Automata and Formal Languages, #course[15103]

Regular Languages, Expressions and Applications

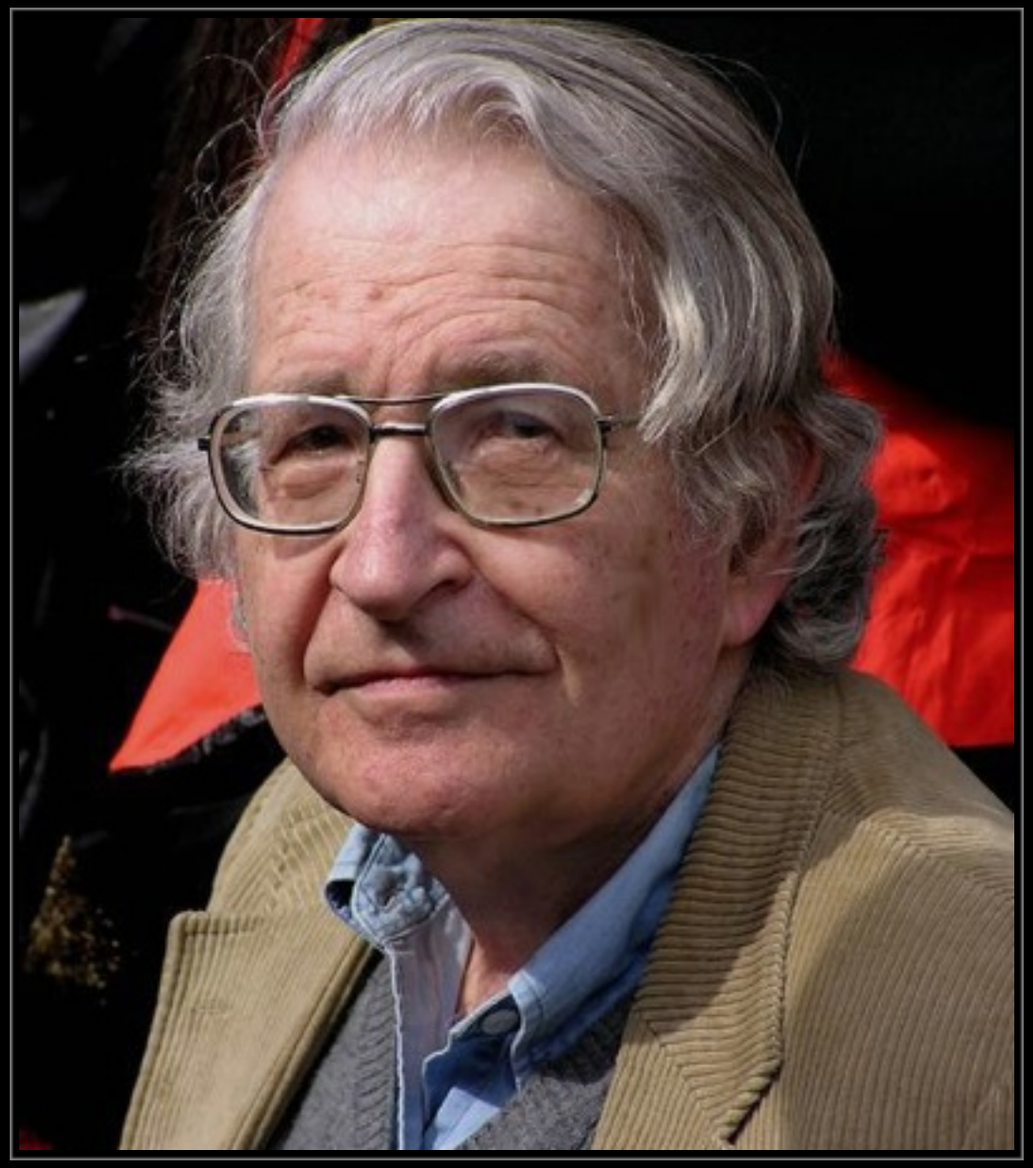
Dr. Vadim Zaytsev a.k.a. @grammarware

Roadmap

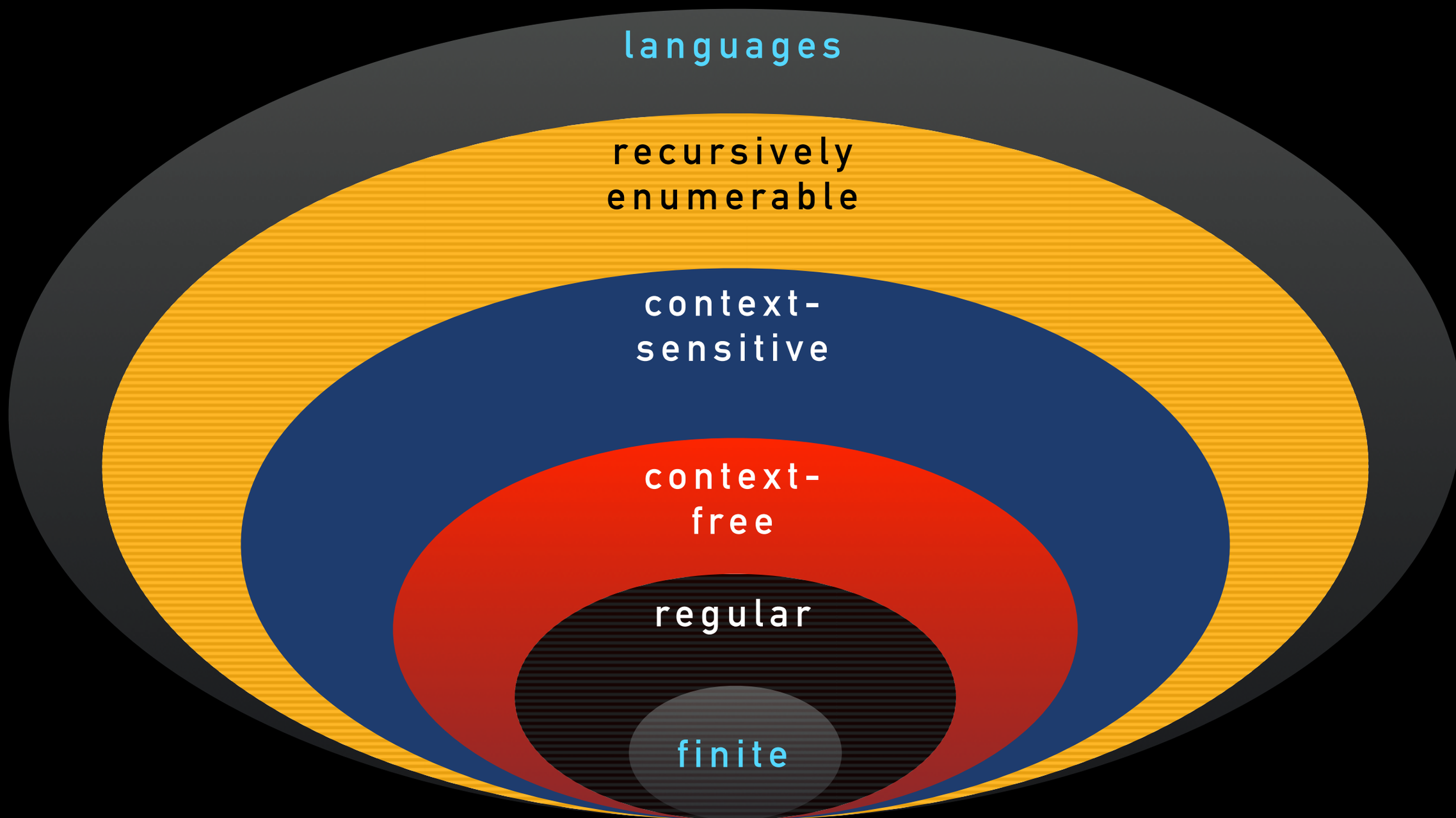
- Chomsky hierarchy revisited
- How to see if the language is regular?
- The class of regular languages
- Tools to work with regular languages
- ~~Advanced methods~~

source is given at the bottom of each slide

CHOMSKY HIERARCHY

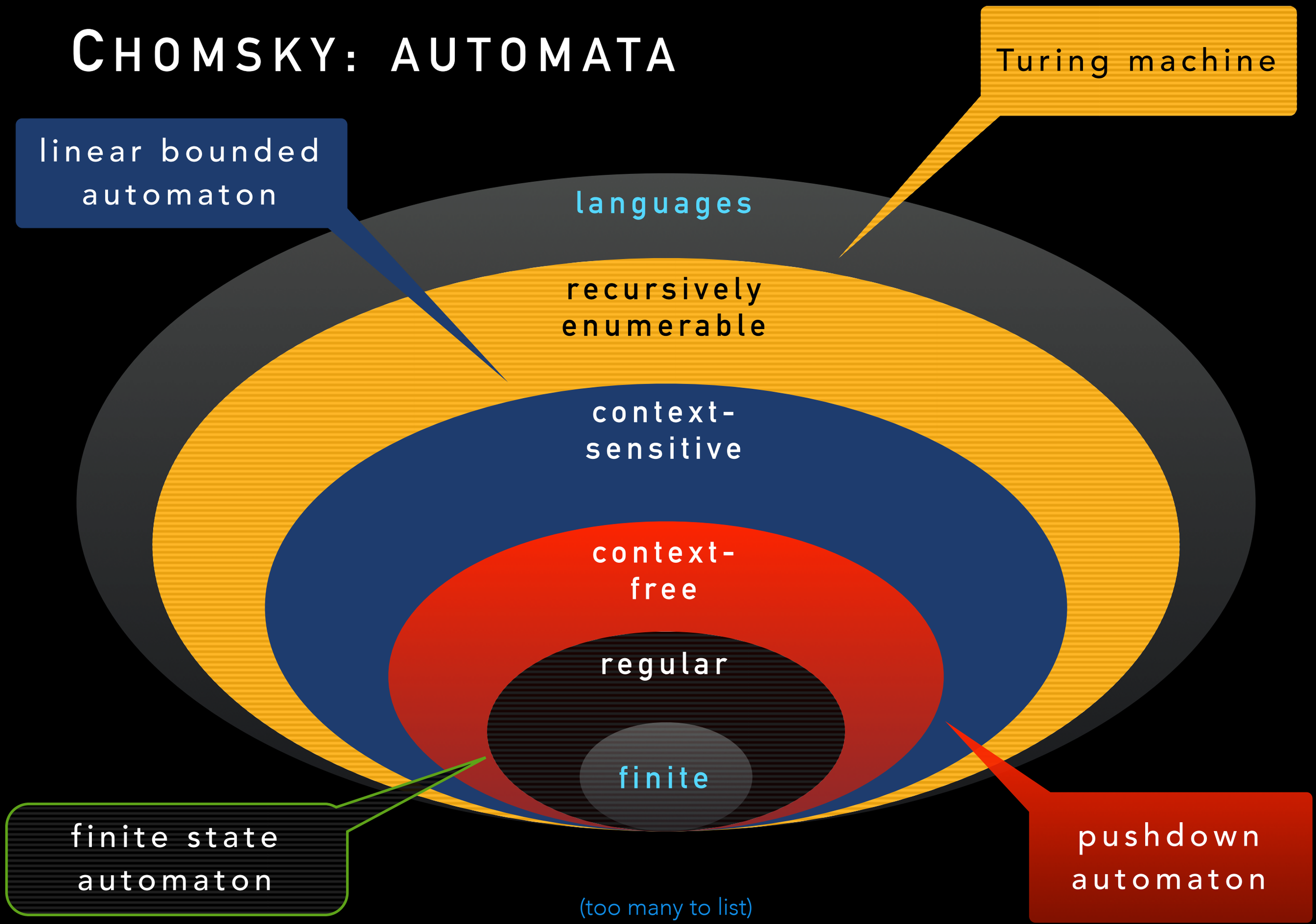


Chomsky hierarchy

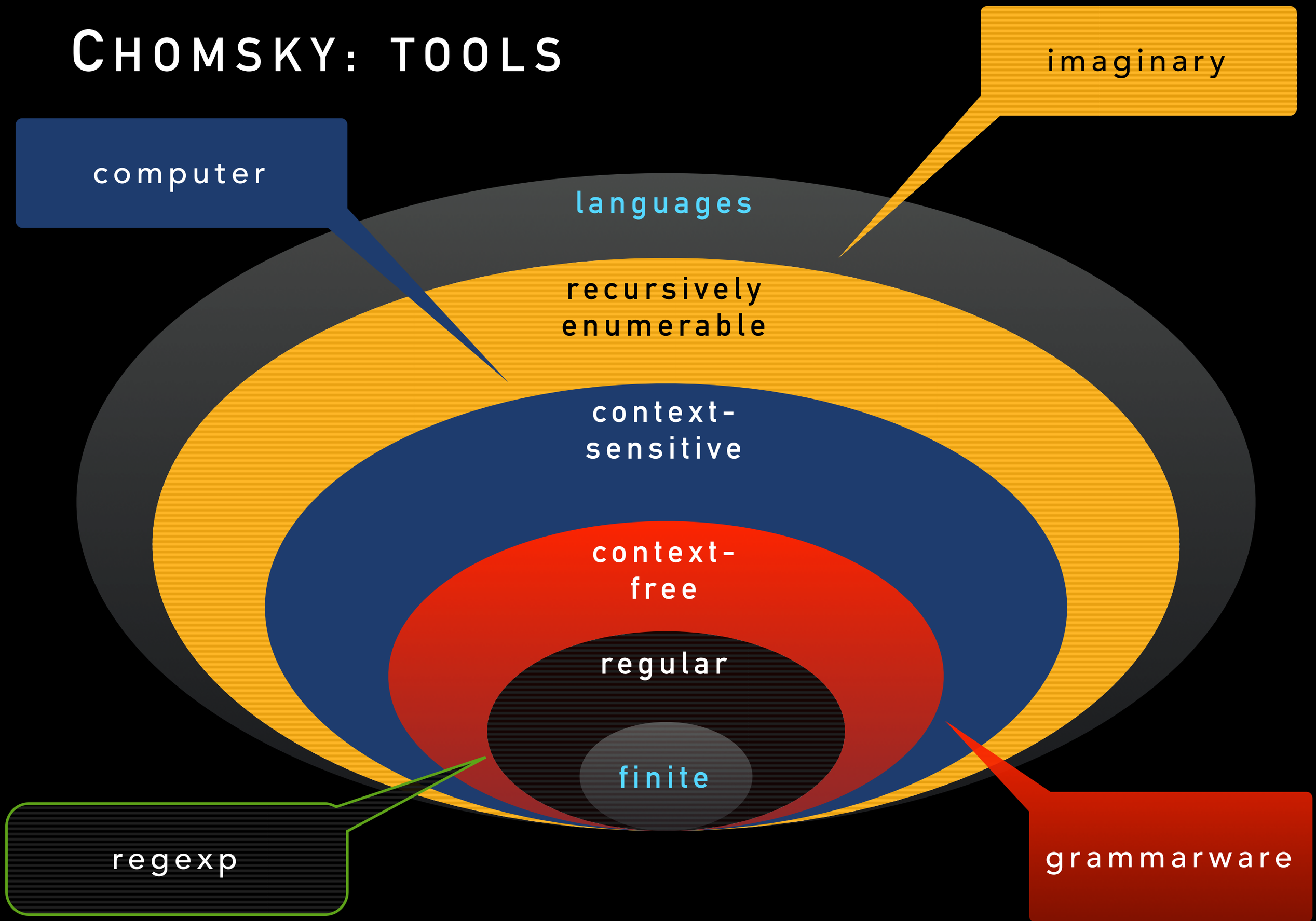


Noam Chomsky. *On Certain Formal Properties of Grammars*, Information & Control 2(2):137–167, 1959.

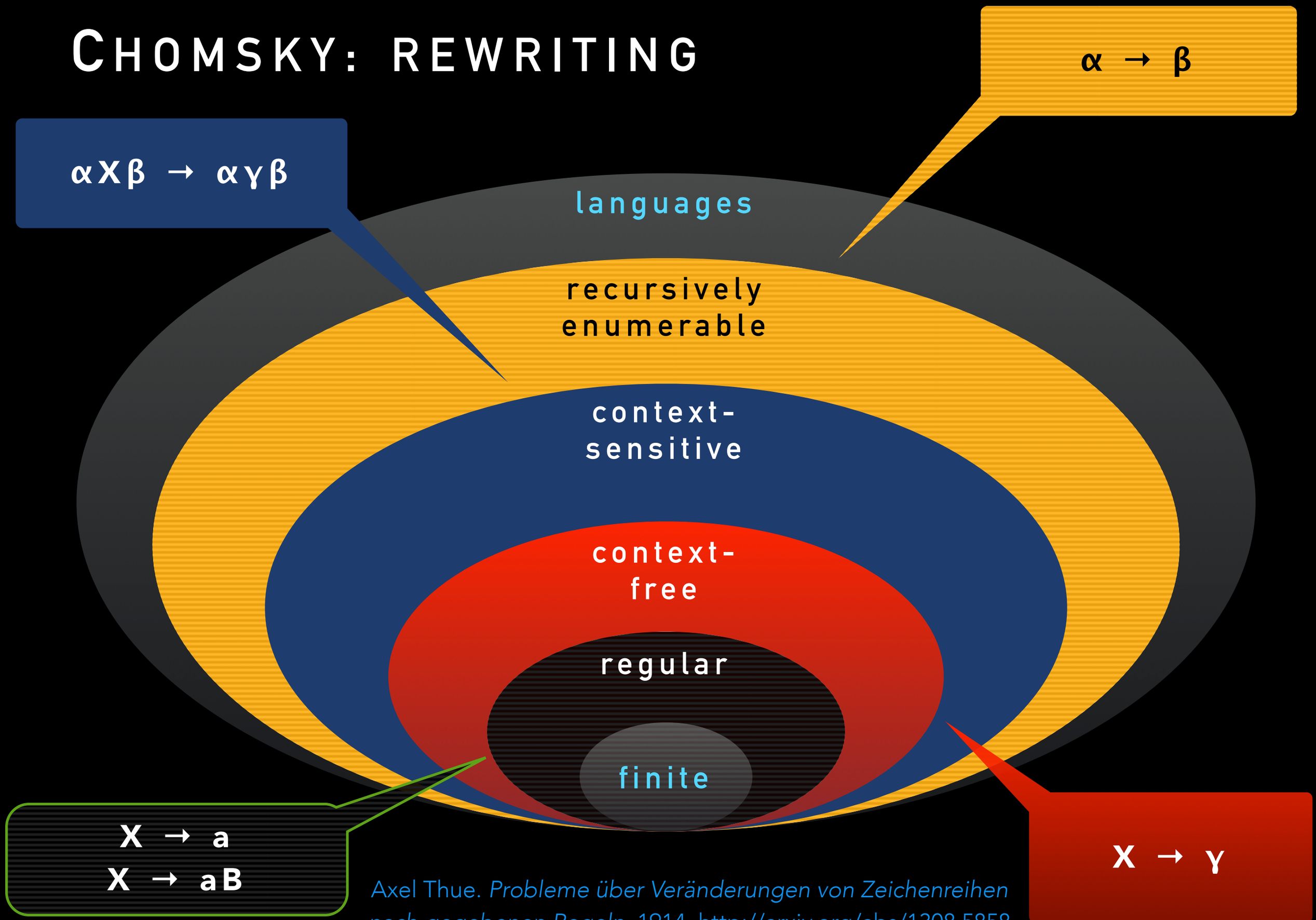
Chomsky: automata



Chomsky: tools



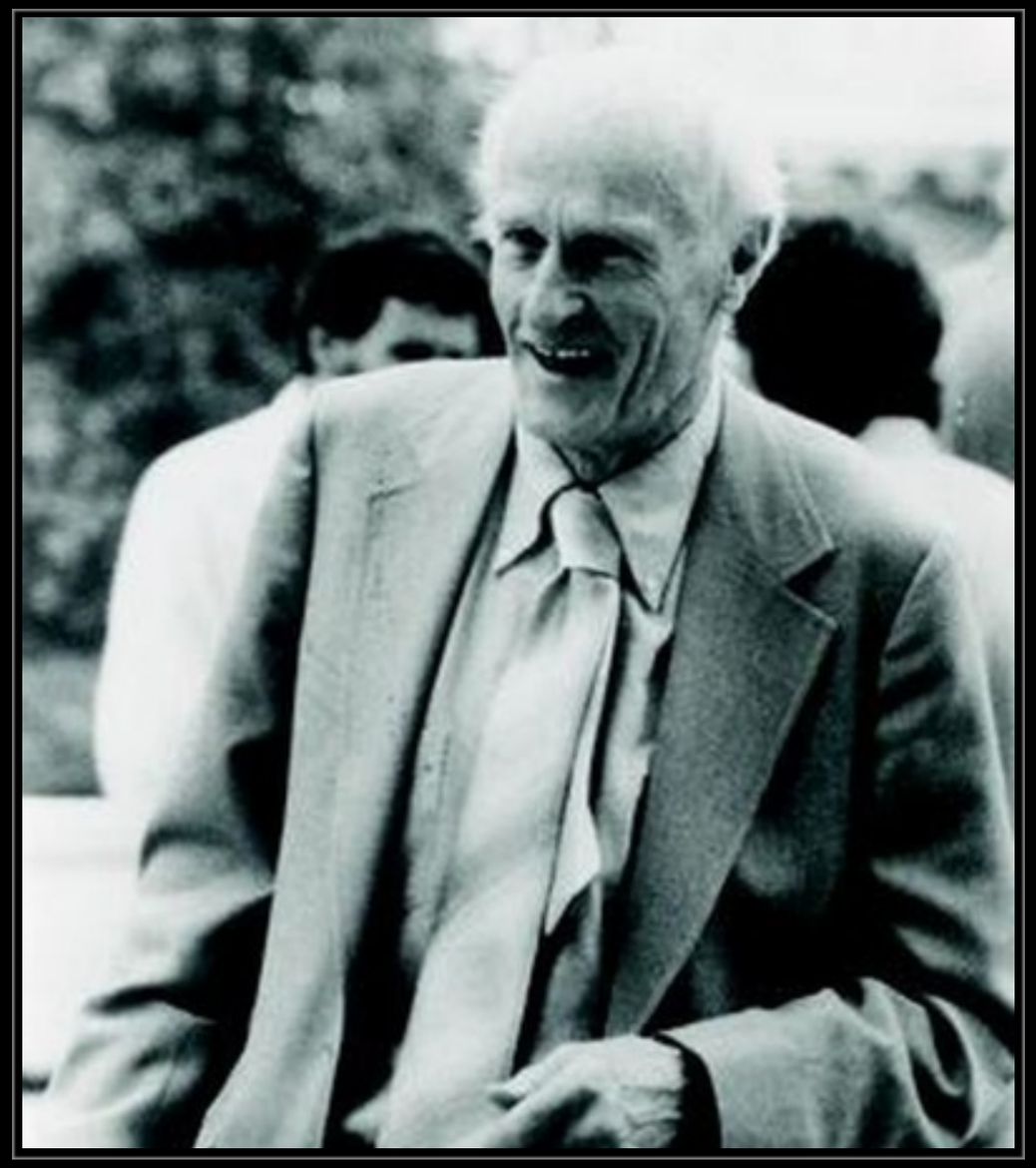
Chomsky: rewriting



Axel Thue. *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln*, 1914. http://arxiv.org/abs/1308.5858

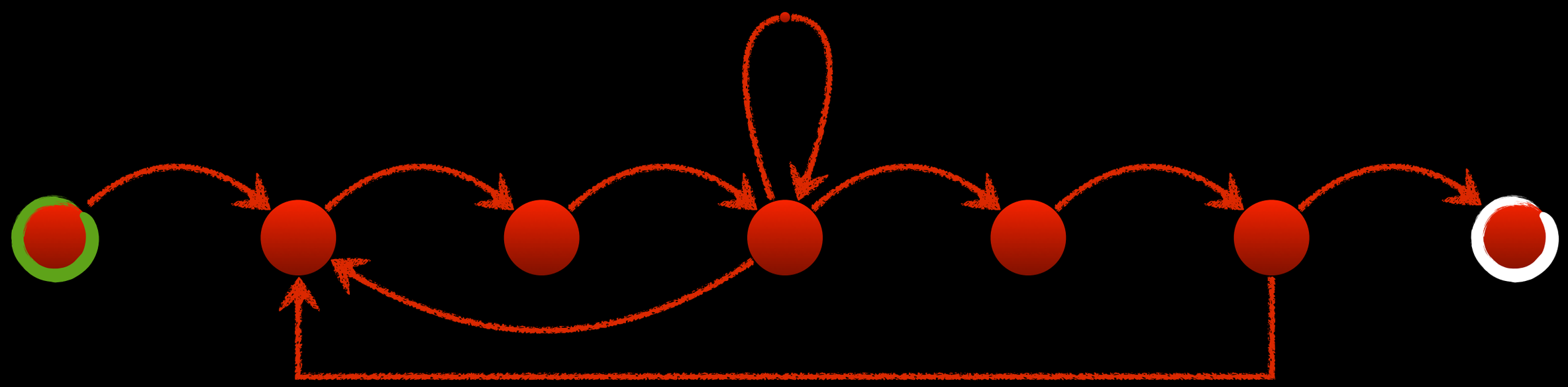
Regexps revisited

- Regular sets by Stephen Kleene in 1956
- ∅, ε, letters from Σ
- concatenation
- iteration
- alternation
- Precisely fit the regular class



S. C. Kleene, *Representation of Events in Nerve Nets and Finite Automata*. In *Automata Studies*, pp. 3–42, 1956.
photo from: Konrad Jacobs, S. C. Kleene, 1978, MFO.

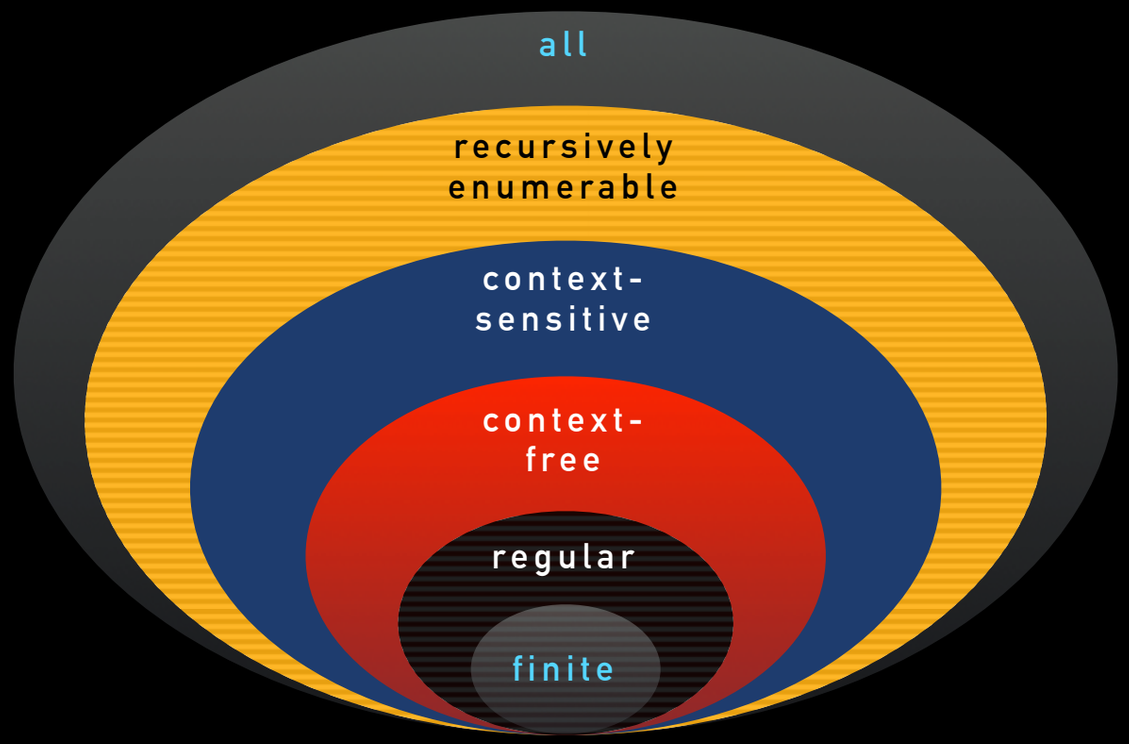
Deterministic Finite Automaton



C. E. Shannon, W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, 1949.
(finite state grammars and finite diagrams and finite state Markov processes)

To which class do languages belong?

- $\varnothing$ FINITE
- $\{\varepsilon\}$ FINITE
- $\{\varepsilon\}$ in a non-empty alphabet FINITE
- $\{x, y, z\}$ FINITE
- $\{0^n \mid n > 1\}$ REGULAR
- decimal numbers REGULAR
- $\{0^n 1^n \mid n > 1\}$ C-FREE
- $\{0^n 1^{2n} \mid n > 1\}$ C-FREE
- $\{0^n 1^n 2^n \mid n > 1\}$ C-SENSITIVE



interactive

Is a task solvable by regular means?

- ✓ Substring search
  - grep, contains(), find(), substring(), …
- ✓ Substring replacement
  - sed, awk, perl, vim, replace(), replaceAll(), …
- ✗ Pretty-printing
  - VS.NET, Sublime, TextMate, …

Is a task solvable by regular means?

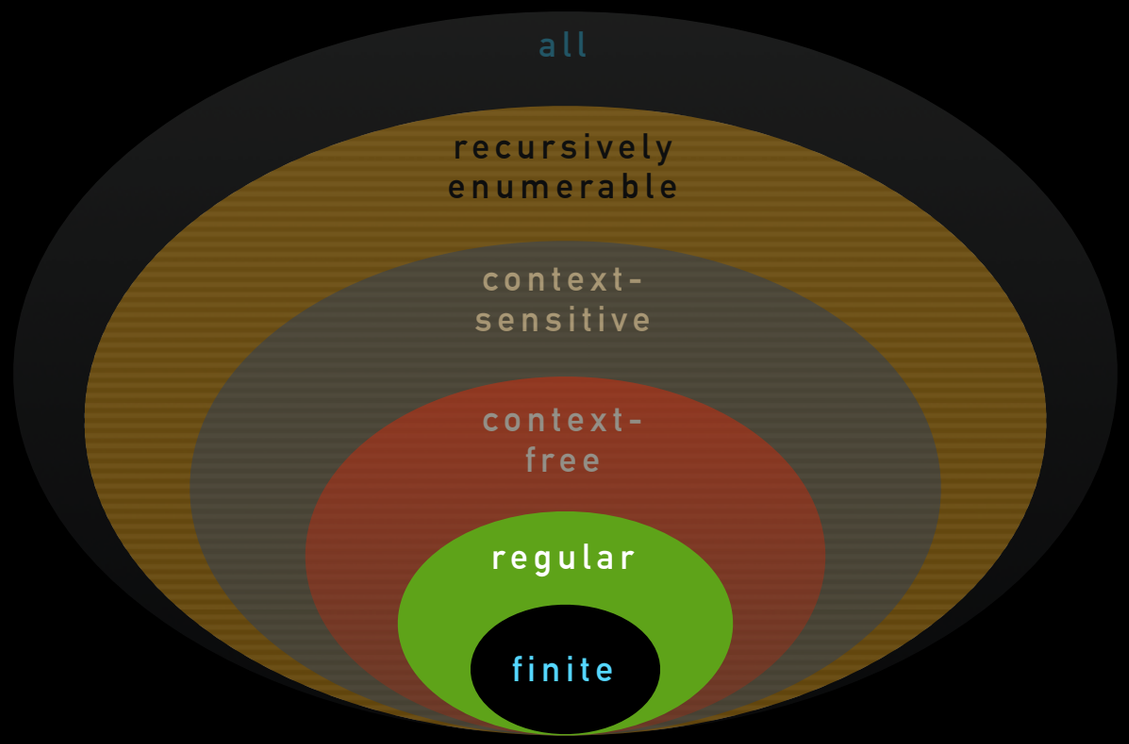
- ✓ Counting [non-empty] lines in a file
  - wc -l, grep -c “”
  - grep -v “^$”, sed -n /./p | wc -l, …
- ✗ Parsing HTML
  - <BODY><TABLE><P><A HREF=…
- ✓ Parsing a postcode
  - 1098 XG, …

interactive

HOW TO PROVE WHICH CLASS A LANGUAGE BELONGS TO

Pumping Lemma

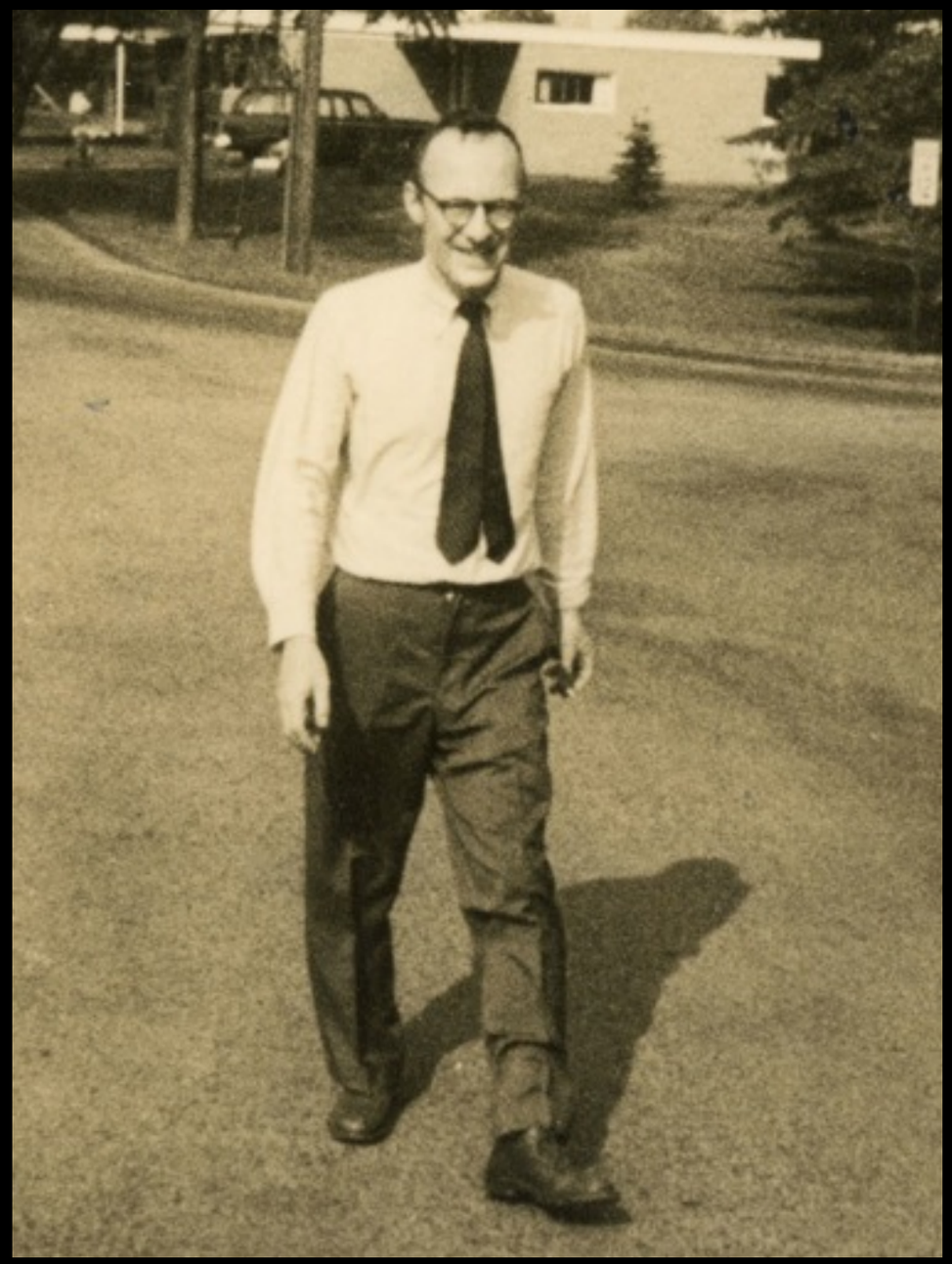
for regular languages

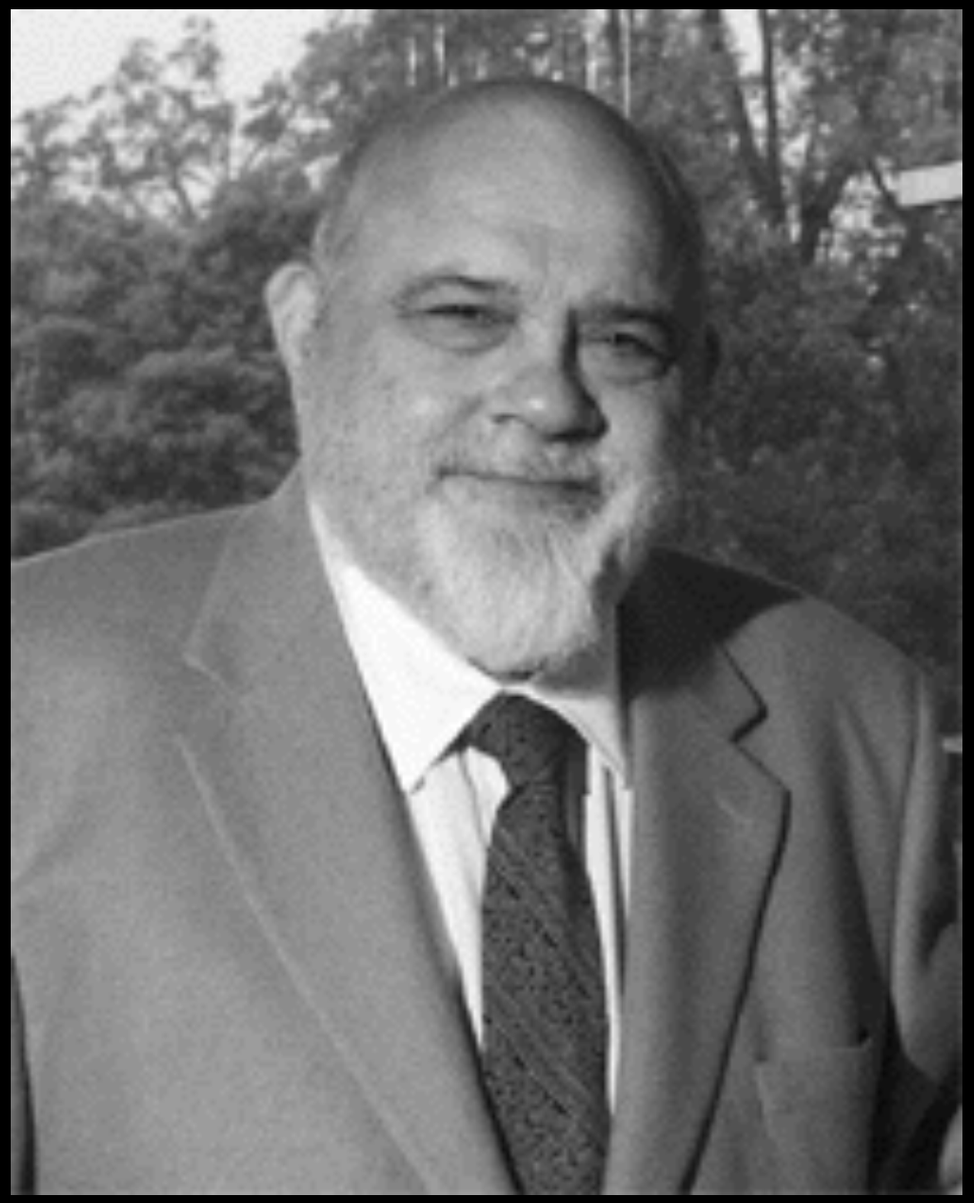


- In simple terms
  - sufficiently long words have repeatable parts
  - (works for all infinite regular languages)
- L is regular ⇒ formula holds
- Formula does not hold ⇒ L is finite or not regular

Jos C.M. Baeten, *Models of Computation: Automata, Formal Languages and Communicating Processes*, §2.9, p.58.

John Myhill and Anil Nerode





Cornell, Faculty and Senior Researcher Profiles.
Who's That Mathematician? Paul R. Halmos Collection - Page 36.

Myhill–Nerode theorem

- Myhill-Nerode equivalence
  - $u \sim v \iff \forall w\colon (uw \in L \wedge vw \in L) \vee (uw \notin L \wedge vw \notin L)$
- Theorem: L is regular iff the number of Myhill-Nerode equivalence classes is finite.
- In simple terms
  - few groups of forgettable prefixes
- Works both ways

Anil Nerode, *Linear Automaton Transformations*, Proceedings of the AMS 9, 1958.

Limited memory

- Advice from teh internetz:
  - how many characters must you remember from the stream?
    - bounded ⇒ regular
    - unbounded ⇒ ?
- Correct or not?

correct!
memory is limited,
alphabet is limited
⇒ prefixes are limited

Brian M. Scott, http://math.stackexchange.com/questions/282216/determine-if-a-language-is-regular-from-the-first-sight

Number of counters

StackExchange

- $\{0^i 1^n \ldots\}$
- no relation between i and n ⇒ regular
- 1 counter ⇒ context-free
- n counters ⇒ context-sensitive
- ∞ counters ⇒ recursively enumerable

Himanshu Saikia, http://math.stackexchange.com/questions/282216/determine-if-a-language-is-regular-from-the-first-sight

Disassemble/massage

- $\{0^n 1^n \mid n > 1\}$
  - $\{0^i 1^n \mid n > 1, i > 1, i \neq n\}$
- matching brackets language not regular
  - $\Rightarrow$ no matching pairs language is regular
- Many combinations of regular languages are regular
- Proving by decomposition is valid

THE CLASS OF REGULAR LANGUAGES

Class closed under complement

- If A is a regular language, then
  - $\bar{A}$ is regular
- Meaning…
  - grep -v “123” file.txt
- (Must know the alphabet Σ)
- (Actually stronger: any finite number of errors)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.
E. Stearns, J. Hartmanis, *Regularity Preserving Modifications of Regular Expressions*, Information & Control 6:55–69, 1963.

Class closed under set union

- If A and B are regular languages, then
  - $A \cup B$ is regular
- Meaning…
  - [a-z]
  - x | y | z (in some notations)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Class closed under intersection

- If A and B are regular languages, then
  - $A \cap B$ is regular
- Meaning…
  - cat file.txt | grep “abc” | grep “xyz”
- (Not true for context-free languages!)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Class closed under difference

- If A and B are regular languages, then
  - $A \setminus B$ is regular
- Meaning…
  - cat file.txt | grep “abc” | grep -v “123”
- (Not true for context-free languages!)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Class closed under iteration

- If A is a regular language, then
  - A^* and A^+ are regular
- Meaning…
  - $[a]^*$
  - $[a]^+$

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Class closed under concatenation

- If A and B are regular languages, then
  - AB is regular
- Meaning…
  - [Bb][Oo][Dd][Yy]
- (Just glue regexps; in practice, watch out for subgroups)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Class [sometimes] closed under decomposition

- If A is a regular language, then
  - “front halves” is regular
  - “tail halves” is regular
  - “middle thirds” is regular
  - “arbitrary halves/thirds” is regular
- NB: glued side thirds is NOT regular

E. Stearns, J. Hartmanis, *Regularity Preserving Modifications of Regular Expressions*, Information & Control 6:55–69, 1963.

Class closed under homomorphism

- If A is a regular language and
  - $h : \Sigma \rightarrow \Sigma^*$
- then
  - h(A) is regular
- Meaning that debugging is feasible
- (Even better for context-free languages: substitutions)

J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Chapter 4.

Regexps YOU need to debug

```
var whitelist =
@"</?p>|<br\s?/?>|</?b>|</?strong>|</?i>|</?em>|
</?s>|</?strike>|</?blockquote>|</?sub>|</?super>|
</?h(1|2|3)>|</?pre>|<hr\s?/?>|</?code>|</?ul>|
</?ol>|</?li>|</a>|<a[^>]+>|<img[^>]+/?>";
```

Jeff Atwood, *If You Like Regular Expressions So Much, Why Don't You Marry Them?*, 22 Mar 2005.
Jeff Atwood, Regular Expressions: Now You Have Two Problems, 27 Jun 2008.

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r
\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\
[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?
[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:
(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r
\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:
\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?
[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?
[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*
\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?
[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?
[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\
[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?
[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?
[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*
\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\
["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r
\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?
[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*
\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\
["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r
\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?
[ \t])*))*\>(?:(?:\r\n)?[ \t])*))*)?;\s*)
```

“somewhat pushes the limits of what it is sensible to do with regular expressions”

Jeff Atwood, Regex use vs. Regex abuse, 16 Feb 2005. RFC822.

Paul Warren, Mail::RFC822::Address: regexp-based address validation, 17/09/2012.

TOOLS OVERVIEW

ALL TOOLS ARE DIFFERENT

POSIX standard since 1993

(who the hell uses [[:digit:]] anyway?)

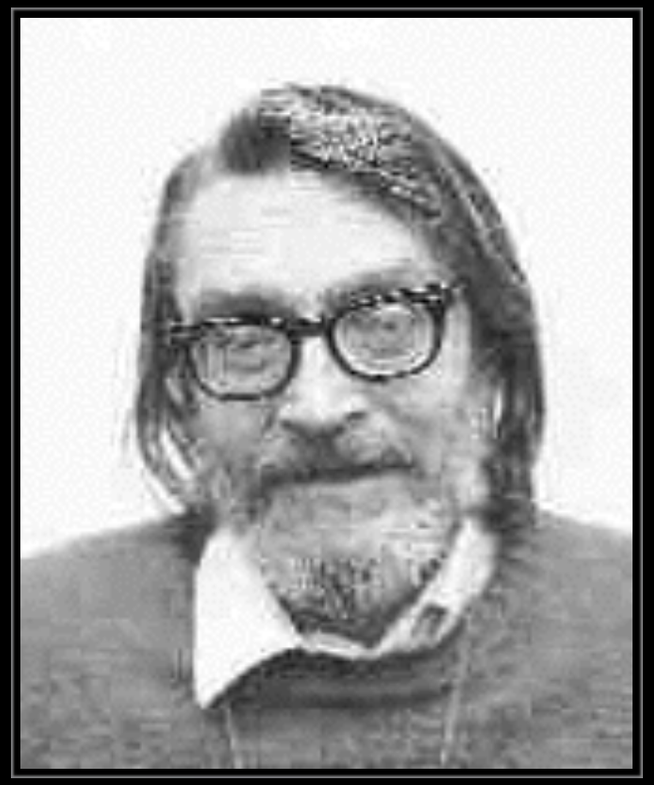
GREP

- Ken Thompson: qed, ed, grep
- grep “abc” program.c
- grep \\d file.txt
- grep ^#*\ \\w README.md

photo from: Archetypal hackers ken (left) and dmr (right).

Sed

- sed 's/Finite/Regular/g' oldfile >newfile
- sed -n 12,18p myfile
- sed 12,18d myfile
- sed 12q myfile
- sed 12,/foo/d myfile
- sed ‘$d’ myfile
- sed -n '/[0-9]\{2\}/p' myfile
- sed ‘5!s/Finite/Regular/g' oldfile >newfile
- sed ‘/Automaton/!s/Finite/Regular/g’ oldfile >newfile



Lee E. McMahon, sed, Stream EDitor, 1973 or 1974, http://www.columbia.edu/~rh120/ch106.x09
photo from: http://merdivengo.blogspot.com/2012/03/turnuva-sistemleri-uzerine.html

Origins of AWK

src — bash — 64×16

```
Last login: Sat Apr 26 15:46:02 on ttys010
>>>>>15:46<<<<< ~ zaytsev$ cd projects/zoo/src/
>>>>>15:46<<<<< src zaytsev$ ls -al
total 8
drwxr-xr-x   6 zaytsev  staff  204 Mar 31 20:57 .
drwxr-xr-x  13 zaytsev  staff  442 Mar 27 22:13 ..
-rw-r--r--   1 zaytsev  staff  281 Mar 26 17:19 Plugin.rsc
drwxr-xr-x  12 zaytsev  staff  408 Apr  9 22:46 framework
drwxr-xr-x   4 zaytsev  staff  136 Mar 26 17:19 javascript
drwxr-xr-x   3 zaytsev  staff  102 Mar 31 21:04 tryout
>>>>>15:46<<<<< src zaytsev$
```

AWK

- Turing-complete one-liner language with regexps
- Built-in variables
  - $0, NF, $1, $2, $3, …, $NF
  - FILENAME, NR, FS, OFS, RS, ORS
- Built-in operators
  - print, printf, length, $
- Can define own functions & variables

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *AWK — A Pattern Scanning and Processing Language*. SPE, 9(4): 267-279, 1979.

AWK IN ACTION

src — bash — 77×16

```
Last login: Sat Apr 26 15:46:02 on ttys010
>>>>>15:46<<<<< ~ zaytsev$ cd projects/zoo/src/
>>>>>15:46<<<<< src zaytsev$ ls -al
total 8
drwxr-xr-x   6 zaytsev  staff  204 Mar 31 20:57 .
drwxr-xr-x  13 zaytsev  staff  442 Mar 27 22:13 ..
-rw-r--r--   1 zaytsev  staff  281 Mar 26 17:19 Plugin.rsc
drwxr-xr-x  12 zaytsev  staff  408 Apr  9 22:46 framework
drwxr-xr-x   4 zaytsev  staff  136 Mar 26 17:19 javascript
drwxr-xr-x   3 zaytsev  staff  102 Mar 31 21:04 tryout
>>>>>15:46<<<<< src zaytsev$ ls -al | awk '!/total/{print $6}' | sort | uniq
Apr
Mar
>>>>>15:53<<<<< src zaytsev$
```

AWK EXAMPLES

- ```
  {
   w += NF
   c += length + 1
  }
  END { print NR, w, c }
  ```
- ```
  yes Wikipedia | awk 'NR % 4 == 1 { printf "%6d  %s\n", NR, $0 }' | sed 5q
  ```
- ```
  #!/usr/bin/awk -f
  BEGIN { print "Hello, world!" }
  ```

https://en.wikipedia.org/wiki/AWK
  ```

AWK OUTPUT

grammarkinds — bash — 77×16

```
>>>>>19:35<<<<< grammarkinds zaytsev$ yes Wikipedia | awk 'NR % 4 == 1 { prin
tf "%6d %s\n", NR, $0 }' | sed 5q
     1 Wikipedia
     5 Wikipedia
     9 Wikipedia
    13 Wikipedia
    17 Wikipedia
>>>>>19:36<<<<< grammarkinds zaytsev$
```

Lex

- Regexps used for the first phase of parsing since 1968.
- Wikipedia explains why it is used together with yacc/bison:

> (Lex is limited to simple finite state automata).[*clarification needed*]

- Collection of regexp patterns with actions
  - [a-zA-Z]+ { printf("Word: %s\n", yytext); }
  - .|\n {}
- Easy to write a tokeniser

W. L. Johnson, J. H. Porter, S. I. Ackley, D. T. Ross, Automatic Generation of Efficient Lexical Processors Using Finite State Techniques. Communications of the ACM 11 (12): 805–813, 1968.

M. E. Lesk, *LEX - A Lexical Analyzer Generator*, CSTR 39, Bell Laboratories, 1975.

Lex example

character-level grammar

```
int lineno=1;

letter          ::=   [a-zA-Z]
digit           ::=   [0-9]
id              ::=   {letter}({letter}|{digit})*
number          ::=   {digit}+
%%
    printf("\nTokeniser running -- ^D to exit\n");

^{id}                 {line();printf("<id>");}
{id}                  printf("<id>");
^{number}             {line();printf("<number>");}
{number}              printf("<number>");
^[ \t]+               line();
[ \t]+                printf(" ");
[\n]                  ECHO;
^[^a-zA-Z0-9 \t\n]+   {line();printf("\\%s\\",yytext);}
[^a-zA-Z0-9 \t\n]+    printf("\\%s\\",yytext);
%%
line()
{
    printf("%4d: ",lineno++);
}
```

M. G. Roth, CS 631, Lex example, https://www.cs.uaf.edu/~cs631/lex_token.txt

Perl [, TCL, Python, ...]

- Henry Spencer made advanced regex in 1986
  - his DFA/NFA-based TCL version is faster!
- Can be used as sed:
  - perl -pi -w -e 's/Perl/Python/g;' *
- Or, in programs:
  - $bar =~ /foo/
- Redesigned in Perl 6 (merged with PEG)

Perl regex examples

- Match
  - my ($hs, $ms, $ss) = ($time =~ m/(\d+):(\d+):(\d+)/);
- Substitute
  - $s =~ s/dog/cat/;
- Transliterate
  - $uc =~ tr/a-z/A-Z/;

Tutorialspoint, PERL Regular Expressions, http://www.tutorialspoint.com/perl/perl_regular_expression.htm

PCRE

- “Perl Compatible Regular Expressions”
  - P.S.: not compatible with Perl
  - P.P.S.: not regular
- C library by Philip Hazel (stable release Dec. 2013)
- PCREs are used in other languages
  - PHP, Ruby, JavaScript, …
- Way beyond regular: backrefs, recursion, assertions, …
  - <(\w+)>.*<\/\1>, \((?R)*\)

http://www.pcre.org

Rascal (metaprogramming)

- Java Regex
  - /xyz/ := “xyz”
  - if (/xyz/ := s) {…}
  - if (/x<m:y+>z/ := s) println(m);
  - /[0-9]+ \w*/ := “1098 XG”
- Lexical grammars
  - lexical Number = [1-9][0-9]*;
  - parse(#Number, file);

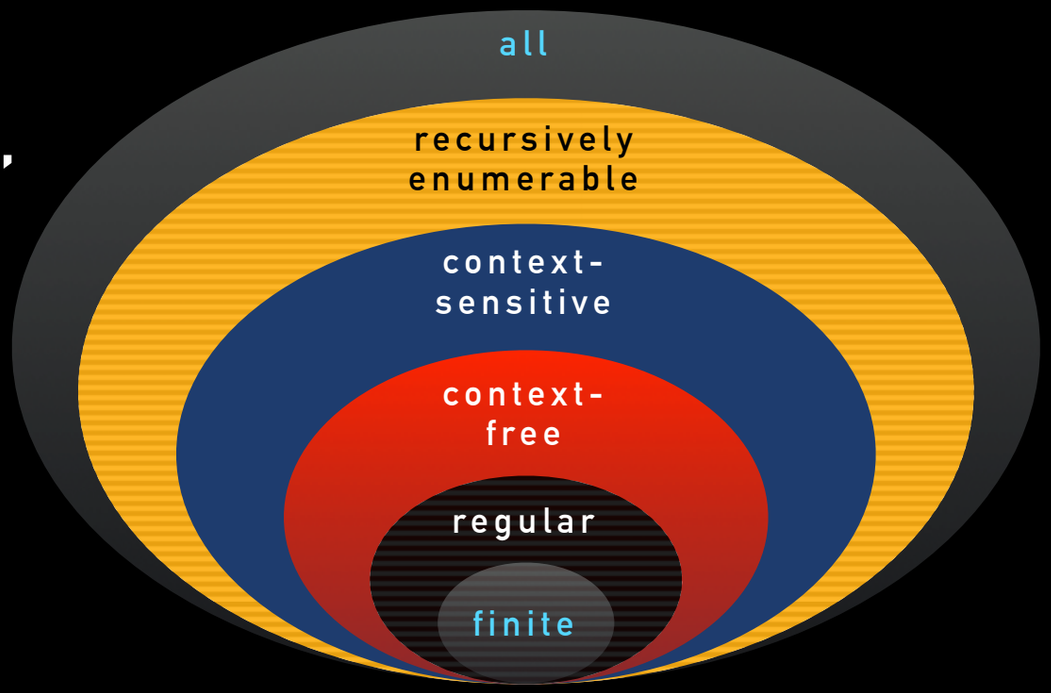
http://rascal-mpl.org/

Conclusion

- Benefits of regular languages:
  - lexical tools are fast & always applicable
  - (relatively) easy to develop
- Drawbacks:
  - very limited context
  - (usually) many false positives, requires tweaking

Summary

- Chomsky hierarchy
  - languages, automata, algorithms, rewriting systems, hardware
- Judging if regular
  - pumping lemma, Myhill-Nerode, memory, counters, disassembly
- Class closed under
  - complement, union, intersection,
  - difference, concatenation,
  - homomorphism
- Tools
  - grep, perl, sed, awk, rsc



- To read: Jeffrey E. F. Friedl, Mastering Regular Expressions: Understand Your Data and Be More Productive, O'Reilly, 2006.

Thanks for your attention!

- This was Dr. Vadim Zaytsev a.k.a. grammarware
  - grammarware.net, twitter.com/grammarware, grammarware.github.com, …
- I usually teach at Master Software Engineering
  - and do research on grammars and software languages
- Affiliations
  - UvA (2013–2014), CWI (2000, 2010–2013), Uni Koblenz (2008–2010), VU (2004–2008), UTwente (2002–2004), Rostov State Transport University (1999–2008), Rostov State University (1998–2003), Desk.nl (1999, 2001)
- Slides are CC-BY-SA: grammarware.net/slides/2014/regular.pdf