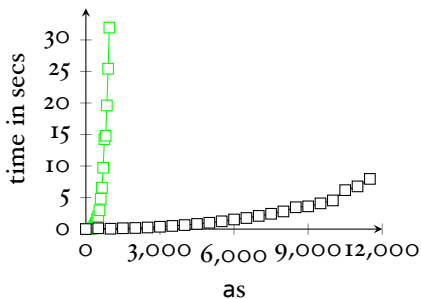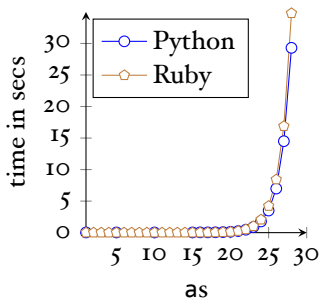# Automata and
Formal Languages (2)

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

  $$[], abc \qquad \text{(Pattern match: } c::s)$$

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

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

- **Concatenation** of strings and sets

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

  $$A \ @ \ B \ \stackrel{\text{def}}{=} \ \{s_1 \ @ \ s_2 \ | \ s_1 \in A \land s_2 \in B\}$$

Regular Expressions

Their inductive definition:

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

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
```

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

The Meaning of a Regular Expression

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

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

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

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

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

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

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

$$a \cdot \varnothing \quad \not\equiv \quad a$$
$$a + \epsilon \quad \not\equiv \quad a$$
$$\epsilon \quad \equiv \quad \varnothing^*$$
$$\epsilon^* \quad \equiv \quad \epsilon$$
$$\varnothing^* \quad \not\equiv \quad \varnothing$$

Simplification Rules

$$r + \varnothing \equiv r$$
$$\varnothing + r \equiv r$$
$$r \cdot \epsilon \equiv r$$
$$\epsilon \cdot r \equiv r$$
$$r \cdot \varnothing \equiv \varnothing$$
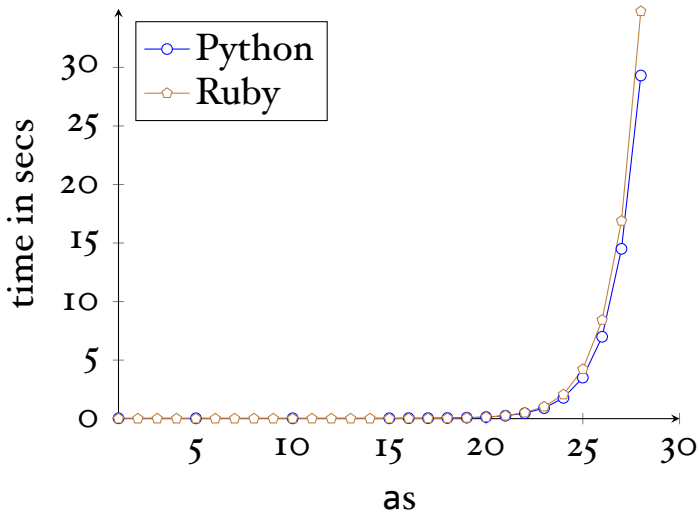$$\varnothing \cdot r \equiv \varnothing$$
$$r + r \equiv r$$

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:

$$nullable(\varnothing) \quad \overset{\text{def}}{=} \quad false$$

$$nullable(\epsilon) \quad \overset{\text{def}}{=} \quad true$$

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

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

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

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

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)

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

$$der\,c\,(\epsilon) \stackrel{\text{def}}{=} \varnothing$$

$$der\,c\,(d) \stackrel{\text{def}}{=} \text{if } c = d \text{ then } \epsilon \text{ else } \varnothing$$

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

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

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

The Derivative of a Rexp (2)

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

$$der\,c\,(\epsilon) \overset{\text{def}}{=} \varnothing$$

$$der\,c\,(d) \overset{\text{def}}{=} \text{if } c = d \text{ then } \epsilon \text{ else } \varnothing$$

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

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

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

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

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

Examples

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

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

The Algorithm

Input: r_1, *abc*

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

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

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

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

Output: result of the test
$\Rightarrow$ *true* or *false*

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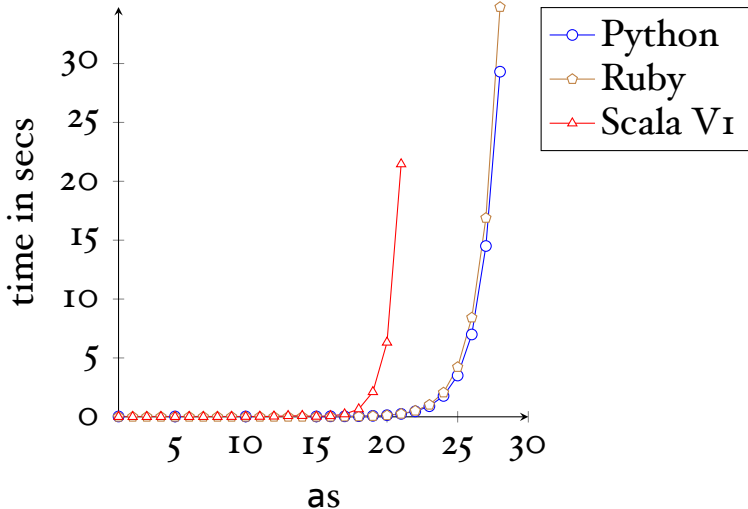
A Problem

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 $\epsilon + a$.
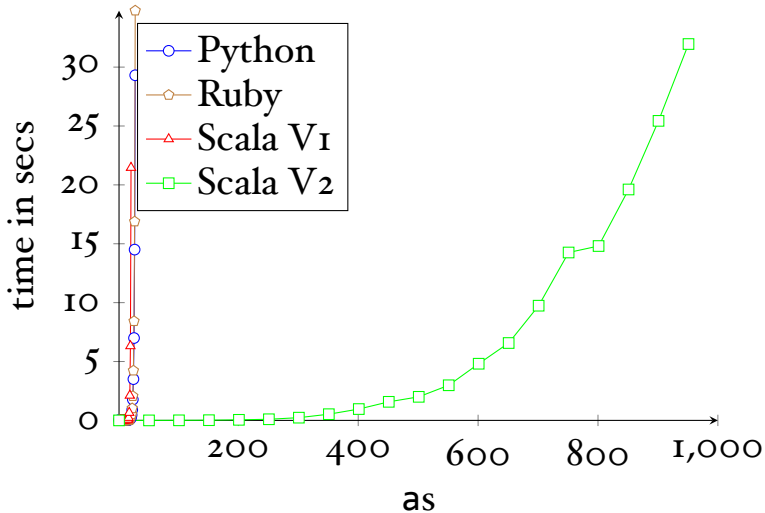
Solving the Problem

What happens if we extend our regular expressions

$$r \quad ::= \quad ...$$
$$| \quad r\{n\}$$
$$| \quad r?$$

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

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

Examples

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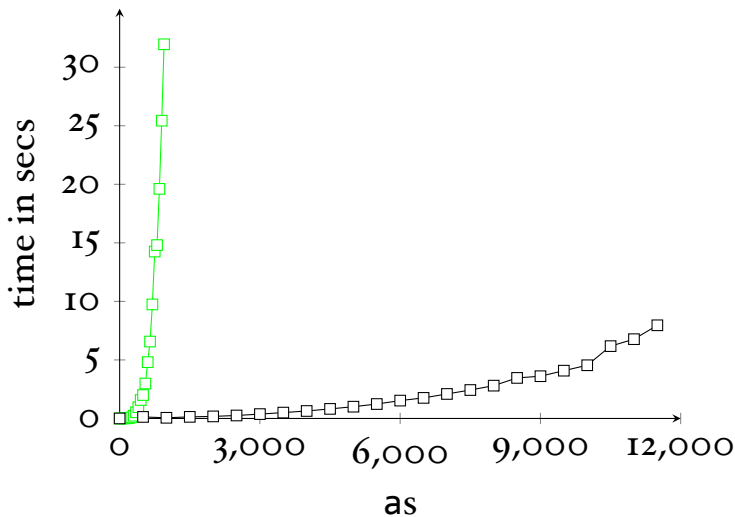
$$der\,a\,r = ((\epsilon \cdot b) + \varnothing) \cdot r$$
$$der\,b\,r = ((\varnothing \cdot b) + \epsilon) \cdot r$$
$$der\,c\,r = ((\varnothing \cdot b) + \varnothing) \cdot r$$

What are these regular expressions equivalent to?

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

Proofs about Rexps

Remember their inductive definition:

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

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:

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

Proofs about Rexp (4)

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

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

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