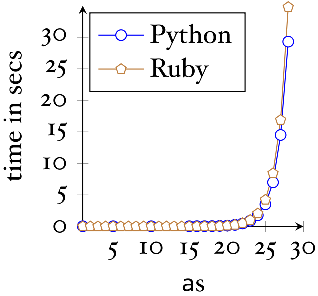
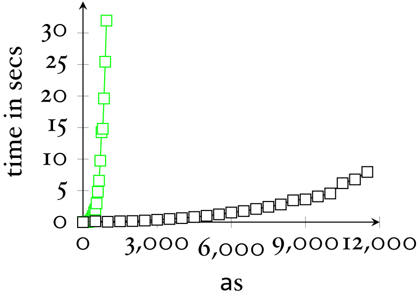
Automata and Formal Languages (2)

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

An Efficient Regular Expression Matcher





Languages

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

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

- **Concatenation** of strings and languages

$$\begin{array}{rcl} foo \mathbin{@} bar & = & foobar \\ A \mathbin{@} B & \stackrel{\text{def}}{=} & \{s_1 \mathbin{@} s_2 \mid s_1 \in A \wedge s_2 \in B\} \end{array}$$

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

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

The Power Operation

- The **Power** of a language:

$$\begin{array}{lcl} A^0 & \stackrel{\text{def}}{=} & \{[]\} \\ A^{n+1} & \stackrel{\text{def}}{=} & A\,@\,A^n \end{array}$$

For example

$$A^4 = A\,@\,A\,@\,A\,@\,A$$
$$A^0 \stackrel{\text{def}}{=} \{[]\}$$

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 **Star** of a language:

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

This expands to

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

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

Semantic Derivative

- The **Semantic Derivative** of a language wrt to a character c:

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

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

$$\begin{aligned} Der\,f\,A &= \{oo, rak\} \\ Der\,b\,A &= \{ar\} \\ Der\,a\,A &= \varnothing \end{aligned}$$

Semantic Derivative

- The **Semantic Derivative** of a language wrt to a character c:

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

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

$$\begin{aligned} Der\, f\, A &= \{oo, rak\} \\ Der\, b\, A &= \{ar\} \\ Der\, a\, A &= \varnothing \end{aligned}$$

We can extend this definition to strings

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

Regular Expressions

Their inductive definition:

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)

Th

```
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

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

L is a function from regular expressions to sets of strings

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

What is $L(a^*)$?

When Are Two Regular Expressions Equivalent?

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

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

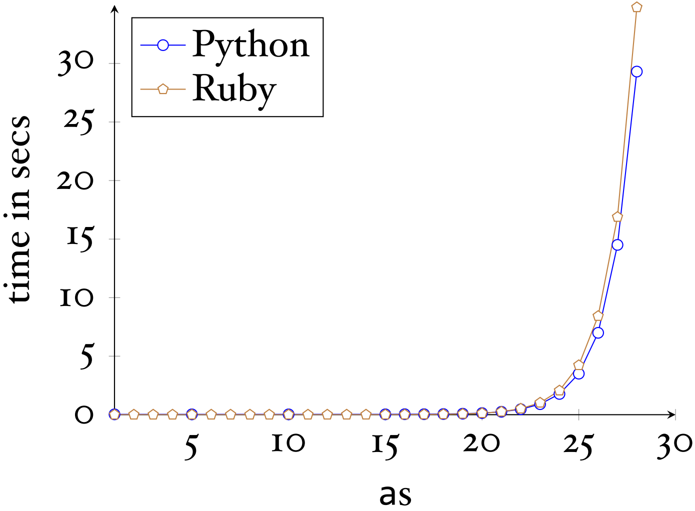
$$
\begin{array}{rcl}
r + \varnothing & \equiv & r \\
\varnothing + r & \equiv & r \\
r \cdot \epsilon & \equiv & r \\
\epsilon \cdot r & \equiv & r \\
r \cdot \varnothing & \equiv & \varnothing \\
\varnothing \cdot r & \equiv & \varnothing \\
r + r & \equiv & r
\end{array}
$$

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:

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

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

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

The Derivative of a Rexp

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

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 $false$	

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

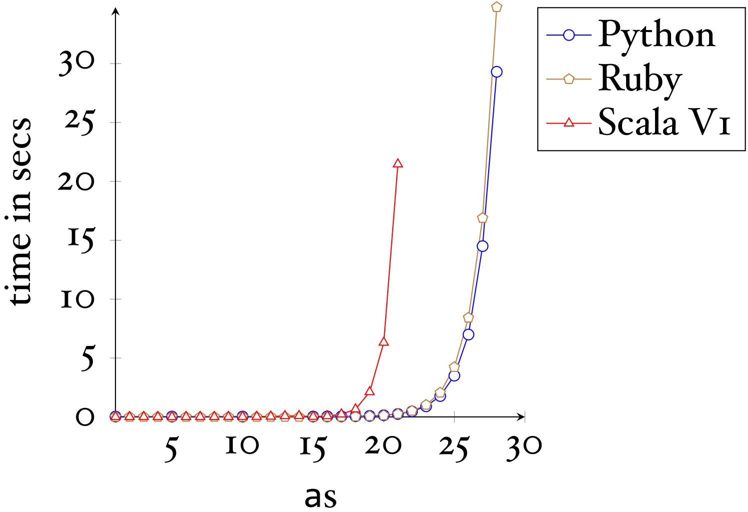
If we want to recognise the string abc with regular expression r_1 then

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.

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



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 $\epsilon + a$.

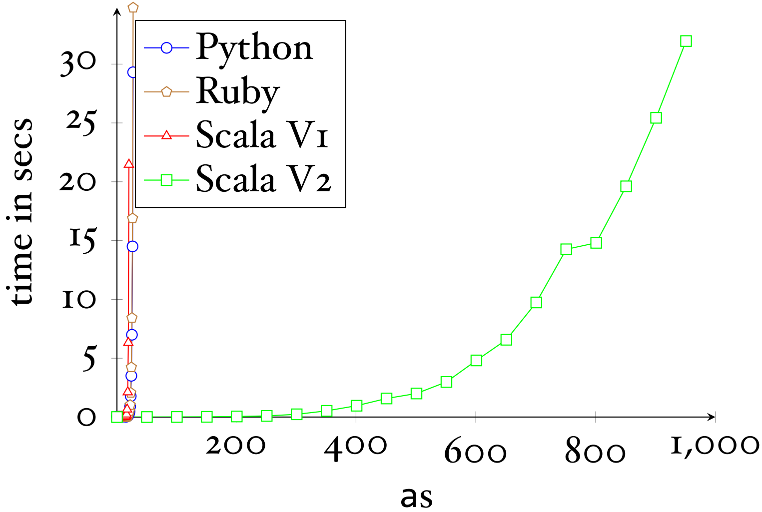
Solving the Problem

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
0
5
10
15
20
25
30
200
400
600
800
1,000
as

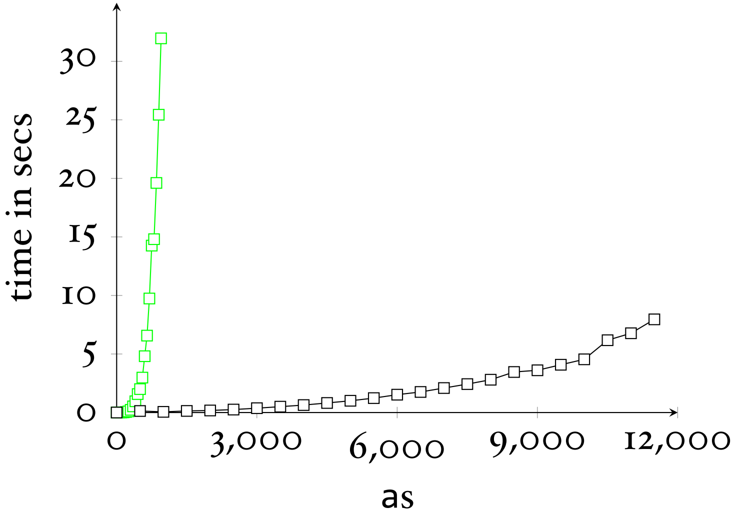
Examples

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?

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



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 brand new work)
- we can prove its correctness...

Proofs about Rexps

Remember their inductive definition:

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

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 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:

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

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.

Correctness Proof for our Matcher

- We started from

$$\begin{array}{rl} & s \in L(r) \\ \Leftrightarrow & [] \in Ders\,s\,(L(r)) \end{array}$$

Correctness Proof for our Matcher

- We started from

$$\begin{aligned} & s \in L(r) \\ \Leftrightarrow \quad & [] \in Ders\, s\, (L(r)) \end{aligned}$$

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

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

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

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