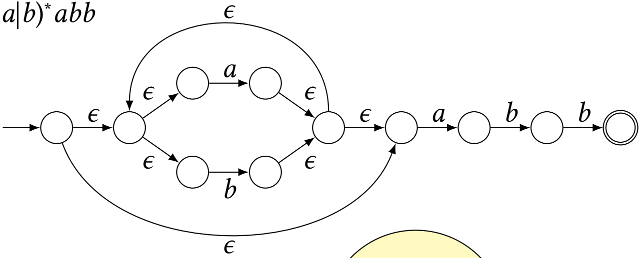


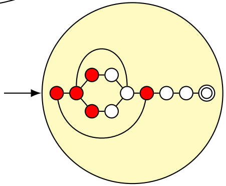
Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, June 1968.

$(a|b)^* abb$

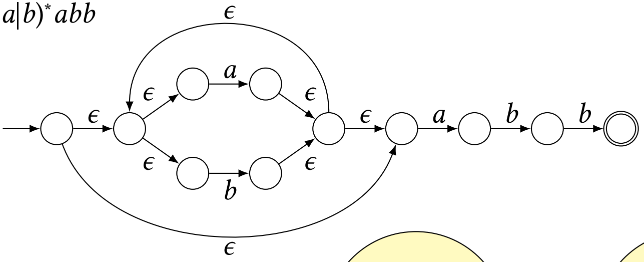


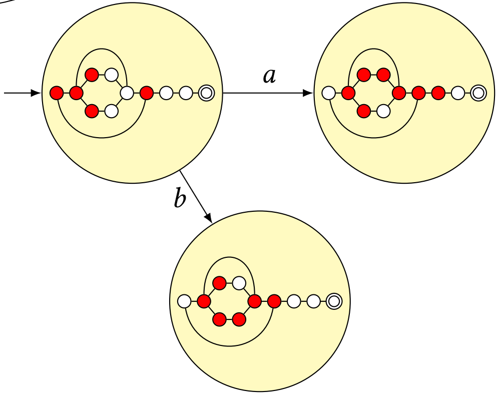
$(a|b)^*abb$



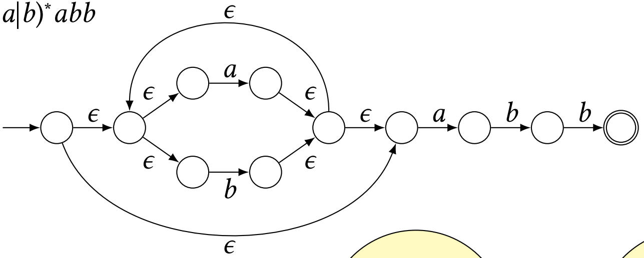


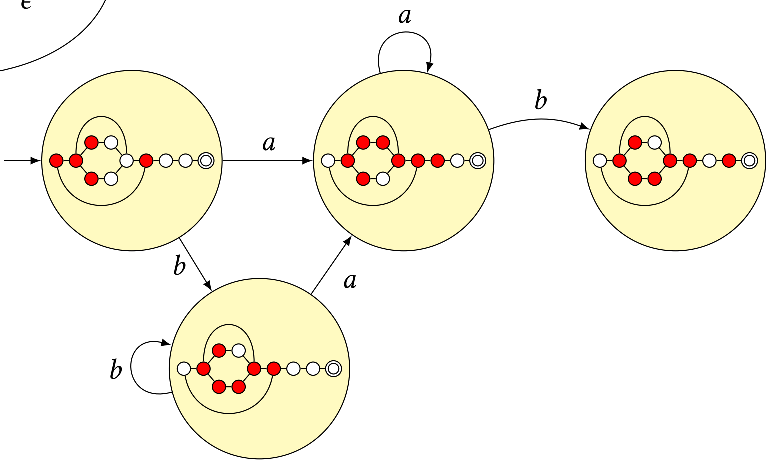
$(a|b)^*abb$



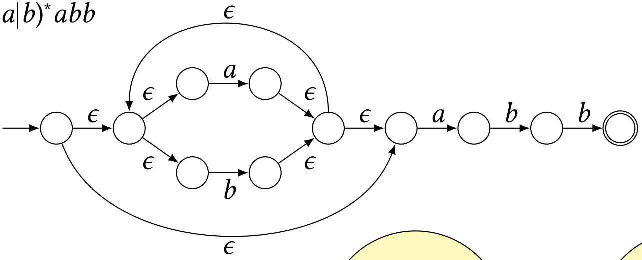


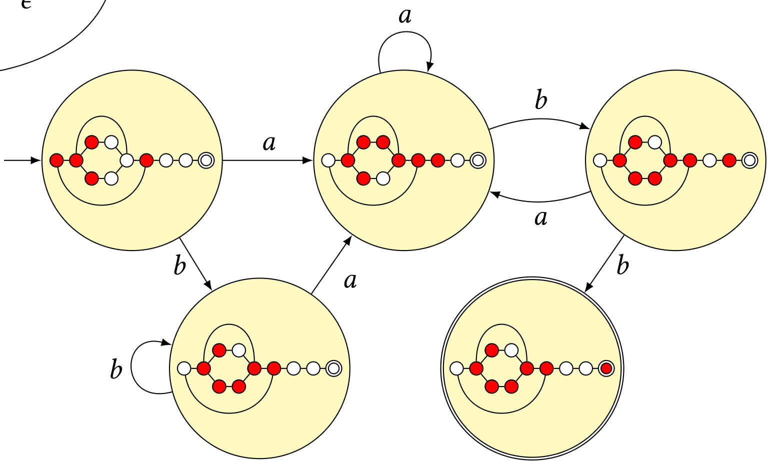
$(a|b)^*abb$



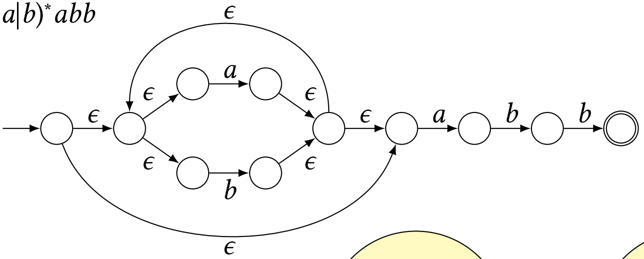


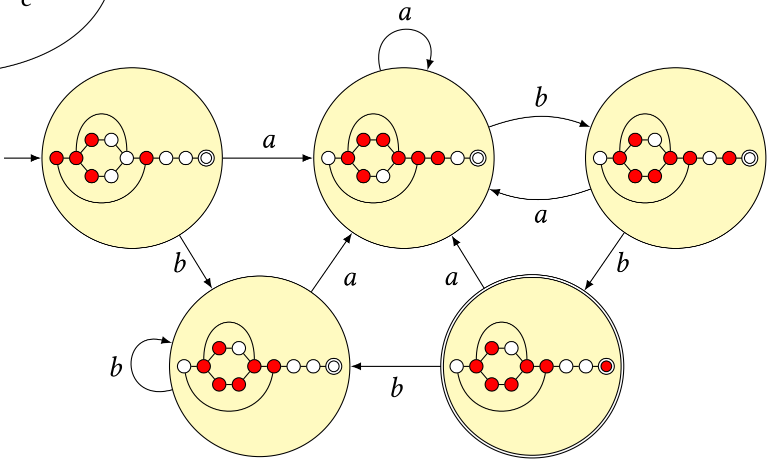
$(a|b)^* abb$





$(a|b)^*abb$





Brzozowski derivatives

$\partial_s R$ is the derivative of regular expression R w.r.t. the string s
"Every string that can follow s to match R"
$\partial_{s_1} R = \{ s_2 \mid s_1 s_2 \in L(R) \}$, where $L(R)$ is the language of R

Janusz A. Brzozowski. Derivatives of regular expressions.
Journal of the Association for Computing Machinery, 11(4):481–494, October 1964.

Brzozowski derivatives

$\partial_s R$ is the derivative of regular expression R w.r.t. the string s

"Every string that can follow s to match R"

$\partial_{s_1} R = \{ s_2 \mid s_1 s_2 \in L(R) \}$, where $L(R)$ is the language of R

$\partial_{\mathsf{a}}\, \mathsf{a} = \epsilon$ $\quad \partial_{\mathsf{a}}\, \mathsf{aa} = \mathsf{a}$ $\quad \partial_{\mathsf{a}}\, \mathsf{abc} = \mathsf{bc}$ $\quad \partial_{\mathsf{b}}\, \mathsf{abc} = \emptyset$ $\quad \partial_{\mathsf{a}}\, \mathsf{ab|cd} = \mathsf{b}$

$\partial_{\mathsf{a}}\, \mathsf{abc|acd} = \mathsf{bc|cd}$ $\quad \partial_{\mathsf{a}}\, \mathsf{a{*}bc} = \mathsf{a{*}bc}$ $\quad \partial_{\mathsf{a}}\, \mathsf{a{*}ac} = \mathsf{a{*}ac|c}$

Janusz A. Brzozowski. Derivatives of regular expressions.
Journal of the Association for Computing Machinery, 11(4):481–494, October 1964.

Brzozowski derivatives

$\partial_s R$ is the derivative of regular expression R w.r.t. the string s
"Every string that can follow s to match R"
$\partial_{s_1} R = \{ s_2 \mid s_1 s_2 \in L(R) \}$, where $L(R)$ is the language of R

$\partial_{\mathsf{a}}\, \mathsf{a} = \epsilon$ $\quad \partial_{\mathsf{a}}\, \mathsf{aa} = \mathsf{a}$ $\quad \partial_{\mathsf{a}}\, \mathsf{abc} = \mathsf{bc}$ $\quad \partial_{\mathsf{b}}\, \mathsf{abc} = \emptyset$ $\quad \partial_{\mathsf{a}}\, \mathsf{ab|cd} = \mathsf{b}$
$\partial_{\mathsf{a}}\, \mathsf{abc|acd} = \mathsf{bc|cd}$ $\quad \partial_{\mathsf{a}}\, \mathsf{a{*}bc} = \mathsf{a{*}bc}$ $\quad \partial_{\mathsf{a}}\, \mathsf{a{*}ac} = \mathsf{a{*}ac|c}$

Theorem: the derivative of a regular expression is a regular expression (including $\emptyset$)

Some subtlety when "" $\sim R$, rules otherwise look like those for polynomials

Janusz A. Brzozowski. Derivatives of regular expressions.
Journal of the Association for Computing Machinery, 11(4):481–494, October 1964.

Brzozowski derivatives

$\partial_s R$ is the derivative of regular expression R w.r.t. the string s
"Every string that can follow s to match R"
$\partial_{s_1} R = \{ s_2 \mid s_1 s_2 \in L(R) \}$, where $L(R)$ is the language of R

$\partial_{\mathsf{a}}\ \mathsf{a} = \epsilon$ $\quad \partial_{\mathsf{a}}\ \mathsf{aa} = \mathsf{a}$ $\quad \partial_{\mathsf{a}}\ \mathsf{abc} = \mathsf{bc}$ $\quad \partial_{\mathsf{b}}\ \mathsf{abc} = \emptyset$ $\quad \partial_{\mathsf{a}}\ \mathsf{ab|cd} = \mathsf{b}$
$\partial_{\mathsf{a}}\ \mathsf{abc|acd} = \mathsf{bc|cd}$ $\quad \partial_{\mathsf{a}}\ \mathsf{a{*}bc} = \mathsf{a{*}bc}$ $\quad \partial_{\mathsf{a}}\ \mathsf{a{*}ac} = \mathsf{a{*}ac|c}$

Theorem: the derivative of a regular expression is a regular expression (including $\emptyset$)

Some subtlety when "" $\sim R$, rules otherwise look like those for polynomials

Use "subset construction" to build a DFA: label states with regular expression derivatives

Janusz A. Brzozowski. Derivatives of regular expressions.
Journal of the Association for Computing Machinery, 11(4):481–494, October 1964.

Scott Owens, John Reppy, and Aaron Turon. Regular-expression derivatives re-examined.
Journal of Functional Programming, 19(2):173–190, March 2009.