# Automata and Formal Languages (2)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS

# Languages

A language is a set of strings.

A regular expression specifies a set of strings, or language.

# Strings

Different ways of writing strings:

$$\text{"}hello\text{"} \qquad [h, e, l, l, o] \qquad h :: e :: l :: l :: o :: Nil$$

$$\text{""} \qquad [] \qquad Nil$$

# Strings

Different ways of writing strings:

$$\text{"}hello\text{"} \qquad [h, e, l, l, o] \qquad h :: e :: l :: l :: o :: Nil$$

$$\text{""} \qquad [] \qquad Nil$$

The concatenation operation on strings and sets of strings:

$$\text{"}foo\text{"} \mathbin{@} \text{"}bar\text{"} = \text{"}foobar\text{"}$$

$$A \mathbin{@} B \stackrel{\text{def}}{=} \{s_1 @ s_2 \mid s_1 \in A \wedge s_2 \in B\}$$

# Regular Expressions

Their inductive definition:

$$
\begin{array}{rcll}
r & ::= & \varnothing & \text{null} \\
 & | & \epsilon & \text{empty string / ”” / []} \\
 & | & c & \text{character} \\
 & | & r_1 \cdot r_2 & \text{sequence} \\
 & | & r_1 + r_2 & \text{alternative / choice} \\
 & | & r^* & \text{star (zero or more)}
\end{array}
$$

# Re

```
1  abstract class Rexp
2
3  case object NULL extends Rexp
4  case object EMPTY extends Rexp
5  case class CHAR(c: Char) extends Rexp
6  case class ALT(r1: Rexp, r2: Rexp) extends Rexp
7  case class SEQ(r1: Rexp, r2: Rexp) extends Rexp
8  case class STAR(r: Rexp) extends Rexp
```

Their indu

$$
\begin{array}{rcll}
r & ::= & \varnothing & \text{null} \\
 & | & \epsilon & \text{empty string / ”” / []} \\
 & | & c & \text{character} \\
 & | & r_1 \cdot r_2 & \text{sequence} \\
 & | & r_1 + r_2 & \text{alternative / choice} \\
 & | & r^* & \text{star (zero or more)}
\end{array}
$$

# The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & L(r_1) \mathbin{@} L(r_2) \\
L(r^*) & \stackrel{\text{def}}{=} & \bigcup_{n \geq 0} L(r)^n
\end{array}
$$

$L$ is a function from regular expressions to sets of strings

$L : \text{Rexp} \Rightarrow \text{Set[String]}$

# The Meaning of a Regular Expression

$$
\begin{array}{rcl}
L(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\
L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\
L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\
L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & L(r_1) \mathbin{@} L(r_2) \\
L(r^*) & \stackrel{\text{def}}{=} & \bigcup_{n \geq 0} L(r)^n
\end{array}
$$

$$
\begin{array}{rcl}
L(r)^0 & \stackrel{\text{def}}{=} & \{""\} \\
L(r)^{n+1} & \stackrel{\text{def}}{=} & L(r) \mathbin{@} L(r)^n
\end{array}
$$

$L$ is a function from regular expressions to sets of strings

$L : \text{Rexp} \Rightarrow \text{Set[String]}$

What is $L(a^*)$?

# Reg Exp Equivalences

$$
\begin{array}{lrcl}
 & (a + b) + c & \equiv^? & a + (b + c) \\
 & a + a & \equiv^? & a \\
 & (a \cdot b) \cdot c & \equiv^? & a \cdot (b \cdot c) \\
 & a \cdot a & \equiv^? & a \\
 & \epsilon^* & \equiv^? & \epsilon \\
 & \varnothing^* & \equiv^? & \varnothing \\
\forall\ \text{r.} & r \cdot \epsilon & \equiv^? & r \\
\forall\ \text{r.} & r + \epsilon & \equiv^? & r \\
\forall\ \text{r.} & r + \varnothing & \equiv^? & r \\
\forall\ \text{r.} & r \cdot \varnothing & \equiv^? & r \\
 & c \cdot (a + b) & \equiv^? & (c \cdot a) + (c \cdot b) \\
 & a^* & \equiv^? & \epsilon + (a \cdot a^*)
\end{array}
$$

# Reg Exp Equivalences

$$(a + b) + c \equiv^{?} a + (b + c) \qquad \text{yes}$$

$$a + a \equiv^{?} a$$

$$(a \cdot b) \cdot c \equiv^{?} a \cdot (b \cdot c)$$

$$a \cdot a \equiv^{?} a$$

$$\epsilon^* \equiv^{?} \epsilon$$

$$\varnothing^* \equiv^{?} \varnothing$$

$\forall$ r. $\quad r \cdot \epsilon \equiv^{?} r$

$\forall$ r. $\quad r + \epsilon \equiv^{?} r$

$\forall$ r. $\quad r + \varnothing \equiv^{?} r$

$\forall$ r. $\quad r \cdot \varnothing \equiv^{?} r$

$$c \cdot (a + b) \equiv^{?} (c \cdot a) + (c \cdot b)$$

$$a^* \equiv^{?} \epsilon + (a \cdot a^*)$$

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | |
| | $a \cdot a$ | $\equiv^?$ | $a$ | |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | no |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | no |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | no |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | no |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | no |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | no |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | yes |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | |

# Reg Exp Equivalences

| | | | | |
|---|---|---|---|---|
| | $(a + b) + c$ | $\equiv^?$ | $a + (b + c)$ | yes |
| | $a + a$ | $\equiv^?$ | $a$ | yes |
| | $(a \cdot b) \cdot c$ | $\equiv^?$ | $a \cdot (b \cdot c)$ | yes |
| | $a \cdot a$ | $\equiv^?$ | $a$ | no |
| | $\epsilon^*$ | $\equiv^?$ | $\epsilon$ | yes |
| | $\varnothing^*$ | $\equiv^?$ | $\varnothing$ | no |
| $\forall$ r. | $r \cdot \epsilon$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r + \epsilon$ | $\equiv^?$ | $r$ | no |
| $\forall$ r. | $r + \varnothing$ | $\equiv^?$ | $r$ | yes |
| $\forall$ r. | $r \cdot \varnothing$ | $\equiv^?$ | $r$ | no |
| | $c \cdot (a + b)$ | $\equiv^?$ | $(c \cdot a) + (c \cdot b)$ | yes |
| | $a^*$ | $\equiv^?$ | $\epsilon + (a \cdot a^*)$ | yes |

# The Specification for Matching

a regular expression $r$ matches a string $s$ if and only if

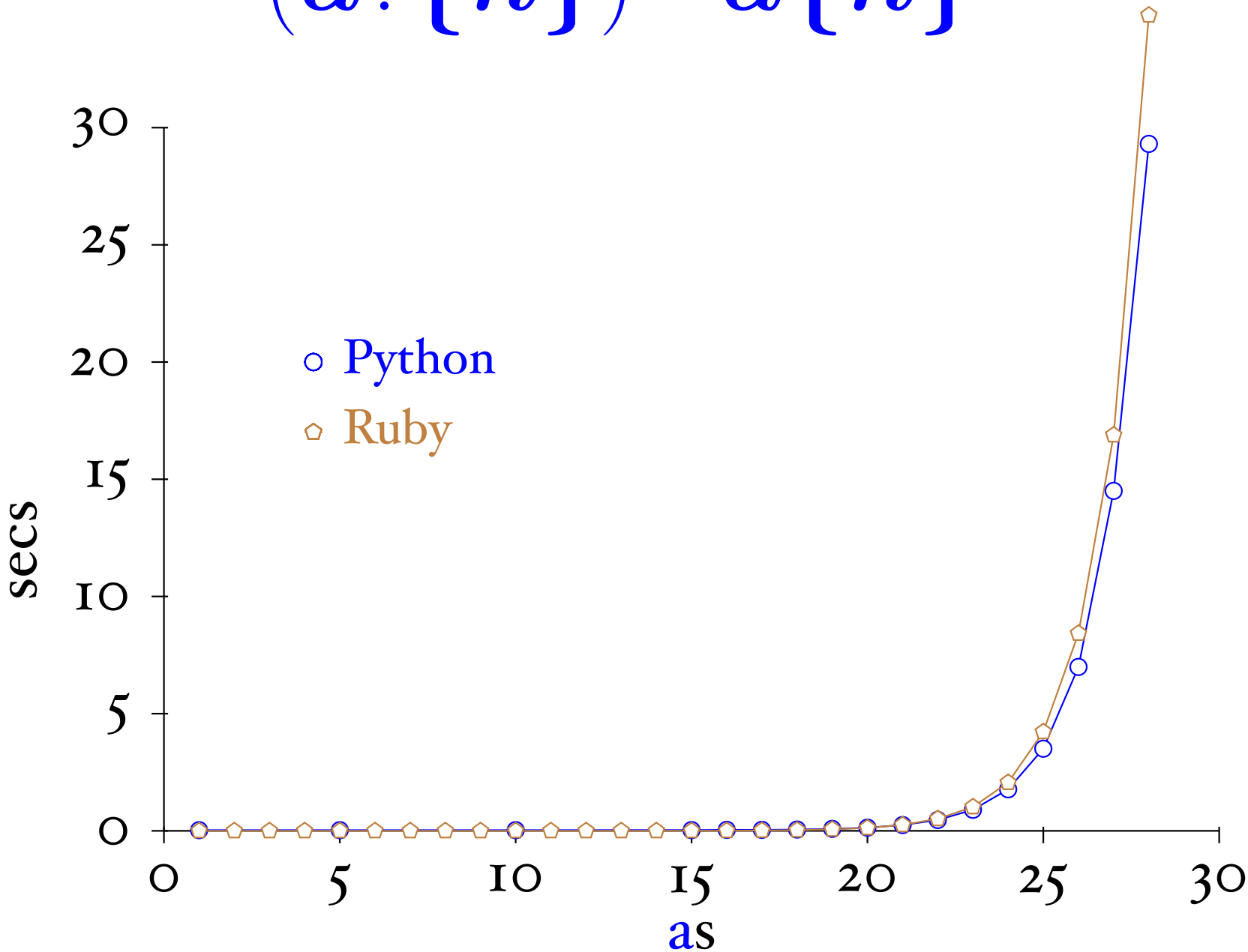
$$s \in L(r)$$

# The Specification for Matching

a regular expression $r$ matches a string $s$ if and only if

$$s \in L(r)$$

# $(a?\{n\}) \cdot a\{n\}$



# Evil Regular Expressions

- Regular expression Denial of Service (ReDoS)
- Evil regular expressions
  - $(a?\{n\}) \cdot a\{n\}$
  - $(a^+)^+$
  - $([a\text{-}z]^+)^*$
  - $(a + a \cdot a)^+$
  - $(a + a?)^+$

# A Matching Algorithm

...whether a regular expression can match the empty string:

$$
\begin{array}{lcl}
nullable(\varnothing) & \stackrel{\text{def}}{=} & false \\
nullable(\epsilon) & \stackrel{\text{def}}{=} & true \\
nullable(c) & \stackrel{\text{def}}{=} & false \\
nullable(r_1 + r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \vee nullable(r_2) \\
nullable(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nullable(r^*) & \stackrel{\text{def}}{=} & true
\end{array}
$$

# A Matching Algorithm

...whether a regular expression can match the empty string:

$$
\begin{array}{lcl}
nullable(\varnothing) & \stackrel{\text{def}}{=} & false \\
nullable(\epsilon) & \stackrel{\text{def}}{=} & true \\
nullable(c) & \stackrel{\text{def}}{=} & false \\
nullable(r_1 + r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \vee nullable(r_2) \\
nullable(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nulla & \stackrel{\text{def}}{} &
\end{array}
$$

```
1  def nullable (r: Rexp) : Boolean = r match {
2    case NULL => false
3    case EMPTY => true
4    case CHAR(_) => false
5    case ALT(r1, r2) => nullable(r1) || nullable(r2)
6    case SEQ(r1, r2) => nullable(r1) && nullable(r2)
7    case STAR(_) => true
8  }
```

# The Derivative of a Rexp

If $r$ matches the string $c :: s$, what is a regular expression that matches $s$?

$der\, c\, r$ gives the answer

# The Derivative of a Rexp (2)

$$
\begin{array}{lcl}
\boldsymbol{der}\, c\, (\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
\boldsymbol{der}\, c\, (\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
\boldsymbol{der}\, c\, (d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \epsilon \text{ else } \varnothing \\
\boldsymbol{der}\, c\, (r_1 + r_2) & \stackrel{\text{def}}{=} & \boldsymbol{der}\, c\, r_1 + \boldsymbol{der}\, c\, r_2 \\
\boldsymbol{der}\, c\, (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } \boldsymbol{nullable}(r_1) \\
 & & \text{then } (\boldsymbol{der}\, c\, r_1) \cdot r_2 + \boldsymbol{der}\, c\, r_2 \\
 & & \text{else } (\boldsymbol{der}\, c\, r_1) \cdot r_2 \\
\boldsymbol{der}\, c\, (r^*) & \stackrel{\text{def}}{=} & (\boldsymbol{der}\, c\, r) \cdot (r^*)
\end{array}
$$

# The Derivative of a Rexp (2)

$$
\begin{array}{lcl}
\boldsymbol{der}\, c\, (\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
\boldsymbol{der}\, c\, (\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
\boldsymbol{der}\, c\, (d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \epsilon \text{ else } \varnothing \\
\boldsymbol{der}\, c\, (r_1 + r_2) & \stackrel{\text{def}}{=} & \boldsymbol{der}\, c\, r_1 + \boldsymbol{der}\, c\, r_2 \\
\boldsymbol{der}\, c\, (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } \boldsymbol{nullable}(r_1) \\
 & & \text{then } (\boldsymbol{der}\, c\, r_1) \cdot r_2 + \boldsymbol{der}\, c\, r_2 \\
 & & \text{else } (\boldsymbol{der}\, c\, r_1) \cdot r_2 \\
\boldsymbol{der}\, c\, (r^*) & \stackrel{\text{def}}{=} & (\boldsymbol{der}\, c\, r) \cdot (r^*) \\
\boldsymbol{ders}\, [] \, r & \stackrel{\text{def}}{=} & r \\
\boldsymbol{ders}\, (c :: s)\, r & \stackrel{\text{def}}{=} & \boldsymbol{ders}\, s\, (\boldsymbol{der}\, c\, r)
\end{array}
$$

# The Derivative of a Rexp (2)

$$der\ c\ (\varnothing) \stackrel{\text{def}}{=} \varnothing$$

$$der\ c\ (\epsilon) \stackrel{\text{def}}{=} \varnothing$$

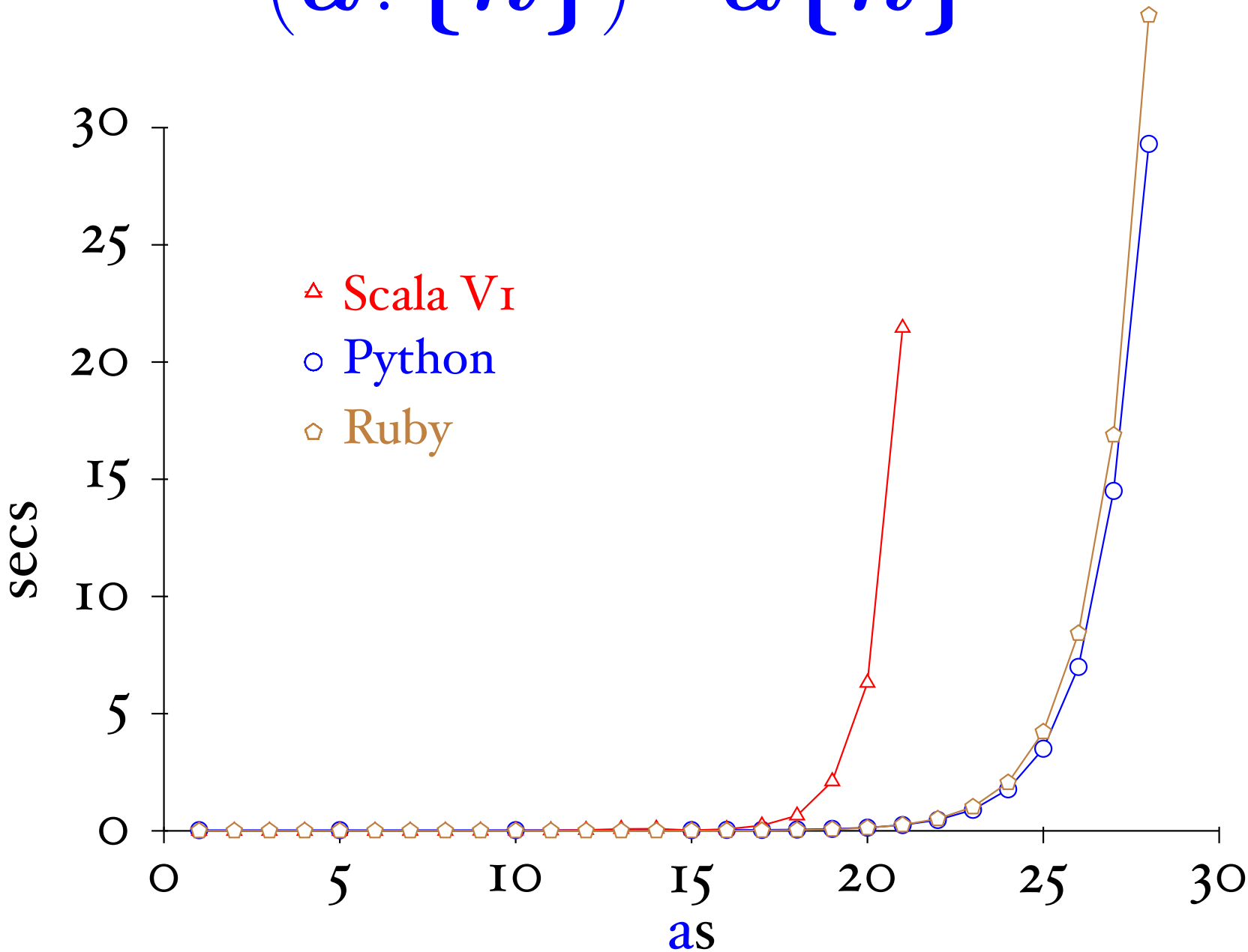
```
1  def der (r: Rexp, c: Char) : Rexp = r match {
2    case NULL => NULL
3    case EMPTY => NULL
4    case CHAR(d) => if (c == d) EMPTY else NULL
5    case ALT(r1, r2) => ALT(der(r1, c), der(r2, c))
6    case SEQ(r1, r2) =>
7      if (nullable(r1)) ALT(SEQ(der(r1, c), r2), der(r2, c))
8      else SEQ(der(r1, c), r2)
9    case STAR(r) => SEQ(der(r, c), STAR(r))
10 }
11
12 def ders (s: List[Char], r: Rexp) : Rexp = s match {
13   case Nil => r
14   case c::s => ders(s, der(c, r))
15 }
```

# Examples

Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ what is

$$der\ a\ r$$
$$der\ b\ r$$

# $(a?\{n\}) \cdot a\{n\}$



# Proofs about Rexps

Remember their inductive definition:

$$
\begin{array}{rcl}
r & ::= & \varnothing \\
 & | & \epsilon \\
 & | & c \\
 & | & r_1 \cdot r_2 \\
 & | & r_1 + r_2 \\
 & | & r^*
\end{array}
$$

If we want to prove something, say a property $P(r)$, for all regular expressions $r$ then ...

# Proofs about Rexp (2)

- $P$ holds for $\varnothing$, $\epsilon$ and $c$
- $P$ holds for $r_1 + r_2$ under the assumption that $P$ already holds for $r_1$ and $r_2$.
- $P$ holds for $r_1 \cdot r_2$ under the assumption that $P$ already holds for $r_1$ and $r_2$.
- $P$ holds for $r^*$ under the assumption that $P$ already holds for $r$.

# Proofs about Rexp (3)

Assume $P(r)$ is the property:

$$nullable(r) \text{ if and only if } "" \in L(r)$$

# Proofs about Rexp (4)

Let $Der\, c\, A$ be the set defined as

$$Der\, c\, A \stackrel{\text{def}}{=} \{s \mid c :: s \in A\}$$

We can prove

$$L(der\, c\, r) = Der\, c\, (L(r))$$

by induction on $r$.

# Proofs about Strings

If we want to prove something, say a property $P(s)$, for all strings $s$ then ...

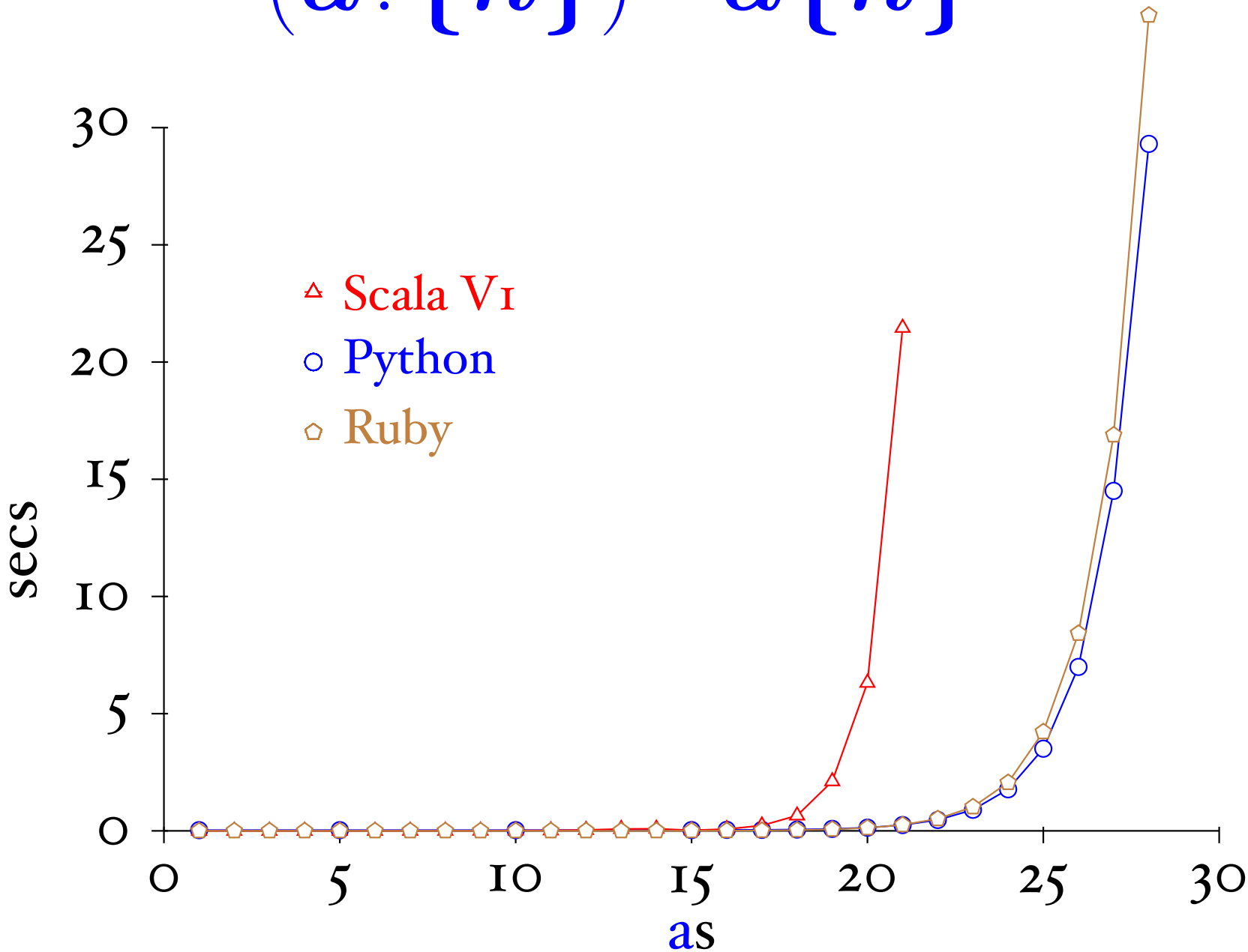
- $P$ holds for the empty string, and
- $P$ holds for the string $c :: s$ under the assumption that $P$ already holds for $s$

# Proofs about Strings (2)

We can finally prove

$$matcher(r, s) \text{ if and only if } s \in L(r)$$

# $(a?\{n\}) \cdot a\{n\}$

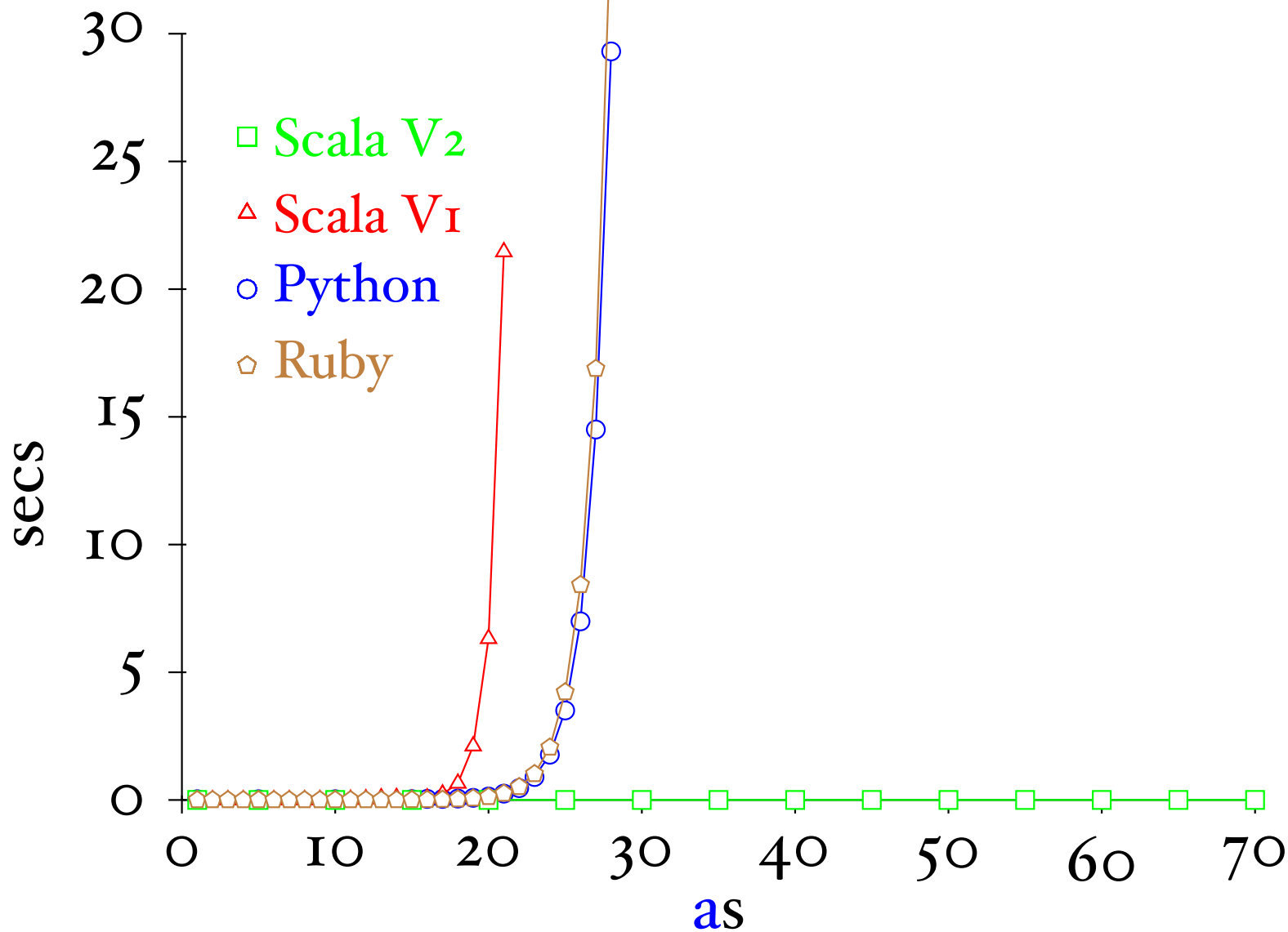


# Problem

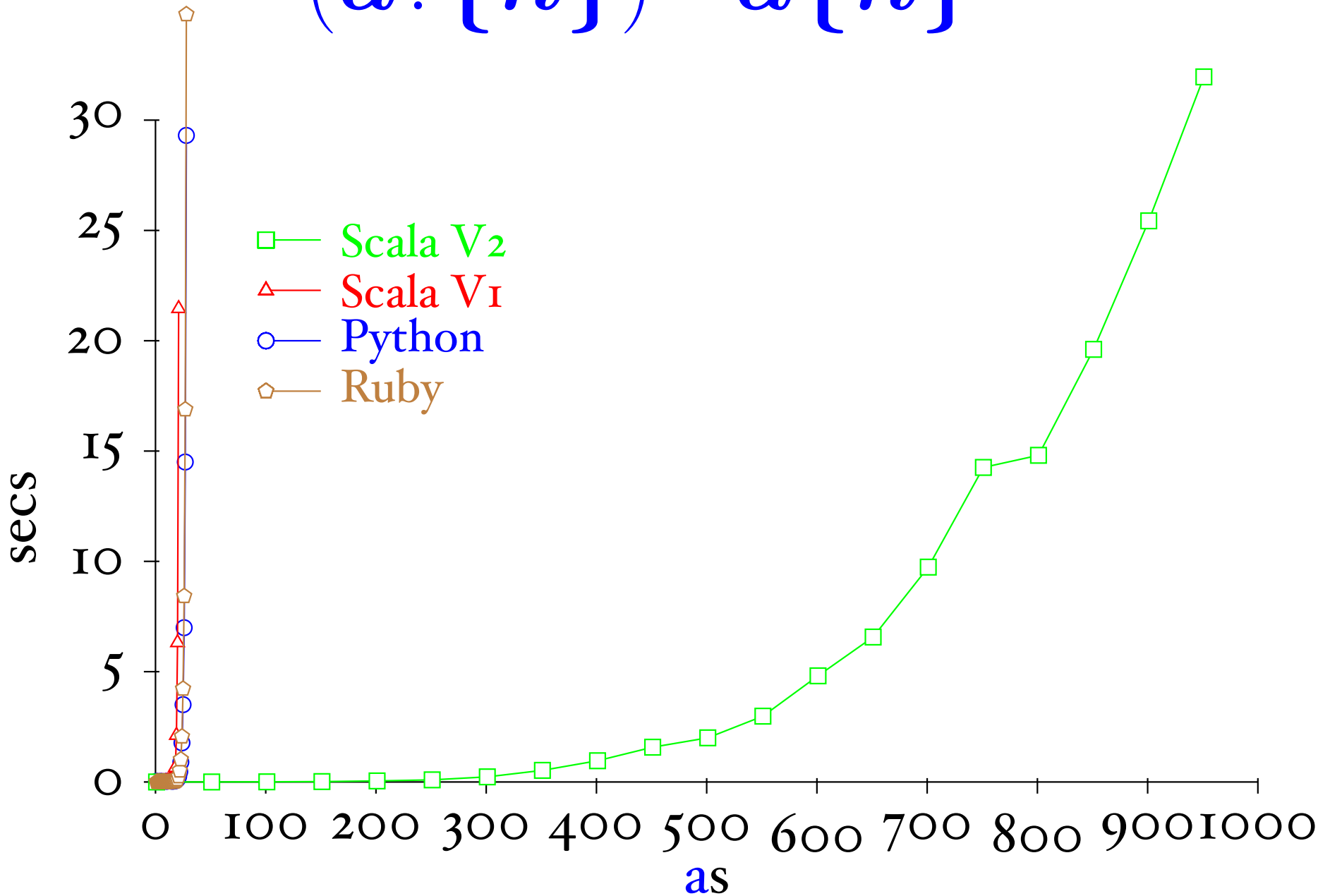
We represented "n-times" as a sequence regular expression:

$$1:$$

# $(a?\{n\}) \cdot a\{n\}$


30
25
20
15
10
5
0
secs
Scala V2
Scala V1
Python
Ruby
0
10
20
30
40
50
60
70
as

# $(a?\{n\}) \cdot a\{n\}$


Scala V2
Scala V1
Python
Ruby
secs
0
5
10
15
20
25
30
0
100
200
300
400
500
600
700
800
900
1000
as

# Regular Languages

A language (set of strings) is regular iff there exists a regular expression that recognises all its strings.

# Automata

A deterministic finite automaton consists of:

- a set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition function

  which takes a state as argument and a character and produces a new state

  this function might not always be defined

# The Rexp Matcher

```
1  abstract class Parser[I, T] {
2    def parse(ts: I): Set[(T, I)]
3
4    def parse_all(ts: I) : Set[T] =
5      for ((head, tail) <- parse(ts); if (tail.isEmpty))
6        yield head
7
8    def || (right : => Parser[I, T]) : Parser[I, T] =
9        new AltParser(this, right)
10   def ==>[S] (f: => T => S) : Parser [I, S] =
11       new FunParser(this, f)
12    def ~[S] (right : => Parser[I, S]) : Parser[I, (T, S)] =
13       new SeqParser(this, right)
14 }
```