Automata and Formal Languages (6)

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

"I hate coding. I do not want to look at code."

"I am appalled. You do not show code anymore."

ReDoS

- Regular expression Denial of Service
- "Regular Expressions Will Stab You in the Back"
- Evil regular expressions
  - $(a?\{n\})a\{n\}$
  - $(a^+)^+$
  - $([a-zA-Z]^+)^*$
  - $(a+aa)^+$
  - $(a+a?)^+$

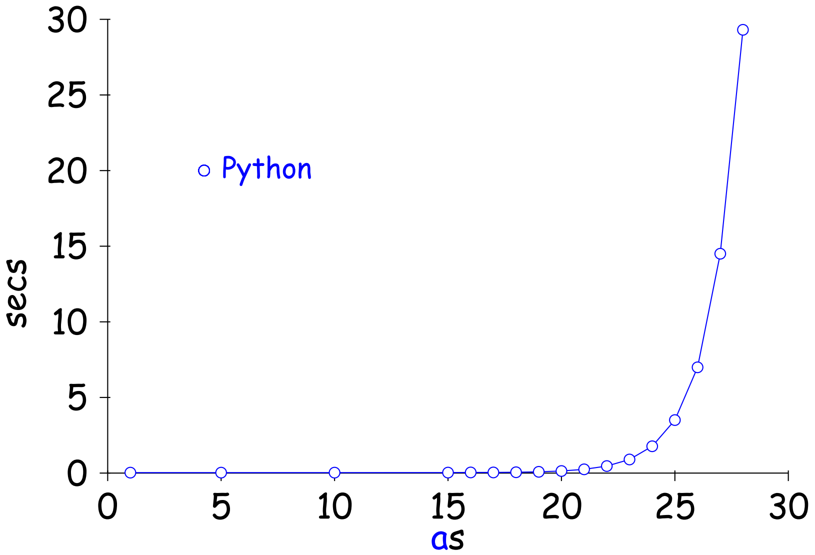
Regexp Matching

Given a regular expression

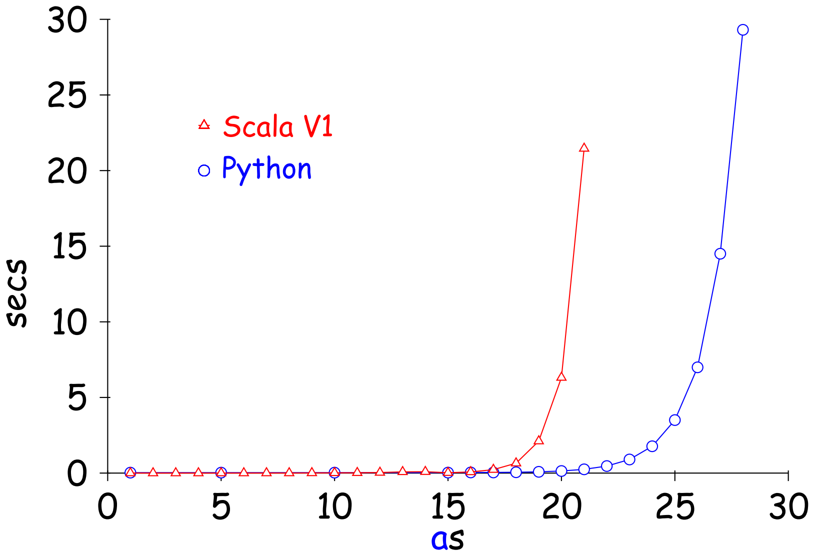
1. you might convert it into a DFA (subset construction)
2. you might try all possible paths in an NFA via backtracking
3. you might try all paths in an NFA in parallel
4. you might try to convert the DFA "lazily"

Often No 2 is implemented (sometimes there are even good reasons for doing this).

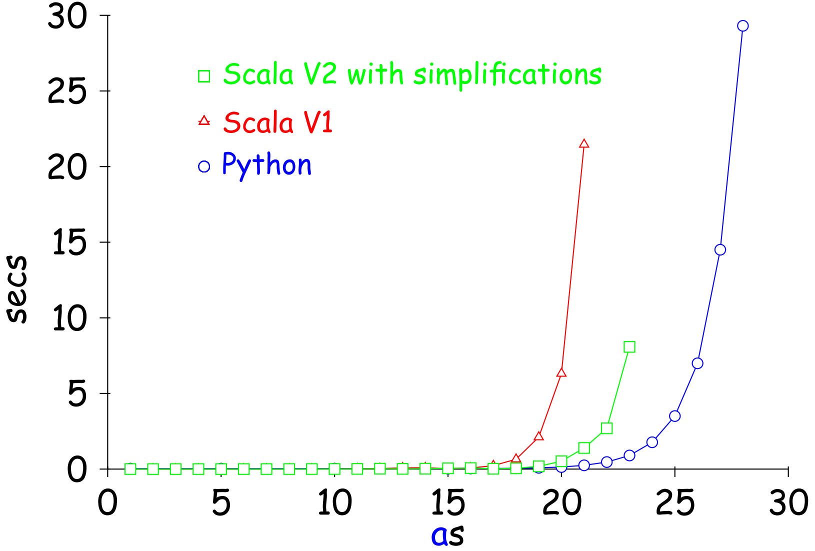
$(a?\{n\})a\{n\}$ in Python

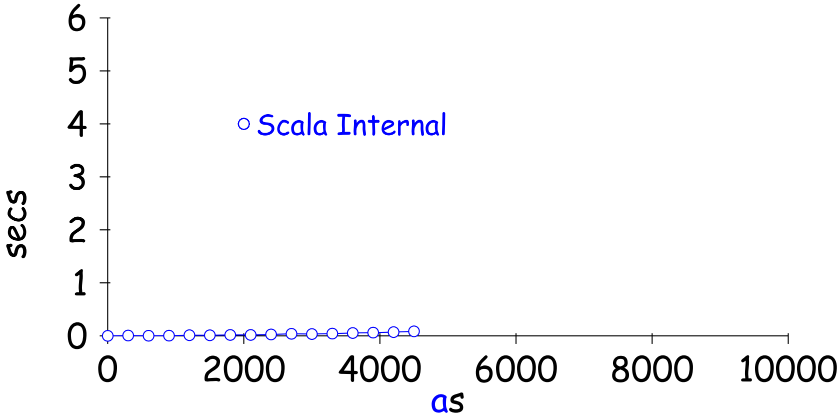


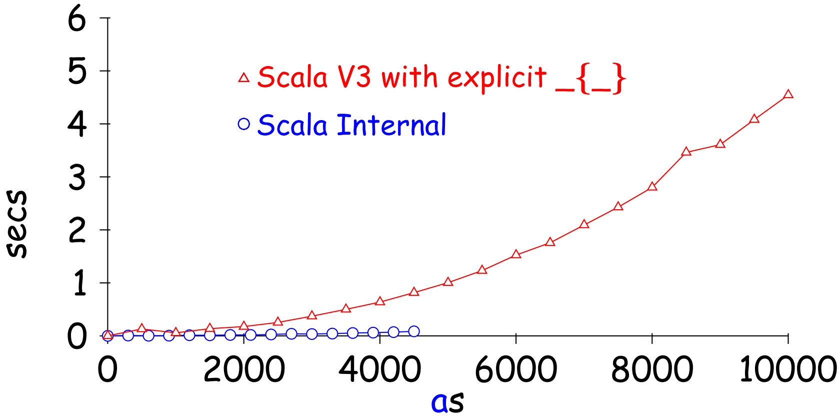
$(a?\{n\})a\{n\}$ in Python



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




Scala Internal
secs
as
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000


Scala V3 with explicit _{_}
Scala Internal
secs
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000
as

Grammars

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.

Grammars

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.

We can also allow rules

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

Palindromes

$$\begin{array}{lcl} S & \rightarrow & \epsilon \\ S & \rightarrow & a \cdot S \cdot a \\ S & \rightarrow & b \cdot S \cdot b \end{array}$$

Palindromes

$$
\begin{array}{lcl}
S & \rightarrow & \epsilon \\
S & \rightarrow & a \cdot S \cdot a \\
S & \rightarrow & b \cdot S \cdot b
\end{array}
$$

or

$$S \rightarrow \epsilon \mid a \cdot S \cdot a \mid b \cdot S \cdot b$$

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

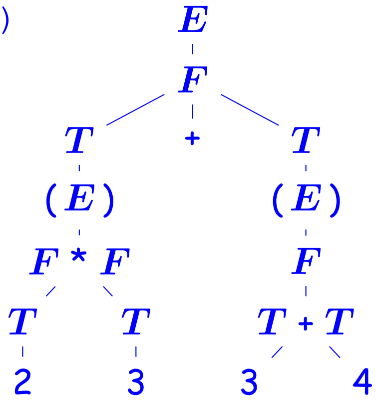
$$\begin{array}{lcl} E & \rightarrow & num\_token \\ E & \rightarrow & E \cdot + \cdot E \\ E & \rightarrow & E \cdot - \cdot E \\ E & \rightarrow & E \cdot * \cdot E \\ E & \rightarrow & (\cdot E \cdot) \end{array}$$

`1 + 2 * 3 + 4`

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 grammar 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`

Chomsky Normal Form

All rules must be of the form

$$A \rightarrow a$$

or

$$A \rightarrow B \cdot C$$

CYK Algorithm

$$
\begin{array}{lcl}
S & \rightarrow & N \cdot P \\
P & \rightarrow & V \cdot N \\
N & \rightarrow & N \cdot N \\
N & \rightarrow & \texttt{students} \,|\, \texttt{Jeff} \,|\, \texttt{geometry} \,|\, \texttt{trains} \\
V & \rightarrow & \texttt{trains}
\end{array}
$$

`Jeff trains geometry students`

CYK Algorithm

- runtime is $O(n^3)$
- grammars need to be transferred into CNF