Regular Expressions

CS 4100
Gordon Stewart
Ohio University

Quizzes

- Every Tuesday, we'll have a quiz with probability 1/3

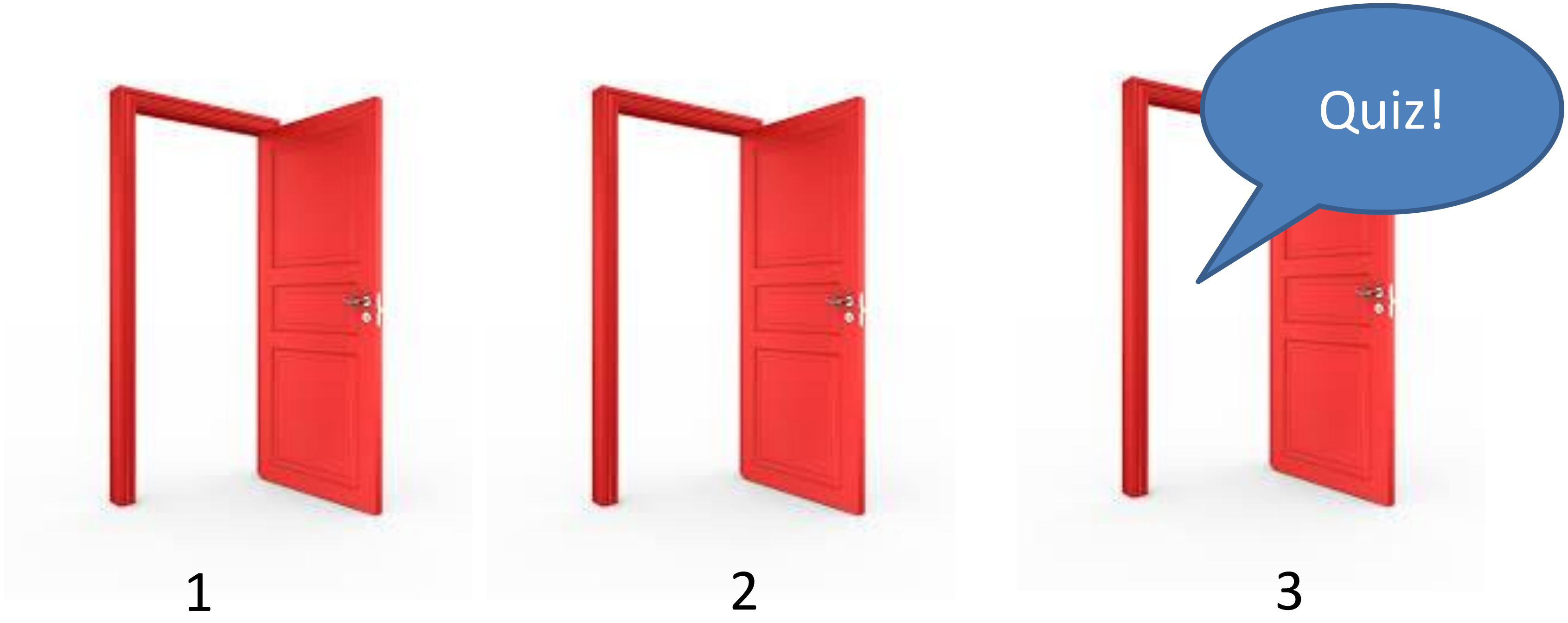
1

2

3

Quizzes

- Every Tuesday, we'll have a quiz with probability 1/3



Quiz 2

On a sheet of paper (or half a sheet borrowed from a friend), write

1. Your name
2. The answer to the following question:

What is the result of
the OCaml expression

```
let b=42 in let b=43 in b-1
```

?

REVIEW: 3000

Sets

- A *set* is a *collection* of elements (of some type *T*) that supports the following operations:
  - empty : set (* the empty set *)
  - contains (S : set) (t : T) : bool (* does set S contain t? *)
  - insert (S : set) (t : T) : set (* add t to set S *)
  - union (S T : set) : set (* all elements in either S or T *)
  - intersect (S T : set) : set (* all elements in S and T *)

Sets

- A *set* is a *collection* of elements (of some type *T*) that supports the following operations:
  - empty : set (* the empty set *)
  - contains (S : set) (t : T) : bool (* does set S contain t? *)
  - insert (S : set) (t : T) : set (* add t to set S *)
  - union (S T : set) : set (* all elements in either S or T *)
  - intersect (S T : set) : set (* all elements in S and T *)
- In math:
  - $\emptyset$ (* the empty set *)
  - $t \in S$ (* does set S contain t? *)
  - $S \cup \{ t \}$ (* all the elements in S, plus t *)
  - $S \cup T$ (* the union of S and T *)
  - $S \cap T$ (* the intersection of S and T *)

Derived Operations on Sets

- Set Inclusion
  - S is a *subset* of T: $S \subseteq T \coloneqq \forall s.\, s \in S \to s \in T$
  - S is a *superset* of T: $S \supseteq T \coloneqq \forall t.\, t \in T \to t \in S$

Derived Operations on Sets

- Set Inclusion
  - S is a *subset* of T: $S \subseteq T \coloneqq \forall s. s \in S \rightarrow s \in T$
  - S is a *superset* of T: $S \supseteq T \coloneqq \forall t. t \in T \rightarrow t \in S$

- Set Equality
  - S *equals* T $S = T \coloneqq S \subseteq T \ \wedge \ T \subseteq S$

Derived Operations on Sets

- Set Inclusion
  - S is a *subset* of T: $S \subseteq T \coloneqq \forall s.\, s \in S \rightarrow s \in T$
  - S is a *superset* of T: $S \supseteq T \coloneqq \forall t.\, t \in T \rightarrow t \in S$

- Set Equality
  - S *equals* T $S = T \coloneqq S \subseteq T \ \wedge \ T \subseteq S$

- Properties of set inclusion
  - Reflexivity $\forall S.\, S \subseteq S$
  - Transitivity $\forall S\ T\ R.\, S \subseteq T \ \wedge \ T \subseteq R \ \rightarrow \ S \subseteq R$

Derived Operations on Sets

- Set Inclusion
  - S is a *subset* of T: $S \subseteq T \coloneqq \forall s. s \in S \to s \in T$
  - S is a *superset* of T: $S \supseteq T \coloneqq \forall t. t \in T \to t \in S$

- Set Equality
  - S *equals* T $S = T \coloneqq S \subseteq T \ \wedge \ T \subseteq S$

- Properties of set inclusion
  - Reflexivity $\forall S. S \subseteq S$
  - Transitivity $\forall S\ T\ R. S \subseteq T \ \wedge \ T \subseteq R \ \to \ S \subseteq R$

- Set equality?
  - Reflexive, transitive, *and symmetric* (equivalence relation)

A Couple Questions…

$$\{a\} \cup \{b\} \subseteq \{a\}?$$

A Couple Questions…

$$\{a\} \cup \{b\} \subseteq \{a\}?$$

No…Why?

A Couple Questions…

$$\{a\} \cup \{b\} \subseteq \{a\}?$$

No…Why?

$$\{a\} \cup \{b\} \subseteq \{a\} \coloneqq \forall x \in \{a\} \cup \{b\}.\, x \in \{a\}$$

$$But\ b \in \{a\} \cup \{b\}\ \land\ b \notin \{a\}$$

A Couple Questions...

$$\{a\} \cup \{b\} \subseteq \{a\}?$$

No...Why?

$$\{a\} \cup \{b\} \subseteq \{a\} \coloneqq \forall x \in \{a\} \cup \{b\}.\, x \in \{a\}$$

$$But\ b \in \{a\} \cup \{b\}\ \wedge\ b \notin \{a\}$$

$$(\{a\} \cup \{b\}) \cap \emptyset \subseteq \emptyset?$$

A Couple Questions…

$$\{a\} \cup \{b\} \subseteq \{a\}?$$

No…Why?

$$\{a\} \cup \{b\} \subseteq \{a\} \coloneqq \forall x \in \{a\} \cup \{b\}. x \in \{a\}$$

$$But\ b \in \{a\} \cup \{b\}\ \wedge\ b \notin \{a\}$$

$$(\{a\} \cup \{b\}) \cap \emptyset \subseteq \emptyset?$$

Yes!…Why?

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)
  - $\{ n \in N \mid n\ \boldsymbol{mod}\ 2 = 0 \}$ The even natural numbers

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)
  - $\{ n \in N \mid n \ \boldsymbol{mod}\ 2 = 0 \}$ The even natural numbers
  - $\{ a \in A \mid a \in S \ \wedge \ a \notin T \}$ The set of elements in **A** that are also in **S** but not in **T**

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)
  - $\{ n \in N \mid n \ \boldsymbol{mod}\ 2 = 0 \}$ The even natural numbers
  - $\{ a \in A \mid a \in S \ \wedge \ a \notin T \}$ The set of elements in **A** that are also in **S** but not in **T**

- Set Difference
  - $S - T \coloneqq \{s \in S \mid s \notin T \}$

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)
  - $\{ n \in N \mid n \ \boldsymbol{mod}\ 2 = 0 \}$ The even natural numbers
  - $\{ a \in A \mid a \in S \ \wedge \ a \notin T \}$ The set of elements in **A** that are also in **S** but not in **T**

- Set Difference
  - $S - T \coloneqq \{s \in S \mid s \notin T \}$
  - *Examples:*

    $\{ n \in \boldsymbol{Z} \mid n \geq 0 \} - \{ n \in \boldsymbol{N} \mid n \ \boldsymbol{mod}\ 2 = 0 \}$

Operations on Sets (II)

- Set Comprehension
  - $\{ n \in \boldsymbol{Z} \mid n \geq 0 \}$ The natural numbers (**N**)
  - $\{ n \in N \mid n \boldsymbol{mod}\ 2 = 0 \}$ The even natural numbers
  - $\{ a \in A \mid a \in S \ \wedge \ a \notin T \}$ The set of elements in **A** that are also in **S** but not in **T**

- Set Difference
  - $S - T \coloneqq \{s \in S \mid s \notin T \}$
  - *Examples:*

    $\{ n \in \boldsymbol{Z} \mid n \geq 0 \} - \{ n \in \boldsymbol{N} \mid n \boldsymbol{mod}\ 2 = 0 \}$

    $\{a\} \cup \{b\} - \emptyset$

A Simple Proof

Theorem: Set Equality is Symmetric.

A Simple Proof

Theorem: Set Equality is Symmetric.

By the definition of symmetric, we need to show that:

(1) $\forall S\ T.\, S = T \rightarrow T = S$

A Simple Proof

Theorem: Set Equality is Symmetric.

By the definition of symmetric, we need to show that:

(1) $\forall S\ T.\ S = T \rightarrow T = S$

By the definition of set equality, (1) is equivalent to:

(2) $\forall S\ T.\ (S \subseteq T \wedge T \subseteq S) \rightarrow (T \subseteq S \wedge S \subseteq T)$

A Simple Proof

Theorem: Set Equality is Symmetric.

By the definition of symmetric, we need to show that:

(1) $\forall S\,T.\, S = T \to T = S$

By the definition of set equality, (1) is equivalent to:

(2) $\forall S\,T.\, (S \subseteq T \land T \subseteq S) \to (T \subseteq S \land S \subseteq T)$

Labeling our hypotheses and conclusions, we have:

(3) $\forall S\,T.\, (a)\; S \subseteq T \land (b)\; T \subseteq S \to (c)\; T \subseteq S \land (d)\; S \subseteq T$

A Simple Proof

Theorem: Set Equality is Symmetric.

By the definition of symmetric, we need to show that:

(1) $\forall S\, T.\, S = T \rightarrow T = S$

By the definition of set equality, (1) is equivalent to:

(2) $\forall S\, T.\, (S \subseteq T \wedge T \subseteq S) \rightarrow (T \subseteq S \wedge S \subseteq T)$

Labeling our hypotheses and conclusions, we have:

(3) $\forall S\, T.\, (a)\; S \subseteq T \wedge (b)\; T \subseteq S \rightarrow (c)\; T \subseteq S \wedge (d)\; S \subseteq T$

(c) follows directly from (b).

(d) follows directly from (a). ∎

REGULAR EXPRESSIONS IN OCAML

Regular Expressions

- In OCaml:

```
# #load "str.cma";;          (*load the regexp library*)

# open Str;;
```

Regular Expressions

- In OCaml:

```
# #load "str.cma";;        (*load the regexp library*)

# open Str;;

# let r = regexp "hello, .*";;
- : val r : Str.regexp = <abstr>
```

Regular Expressions

- In OCaml:

```
# #load “str.cma”;;          (*load the regexp library*)

# open Str;;

# let r = regexp “hello, .*”;;
- : val r : Str.regexp = <abstr>

# string_match r "hello world!" 0;;
```

Regular Expressions

- In OCaml:

```
# #load “str.cma”;;         (*load the regexp library*)

# open Str;;

# let r = regexp “hello, .*”;;
- : val r : Str.regexp = <abstr>

# string_match r "hello world!" 0;;
- : bool = false
```

Regular Expressions

- In OCaml:

```
# #load “str.cma”;;        (*load the regexp library*)

# open Str;;

# let r = regexp “hello, .*”;;
- : val r : Str.regexp = <abstr>

# string_match r "hello world!" 0;;
- : bool = false

# string_match r "hello, cs4100" 0;;
```

Regular Expressions

- In OCaml:

```
# #load “str.cma”;;        (*load the regexp library*)

# open Str;;

# let r = regexp “hello, .*”;;
- : val r : Str.regexp = <abstr>

# string_match r "hello world!" 0;;
- : bool = false

# string_match r "hello, cs4100" 0;;
- : bool = true
```

OCaml Regexp Syntax

- .   Matches any character except newline.
- r*   Matches the expression r zero, one or several times
- r+   Matches the expression r one or several times
- r?   Matches the expression r once or not at all
- [..]   Character set. Ranges are denoted with -, as in [a-z].
  An initial ^, as in [^0-9], complements the set.
- ^   Matches at beginning of line
- $ Matches at end of line
- r1|r2   Alternative between two expressions r1 and r2
- …

Examples

- A regexp that matches the *positive integers*

  … rejecting integers that begin with leading zeros.

  One possibility:

  ```
  let posints = "[1-9]+[0-9]*"
  in
  ```

- A regexp that matches the *positive floating-point numbers*

  … rejecting floats with leading zeros.

  One possibility:

  ```
  let posfloats = posints ^ "\.[0-9]+"
  ```

REGULAR EXPRESSIONS, FORMALLY

Backus-Naur Form (BNF)

- A notation for expressing *context-free grammars (CFGs)*
- If you don’t know what a CFG is, don’t worry!
  - A lot more about CFGs in the next two weeks…

Backus-Naur Form (BNF)

- A notation for expressing *context-free grammars (CFGs)*
- If you don't know what a CFG is, don't worry!
  - A lot more about CFGs in the next two weeks…
- A small example:

The name of the piece of syntax we're defining (in this case, <expr>, for expression)

```
<expr> ::= <int>                 (*integers*)
         | <expr> + <expr>  (*addition*)
         | <expr> * <expr>  (*multiplication*)
         | ( <expr> )
```

Each "production" after a pipe "|" corresponds to a different way to build an <expr>

Backus-Naur Form (BNF)

- A small example:

```
<expr> ::= <int>                   (*integers*)
         | <expr> + <expr>         (*addition*)
         | <expr> * <expr>         (*multiplication*)
         | ( <expr> )
```

- Strings in the “language” defined by the grammar above:

```
1
(1)
(1 + 2) * 3 + (4 * 5)
…
```

BNF in OCaml

- Traditional BNF

```
<expr> ::= <int>                  (*integers*)
         | <expr> + <expr>        (*addition*)
         | <expr> * <expr>        (*multiplication*)
         | ( <expr> )
```

BNF in OCaml

- Traditional BNF

```
<expr> ::= <int>                    (*integers*)
         | <expr> + <expr>  (*addition*)
         | <expr> * <expr>  (*multiplication*)
         | ( <expr> )
```

- The same data type in OCaml

```
type expr =
  | Eint of int
  | Eplus of (expr * expr)
  | Etimes of (expr * expr
  | Eparen of expr
```

A Language of Regular Expressions

- Regular Expressions in BNF Style

```
r ::= empty                  (*match no strings*)
    | ϵ                      (*match the empty string*)
    | ‘A’ | ‘B’ | ‘C’        (*characters A, B, C*)
    | r1 . r2                (*concatenation: r1 then r2*)
    | r*                     (*Kleene star: 0 or more*)
    | r1 || r2               (*r1 or r2*)
    | r1 && r2               (*r1 and r2*)
    | ¬r                     (*not r*)
    | ( r )                  (*paranthesized r*)
```

- Examples:
  - `‘A’ . ‘B’ . ‘C’`
  - ¬*empty*
  - `(‘A’ . ‘B’)*`

OCaml Regexp Syntax

- . Matches any character except newline.
- r* Matches the expression r zero, one or several times
- r+ Matches the expression r one or several times
- r? Matches the expression r once or not at all
- …

OCaml Regexp Syntax

- . Matches any character except newline.
- r* Matches the expression r zero, one or several times
- r+ Matches the expression r one or several times
- r? Matches the expression r once or not at all
- …

Desugaring

```
desugar(.)  = ('A' || ('B' || 'C'))
desugar(r+) = let s = desugar(r) in s . s*
desugar(r?) = let s = desugar(r) in (s||ϵ)
```

INTERPRETING
REGULAR EXPRESSIONS

What Do Regexps *Mean*?

- Every regexp r is denoted by a *set of strings* (language)

“is interpreted as”

de·no·ta·tion
ˌdēnōˈtāSHən/
noun
noun: **denotation**; plural noun: **denotations**
PHILOSOPHY the object or concept to which a term refers, or the set of objects of which a predicate is true.
<Google Search Result>

- Alternatively, the ***denotation*** of a regular expression is some set of strings, or language
- A string is a *sequence of characters* (0 or more) drawn from some *alphabet* (typically written Σ)
- Σ^*: The set of ***all possible strings*** over alphabet Σ

Enumerate the strings of…

- $\Sigma = \{a\}$

Enumerate the strings of…

- $\Sigma = \{a\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}aa\text{”}, \text{“}aaa\text{”}, \ldots\}$

Enumerate the strings of…

- $\Sigma = \{a\}$
  - $\Sigma^* = \{``", ``a", ``aa", ``aaa", \dots\}$

- $\Sigma = \{a, b\}$

Enumerate the strings of…

- $\Sigma = \{a\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}aa\text{”}, \text{“}aaa\text{”}, \dots\}$

- $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}b\text{”}, \text{“}aa\text{”}, \text{“}ab\text{”}, \text{“}ba\text{”}, \text{“}bb\text{”}, \text{“}aaa\text{”}, \text{“}aab\text{”}, \dots\}$

Enumerate the strings of...

- $\Sigma = \{a\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}aa\text{”}, \text{“}aaa\text{”}, \dots\}$

- $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}b\text{”}, \text{“}aa\text{”}, \text{“}ab\text{”}, \text{“}ba\text{”}, \text{“}bb\text{”}, \text{“}aaa\text{”}, \text{“}aab\text{”}, \dots\}$
- $\Sigma = \emptyset$

Enumerate the strings of…

- $\Sigma = \{a\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}aa\text{”}, \text{“}aaa\text{”}, \ldots\}$
- $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\text{“”}, \text{“}a\text{”}, \text{“}b\text{”}, \text{“}aa\text{”}, \text{“}ab\text{”}, \text{“}ba\text{”}, \text{“}bb\text{”}, \text{“}aaa\text{”}, \text{“}aab\text{”}, \ldots\}$
- $\Sigma = \emptyset$
  - $\Sigma^* = \{\text{“”}\}$

Interpreting Regexps

- Define a function, ***L***, that maps regular expressions to the set of strings they denote
- Mathematically

$$L(r) \subseteq \Sigma^*$$

- In Ocaml

$$\textbf{type}\ \text{string} = \Sigma\ list$$
$$L : regexp \rightarrow string\ set$$

- Examples:

$$Let\ \Sigma = \{a, b, c\}.$$
$$L(a) = \{``a"\}$$
$$L(a^*) = \{``", ``a", ``aa", ``aaa", \ldots\}$$
$$L(\neg empty) = \Sigma^* - L(empty) = \Sigma^* - \emptyset = \Sigma^*$$

Interpreting Regexps *In General*

```
r ::= empty | ϵ | c          (*c ∈ Σ*)
    | r1 . r2                (*concatenation*)
    | r*                     (*Kleene star*)
    | r1 || r2               (*r1 or r2*)
    | r1 && r2               (*r1 and r2*)
    | ¬r                     (*not r*)
```

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\text{“”}\} \\
L(c) &= \{\text{“}c\text{”}\} \\
L(r1 \,.\, r2) &= \{ s1 \wedge s2 \mid s1 \in L(r1) \text{ and } s2 \in L(r2) \} \\
L(r^*) &= \{\text{“”}\} \cup L(r \,.\, r^*) \\
L(r1 \,||\, r2) &= L(r1) \cup L(r2) \\
L(r1 \,\&\&\, r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Some Examples

$$\Sigma = \{a, b, c\}$$

$$L(a\,.\,b) =$$

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\text{ “” }\} \\
L(c) &= \{\text{ “}c\text{” }\} \\
L(r1\,.\,r2) &= \{\, s1 \wedge s2 \mid s1 \in L(r1) \; and \; s2 \in L(r2) \,\} \\
L(r^*) &= \{\text{ “” }\} \cup L(r\,.\,r^*) \\
L(r1 \,||\, r2\,) &= L(r1) \cup L(r2) \\
L(r1 \,\&\&\, r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Some Examples

$$\Sigma = \{a, b, c\}$$
$$L(a\,.\,b) = \{\, s1 \hat{}\, s2 \mid s1 \in L(a) \; and \; s2 \in L(b) \,\}$$

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\text{ ""}\} \\
L(c) &= \{\text{ "}c\text{"}\} \\
L(r1\,.\,r2) &= \{\, s1 \wedge s2 \mid s1 \in L(r1) \; and \; s2 \in L(r2) \,\} \\
L(r^*) &= \{\text{ ""}\} \cup L(r\,.\,r^*) \\
L(r1 \,||\, r2\,) &= L(r1) \cup L(r2) \\
L(r1 \,\&\&\, r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Some Examples

$$\Sigma = \{a, b, c\}$$

$$L(a\,.\,b) = \{\, s1\text{^}s2 \mid s1 \in L(a)\ and\ s2 \in L(b)\, \}$$

$$= \{\, s1\text{^}s2 \mid s1 \in \{\text{“}a\text{”}\}\ and\ s2 \in \{\text{“}b\text{”}\, \}\}$$

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\, \text{“”}\, \} \\
L(c) &= \{\, \text{“}c\text{”}\, \} \\
L(r1\,.\,r2) &= \{\, s1 \text{ ^ } s2 \mid s1 \in L(r1)\ and\ s2 \in L(r2)\, \} \\
L(r^*) &= \{\, \text{“”}\, \} \cup L(r\,.\,r^*) \\
L(r1 \mid\mid r2\,) &= L(r1) \cup L(r2) \\
L(r1\ \&\&\ r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Some Examples

$$\Sigma = \{a, b, c\}$$

$$
\begin{aligned}
L(a\,.\,b) &= \{\, s1\text{^}s2 \mid s1 \in L(a) \; and \; s2 \in L(b) \,\} \\
&= \{\, s1\text{^}s2 \mid s1 \in \{\text{“}a\text{”}\} \; and \; s2 \in \{\text{“}b\text{”}\,\}\} \\
&= \{\, \text{“}a\text{”} \wedge \text{“}b\text{”} \,\}
\end{aligned}
$$

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\, \text{“”} \,\} \\
L(c) &= \{\, \text{“}c\text{”} \,\} \\
L(r1\,.\,r2) &= \{\, s1 \wedge s2 \mid s1 \in L(r1) \; and \; s2 \in L(r2) \,\} \\
L(r^*) &= \{\, \text{“”} \,\} \cup L(r\,.\,r^*) \\
L(r1 \mid\mid r2\,) &= L(r1) \cup L(r2) \\
L(r1 \;\&\&\; r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Some Examples

$$\Sigma = \{a, b, c\}$$

$$\begin{aligned} L(a\,.\,b) &= \{\, s1^\wedge s2 \mid s1 \in L(a)\ and\ s2 \in L(b)\,\} \\ &= \{\, s1^\wedge s2 \mid s1 \in \{``a"\}\ and\ s2 \in \{``b"\,\}\} \\ &= \{\, ``a" \wedge ``b"\,\} \\ &= \{\, ``ab"\,\} \end{aligned}$$

$$\begin{aligned} L(empty) &= \emptyset \\ L(\epsilon) &= \{\, ``" \,\} \\ L(c) &= \{\, ``c" \,\} \\ L(r1\,.\,r2) &= \{\, s1 \wedge s2 \mid s1 \in L(r1)\ and\ s2 \in L(r2)\,\} \\ L(r^*) &= \{\, ``" \,\} \cup L(r\,.\,r^*) \\ L(r1 \mid\mid r2\,) &= L(r1) \cup L(r2) \\ L(r1\ \&\&\ r2) &= L(r1) \cap L(r2) \\ L(\neg r) &= \Sigma^* - L(r) \end{aligned}$$

A Trickier One…

$\Sigma = \{a, b, c\}$

$L(empty^*. a)$

A Trickier One…

$\Sigma = \{a, b, c\}$

$L(empty^*. a)$

$= \{\ s1\hat{}s2 \mid s1 \in L(empty^*)\ and\ s2 \in L(a)\ \}$

A Trickier One...

$\Sigma = \{a, b, c\}$

$L(empty^*.a)$

$= \{ s1 \wedge s2 \mid s1 \in L(empty^*) \; and \; s2 \in L(a) \}$

$= \{ s1 \wedge s2 \mid s1 \in (\{``"\} \cup L(empty \, . \, empty^*)) \; and \; s2 \in \{``a"\}\}$

A Trickier One...

$\Sigma = \{a, b, c\}$

$L(empty^*.a)$

$= \{\ s1 \wedge s2 \mid s1 \in L(empty^*)\ and\ s2 \in L(a)\ \}$

$= \{\ s1 \wedge s2\ \mid s1 \in (\{\text{“”}\} \cup L(empty\ .\ empty^*))\ and\ s2 \in \{\text{“}a\text{”}\ \}\}$

$= \{\ s1 \wedge \text{“}a\text{”} \mid s1 \in \left(\{\text{“”}\} \cup L(empty\ .\ empty^*)\right)\}$

A Trickier One...

$\Sigma = \{a, b, c\}$

$L(empty^*.a)$

$= \{\ s1 \wedge s2 \mid s1 \in L(empty^*)\ and\ s2 \in L(a)\ \}$

$= \{\ s1 \wedge s2\ \mid s1 \in (\{``"\} \cup L(empty\ .\ empty^*))\ and\ s2 \in \{``a"\ \}\}$

$= \{\ s1 \wedge ``a" \mid s1 \in \left(\{``"\} \cup L(empty\ .\ empty^*)\right)\}$

$= \{\ s1 \wedge ``a" \mid s1 \in (\{``"\} \cup \emptyset)\}$

A Trickier One…

$\Sigma = \{a, b, c\}$

$L(empty^{*}.a)$

$= \{ s1 \hat{} s2 \mid s1 \in L(empty^{*}) \ and \ s2 \in L(a) \}$

$= \{ s1 \hat{} s2 \mid s1 \in (\{\text{“”}\} \cup L(empty . empty^{*})) \ and \ s2 \in \{\text{“}a\text{”}\}\}$

$= \{ s1 \hat{} \text{“}a\text{”} \mid s1 \in \left(\{\text{“”}\} \cup L(empty . empty^{*})\right)\}$

$= \{ s1 \hat{} \text{“}a\text{”} \mid s1 \in (\{\text{“”}\} \cup \emptyset)\}$

$= \{ s1 \hat{} \text{“}a\text{”} \mid s1 \in (\{\text{“”}\})\}$

A Trickier One...

$\Sigma = \{a, b, c\}$

$L(empty^*.a)$

$= \{ s1 \wedge s2 \mid s1 \in L(empty^*) \; and \; s2 \in L(a) \}$

$= \{ s1 \wedge s2 \mid s1 \in (\{``"\} \cup L(empty\,.\,empty^*)) \; and \; s2 \in \{``a"\}\}$

$= \{ s1 \wedge ``a" \mid s1 \in \left(\{``"\} \cup L(empty\,.\,empty^*)\right)\}$

$= \{ s1 \wedge ``a" \mid s1 \in (\{``"\} \cup \emptyset)\}$

$= \{ s1 \wedge ``a" \mid s1 \in (\{``"\})\}$

$= \{ ``" \wedge ``a" \} = \{ ``a" \}$

Interpreting Regexps *In General*

```
r ::= empty | ϵ | c          (*c ∈ Σ*)
    | r1 . r2                (*concatenation*)
    | r*                     (*Kleene star*)
    | (r1 || r2)             (*r1 or r2*)
    | (r1 && r2)             (*r1 and r2*)
    | ¬r                     (*not r*)
```

$$L(empty) = \emptyset$$
$$L(\epsilon) = \{\text{“”}\}$$
$$L(c) = \{\text{“}c\text{”}\}$$
$$L(r1 \,.\, r2) = \{\, s1 \wedge s2 \mid s1 \in L(r1) \; and \; s2 \in L(r2) \,\}$$
$$\boxed{L(r^*) = \{\text{“”}\} \cup L(r \,.\, r^*)}$$
$$L(r1 \,||\, r2) = L(r1) \cup L(r2)$$
$$L(r1 \,\&\&\, r2) = L(r1) \cap L(r2)$$
$$L(\neg r) = \Sigma^* - L(r)$$

Interpreting Regexps *In General*

`r ::=` $empty \mid \epsilon$ `| c          (*c` $\in \Sigma$`*)`

$$L(r^*) = \{\text{“”}\} \cup L(r\,.\,r^*)$$
$$= \{\text{“”}\} \cup \{s1\,\hat{}\,s2 \mid s1 \in L(r) \text{ and } s2 \in L(r^*)\}$$

L(r*) is defined in terms of itself!

$$
\begin{aligned}
L(empty) &= \emptyset \\
L(\epsilon) &= \{\text{“”}\} \\
L(c) &= \{\text{“}c\text{”}\} \\
L(r1\,.\,r2) &= \{ s1 \,\hat{}\, s2 \mid s1 \in L(r1) \text{ and } s2 \in L(r2) \} \\
\boxed{L(r^*) = \{\text{“”}\} \cup L(r\,.\,r^*)} \\
L(r1 \mid\mid r2) &= L(r1) \cup L(r2) \\
L(r1 \mathbin{\&\&} r2) &= L(r1) \cap L(r2) \\
L(\neg r) &= \Sigma^* - L(r)
\end{aligned}
$$

Kleene Star Take 2

Kleene Star Take 1

$$L(r^*) = \{\text{“”}\} \cup L(r\,.\,r^*)$$

Kleene Star Take 2

$$L(r^*) = \bigcup_{n:nat} iter\ n\ L(r)$$

Iteration

$$iter : nat \rightarrow (\Sigma\ list)\ set \rightarrow (\Sigma\ list)\ set$$

$$iter\ 0\ S = \{\text{“”}\}$$

$$iter\ n\ S = \{s1 \wedge s2 \mid s1 \in S \text{ and } s2 \in iter\ (n-1)\ S\}$$

An Example

Kleene Star Take 2

$$L(r^*) = \bigcup_{n:nat} iter\ n\ L(r)$$

$$iter : nat \rightarrow (\Sigma\ list)\ set \rightarrow (\Sigma\ list)\ set$$

$$iter\ 0\ S = \{\text{“”}\}$$

$$iter\ n\ S = \{s1 \wedge s2 \mid s1 \in S\ and\ s2 \in iter\ (n-1)\ S\}$$

$$\text{“}aa\text{”} \in L(a^*)?$$

$$\underbrace{iter\ 0\ L(a) \cup iter\ 1\ L(a) \cup iter\ 2\ L(a) \cup iter\ 3\ L(a) \cup \cdots}$$

$$\{\text{“”}\} \cup \{\text{“}a\text{”}\} \cup \{\text{“}aa\text{”}\} \cup \{\text{“}aaa\text{”}\} \cup \cdots$$

$$\text{“}aa\text{”} \in \{\text{“}aa\text{”}\}\ !$$

A Theorem

$$L(r^*) = \bigcup_{n:nat} iter\ n\ L(r)$$

$$iter\ 0\ S = \{``"\}$$

$$iter\ n\ S = \{s1\ \hat{}\ s2 \mid s1 \in S\ and\ s2 \in iter\ (n-1)\ S\}$$

Theorem: $``aa" \in L(a^*)$.

Proof: Suffices $\exists n.\ ``aa" \in iter\ n\ L(a)$.

Let $n = 2$. *N.T.S.* $``aa" \in iter\ 2\ L(a)$.

$$\begin{aligned} iter\ 2\ L(a) &= \{s1 \hat{} s2 \mid s1 \in L(a), s2 \in iter\ 1\ L(a)\} \\ &= \{s1 \hat{} s2 \mid s1 \in L(a), \\ &\qquad s2 \in \{x1 \hat{} x2 \mid x1 \in L(a), x2 \in iter\ 0\ L(a)\} \\ &= \{``a" \hat{} s2 \mid s2 \in \{``a" \hat{} x2 \mid x2 \in \{``"\}\}\} \\ &= \{``a" \hat{} s2 \mid s2 \in \{``a" \hat{} ``"\}\} \\ &= \{``aa"\}\ \blacksquare \end{aligned}$$

A Slightly Trickier Theorem

$$L(r^*) = \bigcup_{n:nat} iter\ n\ L(r)$$

$$iter\ 0\ S = \{\text{“”}\}$$

$$iter\ n\ S = \{s1 \wedge s2 \mid s1 \in S\ and\ s2 \in iter\ (n-1)\ S\}$$

Theorem: $\forall r.\ L(r^*) \subseteq L(r^{**})$.

Proof: $Exercise$ ☺ ■

A Slightly Trickier Theorem

$$L(r^*) = \bigcup_{n:nat} iter\ n\ L(r)$$

$$iter\ 0\ S = \{\text{“”}\}$$

$$iter\ n\ S = \{s1 \mathbin{\char`^} s2 \mid s1 \in S\ and\ s2 \in iter\ (n-1)\ S\}$$

Theorem: $\forall r.\ L(r^*) \subseteq L(r^{**})$.

Proof:

$$N.T.S. \forall x \in \bigcup iter\ n\ L(r).\ x \in \bigcup iter\ m\ (\bigcup iter\ m'\, L(r))$$

(1) $\exists n.\, x \in iter\ n\ L(r)$

(2) Let m = 1, m’ = n.

(3) N.T.S. $x \in \{s1 \mathbin{\char`^} s2 \mid s1 \in iter\ n\ L(r)\ and\ s2 \in iter\ 0\ \ldots\}$

(4) $= x \in\ iter\ n\ L(r)$ ■