Regular expressions and automata

Adrien Mogenet

ContentSquare *am@csq.io*

September 9, 2016

Objectives

- Regexes are simple (really? all the regexes?)

Objectives

- Regexes are simple (really? all the regexes?)
- Regexes are complex (no kidding!)

Objectives

- Regexes are simple (really? all the regexes?)
- Regexes are complex (no kidding!)
- Regexes can be fun (at least visual)

Objectives

- Regexes are simple (really? all the regexes?)
- Regexes are complex (no kidding!)
- Regexes can be fun (at least visual)
- There's an history behind the scene

Objectives

- Regexes are simple (really? all the regexes?)
- Regexes are complex (no kidding!)
- Regexes can be fun (at least visual)
- There's an history behind the scene
- Using regexes in your code is not harmless

Introducing Noam Chomsky

- American, born in 1928

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)
- 7 books (**ABOUT** Chomsky)

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)
- 7 books (**ABOUT** Chomsky)
- 20+ movies starring Chomsky

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)
- 7 books (**ABOUT** Chomsky)
- 20+ movies starring Chomsky
- Even got a bee named after him: *Megachile chomskyi* (2013)

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)
- 7 books (**ABOUT** Chomsky)
- 20+ movies starring Chomsky
- Even got a bee named after him: *Megachile chomskyi* (2013)
- Released his *Chomsky hierarchy* in 1950s

Introducing Noam Chomsky

- American, born in 1928
- Linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist
- 50+ books in 50 years at MIT (Linguistics)
- 100+ books (Politics)
- 7 books (**ABOUT** Chomsky)
- 20+ movies starring Chomsky
- Even got a bee named after him: *Megachile chomskyi* (2013)
- Released his *Chomsky hierarchy* in 1950s

Chomsky hierarchy

- Hierarchy of classes of formal grammars

Formal grammars?

In an abstract world, set of production rules for a formal language.

- S: Start symbol
- A: Nonterminals (uppercase)
- a: Terminals (lowercase)
- ϵ: Empty string
- Σ: Alphabet (*e.g.* $\Sigma = a, b, c$)

Formal grammars?

Example of simple arithmetic language:

- $S \to \epsilon$
- $S \to A$
- $A \to n$ (any number)
- $A \to (A)$
- $A \to A + A$
- $A \to A - A$
- $A \to A * A$
- $A \to A/A$

Can generate something like $(2 + (3))/5$

Back to Chomsky hierarchy

Defined in 1956 by Noam Chomsky:

Grammar	Language	Production Rules
Type-0	Recursively enumerable (Turing machine)	$\alpha \to \beta$
Type-1	Context-sensitive	$\alpha A \beta \to \alpha \gamma \beta$
Type-2	Context-free	$A \to \gamma$
Type-3	Regular (Finite state automaton)	$A \to \alpha$ and $A \to \alpha A$

Chomsky hierarchy

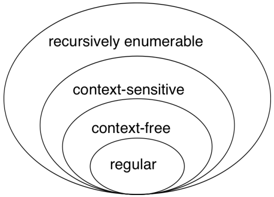
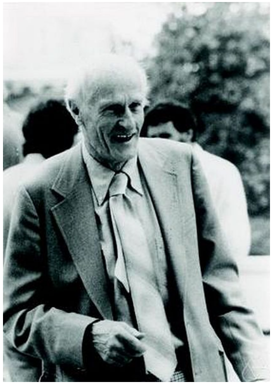


Figure: `https://en.wikipedia.org/wiki/File:Chomsky-hierarchy.svg`

Chomsky hierarchy

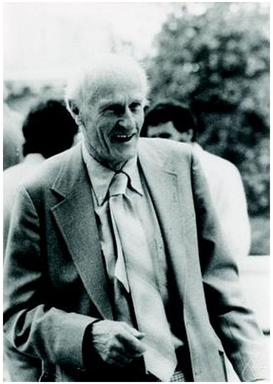
Without surprise, *regular expressions* are strongly linked to regular grammar (Type-3, generating regular languages).

Introducing Stephen Kleene



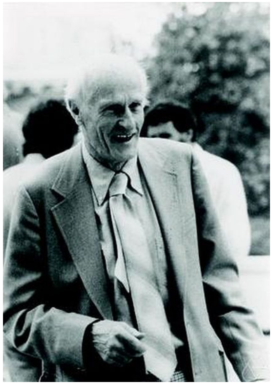
- American mathematician (1909 - 1994)

Introducing Stephen Kleene



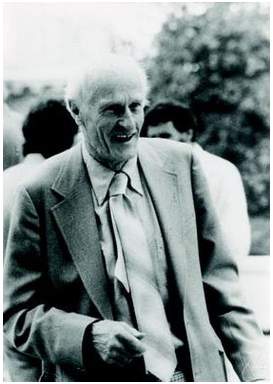
- American mathematician (1909 - 1994)
- Student of Alonzo Church (along with Alan Turing, Emil Post)

Introducing Stephen Kleene



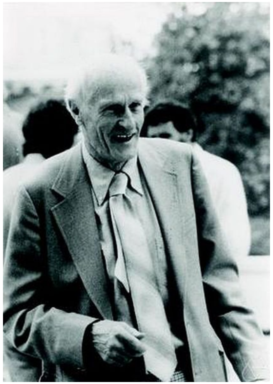
- American mathematician (1909 - 1994)
- Student of Alonzo Church (along with Alan Turing, Emil Post)
- got the National Medal of Science

Introducing Stephen Kleene



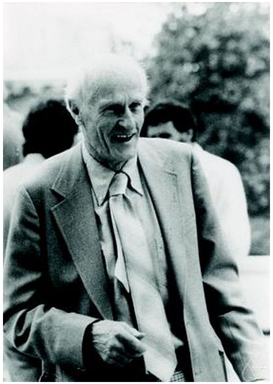
- American mathematician (1909 - 1994)
- Student of Alonzo Church (along with Alan Turing, Emil Post)
- got the National Medal of Science
- Formalized *regular languages*

Introducing Stephen Kleene



- American mathematician (1909 - 1994)
- Student of Alonzo Church (along with Alan Turing, Emil Post)
- got the National Medal of Science
- Formalized *regular languages*
- ...and invented regular expressions (1956)

Introducing Stephen Kleene



- American mathematician (1909 - 1994)
- Student of Alonzo Church (along with Alan Turing, Emil Post)
- got the National Medal of Science
- Formalized *regular languages*
- ...and invented regular expressions (1956)
- (Right after writing alternative proof to the Gödel's incompleteness theorems)

Regular language?

Definition

Regular language (or *rational language*) can be defined as a language recognized by a finite automaton. *i.e.* (equivalent properties):

- language of a regular expression

Regular language?

Definition

Regular language (or *rational language*) can be defined as a language recognized by a finite automaton. *i.e.* (equivalent properties):

- language of a regular expression
- accepted by a read-only Turing machine

Regular language?

Definition

Regular language (or *rational language*) can be defined as a language recognized by a finite automaton. *i.e.* (equivalent properties):

- language of a regular expression
- accepted by a read-only Turing machine
- accepted by a nondeterministic finite automaton (NFA)

Regular language?

Definition

Regular language (or *rational language*) can be defined as a language recognized by a finite automaton. *i.e.* (equivalent properties):

- language of a regular expression
- accepted by a read-only Turing machine
- accepted by a nondeterministic finite automaton (NFA)
- accepted by a deterministic finite automaton (DFA)

Regular language?

Formal definition (Production rules)

$$A \rightarrow a$$
$$A \rightarrow aB$$

Examples

- Empty string language $\{\epsilon\} = \emptyset^*$ //
- Singleton language $\Sigma = \{a\}$ // aaaaa
- Language over $\Sigma = \{a, b\}$ // aaaaabb

NOT a regular language

Type-1 grammar (context-sensitive)

$$\alpha A\beta \to \alpha\gamma\beta$$

NOT a regular language

Type-1 grammar (context-sensitive)

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

Famous example

- Set of strings $\{a^n b^b | n \geq 0\}$ // aaabbb

NOT a regular language

Type-1 grammar (context-sensitive)

$$\alpha A\beta \rightarrow \alpha\gamma\beta$$

Famous example

- Set of strings $\{a^n b^b | n \geq 0\}$ // aaabbb

And so...

THIS CAN'T BE ACCEPTED BY A REGEX!

Translations

- French Wikipedia page title: “Expression rationnelle”

Translations

- French Wikipedia page title: “Expression rationnelle”
- Other available translation that sounds correct (to me): “Expression normale”

Translations

- French Wikipedia page title: “Expression rationnelle”
- Other available translation that sounds correct (to me): “Expression normale”
- These grammars and languages are everything but “régulier”

Translations

- French Wikipedia page title: “Expression rationnelle”
- Other available translation that sounds correct (to me): “Expression normale”
- These grammars and languages are everything but “régulier”
- Fun fact: other european countries chose equivalent of “régulier”:
  Espressione regolare (it), Expressão regular (pt), Reguljära uttryck (sw), Regulārā izteiksme (le)

Wikipedia War

Please feed the troll

“Ah oui ? Qui d’autre que l’Office Québécois pour critiquer cette horreur de mot qui ressemble á l’anglais ?”

→ *Moi, modeste informaticien et Français (...)*

`https://fr.wikipedia.org/wiki/Discussion:Expression_rationnelle`

Are regexes really regular?

Meanwhile, in 2000s:

- Regexes exceed regular languages (Type-3) and became context-sensitive (Type-1):
- This: `(.+)\1` $\mapsto$ will match *papa* or *Pika!Pika!*.

Perl regexes performances

Exemple: recognize $a?^n a^n$ against strings a^n.

```
time perl -e '("a" x 20) =~ /^(a?){20}(a){20}$/;'
real    0m0.132s
```

```
time perl -e '("a" x 26) =~ /^(a?){26}(a){26}$/;'
real    ???????
```

Perl regexes performances

Exemple: recognize $a?^n a^n$ against strings a^n.

```
time perl -e '("a" x 20) =~ /^(a?){20}(a){20}$/;'
real    0m0.132s
```

```
time perl -e '("a" x 26) =~ /^(a?){26}(a){26}$/;'
real    0m11.507s
```

From regexes to automata

Regexp = `kook`

$$\begin{tikzcd}
\text{start} \arrow[r] & s_0 \arrow[r, "\text{k}"] & s_1 \arrow[r, "\text{o}"] & s_2 \arrow[r, "\text{o}"] & s_3 \arrow[r, "\text{k}"] & s_4
\end{tikzcd}$$

Nondeterministic finite automaton

- Introduced in 1959

Nondeterministic finite automaton

- Introduced in 1959
- Recognize regular languages

Nondeterministic finite automaton

- Introduced in 1959
- Recognize regular languages
- Reading an input symbol is required for each state transition

Nondeterministic finite automaton

- Introduced in 1959
- Recognize regular languages
- Reading an input symbol is required for each state transition
- Its transitions are not uniquely determined by its source state and input symbol

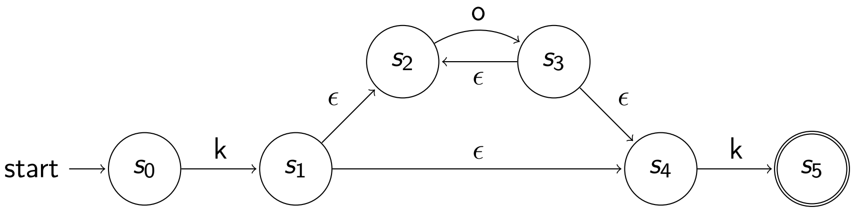
Nondeterministic finite automaton

- Introduced in 1959
- Recognize regular languages
- Reading an input symbol is required for each state transition
- Its transitions are not uniquely determined by its source state and input symbol

And exist under many, many variations.

Regexp? Automata?

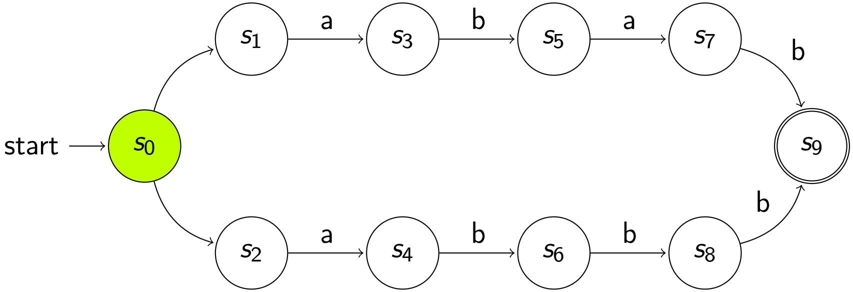
Regexp = `ko*k`



Another example

Regexp = `abab|abbb`

Input = `⟩abbb`

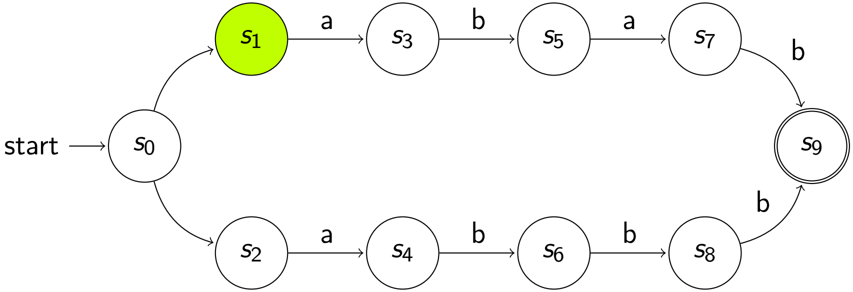


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Another example

Regexp = `abab|abbb`

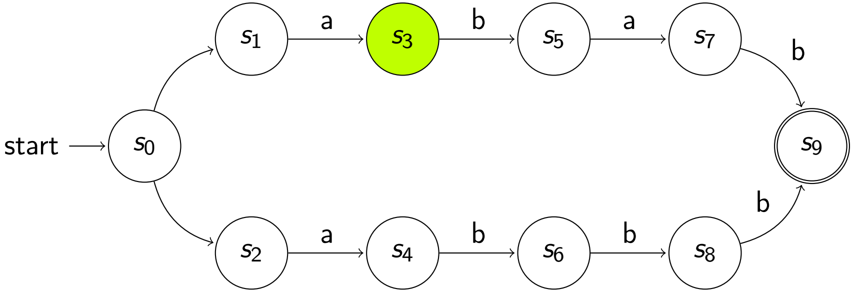
Input = `⟩abbb`



Another example

Regexp = `abab|abbb`

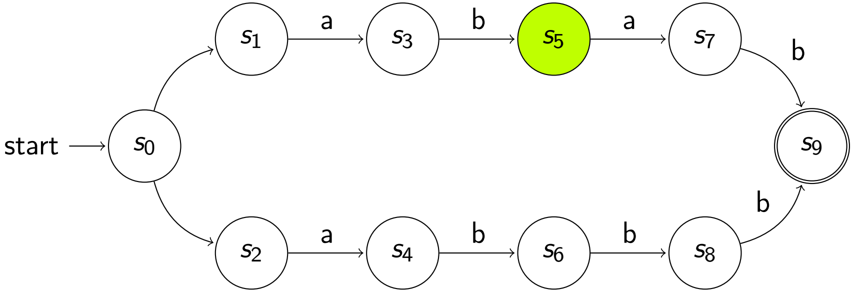
Input = `a⟩bbb`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Another example

Regexp = `abab|abbb`

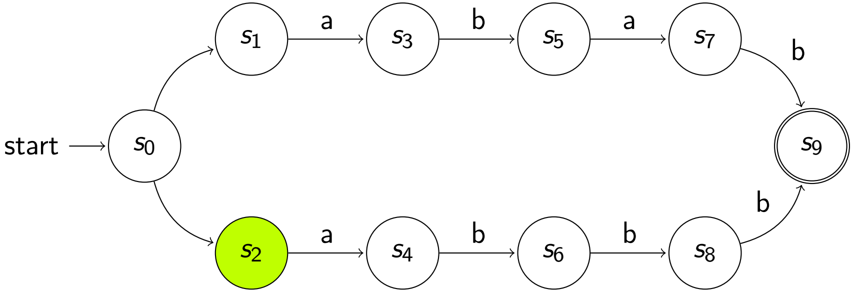
Input = `ab⟩bb`



Another example

Regexp = `abab|abbb`

Input = `⟩abbb`

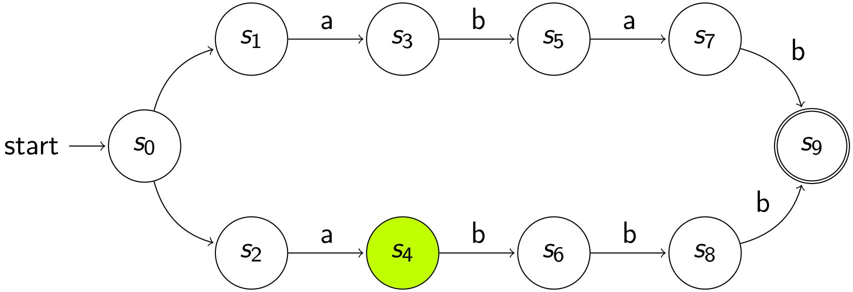


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"] &
\end{tikzcd}$$

Another example

Regexp = `abab|abbb`

Input = `a⟩bbb`

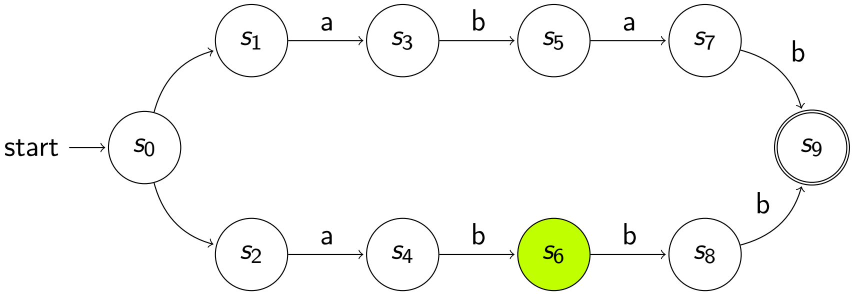


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Another example

Regexp = `abab|abbb`

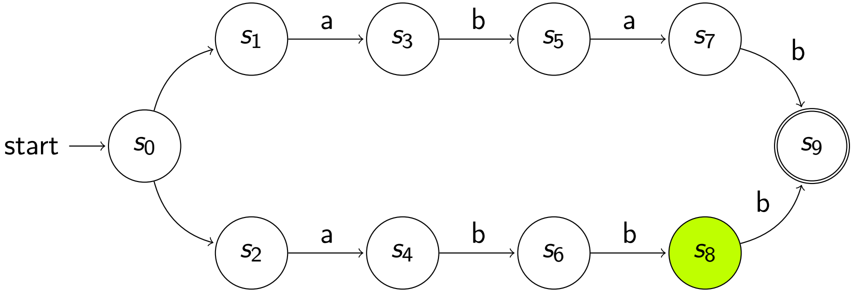
Input = `ab⟩bb`



Another example

Regexp = `abab|abbb`

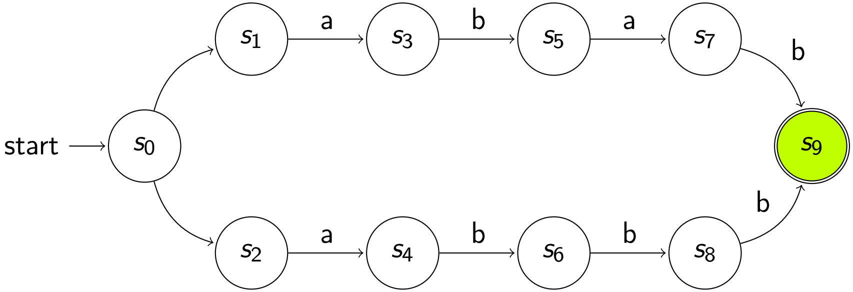
Input = `abb⟩b`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Another example

Regexp = `abab|abbb`

Input = `abbb⟩`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Problem

Problem

Backtracking!

This is what happened with `(a?)26(a)26`. (At least) 2 alternatives:

- Thompson's NFA
- DFA (Deterministic Finite Automaton)

Alternative 1: Thompson's NFA

- Optimized by Ken Thompson, (Bell Telphone Labs Inc.), in a 1968 CACM paper
- Allows several possible states at the same moment

Ken Thompson

- American computer scientist (1943)

Ken Thompson

- American computer scientist (1943)
- Designed and implemented the original Unix operating system

Ken Thompson

- American computer scientist (1943)
- Designed and implemented the original Unix operating system
- Co-invented the Go programming language

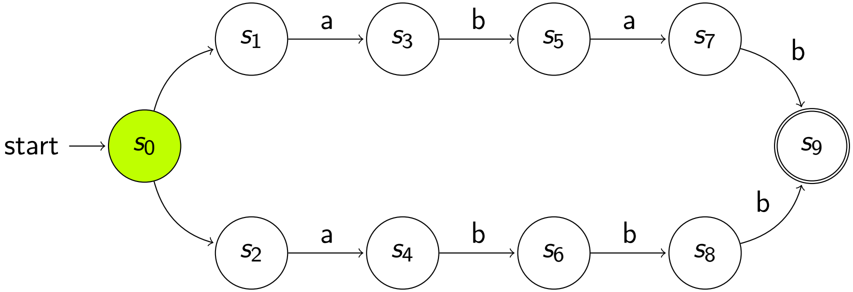
Ken Thompson

- American computer scientist (1943)
- Designed and implemented the original Unix operating system
- Co-invented the Go programming language
- Turing Award (for *backdoor attack*)

Thompson's NFA example

Regexp = `abab|abbb`

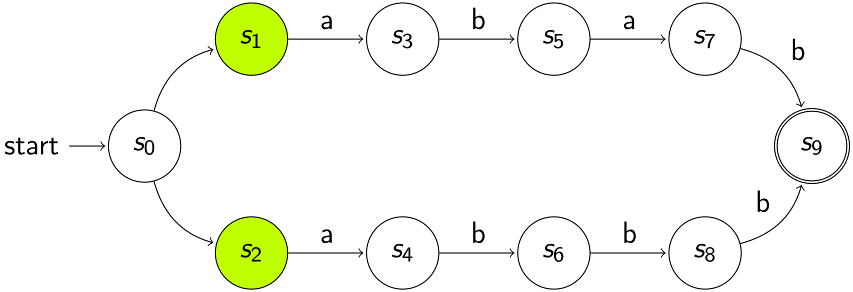
Input = `⟩abbb`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"] &
\end{tikzcd}$$

Thompson's NFA example

Regexp = `abab|abbb`

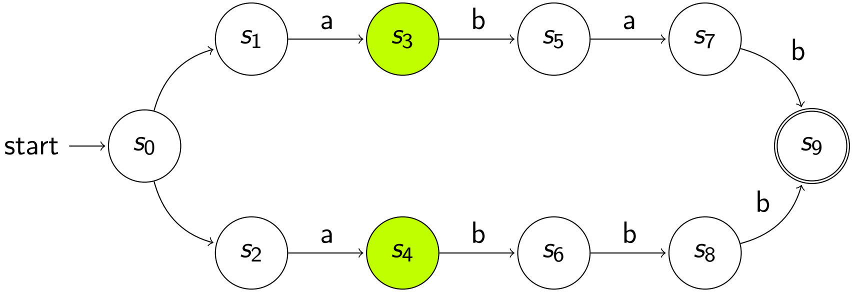
Input = `⟩abbb`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Thompson's NFA example

Regexp = `abab|abbb`

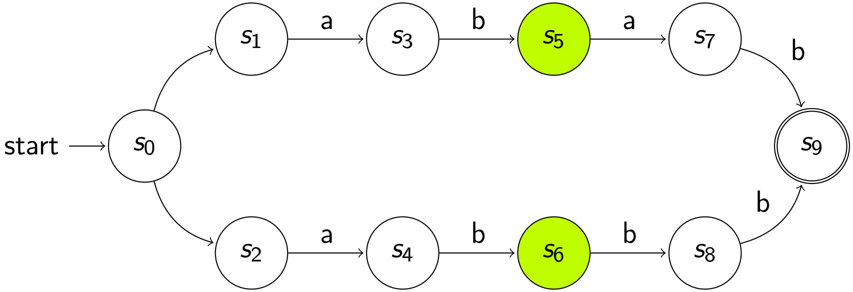
Input = `a⟩bbb`



Thompson's NFA example

Regexp = `abab|abbb`

Input = `ab⟩bb`

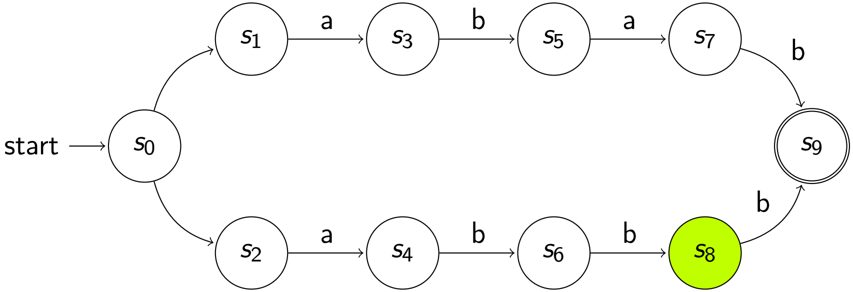


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & \boxed{s_5} \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & \boxed{s_6} \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"] &
\end{tikzcd}$$

Thompson's NFA example

Regexp = `abab|abbb`

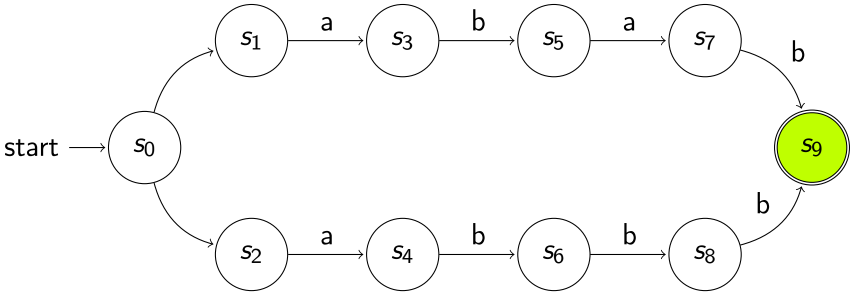
Input = `abb⟩b`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[u, bend left] \arrow[d, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

Thompson's NFA example

Regexp = `abab|abbb`

Input = `abbb⟩`


$$\begin{tikzcd}
 & s_1 \arrow[r, "a"] & s_3 \arrow[r, "b"] & s_5 \arrow[r, "a"] & s_7 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_0 \arrow[ur, bend left] \arrow[dr, bend right] & & & & s_9 \\
 & s_2 \arrow[r, "a"] & s_4 \arrow[r, "b"] & s_6 \arrow[r, "b"] & s_8 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

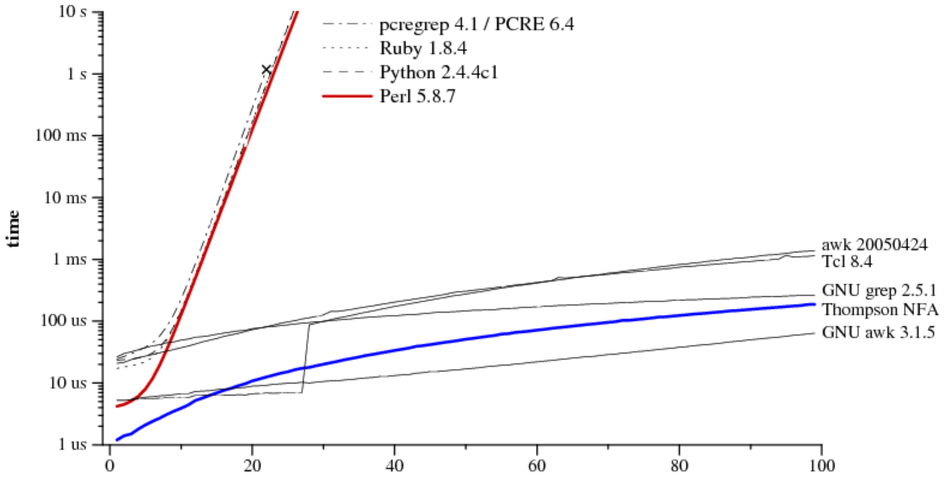


Figure: `https://swtch.com/~rsc/regexp/regexp1.html`

Alternative 2: DFA

Property

Each of its transitions is **uniquely** determined by its source state and input symbol

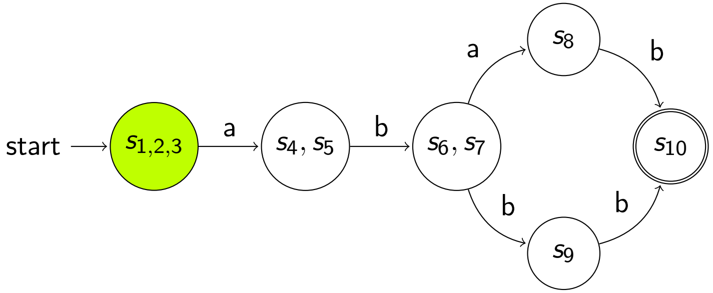
Theorem

Every NFA as an equivalent DFA [proof]

DFA example

Regexp = `abab|abbb`

Input = `⟩abbb`

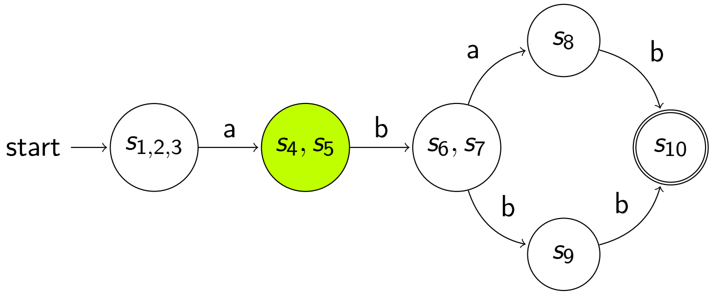


$$\begin{tikzcd}
 & & & s_8 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_{1,2,3} \arrow[r, "a"] & s_4, s_5 \arrow[r, "b"] & s_6, s_7 \arrow[u, bend left, "a"] \arrow[d, bend right, "b"'] & s_{10} \\
 & & & s_9 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

DFA example

Regexp = `abab|abbb`

Input = `a⟩bbb`

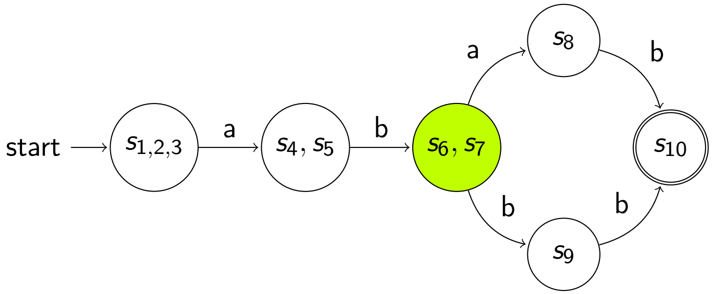


$$\begin{tikzcd}
 & & & s_8 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_{1,2,3} \arrow[r, "a"] & s_4, s_5 \arrow[r, "b"] & s_6, s_7 \arrow[u, bend left, "a"] \arrow[d, bend right, "b"'] & s_{10} \\
 & & & s_9 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

DFA example

Regexp = `abab|abbb`

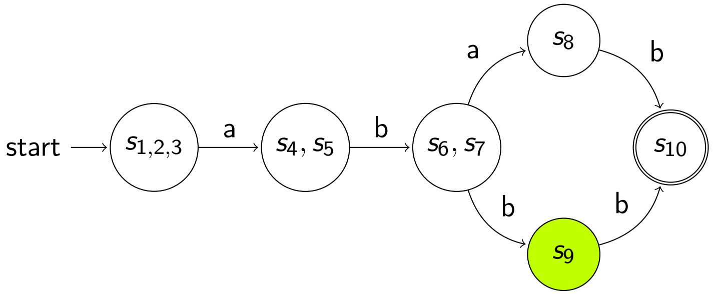
Input = `ab⟩bb`


$$\begin{tikzcd}
 & & & s_8 \arrow[dr, bend left, "b"] & \\
\text{start} \arrow[r] & s_{1,2,3} \arrow[r, "a"] & s_4, s_5 \arrow[r, "b"] & s_6, s_7 \arrow[u, bend left, "a"] \arrow[d, bend right, "b"'] & s_{10} \\
 & & & s_9 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

DFA example

Regexp = `abab|abbb`

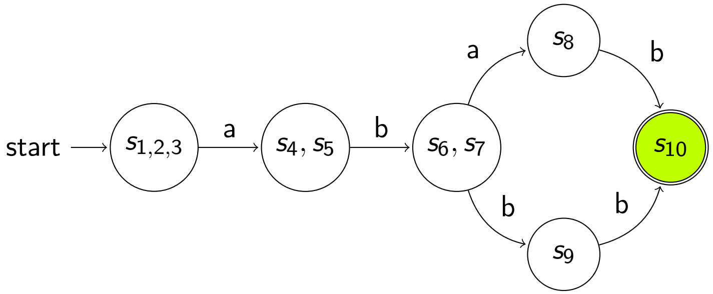
Input = `abb⟩b`


$$\begin{tikzcd}
\text{start} \arrow[r] & s_{1,2,3} \arrow[r, "a"] & s_4, s_5 \arrow[r, "b"] & s_6, s_7 \arrow[r, bend left, "a"] \arrow[r, bend right, "b"'] & s_8 \arrow[r, bend left, "b"] & s_{10} \\
 & & & & s_9 \arrow[ur, bend right, "b"'] &
\end{tikzcd}$$

DFA example

Regexp = `abab|abbb`

Input = `abbb⟩`



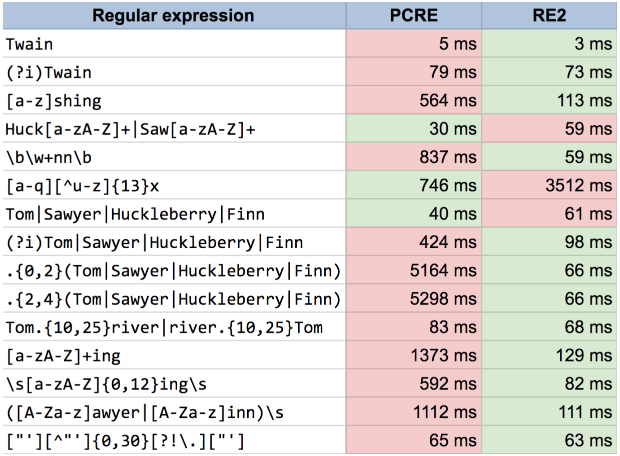
Mistery from History

Why don't everyone use Thompson's NFA or DFA???

You're not alone

- Java's `java.util.regex` uses backtracking too, so does PHP (PCRE library).
- Tcl, Awk, GNU Awk and GNU grep all use... DFA's!
- Google RE2 offers a C++ implementation (+ wrappers)

PCRE vs Google RE2



Regular expression	PCRE	RE2
`Twain`	5 ms	3 ms
`(?i)Twain`	79 ms	73 ms
`[a-z]shing`	564 ms	113 ms
`Huck[a-zA-Z]+|Saw[a-zA-Z]+`	30 ms	59 ms
`\b\w+nn\b`	837 ms	59 ms
`[a-q][^u-z]{13}x`	746 ms	3512 ms
`Tom|Sawyer|Huckleberry|Finn`	40 ms	61 ms
`(?i)Tom|Sawyer|Huckleberry|Finn`	424 ms	98 ms
`.{0,2}(Tom|Sawyer|Huckleberry|Finn)`	5164 ms	66 ms
`.{2,4}(Tom|Sawyer|Huckleberry|Finn)`	5298 ms	66 ms
`Tom.{10,25}river|river.{10,25}Tom`	83 ms	68 ms
`[a-zA-Z]+ing`	1373 ms	129 ms
`\s[a-zA-Z]{0,12}ing\s`	592 ms	82 ms
`([A-Za-z]awyer|[A-Za-z]inn)\s`	1112 ms	111 ms
`["'][^"']{0,30}[?!\.]["']`	65 ms	63 ms

Figure: Numbers from `http://sljit.sourceforge.net/regex_perf.html`

Limitations

- Backreferences (*e.g.* `<(.+)>(.*)</(\1)>` to identify a closing XML tag)
- Space consumption of DFA can be way higher:
  NFA: $O(n)$, DFA: $O(2^n)$, n is for regexp's length

Conclusion

- Be careful when using regexes
- Think about underlying engine
- Think about worst case usage (user's inputs...)
- Think about `strstr()`

Bibliography

- https://en.wikipedia.org/wiki/Regular_language
- https://en.wikipedia.org/wiki/Chomsky_hierarchy
- https://en.wikipedia.org/wiki/Stephen_Cole_Kleene
- http://www.cs.ucr.edu/~jiang/cs215/tao-new.pdf
- https://www.debuggex.com/
- http://www.borntosegfault.com/2013/03/regexp-think-dfa.html
- http://sljit.sourceforge.net/regex_perf.html
- https://en.wikipedia.org/wiki/Thompson's_construction

~~Trolls~~ Discussions

- `http://arstechnica.com/civis/viewtopic.php?f=20&t=1195549`
- `http://www.perlmonks.org/?node_id=597262`

Illustrations

- https://en.wikipedia.org/wiki/File:Chomsky-hierarchy.svg
- https://en.wikipedia.org/wiki/File:Kleene.jpg
- https://en.wikipedia.org/wiki/File:Noam_Chomsky_portrait_2015.jpg