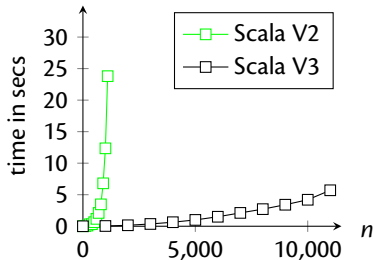
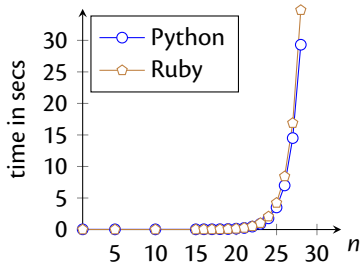


# Compilers and Formal Languages (2)

Email: christian.urban at kcl.ac.uk  
Office Hours: Thursdays 12 – 14  
Location: N7.07 (North Wing, Bush House)  
Slides & Progs: KEATS (also homework is there)

# Lets Implement an Efficient Regular Expression Matcher

Graphs:  $a^{\{n\}} \cdot a^{\{n\}}$  and strings  $\underbrace{a \dots a}_n$



In the handouts is a similar graph for  $(a^*)^* \cdot b$  and Java 8, JavaScript and Python.

# (Basic) Regular Expressions

Their inductive definition:

$r ::= 0$	nothing
$1$	empty string / "" / []
$c$	character
$r_1 + r_2$	alternative / choice
$r_1 \cdot r_2$	sequence
$r^*$	star (zero or more)

Q: What about  $r \cdot 0$  ?

# Languages (Sets of Strings)

- A **Language** is a set of strings, for example

$$\{\ [], \text{hello}, \text{foobar}, a, abc \}$$

- **Concatenation** for strings and languages

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

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

For example  $A = \{\text{foo}, \text{bar}\}$ ,  $B = \{a, b\}$

$$A @ B = \{\text{fooa}, \text{foob}, \text{bar a}, \text{bar b}\}$$

# Two Corner Cases

$$A @ \{ [] \} = ?$$

# Two Corner Cases

$$A @ \{ [] \} = ?$$

$$A @ \{ \} = ?$$

# The Meaning of a Regular Expression

...all the strings a regular expression can match.

$$\begin{aligned}L(\mathbf{0}) &\stackrel{\text{def}}{=} \{\} \\L(\mathbf{1}) &\stackrel{\text{def}}{=} \{\epsilon\} \\L(c) &\stackrel{\text{def}}{=} \{[c]\} \\L(r_1 + r_2) &\stackrel{\text{def}}{=} L(r_1) \cup L(r_2) \\L(r_1 \cdot r_2) &\stackrel{\text{def}}{=} L(r_1) @ L(r_2) \\L(r^*) &\stackrel{\text{def}}{=} \end{aligned}$$

$L$  is a function from regular expressions to sets of strings (languages):

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

# The Power Operation

- The ***n*th Power** of a language:

$$A^0 \stackrel{\text{def}}{=} \{\epsilon\}$$
$$A^{n+1} \stackrel{\text{def}}{=} A @ A^n$$

For example

$$A^4 = A @ A @ A @ A \quad (@ \{\epsilon\})$$
$$A^1 = A \quad (@ \{\epsilon\})$$
$$A^0 = \{\epsilon\}$$



# Homework Question

- Say  $A = \{[a], [b], [c], [d]\}$ .

How many strings are in  $A^4$ ?

# Homework Question

- Say  $A = \{[a], [b], [c], [d]\}$ .

How many strings are in  $A^4$ ?

What if  $A = \{[a], [b], [c], []\}$ ;  
how many strings are then in  $A^4$ ?

# The Star Operation

- The **Kleene Star** of a language:

$$A^* \stackrel{\text{def}}{=} \bigcup_{0 \leq n} A^n$$

This expands to

$$A^0 \cup A^1 \cup A^2 \cup A^3 \cup A^4 \cup \dots$$

or

$$\{\epsilon\} \cup A \cup A @ A \cup A @ A @ A \cup A @ A @ A @ A \cup \dots$$

# The Meaning of a Regular Expression

...all the strings a regular expression can match.

$$\begin{aligned}L(\mathbf{0}) &\stackrel{\text{def}}{=} \{\} \\L(\mathbf{1}) &\stackrel{\text{def}}{=} \{\epsilon\} \\L(c) &\stackrel{\text{def}}{=} \{[c]\} \\L(r_1 + r_2) &\stackrel{\text{def}}{=} L(r_1) \cup L(r_2) \\L(r_1 \cdot r_2) &\stackrel{\text{def}}{=} L(r_1) @ L(r_2) \\L(r^*) &\stackrel{\text{def}}{=} (L(r))^\star \stackrel{\text{def}}{=} \bigcup_{0 \leq n} L(r)^n\end{aligned}$$

# When Are Two Regular Expressions Equivalent?

Two regular expressions  $r_1$  and  $r_2$  are equivalent provided:

$$r_1 \equiv r_2 \stackrel{\text{def}}{=} L(r_1) = L(r_2)$$

# Some Concrete Equivalences

$$(a + b) + c \equiv a + (b + c)$$

$$a + a \equiv a$$

$$a + b \equiv b + a$$

$$(a \cdot b) \cdot c \equiv a \cdot (b \cdot c)$$

$$c \cdot (a + b) \equiv (c \cdot a) + (c \cdot b)$$

# Some Concrete Equivalences

$$(a + b) + c \equiv a + (b + c)$$

$$a + a \equiv a$$

$$a + b \equiv b + a$$

$$(a \cdot b) \cdot c \equiv a \cdot (b \cdot c)$$

$$c \cdot (a + b) \equiv (c \cdot a) + (c \cdot b)$$

$$a \cdot a \not\equiv a$$

$$a + (b \cdot c) \not\equiv (a + b) \cdot (a + c)$$

# Some Corner Cases

$$\begin{array}{l} a \cdot 0 \not\equiv a \\ a + 1 \not\equiv a \\ 1 \equiv 0^* \\ 1^* \equiv 1 \\ 0^* \not\equiv 0 \end{array}$$



# Some Simplification Rules

$$r + 0 \equiv r$$

$$0 + r \equiv r$$

$$r \cdot 1 \equiv r$$

$$1 \cdot r \equiv r$$

$$r \cdot 0 \equiv 0$$

$$0 \cdot r \equiv 0$$

$$r + r \equiv r$$

# The Specification for Matching

A regular expression  $r$  matches a string  $s$  provided:

$$s \in L(r)$$

...and the point of the this lecture is to decide this problem as fast as possible (unlike Python, Ruby, Java etc)

# Questions?

homework (written exam 80%)

coursework (20%; first one today)

submission Fridays @ 18:00 – accepted until Mondays

# Semantic Derivative

- The **Semantic Derivative** of a language w.r.t. to a character  $c$ :

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

For  $A = \{\text{foo}, \text{bar}, \text{frak}\}$  then

$$\text{Der } f A = \{\text{oo}, \text{rak}\}$$

$$\text{Der } b A = \{\text{ar}\}$$

$$\text{Der } a A = \{\}$$

# Semantic Derivative

- The **Semantic Derivative** of a language w.r.t. to a character  $c$ :

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

For  $A = \{\text{foo}, \text{bar}, \text{frak}\}$  then

$$\text{Der } f A = \{\text{oo}, \text{rak}\}$$

$$\text{Der } b A = \{\text{ar}\}$$

$$\text{Der } a A = \{\}$$

We can extend this definition to strings

$$\text{Der } s A = \{s' \mid s @ s' \in A\}$$

# Brzozowski's Algorithm (1)

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

$$\text{nullable}(\mathbf{0}) \stackrel{\text{def}}{=} \text{false}$$

$$\text{nullable}(\mathbf{1}) \stackrel{\text{def}}{=} \text{true}$$

$$\text{nullable}(c) \stackrel{\text{def}}{=} \text{false}$$

$$\text{nullable}(r_1 + r_2) \stackrel{\text{def}}{=} \text{nullable}(r_1) \vee \text{nullable}(r_2)$$

$$\text{nullable}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \text{nullable}(r_1) \wedge \text{nullable}(r_2)$$

$$\text{nullable}(r^*) \stackrel{\text{def}}{=} \text{true}$$

# The Derivative of a Rexp

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

$der\ c\ r$  gives the answer, Brzozowski 1964

# The Derivative of a Rexp

$$\text{der } c(0) \stackrel{\text{def}}{=} 0$$

$$\text{der } c(1) \stackrel{\text{def}}{=} 0$$

$$\text{der } c(d) \stackrel{\text{def}}{=} \text{if } c = d \text{ then } 1 \text{ else } 0$$

$$\text{der } c(r_1 + r_2) \stackrel{\text{def}}{=} \text{der } c r_1 + \text{der } c r_2$$

$$\text{der } c(r_1 \cdot r_2) \stackrel{\text{def}}{=} \text{if nullable}(r_1) \\ \text{then } (\text{der } c r_1) \cdot r_2 + \text{der } c r_2 \\ \text{else } (\text{der } c r_1) \cdot r_2$$

$$\text{der } c(r^*) \stackrel{\text{def}}{=} (\text{der } c r) \cdot (r^*)$$



# The Derivative of a Rexp

$$\text{der } c \ (0) \quad \stackrel{\text{def}}{=} \ 0$$

$$\text{der } c \ (1) \quad \stackrel{\text{def}}{=} \ 0$$

$$\text{der } c \ (d) \quad \stackrel{\text{def}}{=} \ \text{if } c = d \ \text{then } 1 \ \text{else } 0$$

$$\text{der } c \ (r_1 + r_2) \quad \stackrel{\text{def}}{=} \ \text{der } c \ r_1 + \text{der } c \ r_2$$

$$\text{der } c \ (r_1 \cdot r_2) \quad \stackrel{\text{def}}{=} \ \text{if nullable}(r_1) \\ \text{then } (\text{der } c \ r_1) \cdot r_2 + \text{der } c \ r_2 \\ \text{else } (\text{der } c \ r_1) \cdot r_2$$

$$\text{der } c \ (r^*) \quad \stackrel{\text{def}}{=} \ (\text{der } c \ r) \cdot (r^*)$$

$$\text{ders } [] \ r \quad \stackrel{\text{def}}{=} \ r$$

$$\text{ders } (c :: s) \ r \quad \stackrel{\text{def}}{=} \ \text{ders } s \ (\text{der } c \ r)$$

# Examples

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

$$\text{der } a \ r = ?$$

$$\text{der } b \ r = ?$$

$$\text{der } c \ r = ?$$

# The Brzozowski Algorithm

*matches*  $r s \stackrel{\text{def}}{=} \text{nullable}(\text{ders } s r)$

# Brzowski: An Example

Does  $r_1$  match  $abc$ ?

Step 1: build derivative of  $a$  and  $r_1$  ( $r_2 = \text{der } a r_1$ )

Step 2: build derivative of  $b$  and  $r_2$  ( $r_3 = \text{der } b r_2$ )

Step 3: build derivative of  $c$  and  $r_3$  ( $r_4 = \text{der } c r_3$ )

Step 4: the string is exhausted: ( $\text{nullable}(r_4)$ )  
test whether  $r_4$  can recognise  
the empty string

Output: result of the test  
 $\Rightarrow$   $\text{true}$  or  $\text{false}$

# The Idea of the Algorithm

If we want to recognise the string  $abc$  with regular expression  $r_1$  then

1  $Der a(L(r_1))$

# The Idea of the Algorithm

If we want to recognise the string  $abc$  with regular expression  $r_1$  then

- 1  $Der a (L(r_1))$
- 2  $Der b (Der a (L(r_1)))$

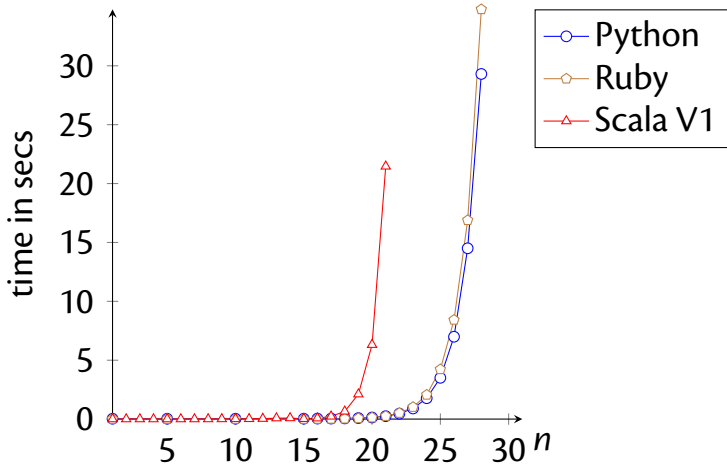
# The Idea of the Algorithm

If we want to recognise the string  $abc$  with regular expression  $r_1$  then

- 1  $Der a (L(r_1))$
- 2  $Der b (Der a (L(r_1)))$
- 3  $Der c (Der b (Der a (L(r_1))))$
- 4 finally we test whether the empty string is in this set; same for  $Der abc (L(r_1))$ .

The matching algorithm works similarly, just over regular expressions instead of sets.

# Oops... $a^{a^n} \cdot a^n$





# A Problem

We represented the “n-times”  $a^{\{n\}}$  as a sequence regular expression:

1:  $a$

2:  $a \cdot a$

3:  $a \cdot a \cdot a$

...

13:  $a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a$

...

20:

This problem is aggravated with  $a^?$  being represented as  $a + 1$ .

# Solving the Problem

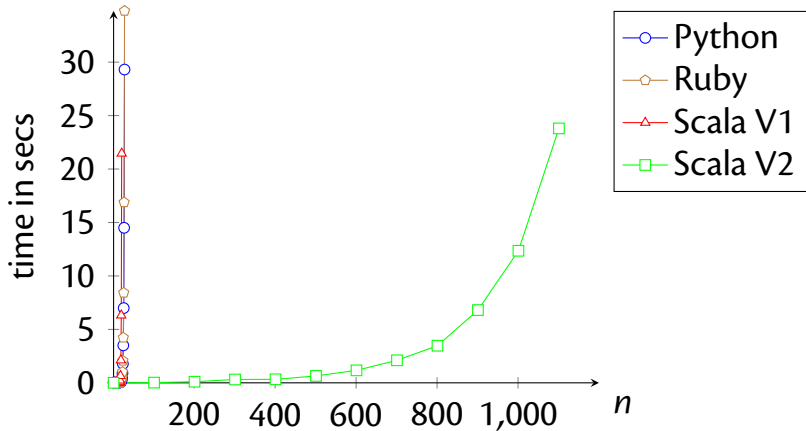
What happens if we extend our regular expressions with explicit constructors

$$r ::= \dots$$
$$| r^{\{n\}}$$
$$| r^?$$

What is their meaning?

What are the cases for *nullable* and *der*?

# Brzozowski: $a^{\{n\}} \cdot a^{\{n\}}$



# Examples

Recall the example of  $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$  with

$$\text{der } a r = ((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r$$

$$\text{der } b r = ((\mathbf{0} \cdot b) + \mathbf{1}) \cdot r$$

$$\text{der } c r = ((\mathbf{0} \cdot b) + \mathbf{0}) \cdot r$$

What are these regular expressions equivalent to?

# Simplification Rules

$$r + 0 \Rightarrow r$$

$$0 + r \Rightarrow r$$

$$r \cdot 1 \Rightarrow r$$

$$1 \cdot r \Rightarrow r$$

$$r \cdot 0 \Rightarrow 0$$

$$0 \cdot r \Rightarrow 0$$

$$r + r \Rightarrow r$$

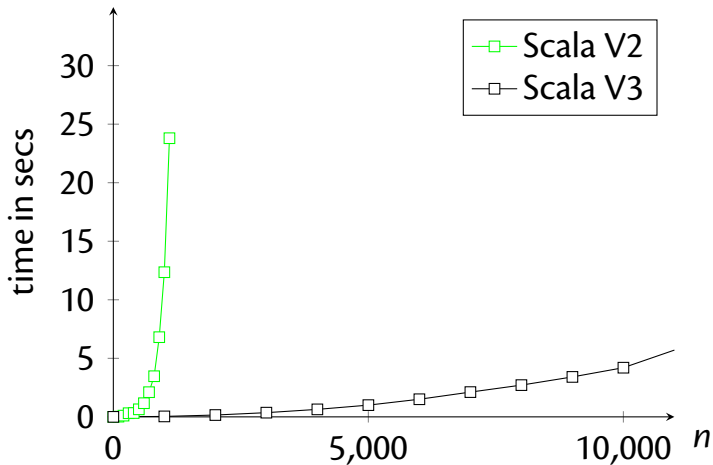
```
def ders(s: List[Char], r: Rexp) : Rexp = s match {  
  case Nil => r  
  case c::s => ders(s, simp(der(c, r)))  
}
```

```

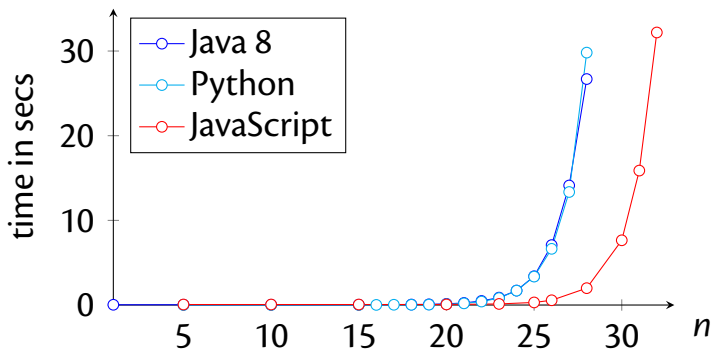
def simp(r: Rexp) : Rexp = r match {
  case ALT(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, r2s) => r2s
      case (r1s, ZERO) => r1s
      case (r1s, r2s) =>
        if (r1s == r2s) r1s else ALT(r1s, r2s)
    }
  }
  case SEQ(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, _) => ZERO
      case (_, ZERO) => ZERO
      case (ONE, r2s) => r2s
      case (r1s, ONE) => r1s
      case (r1s, r2s) => SEQ(r1s, r2s)
    }
  }
  case r => r
}

```

# Brzozowski: $a^{\{n\}} \cdot a^{\{n\}}$



# Another Example in Java 8, Python and JavaScript

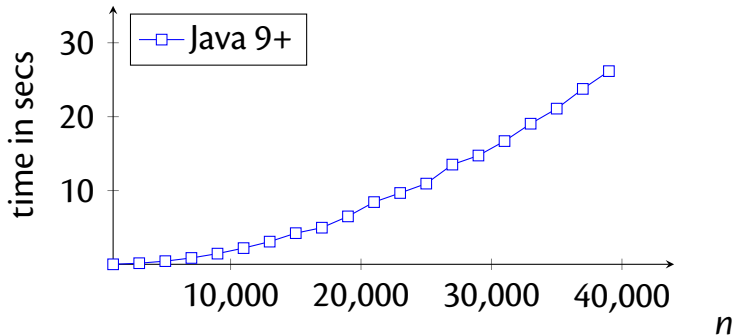


Regex:  $(a^*)^* \cdot b$

Strings of the form  $\underbrace{a \dots a}_n$



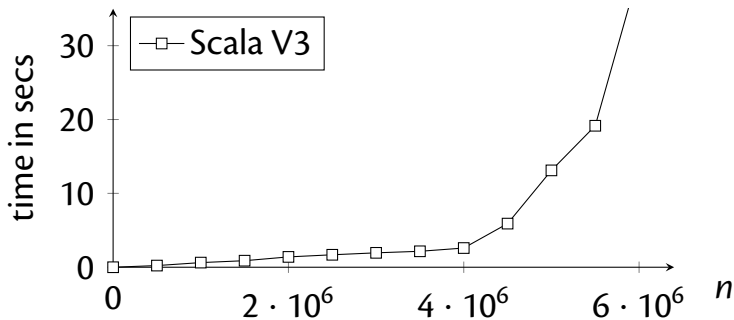
# Same Example in Java 9+



Regex:  $(a^*)^* \cdot b$

Strings of the form  $\underbrace{a \dots a}_n$

# ...and with Brzozowski



Regex:  $(a^*)^* \cdot b$

Strings of the form  $\underbrace{a \dots a}_n$

# What is good about this Alg.

- extends to most regular expressions, for example  $\sim r$  (next slide)
- is easy to implement in a functional language (slide after)
- the algorithm is already quite old; there is still work to be done to use it as a tokenizer (that is relatively new work)
- we can prove its correctness...(several slides later)

# Negation of Regular Expr's

- $\sim r$  (everything that  $r$  cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} not(nullable(r))$
- $derc(\sim r) \stackrel{\text{def}}{=} \sim(derc r)$

# Negation of Regular Expr's

- $\sim r$  (everything that  $r$  cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} \text{not}(nullable(r))$
- $derc(\sim r) \stackrel{\text{def}}{=} \sim(derc r)$

Used often for recognising comments:

$$/ \cdot * \cdot (\sim ([a-z]^* \cdot * \cdot / \cdot [a-z]^*)) \cdot * \cdot /$$

# Coursework

## Strand 1:

- Submission on Friday 11 October accepted until Monday 14 @ 18:00
- source code needs to be submitted as well
- you can re-use my Scala code from KEATS or use any programming language you like
- <https://nms.kcl.ac.uk/christian.urban/ProgInScala2ed.pdf>

# Proofs about Rexprs

Remember their inductive definition:

$$r ::= \begin{array}{l} 0 \\ 1 \\ c \\ r_1 \cdot r_2 \\ r_1 + r_2 \\ r^* \end{array}$$

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

# Proofs about Rexp (2)

- $P$  holds for  $0$ ,  $1$  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:

$nullable(r)$  if and only if  $[\ ] \in L(r)$

# Proofs about Rexp (4)

$$\text{rev}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$$

$$\text{rev}(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1}$$

$$\text{rev}(c) \stackrel{\text{def}}{=} c$$

$$\text{rev}(r_1 + r_2) \stackrel{\text{def}}{=} \text{rev}(r_1) + \text{rev}(r_2)$$

$$\text{rev}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \text{rev}(r_2) \cdot \text{rev}(r_1)$$

$$\text{rev}(r^*) \stackrel{\text{def}}{=} \text{rev}(r)^*$$

We can prove

$$L(\text{rev}(r)) = \{s^{-1} \mid s \in L(r)\}$$

by induction on  $r$ .

# Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$

$$\Leftrightarrow \square \in \text{Ders } s (L(r))$$

# Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$

$$\Leftrightarrow [] \in Ders\ s\ (L(r))$$

- if we can show  $Ders\ s\ (L(r)) = L(ders\ s\ r)$  we have

$$\Leftrightarrow [] \in L(ders\ s\ r)$$

$$\Leftrightarrow nullable(ders\ s\ r)$$

$$\stackrel{\text{def}}{=} matches\ s\ r$$

# Proofs about Rexp (5)

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

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

We can prove

$$L(\text{der}\ c\ r) = Der\ c\ (L(r))$$

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 then prove

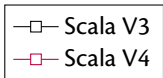
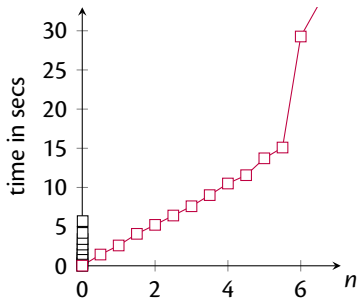
$$\text{Ders } s (L(r)) = L(\text{ders } s r)$$

We can finally prove

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

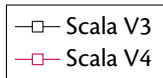
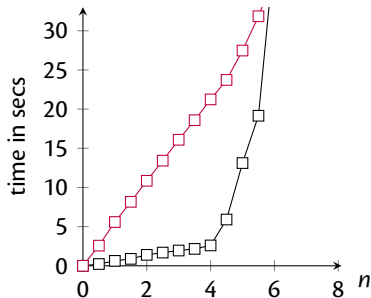
# Epilogue

Graph:  $a^{2\{n\}} \cdot a\{n\}$



$\cdot 10^6$

Graph:  $(a^*)^* \cdot b$

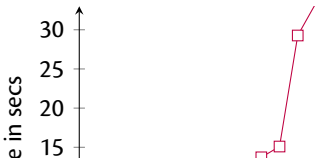


$\cdot 10^6$

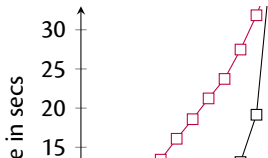


# Epilogue

Graph:  $a^{? \{n\}} \cdot a^{\{n\}}$



Graph:  $(a^*)^* \cdot b$



```
def ders2(s: List[Char], r: Rexp) : Rexp = (s, r) match {  
  case (Nil, r) => r  
  case (s, ZERO) => ZERO  
  case (s, ONE) => if (s == Nil) ONE else ZERO  
  case (s, CHAR(c)) => if (s == List(c)) ONE else  
    if (s == Nil) CHAR(c) else ZERO  
  case (s, ALT(r1, r2)) => ALT(ders2(s, r1), ders2(s, r2))  
  case (c::s, r) => ders2(s, simp(der(c, r)))  
}
```

# Another Homework Question

- How many basic regular expressions are there to match the string *abcd*?

# Another Homework Question

- How many basic regular expressions are there to match the string *abcd*?
- How many if they cannot include **1** and **0**?

# Another Homework Question

- How many basic regular expressions are there to match the string *abcd*?
- How many if they cannot include **1** and **0**?
- How many if they are also not allowed to contain stars?

# Another Homework Question

- How many basic regular expressions are there to match the string *abcd*?
- How many if they cannot include **1** and **0**?
- How many if they are also not allowed to contain stars?
- How many if they are also not allowed to contain **\_ + \_**?