Compilers and Formal Languages

Email:	christian.urban at kcl.ac.uk
Office Hour:	Fridays 12 – 14
Location:	N7.07 (North Wing, Bush House)
Slides & Progs:	KEATS
Pollev:	https://pollev.com/cfltutoratki576

The Goal of this Module...

... you write a compiler

input
program

binary
code

lexer → parser → code gen

The Goal of this Module...

lexer input: a string

```
"read(n);"
```

lexer output: a sequence of tokens

```
key(read) lpar id(n) rpar semi
```

input program → **lexer** → **parser** → **code gen** → binary code

The Goal of this Module...

lexer input: a string

`"read(n);"`

lexer output: a sequence of tokens

`key(`**`read`**`) lpar id(n) rpar semi`

input program

binary code

lexer

par...

lexing ⇒ recognising words (Stone of Rosetta)

The Goal of this Module...

lexer input: a string
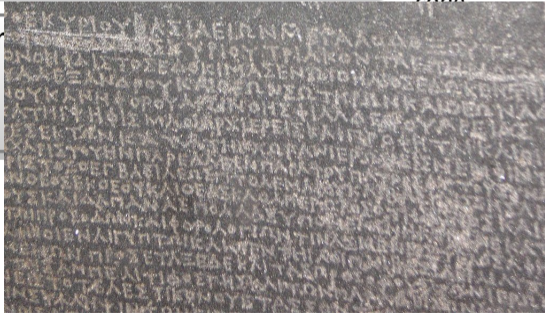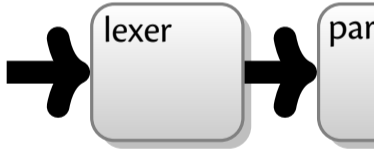
    "read(n);"

lexer output: a sequence of tokens

    key(**read**) lpar id(n) rpar semi

input
program

binary
code

lexer ➔ par...

if   ⟹ keyword
iffoo ⟹ identifier

lexing ⟹ recognising words (Stone of Rosetta)

The Goal of this Module...

... you write a compiler

input program → lexer → parser → code gen → binary code

parser input: a sequence of tokens

```
key(read) lpar id(n) rpar semi
```

parser output: an abstract syntax tree

input
pro

binary
code

The Goal of this Module...

... you write a compiler

input
program

binary
code

lexer → parser → code gen

write a compiler

code generation:

```
istore 2
iload 2
ldc 10
isub
ifeq Label2
iload 2
...
```

input
pro...

parser → code gen → binary code

The Goal of this Module...

write a compiler

code generation:

```
istore 2
iload 2
ldc 10
isub
ifeq Label2
iload 2
...
```

inp
pro

parser

The Goal of this Module...

Compiler explorers, e.g.: https://gcc.godbolt.org

input
pro

source ⟶ binary

The Goal of this Module...

Compiler explorer for Java: https://javap.yawk.at

input
program

source ⟶ byte code

The Goal of this Module...
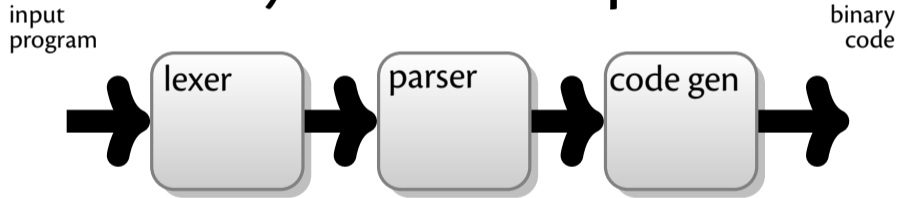
... you write a compiler

input
program

binary
code

lexer

parser

code gen

Why Study Compilers?

John Regehr (Univ. Utah, LLVM compiler hacker) 👍

> **"…It's effectively a perpetual employment act for solid compiler hackers."**

Why Study Compilers?

John Regehr (Univ. Utah, LLVM compiler hacker) 👆

> **"...It's effectively a perpetual employment act for solid compiler hackers."**

- **Hardware is getting weirder rather than getting clocked faster.**

  "Almost all processors are multicores nowadays and it looks like there is increasing asymmetry in resources across cores. Processors come with vector units, crypto accelerators etc. We have DSPs, GPUs, ARM big.little, and Xeon Phi. This is only scratching the surface."

Why Study Compilers?

John Regehr (Univ. Utah, LLVM compiler hacker) 👍

> "...It's effectively a perpetual employment act for solid compiler hackers."

- **We're getting tired of low-level languages and their associated security disasters.**

  "We want to write new code, to whatever extent possible, in safer, higher-level languages. Compilers are caught right in the middle of these opposing trends: one of their main jobs is to help bridge the large and growing gap between increasingly high-level languages and increasingly wacky platforms."

Why Bother with Compilers?

Boeing 777's: First flight in 1994. They want to achieve triple redundancy for potential hardware faults. 👈

They compile 1 Ada program to

- Intel 80486
- Motorola 68040 (old Macintosh's)
- AMD 29050 (RISC chips used often in laser printers)

using 3 independent compilers.
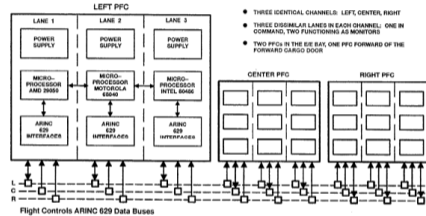
Why Bother with Compilers?

Boeing 777's: First flight in 1994. They want to achieve triple redundancy for potential hardware faults. 👉

They compile 1 Ada program to

- Intel 80486
- Motorola 68040 (old Macintosh's)
- AMD 29050 (RISC chips used often in laser printers)

using 3 independent compilers.

Airbus uses C and static analysers. Recently started using CompCert.
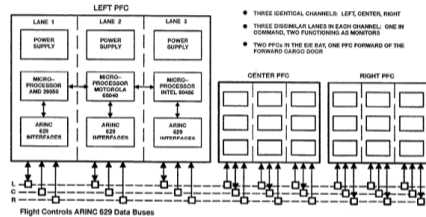
What Do Compilers Do?

Remember BF*** from PEP?

>	$\Rightarrow$	move one cell right
<	$\Rightarrow$	move one cell left
+	$\Rightarrow$	increase cell by one
-	$\Rightarrow$	decrease cell by one
.	$\Rightarrow$	print current cell
,	$\Rightarrow$	input current cell
[	$\Rightarrow$	loop begin
]	$\Rightarrow$	loop end
	$\Rightarrow$	everything else is a comment

A "Compiler" for BF*** to C

```
>  ⇒  ptr++
<  ⇒  ptr--
+  ⇒  (*ptr)++
-  ⇒  (*ptr)--
.  ⇒  putchar(*ptr)
,  ⇒  *ptr = getchar()
[  ⇒  while(*ptr){
]  ⇒  }
   ⇒  ignore everything else
```

```
char field[30000]
char *ptr = &field[15000]
```

Another "Compiler" for BF to C

$$>...> \quad \Rightarrow \quad \texttt{ptr += n}$$
$$<...< \quad \Rightarrow \quad \texttt{ptr -= n}$$
$$+...+ \quad \Rightarrow \quad \texttt{(*ptr) += n}$$
$$-...- \quad \Rightarrow \quad \texttt{(*ptr) -= n}$$
$$. \quad \Rightarrow \quad \texttt{putchar(*ptr)}$$
$$, \quad \Rightarrow \quad \texttt{*ptr = getchar()}$$
$$[\quad \Rightarrow \quad \texttt{while(*ptr)\{}$$
$$] \quad \Rightarrow \quad \texttt{\}}$$

$$\Rightarrow \quad \text{ignore everything else}$$

```
char field[30000]
char *ptr = &field[15000]
```

A Brief Compiler History

- Turing Machines, 1936 (a tape as memory)
- Regular Expressions, 1956
- The first compiler for COBOL, 1957
  (Grace Hopper)

- But surprisingly research papers are still published
  nowadays
- "Parsing: The Solved Problem That Isn't" 👉

Grace Hopper

(she made it to David Letterman's Tonight Show 👉)

Some Housekeeping

Exam will be computer-based, invigilated in some big examination hall:

- final exam in January (~~35%~~ **40%**)
- coursework (~~65%~~ **60%- very first part is now optional**)

Some Housekeeping

Exam will be computer-based, invigilated in some big examination hall:

- final exam in January (~~35%~~ **40%**)
- coursework (~~65%~~ **60%- very first part is now optional**)

Weekly Homework (optional):

- uploaded on KEATS - solutions will be discussed during the SGTs
- **all** questions in the exam will be in some close shape or form from the HWs!!

Homework

Until 3 years ago: I did not give out solutions; students sent emails to me and I responded to them individually.

Now: We will review the homework mainly during the SGTs.

Homework

Until 3 years ago: I did not give out solutions; students sent emails to me and I responded to them individually.

Now: We will review the homework mainly during the SGTs.

I will still choose the questions from the HW for the exam, but there might be some larger amount of deviation.

Homework

Until 3 years ago: I did not give out solutions; students sent emails to me and I responded to them individually.

Now: We will review the homework mainly during the SGTs.

I will still choose the questions from the HW for the exam, but there might be some larger amount of deviation.

Do not harass your TAs for the solutions!

Students in CFL
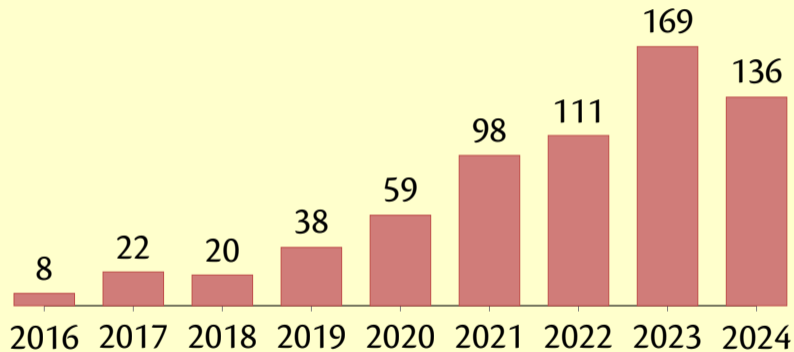
Student numbers since the start of the compiler module.

Students in CFL
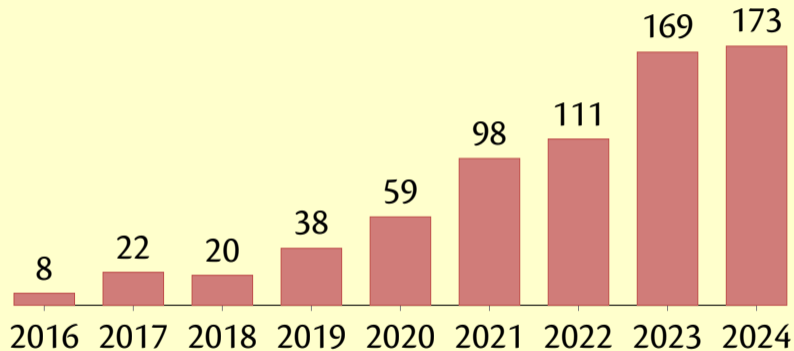
Student numbers since the start of the compiler module.

Some Housekeeping

Coursework (4 parts accounting for 60%; submission deadline <u>2nd January</u>):

- matcher ~~(5%)~~ optional from this year
- lexer (10%)
- parser / interpreter (10%)
- JVM compiler (15%)
- LLVM compiler (25%)

Some Housekeeping

Coursework (4 parts accounting for 60%; submission deadline 2nd January):

- matcher (~~5%~~) optional from this year
- lexer (10%)
- parser / interpreter (10%)
- JVM compiler (15%)
- LLVM compiler (25%)

you can use any programming language you like
(Haskell, Rust, Swift)

Some Housekeeping

Coursework (4 parts accounting for 60%; submission deadline 2nd January):

- matcher (5%) optional from this year
- lexer (10%)
- parser / interpreter (10%)
- JVM compiler (15%)
- LLVM compiler (25%)

you can use any programming language you like (Haskell, Rust, Swift)

you can use any code I show you and is uploaded to KEATS...**BUT NOTHING ELSE!**

I will show you all my code in Scala 3

```
$ scala-cli
Welcome to Scala 3.5.0 (21.0.4, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> 1 + 2
res0: Int = 3
```

I will show you all my code in Scala 3

```
$ scala-cli
Welcome to Scala 3.5.0 (21.0.4, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> 1 + 2
res0: Int = 3
```

Since Scala 3.5.0, scala-cli is included in "plain" Scala

```
$ scala
Welcome to Scala 3.5.1 (21.0.4, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala>
```

Ammonite & Scala 3

Actually in CFL, I will use Amm / Scala 3

```
$ amm
Loading...
Welcome to the Ammonite Repl 3.0.0-M2 (Scala 3.3.3 Java 21.0.4)
scala> 1 + 2
res0: Int = 3
```

Ammonite & Scala 3

Actually in CFL, I will use Amm / Scala 3

```
$ amm
Loading...
Welcome to the Ammonite Repl 3.0.0-M2 (Scala 3.3.3 Java 21.0.4)
scala> 1 + 2
res0: Int = 3
```

Do not use Amm + Scala 2!

```
$ amm2
Loading...
Welcome to the Ammonite Repl 2.5.9 (Scala 2.13.11 Java 17.0.7)
scala>
```

For Install Problems

- Harry Dilnot (harry.dilnot@kcl.ac.uk)
    Windows expert
- Oliver Iliffe (oliver.iliffe@kcl.ac.uk)

> *Unequivocally the worst module I've taken on this course. The subject matter is fascinating, however the insistence on the use of this abomination of a language "Scala" completely ruins it. If you're going to teach something as complex as this, use a proper language, not some "object oriented functional" abomination. Use C, you know, the language that real compilers are written in. I will go to the end of the earth to dissuade others from taking this module so long as Scala is still being used.*
>
> *— Lone voice in the end-of-year feedback in 2019*

(for alternative opinions check "What the students say" on KEATS)

Lectures 1 - 5

transforming strings into structured data

Lexing   based on regular expressions

(recognising "words")

Parsing

(recognising "sentences")
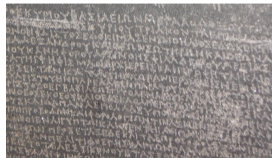
Stone of Rosetta

Lectures 1 - 5

transforming strings into structured data

Lexing

based on regular expressions

(recognising "words")

Parsing

(recognising "sentences")
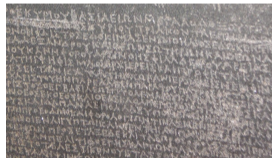
Stone of Rosetta

Lectures 5 - 10

code generation for a small imperative and a small functional language

Interpreters

(directly runs a program)

Compilers

(generate JVM code and LLVM-IR code)

Familiar Regular Expresssions

```
[a-z0-9_\.-]+ @ [a-z0-9\.-]+ . [a-z\.]{2,6}
```

`re*`	matches 0 or more times
`re+`	matches 1 or more times
`re?`	matches 0 or 1 times
`re{n}`	matches exactly n number of times
`re{n,m}`	matches at least n and at most `m` times
`[...]`	matches any single character inside the brackets
`[^...]`	matches any single character not inside the brackets
`a-z A-Z`	character ranges
`\d`	matches digits; equivalent to `[0-9]`
`.`	matches every character except newline
`(re)`	groups regular expressions and remembers the matched text

Notation for REs

Some "innocent" examples

Let's try two examples

```
(a*)*b          [a?]{n}[a]{n}
```

Some "innocent" examples

Let's try two examples

      `(a*)*b`   `[a?]{n}[a]{n}`

and match them with strings of the form

        a, aa, aaa, aaaa, aaaaa, a...a

$$\underbrace{\qquad}_{n}$$

Why Bother with Regexes?

Ruby, Python, Java 8   Us (after next lecture)

[a?]{n}[a]{n}:
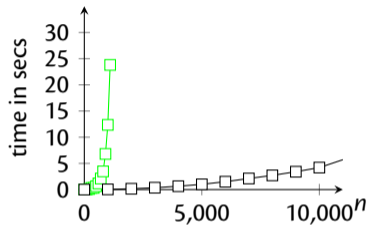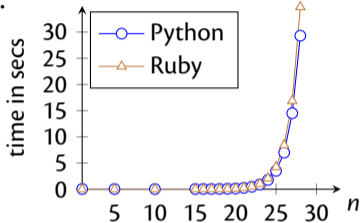
(a*)*b:
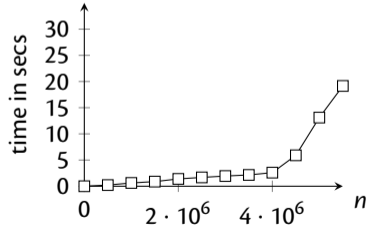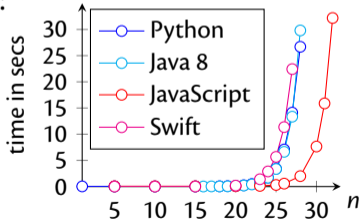
matching with
strings a...a
         n

Incidents

- a global outage on 2 July 2019 at **Cloudflare** (first one for six years)

```
(?:(?:\"|'|\]|\}|\\|\d|(?:nan|infinity|true|false|
null|undefined|symbol|math)|\`|\-|\+)+[)]*;?((?:\s
|-|~|!|{}|\|\||\+)*.*(?:.*=.*)))
```


CLOUDFLARE®

It serves more web traffic than Twitter, Amazon, Apple, Instagram, Bing & Wikipedia combined. 👆

- on 20 July 2016 the **Stack Exchange** webpage went down because of an evil regular expression 👆

Evil Regular Expressions

- Regular expression Denial of Service (ReDoS)

- Some evil regular expressions:

    - `[a?]{n} [a]{n}`
    - `(a*)* b`
    - `([a-z]+)*`
    - `(a + aa)*`
    - `(a + a?)*`

- sometimes also called catastrophic backtracking

- this is a problem for Network Intrusion Detection systems, Cloudflare, StackExchange, Atom editor

- https://vimeo.com/112065252

(Basic) Regular Expressions

Their inductive definition:

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

(B

Their

```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
```

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

Strings

...are lists of characters. For example `"hello"`

$$[h, e, l, l, o] \text{ or just } hello$$

the empty string: $[]$ or `""`

the concatenation of two strings:

$$s_1 \, @ \, s_2$$

$foo \, @ \, bar = foobar$
$baz \, @ \, [] = baz$

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 languages

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

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

Languages, Strings

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

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

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

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

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

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

- **Concatenation** of strings and languages

  $foo @ bar \;=\; foobar$

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

Two Corner Cases

$$A \mathbin{@} \{[]\} = ?$$

Two Corner Cases

$$A \mathbin{@} \{[]\} = ?$$

$$A \mathbin{@} \{\} = ?$$

The Meaning of a Regex

...all the strings a regular expression can match.

$$L(\mathbf{0}) \;\stackrel{\text{def}}{=}\; \{\}$$
$$L(\mathbf{1}) \;\stackrel{\text{def}}{=}\; \{[]\}$$
$$L(c) \;\stackrel{\text{def}}{=}\; \{[c]\}$$
$$L(r_1 + r_2) \;\stackrel{\text{def}}{=}\; L(r_1) \cup L(r_2)$$
$$L(r_1 \cdot r_2) \;\stackrel{\text{def}}{=}\; L(r_1) \,@\, L(r_2)$$
$$L(r^*) \;\stackrel{\text{def}}{=}\;$$

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

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

The Power Operation

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

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

For example

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

The Meaning of a Regex

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

The Meaning of a Regex

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

$$L(\mathbf{1}) \stackrel{\text{def}}{=} \{[]\}$$

$$L(c) \stackrel{\text{def}}{=} \{[c]\}$$

$$L(r_1 + r_2) \stackrel{\text{def}}{=} L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) \stackrel{\text{def}}{=} \{s_1 @ s_2 \mid s_1 \in L(r_1) \wedge s_2 \in L(r_2)\}$$

$$L(r^*) \stackrel{\text{def}}{=} \bigcup_{0 \leq n} L(r)^n$$

The Star Operation

- The **Kleene Star** of a language:

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

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 Regex

$$L(\mathbf{0}) \;\stackrel{\text{def}}{=}\; \{\}$$

$$L(\mathbf{1}) \;\stackrel{\text{def}}{=}\; \{[]\}$$

$$L(c) \;\stackrel{\text{def}}{=}\; \{[c]\}$$

$$L(r_1 + r_2) \;\stackrel{\text{def}}{=}\; L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) \;\stackrel{\text{def}}{=}\; \{s_1 @ s_2 \mid s_1 \in L(r_1) \land s_2 \in L(r_2)\}$$

$$L(r^*) \;\stackrel{\text{def}}{=}\; (L(r))\star$$

The Meaning of Matching

A regular expression *r* matches a string *s* provided

$$s \in L(r)$$

...and the point of the next lecture is to decide this problem as fast as possible (unlike Python, Ruby, Java)

Questions

- Say $A = \{[a], [b], [c], [d]\}$.

  How many strings are in A^4?

Questions

- 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 ?

Questions

- Assume a set A contains 4 strings and a set B contains 7 strings. None of the strings is the empty string.
- How many strings are in $A @ B$?

CH Assume a language A contains 7 strings, and B contains 4 strings. How many strings are in the concatenation of these two languages?

If you want to find the number of strings in the concatenation of two languages A and B, you need to multiply the number of strings in each language. So, if A contains 7 strings and B contains 4 strings, the number of strings in the concatenation of A and B is:

Number of strings in A concatenated with B = |A| * |B| = 7 * 4 = 28

There are 28 strings in the concatenation of these two languages.

...for amusement

CH No this is incorrect. Think again.

I apologize for the incorrect response earlier. You are right; my previous answer was incorrect.

When concatenating two languages A and B, you need to consider all possible combinations of strings from A and B. For each string in A, you can concatenate it with every string in B, resulting in a new string. To find the total number of strings in the concatenation, you multiply the number of strings in A by the number of strings in B.

So, if A contains 7 strings and B contains 4 strings, the number of strings in the concatenation of A and B is:

Number of strings in A concatenated with B = |A| * |B| = 7 * 4 = 28

There are indeed 28 different strings in the concatenation of these two languages.

Questions?

SGT TAs: Flavio Melinte Citea   (was a KURF two summers ago)
              Zishan Rahman
              Harry Dilnot
              Opale Sjostedt

Amm Helpers   Harry Dilnot   (harry.dilnot@kcl.ac.uk)
              Oliver Iliffe   (oliver.iliffe@kcl.ac.uk)

              (was a KURF last summer)

Coursework

Do we need to provide instructions on running the coursework files if we're using languages other than Scala? Thanks

Coursework

Do we need to provide instructions on running the coursework files if we're using languages other than Scala? Thanks

Zip-File for Coursework

Please, please submit a zipfile that generates a subdirectory

```
NameFamilyName
```

What is the trick?

What was the trick to improve the evil regular expressions matcher to have such good results compared to other programming languages? Is it working better on casual regular expressions (the ones that Python and Java handle pretty well), too? Or was it just optimised for these evil ones?
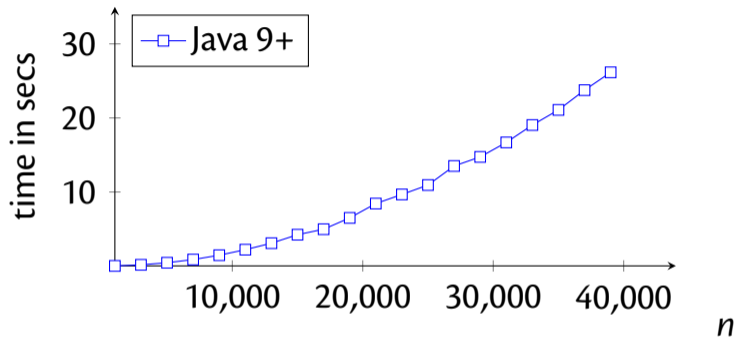
Thanks to Martin Mikusovic
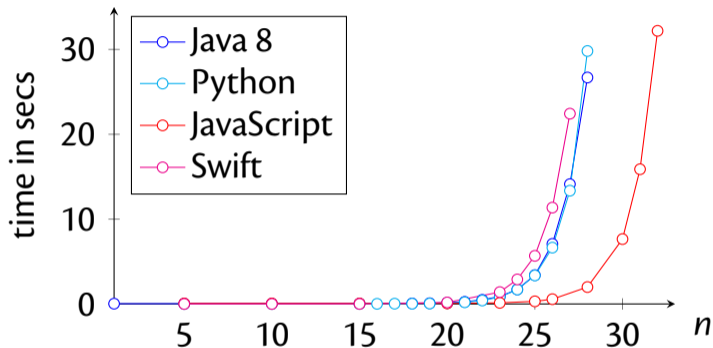
Regex: $(a^*)^* \cdot b$
Strings of the form $\underbrace{a \ldots a}_{n}$

Same Example in Java 9+

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