

Compilers and Formal Languages

Email: christian.urban at kcl.ac.uk
Office Hour: Friday 11:30 – 12:30
Location: N7.07 (North Wing, Bush House)
Slides & Progs: KEATS



Wifi: Lincoln's Inn
Pwd: 0207 4051 393

1 Introduction, Languages	6 While-Language
2 Regular Expressions, Derivatives	7 Compilation, JVM
3 Automata, Regular Languages	8 Compiling Functional Languages
4 Lexing, Tokenising	9 Optimisations
5 Grammars, Parsing	10 LLVM



I try my best, butserver, venue

"if x = 42 then x := x + 1 else x := x - 1"

KEYWORD:

if, then, else,...

WHITESPACE:

" ", \n, \r

IDENTIFIER:

LETTER · (LETTER + DIGIT + $_$)*

NUM:

(NONZERODIGIT · DIGIT*) + 0

NUMBER:

NUM + ("-" · NUM)

OP:

=, :=, +, -, *, %, <, =<,...

COMMENT:

/* · \sim (ALL* · (* /) · ALL*) · */

LETTER: [a-zA-Z]

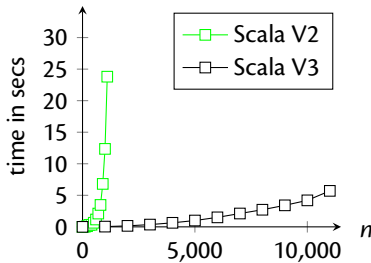
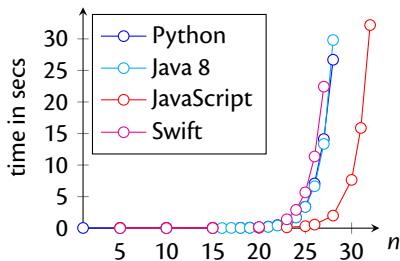
DIGIT: [0-9]

NONZERODIGIT: [1-9]

Let's Implement an Efficient Regular Expression Matcher

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

Graph: $(a^*)^* b$ and strings $\underbrace{a \dots a}_n$



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

(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)

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

$$\begin{array}{rcl} a \cdot 0 & \neq & a \\ a + 1 & \neq & a \\ 1 & \equiv & 0^* \\ 1^* & \equiv & 1 \\ 0^* & \neq & 0 \end{array}$$

Some Simplification Rules

$$r + 0 \equiv r$$

$$0 + r \equiv r$$

$$r \cdot 1 \equiv r$$

$$1 \cdot r \equiv r$$

$$r \cdot 0 \equiv 0$$

$$0 \cdot r \equiv 0$$

$$r + r \equiv r$$

Simplification Example

$$\begin{aligned}((1 \cdot b) + 0) \cdot r &\Rightarrow ((\underline{1 \cdot b}) + 0) \cdot r \\&= (\underline{b + 0}) \cdot r \\&= b \cdot r\end{aligned}$$

Simplification Example

$$((\mathbf{0} \cdot b) + \mathbf{0}) \cdot r \Rightarrow ((\underline{\mathbf{0} \cdot b}) + \mathbf{0}) \cdot r$$

$$= (\underline{\mathbf{0} + \mathbf{0}}) \cdot r$$

$$= \mathbf{0} \cdot r$$

$$= \mathbf{0}$$

Semantic Derivative

- The **Semantic Derivative** of a language 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 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

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

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)

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 ?

$\text{der } c \ r$ gives the answer, Brzozowski 1964

The Derivative of a Rexp

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

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

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

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

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

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

The Derivative of a Rexp

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

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

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

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

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

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

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

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

Examples

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

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

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

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

Derivative Example

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

$$\begin{aligned} \text{der } a ((a \cdot b) + b)^* &\Rightarrow \text{der } a \underline{((a \cdot b) + b)^*} \\ &= (\text{der } a \underline{((a \cdot b) + b)}) \cdot r \\ &= ((\text{der } a \underline{a \cdot b}) + (\text{der } a b)) \cdot r \\ &= (((\text{der } a \underline{a}) \cdot b) + (\text{der } a b)) \cdot r \\ &= ((\mathbf{1} \cdot b) + (\text{der } a \underline{b})) \cdot r \\ &= ((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r \end{aligned}$$

The Brzozowski Algorithm

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

Brzozowski: 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 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.

The Idea with Derivatives

Input: string *abc* and regular expression *r*

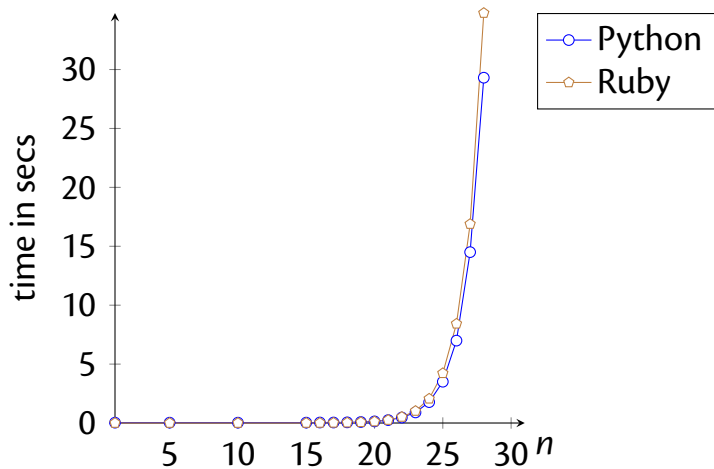
1. *der a r*
2. *der b (der a r)*
3. *der c (der b (der a r))*

The Idea with Derivatives

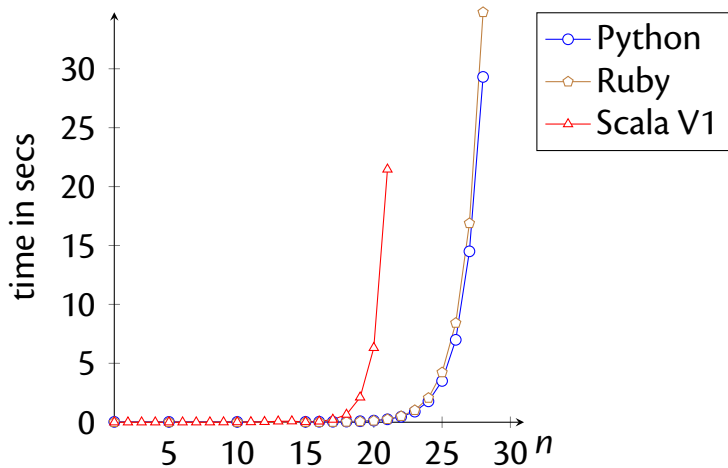
Input: string *abc* and regular expression *r*

1. *der a r*
2. *der b (der a r)*
3. *der c (der b (der a r))*
4. finally check whether the last regular expression can match the empty string

$$a^{\{n\}} \cdot a^{\{n\}}$$



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



A Problem

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

0: 1

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

$$\begin{array}{c} r \quad ::= \quad \dots \\ \quad \quad | \quad r^{\{n\}} \\ \quad \quad | \quad r^? \end{array}$$

What is their meaning?

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

der for n -times

Case $n = 2$ and $r \cdot r$:

$$\begin{aligned} \text{der } c(r \cdot r) &\stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c r) \cdot r + \text{der } c r \\ \text{else } (\text{der } c r) \cdot r \end{array} \end{aligned}$$

der for *n*-times

Case $n = 2$ and $r \cdot r$:

$$\begin{aligned} \textit{der } c(r \cdot r) &\stackrel{\text{def}}{=} \text{if } \textit{nullable}(r) \\ &\quad \text{then } (\textit{der } c r) \cdot r + \textit{der } c r \\ &\quad \text{else } (\textit{der } c r) \cdot r \end{aligned}$$

$$\begin{aligned} \text{my claim} &\equiv (\textit{der } c r) \cdot r \\ \text{(in this case)} & \end{aligned}$$

der for n -times

Case $n = 2$ and $r \cdot r$:

$$\begin{aligned} \text{der } c(r \cdot r) &\stackrel{\text{def}}{=} \text{if } \text{nullable}(r) \\ &\quad \text{then } (\text{der } c r) \cdot r + \text{der } c r \\ &\quad \text{else } (\text{der } c r) \cdot r \end{aligned}$$

$$\begin{aligned} \text{my claim} &\equiv (\text{der } c r) \cdot r \\ \text{(in this case)} & \end{aligned}$$

We know *nullable*(*r*) holds!

We know *nullable*(*r*) holds!

$$(der\ c\ r) \cdot r + der\ c\ r$$

We know *nullable*(*r*) holds!

$$(der\ c\ r) \cdot r + der\ c\ r \equiv (der\ c\ r) \cdot r + (der\ c\ r) \cdot \mathbf{1}$$

We know *nullable*(*r*) holds!

$$\begin{aligned}(\textit{der } c \textit{ } r) \cdot r + \textit{der } c \textit{ } r &\equiv (\textit{der } c \textit{ } r) \cdot r + (\textit{der } c \textit{ } r) \cdot \mathbf{1} \\ &\equiv (\textit{der } c \textit{ } r) \cdot (r + \mathbf{1})\end{aligned}$$

We know *nullable*(*r*) holds!

$$\begin{aligned}(\text{der } c \ r) \cdot r + \text{der } c \ r &\equiv (\text{der } c \ r) \cdot r + (\text{der } c \ r) \cdot \mathbf{1} \\ &\equiv (\text{der } c \ r) \cdot (r + \mathbf{1}) \\ &\equiv (\text{der } c \ r) \cdot r \\ &\quad \text{(remember } r \text{ is nullable)}\end{aligned}$$

We know *nullable*(*r*) holds!

$$\begin{aligned}(\text{der } c \ r) \cdot r + \text{der } c \ r &\equiv (\text{der } c \ r) \cdot r + (\text{der } c \ r) \cdot \mathbf{1} \\ &\equiv (\text{der } c \ r) \cdot (r + \mathbf{1}) \\ &\equiv (\text{der } c \ r) \cdot r \\ &\quad \text{(remember } r \text{ is nullable)}\end{aligned}$$

$$\text{der } c \ (r \cdot r) \stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c \ r) \cdot r + \text{der } c \ r \\ \text{else } (\text{der } c \ r) \cdot r \end{array}$$

We know *nullable*(*r*) holds!

$$\begin{aligned}(\text{der } c \, r) \cdot r + \text{der } c \, r &\equiv (\text{der } c \, r) \cdot r + (\text{der } c \, r) \cdot \mathbf{1} \\ &\equiv (\text{der } c \, r) \cdot (r + \mathbf{1}) \\ &\equiv (\text{der } c \, r) \cdot r \\ &\quad \text{(remember } r \text{ is nullable)}\end{aligned}$$

$$\text{der } c \, (r \cdot r) \stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c \, r) \cdot r \\ \text{else } (\text{der } c \, r) \cdot r \end{array}$$

We know *nullable*(*r*) holds!

$$\begin{aligned}(\text{der } c \, r) \cdot r + \text{der } c \, r &\equiv (\text{der } c \, r) \cdot r + (\text{der } c \, r) \cdot \mathbf{1} \\ &\equiv (\text{der } c \, r) \cdot (r + \mathbf{1}) \\ &\equiv (\text{der } c \, r) \cdot r \\ &\quad \text{(remember } r \text{ is nullable)}\end{aligned}$$

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

	$r\{n\}$	der
$n = 0:$	1	0
$n = 1:$	r	$(der\ c\ r)$
$n = 2:$	$r \cdot r$	$(der\ c\ r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$???
	\vdots	

	$r\{n\}$	der
$n = 0:$	1	0
$n = 1:$	r	$(der\ c\ r)$
$n = 2:$	$r \cdot r$	$(der\ c\ r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$	$(der\ c\ r) \cdot r \cdot r$
	\vdots	

	$r\{n\}$	der
$n = 0:$	1	0
$n = 1:$	r	$(der\ c\ r)$
$n = 2:$	$r \cdot r$	$(der\ c\ r) \cdot r$
$n = 3:$	$r \cdot r \cdot r$	$(der\ c\ r) \cdot r \cdot r$
	\vdots	

$nullable(r\{n\}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } true \text{ else } nullable(r)$

$der\ c\ (r\{n\}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } \mathbf{0} \text{ else } (der\ c\ r) \cdot r\{n - 1\}$

$$\begin{aligned}
 \text{der } c(r \cdot r \cdot r) &\stackrel{\text{def}}{=} \text{if } \text{nullable}(r) \\
 &\quad \text{then } (\text{der } c\,r) \cdot r \cdot r + \text{der } c(r \cdot r) \\
 &\quad \text{else } (\text{der } c\,r) \cdot r \cdot r
 \end{aligned}$$

$$\begin{aligned}
 \text{der } c(r \cdot r \cdot r) &\stackrel{\text{def}}{=} \text{if } \text{nullable}(r) \\
 &\quad \text{then } (\text{der } c r) \cdot r \cdot r + (\text{der } c r) \cdot r \\
 &\quad \text{else } (\text{der } c r) \cdot r \cdot r
 \end{aligned}$$

$$\begin{aligned}
 \text{der } c(r \cdot r \cdot r) &\stackrel{\text{def}}{=} \text{if } \text{nullable}(r) \\
 &\quad \text{then } (\text{der } c\,r) \cdot (r \cdot r + r) \\
 &\quad \text{else } (\text{der } c\,r) \cdot r \cdot r
 \end{aligned}$$

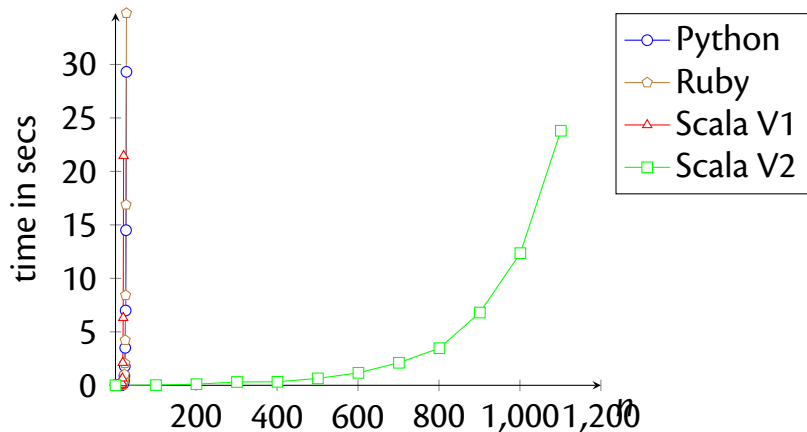
$$\text{der } c(r \cdot r \cdot r) \stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c\,r) \cdot (r \cdot (r + \mathbf{1})) \\ \text{else } (\text{der } c\,r) \cdot r \cdot r \end{array}$$

$$\text{der } c(r \cdot r \cdot r) \stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c\ r) \cdot (r \cdot r) \\ \text{else } (\text{der } c\ r) \cdot r \cdot r \end{array}$$

$$\text{der } c(r \cdot r \cdot r) \stackrel{\text{def}}{=} \begin{array}{l} \text{if } \text{nullable}(r) \\ \text{then } (\text{der } c\ r) \cdot r \cdot r \\ \text{else } (\text{der } c\ r) \cdot r \cdot r \end{array}$$

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

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



Examples

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

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

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

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

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)))  
}
```

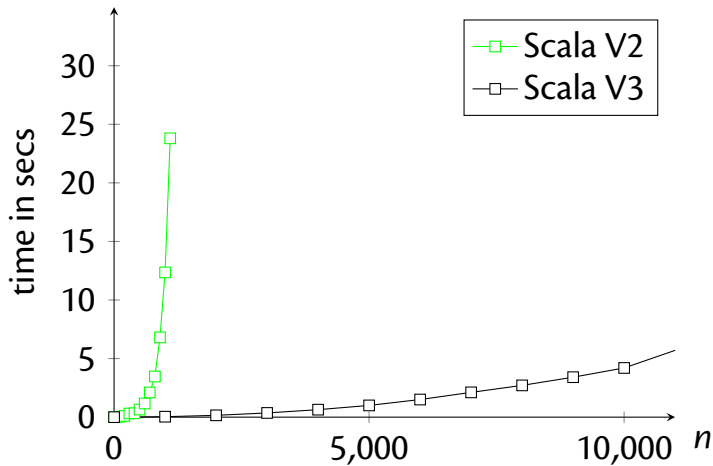


```

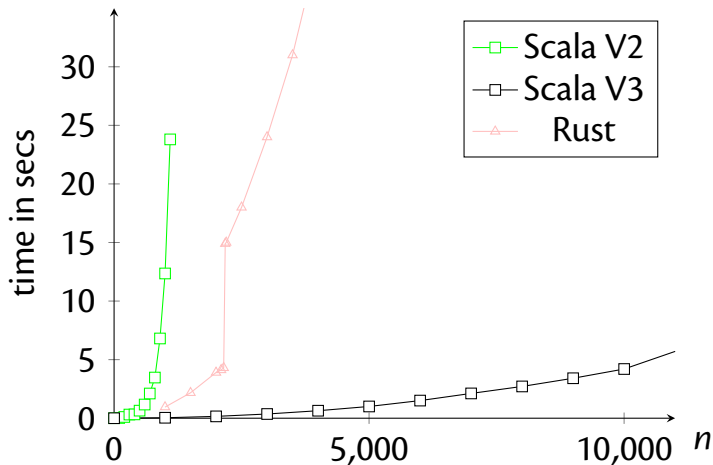
def simp(r: Rexp) : Rexp = r match {
  case ALT(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case (ZERO, r2s) => r2s
      case (r1s, ZERO) => r1s
      case (r1s, r2s) =>
        if (r1s == r2s) r1s else ALT(r1s, r2s)
    }
  }
  case SEQ(r1, r2) => {
    (simp(r1), simp(r2)) match {
      case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

```

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

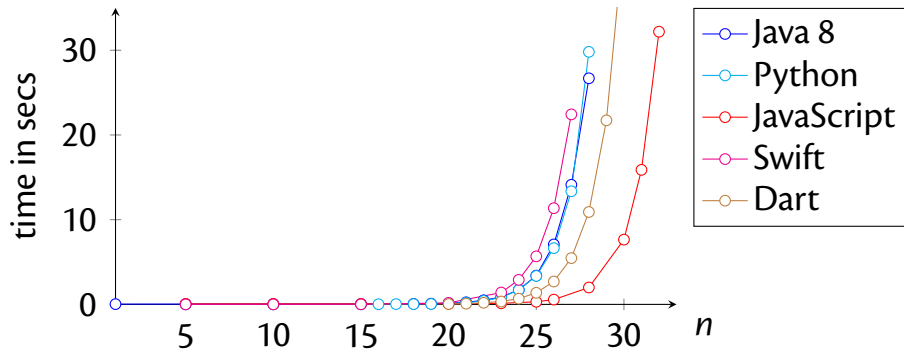


Brzozowski: $a^{\{n\}} \cdot a^{\{n\}}$



code by Archie Collard

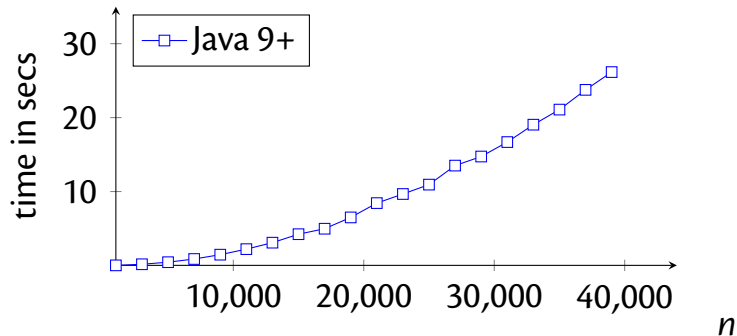
Another Example $(a^*)^* \cdot b$



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

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

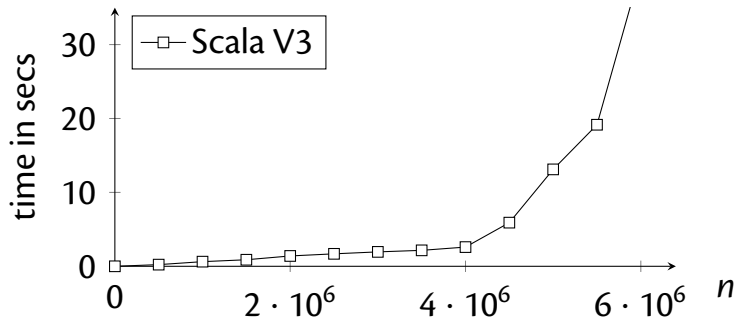
Same Example in Java 9+



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
- 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...(another video)

Negation of Regular Expr's

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

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:

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

The Specification for Matching

A regular expression r matches a string s provided:

$$s \in L(r)$$

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

The Specification for Matching

A regular expression r matches a string s provided:

$$s \in L(r)$$

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

nullable and *der*

The central properties:

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

nullable and *der*

The central properties:

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

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

nullable and *der*

The central properties:

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

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

Proofs about Rexp

Remember their inductive definition:

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

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

Assume $P(r)$ is the property:

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

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

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

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

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

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

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

$nullable(r^{\{n\}}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } true \text{ else } nullable(r)$

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

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

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

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

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

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

$$\text{der } c \ (r^{\{n\}}) \stackrel{\text{def}}{=} \text{if } n = 0 \text{ then } \mathbf{0} \text{ else } (\text{der } c \ r) \cdot r^{\{n-1\}}$$

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

$$s \in L(r)$$

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

Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$

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

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

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

$$\Leftrightarrow \text{nullable}(\text{ders } s r)$$

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

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

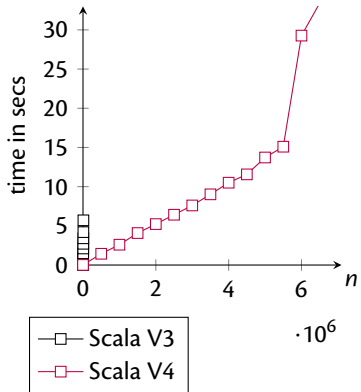
$$\text{matcher } s r \text{ if and only if } s \in L(r)$$

Coursework 1

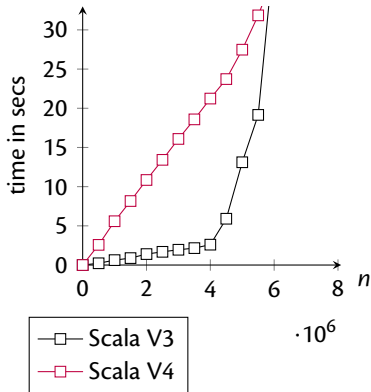
- Submission on Friday 16 October @ 18:00
- source code needs to be submitted as well
- you can re-use my Scala code from KEATS and use any programming language you like
- <https://nms.kcl.ac.uk/christian.urban/ProgInScala2ed.pdf>

Epilogue

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



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

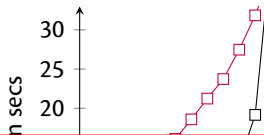


Epilogue

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 *_ + _*?

Questions?

Last Week

Last week I showed you a regular expression matcher that works provably correct in all cases (we only started with the proving part though)

matcher s r if and only if $s \in L(r)$

by Janusz Brzozowski (1964)

Proofs about Rexp

- 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 .

We proved

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

by induction on the regular expression r .

We proved

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

by induction on the regular expression r .

Any Questions?

Proofs about Natural Numbers and Strings

- P holds for 0 and
- P holds for $n + 1$ under the assumption that P already holds for n
- P holds for $[]$ and
- P holds for $c :: s$ under the assumption that P already holds for s

Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$

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

Correctness Proof for our Matcher

- We started from

$$s \in L(r)$$

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

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

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

$$\Leftrightarrow \text{nullable}(\text{ders } s r)$$

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

We need to prove

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

also by induction on the regular expression r .