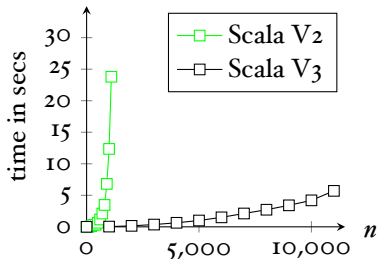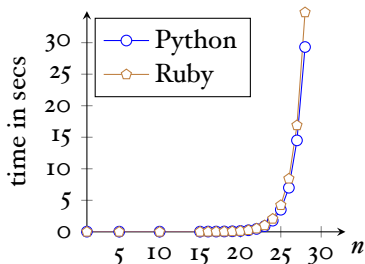# Compilers and Formal Languages (2)

Email: christian.urban at kcl.ac.uk
Office: N7.07 (North Wing, Bush House)
Slides: KEATS (also homework is there)

Lets Implement an Efficient Regular Expression Matcher

Graphs: $a^{?\{n\}} \cdot a^{\{n\}}$ and strings $\underbrace{a \ldots a}_{n}$

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

Evil Regular Expressions

- **R**egular **e**xpression **D**enial **o**f **S**ervice (ReDoS)

- Evil regular expressions

  - $a^{?\{n\}} \cdot a^{\{n\}}$
  - $(a^*)^*$
  - $([a\text{-}z]^+)^*$
  - $(a + a \cdot a)^*$
  - $(a + a^?)^*$

- sometimes also called catastrophic backtracking

Languages

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

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

- **Concatenation** of strings and languages

$$foo \,@\, bar \;=\; foobar$$

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

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

$$A \,@\, B = \{fooa, foob, bara, barb\}$$

The Power Operation

- The **nth Power** of a language:

$$A^0 \overset{\text{def}}{=} \{[]\}$$
$$A^{n+1} \overset{\text{def}}{=} A @ A^n$$

For example

$$A^4 = A @ A @ A @ A \qquad (@ \{[]\})$$
$$A^1 = A \qquad (@ \{[]\})$$
$$A^0 = \{[]\}$$

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:

$$A\star \overset{\text{def}}{=} \bigcup_{0 \leq n} A^n$$

This expands to

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

or

$$\{[]\} \cup A \cup A@A \cup A@A@A \cup A@A@A@A \cup \ldots$$

The Meaning of a Regular Expression

$$L(\mathbf{0}) \quad \stackrel{\text{def}}{=} \quad \{\}$$
$$L(\mathbf{1}) \quad \stackrel{\text{def}}{=} \quad \{[]\}$$
$$L(c) \quad \stackrel{\text{def}}{=} \quad \{[c]\}$$
$$L(r_1 + r_2) \quad \stackrel{\text{def}}{=} \quad L(r_1) \cup L(r_2)$$
$$L(r_1 \cdot r_2) \quad \stackrel{\text{def}}{=} \quad \{s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2)\}$$
$$L(r^*) \quad \stackrel{\text{def}}{=} \quad (L(r))\star \quad \stackrel{\text{def}}{=} \quad \bigcup_{0 \leq n} L(r)^n$$

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

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

Questions?

homework (written exam 80%)
coursework (20%; first one today)
submission Fridays @ 18:00 – accepted until
Mondays

Semantic Derivative

- The **Semantic Derivative** of a <u>language</u> w.r.t. to a character c:

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

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

$$Der\,f\,A = \{oo, rak\}$$
$$Der\,b\,A = \{ar\}$$
$$Der\,a\,A = \{\}$$

Semantic Derivative

- The **Semantic Derivative** of a <u>language</u> w.r.t. to a character c:

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

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

$$Der\,f\,A = \{oo, rak\}$$
$$Der\,b\,A = \{ar\}$$
$$Der\,a\,A = \{\}$$

We can extend this definition to strings

$$Ders\,s\,A = \{s' \mid s@s' \in A\}$$

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)

Regular Expressions

Their inductive definition:

$$
\begin{array}{lll}
r & ::= & \mathbf{0} \qquad\qquad\qquad \text{nothing} \\
  & | & \mathbf{1} \qquad\qquad\qquad \text{empty string / ”” / []} \\
  & | & c \qquad\qquad\qquad \text{single character} \\
  & | & r_1 \cdot r_2 \qquad\qquad \text{sequence} \\
  & | & r_1 + r_2 \qquad\qquad \text{alternative / choice} \\
  & | & r^* \qquad\qquad\qquad \text{star (zero or more)}
\end{array}
$$

Th

```scala
abstract class Rexp
case object ZERO extends Rexp
case object ONE extends Rexp
case class CHAR(c: Char) extends Rexp
case class ALT(r1: Rexp, r2: Rexp) extends Rexp
case class SEQ(r1: Rexp, r2: Rexp) extends Rexp
case class STAR(r: Rexp) extends Rexp
```

$$r \quad ::= \quad \mathbf{0} \qquad \qquad \text{nothing}$$

$$\mid \quad \mathbf{1} \qquad \qquad \text{empty string / ''''' / } []$$

$$\mid \quad c \qquad \qquad \text{single character}$$

$$\mid \quad r_1 \cdot r_2 \qquad \text{sequence}$$

$$\mid \quad r_1 + r_2 \qquad \text{alternative / choice}$$

$$\mid \quad r^* \qquad \qquad \text{star (zero or more)}$$

When Are Two Regular Expressions Equivalent?

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

Concrete Equivalences

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

Concrete Equivalences

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

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

Corner Cases

$$a \cdot \mathbf{0} \;\not\equiv\; a$$

$$a + \mathbf{1} \;\not\equiv\; a$$

$$\mathbf{1} \;\equiv\; \mathbf{0}^*$$

$$\mathbf{1}^* \;\equiv\; \mathbf{1}$$

$$\mathbf{0}^* \;\not\equiv\; \mathbf{0}$$

Simplification Rules

$$r + \mathbf{0} \;\equiv\; r$$
$$\mathbf{0} + r \;\equiv\; r$$
$$r \cdot \mathbf{1} \;\equiv\; r$$
$$\mathbf{1} \cdot r \;\equiv\; r$$
$$r \cdot \mathbf{0} \;\equiv\; \mathbf{0}$$
$$\mathbf{0} \cdot r \;\equiv\; \mathbf{0}$$
$$r + r \;\equiv\; r$$

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 _ + _?

Brzozowski's Algorithm (1)

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

$$nullable(\mathbf{0}) \;\overset{def}{=}\; false$$

$$nullable(\mathbf{1}) \;\overset{def}{=}\; true$$

$$nullable(c) \;\overset{def}{=}\; false$$

$$nullable(r_1 + r_2) \;\overset{def}{=}\; nullable(r_1) \lor nullable(r_2)$$

$$nullable(r_1 \cdot r_2) \;\overset{def}{=}\; nullable(r_1) \land nullable(r_2)$$

$$nullable(r^*) \;\overset{def}{=}\; true$$

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

$$der\, c\ (\mathbf{0}) \quad \overset{\text{def}}{=}\quad \mathbf{0}$$

$$der\, c\ (\mathbf{1}) \quad \overset{\text{def}}{=}\quad \mathbf{0}$$

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

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

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

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

The Derivative of a Rexp

$$der\, c\, (\mathbf{0}) \overset{\text{def}}{=} \mathbf{0}$$

$$der\, c\, (\mathbf{1}) \overset{\text{def}}{=} \mathbf{0}$$

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

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

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

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

$$ders\, [\,]\, r \overset{\text{def}}{=} r$$

$$ders\, (c :: s)\, r \overset{\text{def}}{=} ders\, s\, (der\, c\, r)$$

Examples

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

$$der\,a\,r = ?$$
$$der\,b\,r = ?$$
$$der\,c\,r = ?$$

The Brzozowski Algorithm

$$matches\ r\ s \overset{\text{def}}{=} nullable\,(ders\ s\ r)$$

Brzozowski: An Example

Does r_1 match *abc*?

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

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

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

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

Output: result of the test
$\Rightarrow$ *true* or *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 *Ders abc* $(L(r_1))$.

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

Oops... $a^{?\{n\}} \cdot a^{\{n\}}$
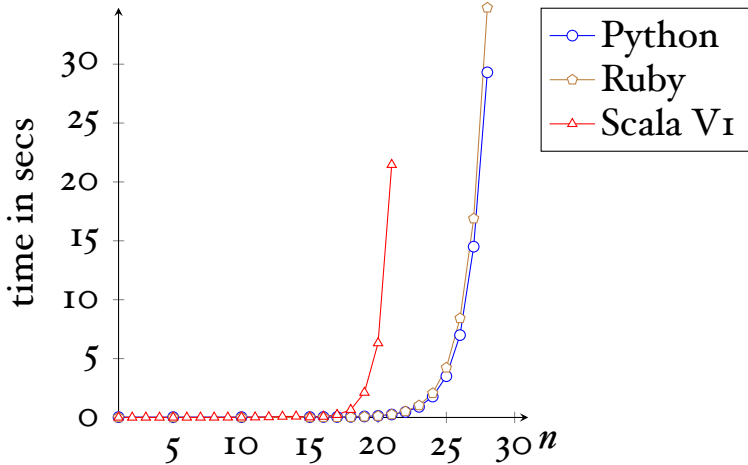
A Problem

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 $a + \mathbf{1}$.

Solving the Problem

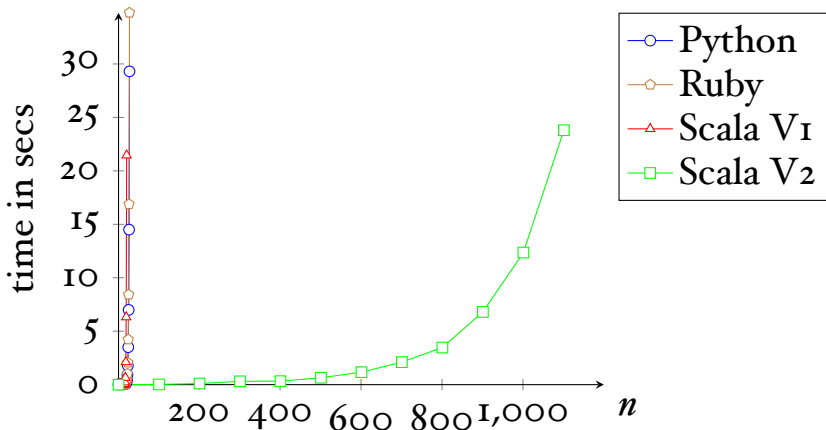What happens if we extend our regular expressions with explicit constructors

$$r ::= \quad \ldots$$
$$| \quad r^{\{n\}}$$
$$| \quad r^{?}$$

What is their meaning?
What are the cases for *nullable* and *der*?

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

Examples

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

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

What are these regular expressions equivalent to?

Simplification Rules

$$r + \mathbf{0} \;\Rightarrow\; r$$
$$\mathbf{0} + r \;\Rightarrow\; r$$
$$r \cdot \mathbf{1} \;\Rightarrow\; r$$
$$\mathbf{1} \cdot r \;\Rightarrow\; r$$
$$r \cdot \mathbf{0} \;\Rightarrow\; \mathbf{0}$$
$$\mathbf{0} \cdot r \;\Rightarrow\; \mathbf{0}$$
$$r + r \;\Rightarrow\; r$$

```
def ders(s: List[Char], r: Rexp) : Rexp = s match {
  case Nil => r
  case c::s => ders(s, simp(der(c, r)))
}
```
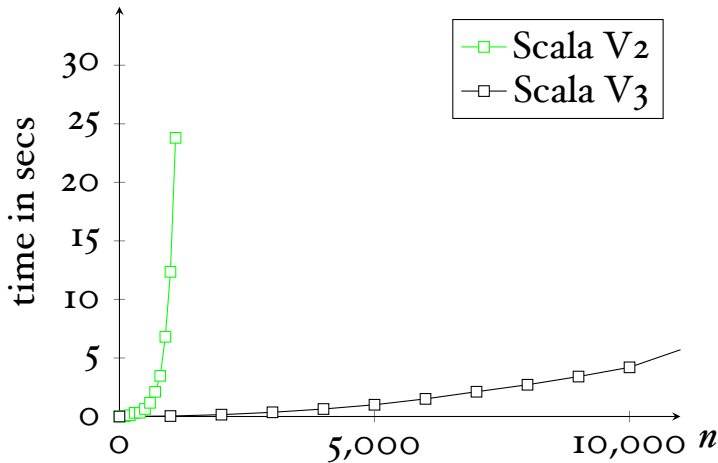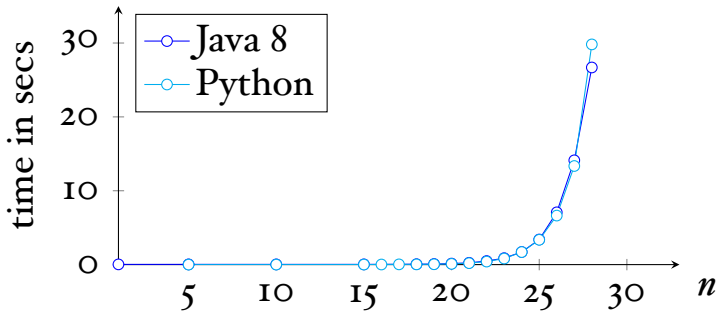
```scala
def simp(r: Rexp) : Rexp = r match {
  case ALT(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, r2s) => r2s
      case (r1s, ZERO) => r1s
      case (r1s, r2s) =>
        if (r1s == r2s) r1s else ALT(r1s, r2s)
    }
  }
  case SEQ(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, _) => ZERO
      case (_, ZERO) => ZERO
      case (ONE, r2s) => r2s
      case (r1s, ONE) => r1s
      case (r1s, r2s) => SEQ(r1s, r2s)
    }
  }
  case r => r
}
```
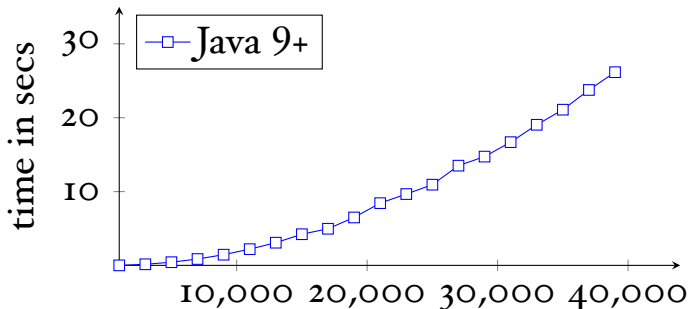
Brzozowski: $a^{?\{n\}} \cdot a^{\{n\}}$

Another Example in Java 8 and Python

Regex: $(a^*)^* \cdot b$
Strings of the form $\underbrace{a \ldots a}_{n}$

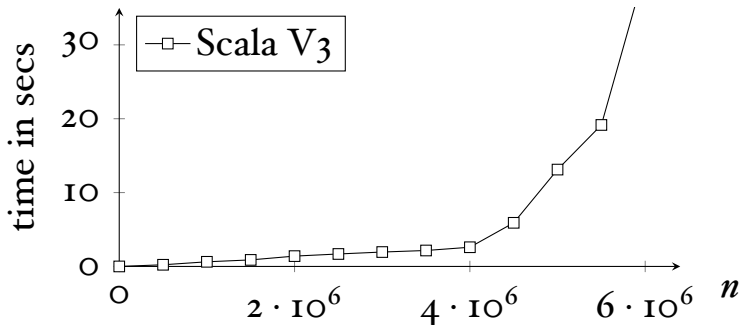# Same Example in Java 9+

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

Strings of the form $\underbrace{a \ldots a}_{n}$

and with Brzozowski

Regex: $(a^*)^* \cdot b$
Strings of the form $\underbrace{a \ldots a}_{n}$

What is good about this Alg.

- extends to most regular expressions, for example $\sim r$

- is easy to implement in a functional language

- 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...

Negation of Regular Expr's

- $\sim r$ (everything that r cannot recognise)

- $L(\sim r) \overset{\text{def}}{=} \mathit{UNIV} - L(r)$

- $\mathit{nullable}(\sim r) \overset{\text{def}}{=} \mathit{not}\,(\mathit{nullable}(r))$

- $\mathit{der}\,c\,(\sim r) \overset{\text{def}}{=} \sim (\mathit{der}\,c\,r)$

Negation of Regular Expr's

- $\sim r$ (everything that r cannot recognise)

- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$

- $nullable(\sim r) \stackrel{\text{def}}{=} not\,(nullable(r))$

- $der\,c\,(\sim r) \stackrel{\text{def}}{=} \sim (der\,c\,r)$

Used often for recognising comments:

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

Coursework

Strand 1:

- Submission on Friday 12 October
  accepted until Monday 15 @ 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 Rexps

Remember their inductive definition:

$$r ::= \quad \mathbf{0}$$
$$| \quad \mathbf{1}$$
$$| \quad c$$
$$| \quad r_1 \cdot r_2$$
$$| \quad r_1 + r_2$$
$$| \quad r^*$$

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

Proofs about Rexp (2)

- P holds for $\mathbf{0}$, $\mathbf{1}$ 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:

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

Proofs about Rexp (4)

$$rev(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$$
$$rev(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1}$$
$$rev(c) \stackrel{\text{def}}{=} c$$
$$rev(r_1 + r_2) \stackrel{\text{def}}{=} rev(r_1) + rev(r_2)$$
$$rev(r_1 \cdot r_2) \stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1)$$
$$rev(r^*) \stackrel{\text{def}}{=} rev(r)^*$$

We can prove

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

by induction on r.

Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$
$$\iff \quad [] \in Ders\, s\, (L(r))$$

Correctness Proof for our Matcher

- We started from

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

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

$$\Leftrightarrow \quad [] \in L(ders\,s\,r)$$
$$\Leftrightarrow \quad nullable(ders\,s\,r)$$
$$\stackrel{\text{def}}{=} \quad matches\,s\,r$$

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

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

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
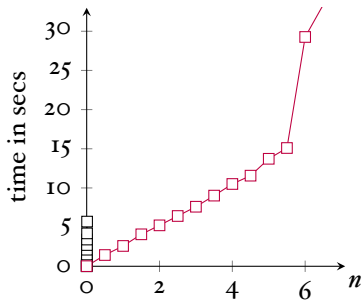
Proofs about Strings (2)

We can then prove

$$Ders\,s\,(L(r)) = L(ders\,s\,r)$$

We can finally prove

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

Epilogue

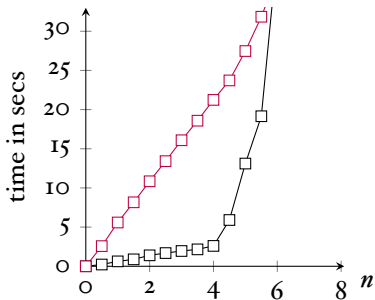Graph: $a^{?\{n\}} \cdot a^{\{n\}}$
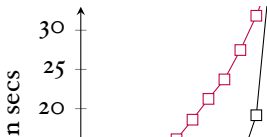

Graph: $(a^*)^* \cdot b$

Epilogue

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

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


```scala
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, r2), ders2(s, r2))
  case (c::s, r) => ders2(s, simp(der(c, r)))
}
```