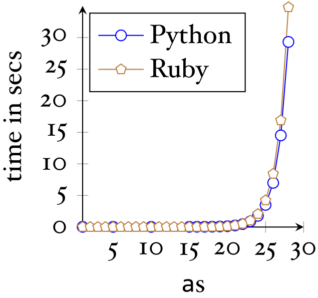
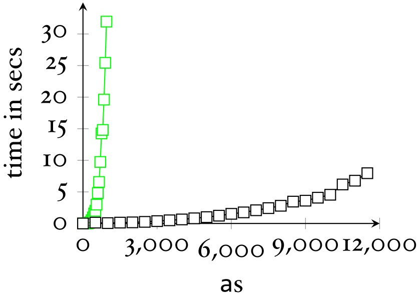
Automata and Formal Languages (2)

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

An Efficient Regular Expression Matcher





Languages, Strings

- **Strings** are lists of characters, for example

  $[], abc$ (Pattern match: $c :: s$)

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

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

- **Concatenation** of strings and sets

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

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

Regular Expressions

Their inductive definition:

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

Th

```scala
abstract class Rexp
case object NULL extends Rexp
case object EMPTY 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	$::=$	$\varnothing$	null
	$\mid$	ϵ	empty string / "" / []
	$\mid$	c	character
	$\mid$	$r_1 \cdot r_2$	sequence
	$\mid$	$r_1 + r_2$	alternative / choice
	$\mid$	r^*	star (zero or more)

The Meaning of a Regular Expression

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

L is a function from regular expressions to sets of strings

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

The Meaning of a Regular Expression

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

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

L is a function from regular expressions to sets of strings

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

What is $L(a^*)$?

Concrete Equivalences

$$
\begin{array}{rcl}
(a + b) + c & \equiv & a + (b + c) \\
a + a & \equiv & a \\
a + b & \equiv & b + a \\
(a \cdot b) \cdot c & \equiv & a \cdot (b \cdot c) \\
c \cdot (a + b) & \equiv & (c \cdot a) + (c \cdot b)
\end{array}
$$

Concrete Equivalences

$$
\begin{array}{rcl}
(a+b)+c & \equiv & a+(b+c) \\
a+a & \equiv & a \\
a+b & \equiv & b+a \\
(a\cdot b)\cdot c & \equiv & a\cdot(b\cdot c) \\
c\cdot(a+b) & \equiv & (c\cdot a)+(c\cdot b)
\end{array}
$$

$$
\begin{array}{rcl}
a\cdot a & \not\equiv & a \\
a+(b\cdot c) & \not\equiv & (a+b)\cdot(a+c)
\end{array}
$$

Corner Cases

$$
\begin{array}{rcl}
a \cdot \varnothing & \not\equiv & a \\
a + \epsilon & \not\equiv & a \\
\epsilon & \equiv & \varnothing^* \\
\epsilon^* & \equiv & \epsilon \\
\varnothing^* & \not\equiv & \varnothing
\end{array}
$$

Simplification Rules

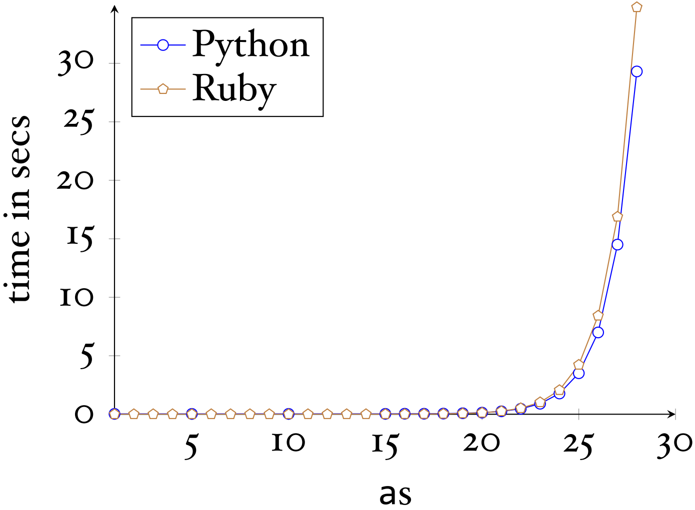
$$
\begin{array}{rcl}
r + \varnothing & \equiv & r \\
\varnothing + r & \equiv & r \\
r \cdot \epsilon & \equiv & r \\
\epsilon \cdot r & \equiv & r \\
r \cdot \varnothing & \equiv & \varnothing \\
\varnothing \cdot r & \equiv & \varnothing \\
r + r & \equiv & r
\end{array}
$$

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:

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

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)

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

The Derivative of a Rexp (2)

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

Examples

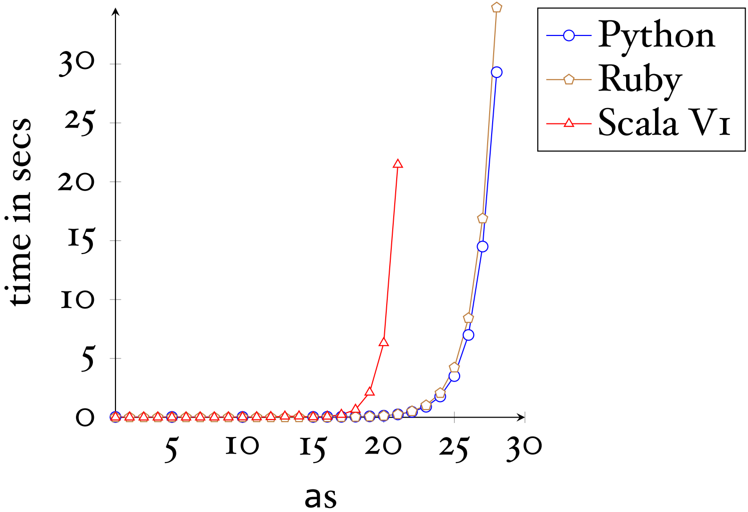
Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ what is

$$
\begin{aligned}
der\,a\,r &= ?\\
der\,b\,r &= ?\\
der\,c\,r &= ?
\end{aligned}
$$

The Algorithm

Input:	r_1, 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	$(r_4 = der\, b\, r_3)$
Step 4:	the string is exhausted; test whether r_4 can recognise the empty string	$(nullable(r_4))$
Output:	result of the test $\Rightarrow$ *true* or *false*	

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



time in secs
0
5
10
15
20
25
30
5
10
15
20
25
30
as
Python
Ruby
Scala V1

A Problem

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

$$
\begin{array}{rl}
1: & a \\
2: & a \cdot a \\
3: & a \cdot a \cdot a \\
 & \ldots \\
13: & a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \\
 & \ldots \\
20: &
\end{array}
$$

This problem is aggravated with $a?$ being represented as $\epsilon + a$.

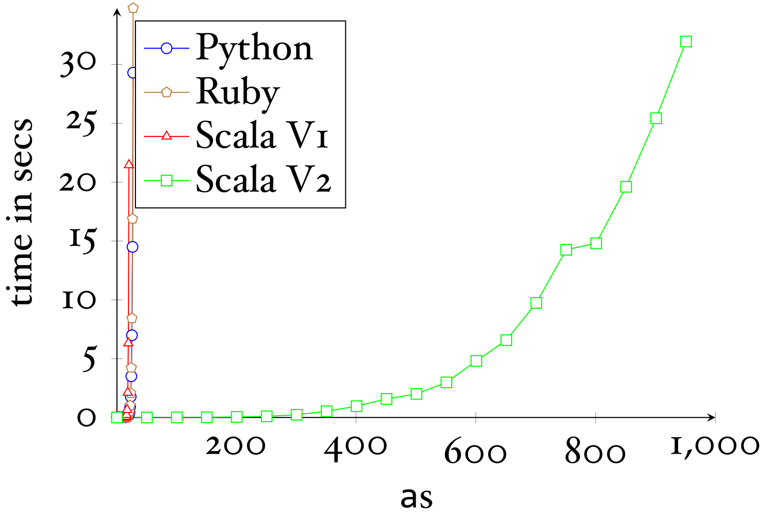
Solving the Problem

What happens if we extend our regular expressions

$$
\begin{array}{rcl}
r & ::= & \ldots \\
  & | & r\{n\} \\
  & | & r?
\end{array}
$$

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

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


Python
Ruby
Scala V1
Scala V2
time in secs
30
25
20
15
10
5
0
200
400
600
800
1,000
as

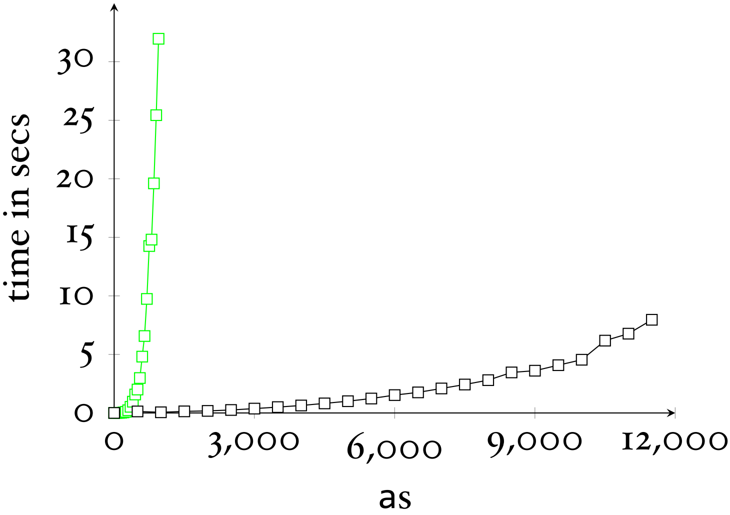
Examples

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

$$
\begin{aligned}
der\,a\,r &= ((\epsilon \cdot b) + \varnothing) \cdot r \\
der\,b\,r &= ((\varnothing \cdot b) + \epsilon) \cdot r \\
der\,c\,r &= ((\varnothing \cdot b) + \varnothing) \cdot r
\end{aligned}
$$

What are these regular expressions equivalent to?

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



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 $\varnothing$, ϵ and $\mathbf{c}$
- 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)

$$
\begin{aligned}
rev(\varnothing) &\stackrel{\text{def}}{=} \varnothing \\
rev(\epsilon) &\stackrel{\text{def}}{=} \epsilon \\
rev(c) &\stackrel{\text{def}}{=} c \\
rev(r_1 + r_2) &\stackrel{\text{def}}{=} rev(r_1) + rev(r_2) \\
rev(r_1 \cdot r_2) &\stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1) \\
rev(r^*) &\stackrel{\text{def}}{=} rev(r)^*
\end{aligned}
$$

We can prove

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

by induction on 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 finally prove

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