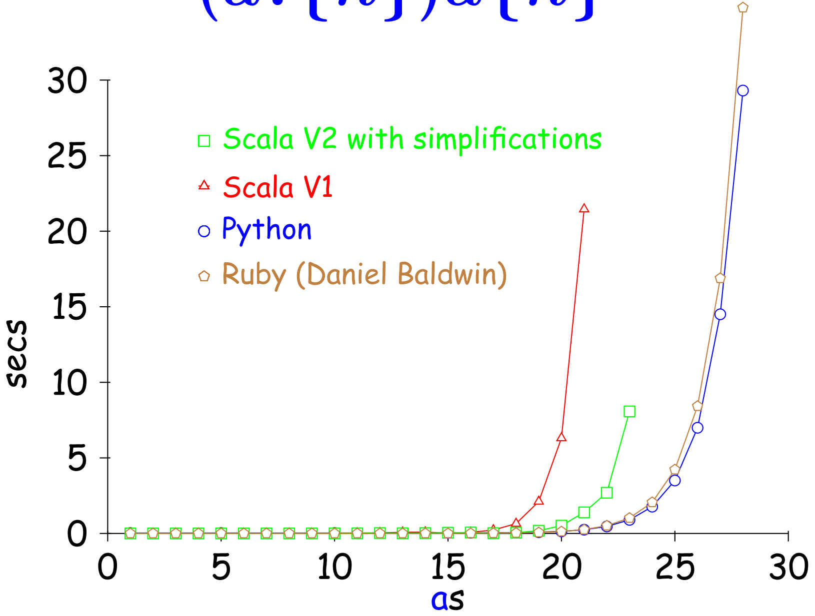
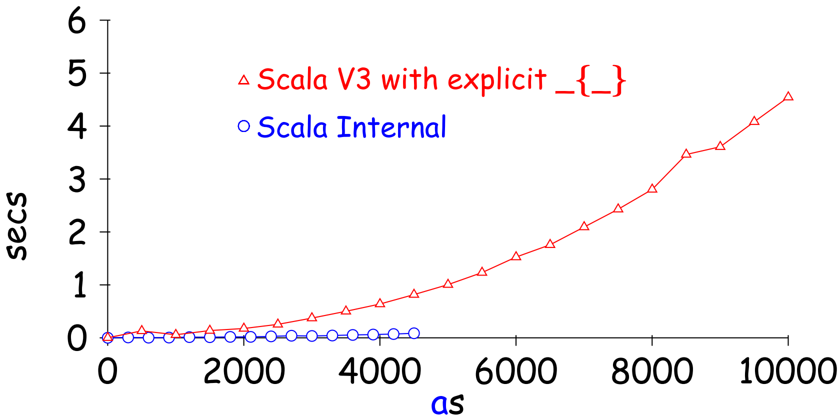
Automata and Formal Languages (7)

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

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



Scala V2 with simplifications
Scala V1
Python
Ruby (Daniel Baldwin)
secs
as
0
5
10
15
20
25
30



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

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

CFG

A context-free grammar G consists of

- a finite set of nonterminal symbols (upper case)
- a finite terminal symbols or tokens (lower case)
- a start symbol (which must be a nonterminal)
- a set of rules

$$A \rightarrow \text{rhs}$$

where rhs are sequences involving terminals and nonterminals (can also be empty).

CFG

A context-free grammar G consists of

- a finite set of nonterminal symbols (upper case)
- a finite terminal symbols or tokens (lower case)
- a start symbol (which must be a nonterminal)
- a set of rules

$$A \rightarrow \text{rhs}$$

where rhs are sequences involving terminals and nonterminals (can also be empty).

We can also allow rules

$$A \rightarrow \text{rhs}_1 | \text{rhs}_2 | \ldots$$

A CFG Derivation

1. Begin with a string with only the start symbol S
2. Replace any non-terminal X in the string by the right-hand side of some production $X \rightarrow$ rhs
3. Repeat 2 until there are no non-terminals

$$S \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots$$

Language of a CFG

Let G be a context-free grammar with start symbol S. Then the language $L(G)$ is:

$$\{c_1 \ldots c_n \mid \forall i.\ c_i \in T \wedge S \rightarrow^* c_1 \ldots c_n\}$$

Language of a CFG

Let G be a context-free grammar with start symbol S. Then the language $L(G)$ is:

$$\{c_1 \ldots c_n \mid \forall i.\ c_i \in T \wedge S \rightarrow^* c_1 \ldots c_n\}$$

- Terminals are so-called because there are no rules for replacing them
- Once generated, terminals are "permanent"
- Terminals ought to be tokens of the language (at least in this course)

Arithmetic Expressions

$$
\begin{array}{lcl}
E & \rightarrow & num\_token \\
E & \rightarrow & E \cdot + \cdot E \\
E & \rightarrow & E \cdot - \cdot E \\
E & \rightarrow & E \cdot * \cdot E \\
E & \rightarrow & (\cdot E \cdot)
\end{array}
$$

Arithmetic Expressions

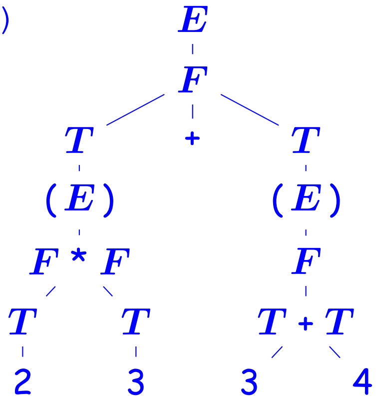
$$
\begin{array}{lcl}
E & \rightarrow & num\_token \\
E & \rightarrow & E \cdot + \cdot E \\
E & \rightarrow & E \cdot - \cdot E \\
E & \rightarrow & E \cdot * \cdot E \\
E & \rightarrow & (\cdot E \cdot)
\end{array}
$$

A CFG is left-recursive if it has a nonterminal E such that $E \rightarrow^{+} E \cdot \ldots$

Parse Trees

$$\begin{array}{lcl} E & \rightarrow & F \mid F \cdot * \cdot F \\ F & \rightarrow & T \mid T \cdot + \cdot T \mid T \cdot - \cdot T \\ T & \rightarrow & num\_token \mid (\cdot E \cdot) \end{array}$$

`(2*3)+(3+4)`



Ambiguous Grammars

A CFG is ambiguous if there is a string that has at least parse trees.

$$
\begin{array}{lcl}
E & \rightarrow & num\_token \\
E & \rightarrow & E \cdot + \cdot E \\
E & \rightarrow & E \cdot - \cdot E \\
E & \rightarrow & E \cdot * \cdot E \\
E & \rightarrow & (\cdot E \cdot)
\end{array}
$$

`1 + 2 * 3 + 4`

Dangling Else

Another ambiguous grammar:

$$\begin{array}{lcl} E & \rightarrow & \text{if } E \text{ then } E \\ & | & \text{if } E \text{ then } E \text{ else } E \\ & | & \text{id} \end{array}$$

`if a then if x then y else c`