Automata and Regular Languages

Oliver Nam Michal Matthew

January 1, 1980

Automata and Regular Languages
Oliver Nam Michal Matthew
January 1, 1980

(30s): Introduction of members

"Real World" Regex

The following regex finds all phone numbers in all files in the current folder, including PDFs:

```
$ rg-all "^(\+\d{1,2}\s)?\(?\d{3}\)?[\s.-]\d{3}[\s.-]\d{4}$"
```

Do not worry if you cannot understand it!

(1min): Show an example of a real world regex with syntactic sugar. This could either be a validation regex or bash script regex. Explains what it does briefly. Audience does not need to understand the syntax, this is just for linking theory to practice.

(style): Dramatic, trying to show off something cool and overwhelm audience with information.

Formal regex example

The following is a formally written regex (without the added syntactic sugar of the real world regex):

$$(A|a)a^*nn^*(y|i|j|h|gh)^*aa^*(h|gh)^*a^*r^*s$$

This one features much less symbols compared to the previous example, although it still looks quite complicated. We will walk you through all the necessary theory to help you understand this.

(30s): Gives a cool example of a formally written regex. This should not have any explanation. Regex should be something cool and at the end, what it represents will be revealed.

(style): Less dramatic, still trying to overwhelm but ease from real world syntax into something that audience might be able to digest.

Formal regex example

The following is a formally written regex (without the added syntactic sugar of the real world regex):

(A|a)a*nn*(y|i|j|h|gh)*aa*(h|gh)*a*r*s

This one features much less symbols compared to the previous example, although it still looks quite complicated. We will walk you through all the necessary theory to help you understand this.

Alphabets, Words, and $\mathcal{A}^*$

Definition

An **alphabet** is a finite non-empty set containing letters/symbols.

For example:

1. The English alphabet

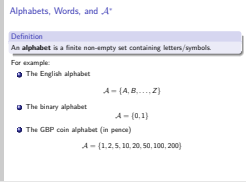
$$\mathcal{A} = \{A, B, \ldots, Z\}$$

2. The binary alphabet

$$\mathcal{A} = \{0, 1\}$$

3. The GBP coin alphabet (in pence)

$$\mathcal{A} = \{1, 2, 5, 10, 20, 50, 100, 200\}$$



(2min): Break down the given example into various sections, which are alphabets, words, and A* (explain language and how it is a language is a subset of A*). For each of these concept, show the formal definition on the slides, but spend most of the time doing a run through of an example with the audience. With each of these concept, remember to link back to the example above.

(style): Change pace depending on whether the audience looks confused or not. In general, go through this slowly while articulating the words clearly. After each concept, pause to let audience have time to think about it. Repeat/ give different example if audience really looks confused.

Alphabets, Words, and $\mathcal{A}^*$

Definition

- A **word** is a concatenations of letters.
- An n-letter word can be described by the Cartesian product $\mathcal{A}^n$.
- A special case is the empty word, ϵ.

Definition

The set of all words using an alphabet $\mathcal{A}$ is $\mathcal{A}^* := \bigcup_{n \geq 0} \mathcal{A}^n$.

For example, using the binary alphabet $\mathcal{A} = \{0, 1\}$

- Some possible words are: 001, 100110, 11000110
- $\mathcal{A}^*$ is the set of all binary strings

Alphabets, Words, and $\mathcal{A}^*$
Definition
A word is a concatenations of letters.
An n-letter word can be described by the Cartesian product $\mathcal{A}^n$.
A special case is the empty word, ϵ.
Definition
The set of all words using an alphabet $\mathcal{A}$ is $\mathcal{A}^* := \bigcup_{n \geq 0} \mathcal{A}^n$.
For example, using the binary alphabet $\mathcal{A} = \{0, 1\}$
Some possible words are: 001, 100110, 11000110
$\mathcal{A}^*$ is the set of all binary strings

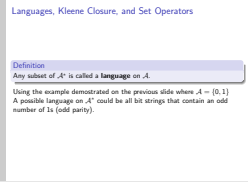
Notes from previous slide

Languages, Kleene Closure, and Set Operators

Definition

Any subset of $\mathcal{A}^*$ is called a **language** on $\mathcal{A}$.

Using the example demostrated on the previous slide where $\mathcal{A} = \{0, 1\}$
A possible language on $\mathcal{A}^*$ could be all bit strings that contain an odd number of 1s (odd parity).



(2.5min): Keep going to explain language set operators. For each, operator, show how it looks like, link back to formal definition above. With each of these concept, remember to link back to the example above. After laying out the groundwork of operators, show what the precedence of operators is through an example.
(style): Go through regex operators quickly since these concept are quite straight forward, read the equivalence of each operators in terms of words slowly since this is what we really care about and need.

Languages, Kleene Closure, and Set Operators

Definition

$\mathcal{L}^n$ is defined by $\mathcal{L}^n = \mathcal{L}\mathcal{L}^{n-1}$ $(n \geq 1)$ with $\mathcal{L}^0 = \{\varepsilon\}$

Definition

The **Kleene closure** of $\mathcal{L}$ is $\mathcal{L}^* = \cup_{n \geq 0} \mathcal{L}^n$.

Languages, Kleene Closure, and Set Operators
Definition
$\mathcal{L}^n$ is defined by $\mathcal{L}^n = \mathcal{L}\mathcal{L}^{n-1}$ $(n \geq 1)$ with $\mathcal{L}^0 = \{\varepsilon\}$
Definition
The **Kleene closure** of $\mathcal{L}$ is $\mathcal{L}^* = \cup_{n \geq 0} \mathcal{L}^n$.

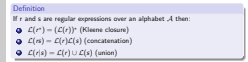
Notes from previous slide

Regex Definitions Using Set Operators

Definition

If r and s are regular expressions over an alphabet $\mathcal{A}$ then:

1. $\mathcal{L}(r^*) = (\mathcal{L}(r))^*$ (Kleene closure)
2. $\mathcal{L}(rs) = \mathcal{L}(r)\mathcal{L}(s)$ (concatenation)
3. $\mathcal{L}(r|s) = \mathcal{L}(r) \cup \mathcal{L}(s)$ (union)



(30s): Define a regex through the set operators shown.
(style): Pretending to be bored, read with a dead tone, try to make this funny and basically say that all of this is the culmination of the concept shown above. Allows to go through formal content quickly while keeping attention of the audience.

Operator Precedence and Un-parenthesising

Definition

$$\text{(i) } (r^*), \quad \text{(ii) } (rs), \quad \text{(iii) } (r|s),$$

Order of priority: (i) $\rightarrow$ (iii)

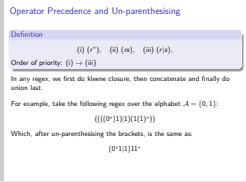
In any regex, we first do kleene closure, then concatenate and finally do union last.

For example, take the following regex over the alphabet $\mathcal{A} = \{0, 1\}$:

$$((((0^*)1)|1)(1(1)^*))$$

Which, after un-parenthesising the brackets, is the same as:

$$(0^*1|1)11^*$$



(1min): Show an example of a regex with more parenthesis as per usual, and go through the act of removing the brackets while explaining why.
(style): Try to be over the top with the precedence example (as there will be a LOT of brackets).

Regex and Specification Regular

Definition

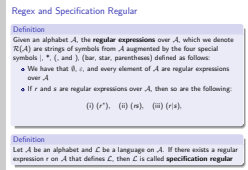
Given an alphabet $\mathcal{A}$, the **regular expressions** over $\mathcal{A}$, which we denote $\mathcal{R}(\mathcal{A})$ are strings of symbols from $\mathcal{A}$ augmented by the four special symbols $|$, *, (, and), (bar, star, parentheses) defined as follows:

- We have that $\emptyset$, ε, and every element of $\mathcal{A}$ are regular expressions over $\mathcal{A}$
- If r and s are regular expressions over $\mathcal{A}$, then so are the following:

$$\text{(i) } (r^*), \quad \text{(ii) } (rs), \quad \text{(iii) } (r|s),$$

Definition

Let $\mathcal{A}$ be an alphabet and $\mathcal{L}$ be a language on $\mathcal{A}$. If there exists a regular expression r on $\mathcal{A}$ that defines $\mathcal{L}$, then $\mathcal{L}$ is called **specification regular**



(1min): Explain specification regular. Say that subset are languages which are specification regular.

(style): Formal tone, read through the definitions properly.

Non-regular Languages: (Informal) Proof

Finite state machines are just that, **finite**.

Languages requiring growing data storage based on input size cannot be encoded. The "data storage" in a finite state machine is constant.

Non-regular Languages: (Informal) Proof

Finite state machines are just that, **finite**.

Languages requiring growing data storage based on input size cannot be encoded. The "data storage" in a finite state machine is constant.

(30s): Quick informal proof by saying that we cannot have a counting system that counts to infinity because a finite state machines can only store a finite amounts of states.

(style): Emphasis on the key point of the proof, which is the idea of a FSM only having finite states, while counting goes infinite.

Non-regular Languages: Example

English! You can't create a finite state automata that only accepts correctly written English sentences. We can hardly approximate that with AI and spellcheckers are still constantly wrong.

More simply, anything were you might need to "count" to arbitrarily large numbers. For example, only accepting an equation with matching parentheses. e.g. ((()(()))) but not ((()(())).

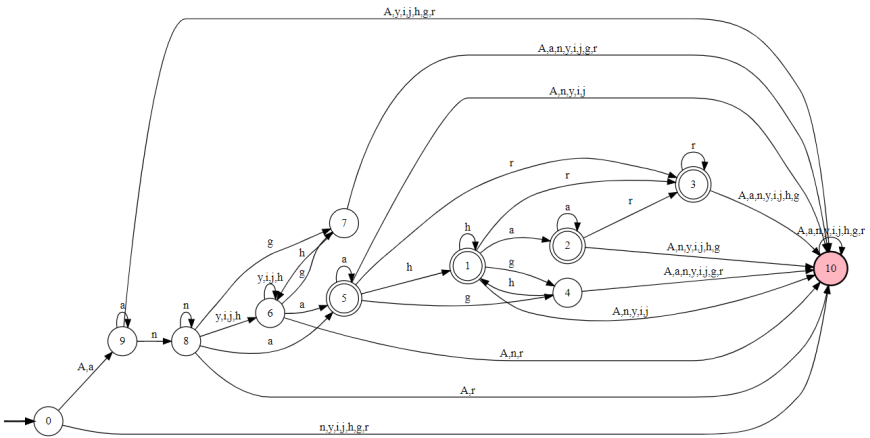
Non-regular Languages: Example

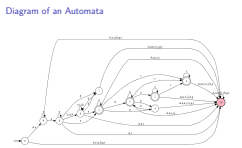
English! You can't create a finite state automata that only accepts correctly written English sentences. We can hardly approximate that with AI and spellcheckers are still constantly wrong.

More simply, anything were you might need to "count" to arbitrarily large numbers. For example, only accepting an equation with matching parentheses. e.g. ((()(())) but not ((()(())).

(30s): Show an example quickly and explain what we expect it to do and why it breaks.
(style): Read in a neutral tone, no gimmick here. Normal paced.

Diagram of an Automata



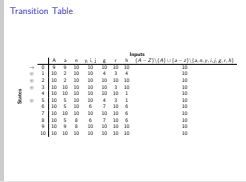


(3min): Show an image of a specific automata (this is the equivalent automata of the regex given as the example before). Go through the formal definition of a FSM by pointing out the respective aspect on the diagram (Input alphabet, states, accepting states, transition function). We wont need to go through every single aspect, just show a few and point to where they are on the image.

(style): Try to impress audience, "tada" and such. Tell them how beautiful it is.

Transition Table

States		Inputs							
		A	a	n	y, i, j	g	r	h	$\{A-Z\}\backslash\{A\} \cup \{a-z\}\backslash\{a,n,y,i,j,g,r,h\}$
→	0	9	9	10	10	10	10	10	10
◎	1	10	2	10	10	4	3	4	10
◎	2	10	2	10	10	10	10	10	10
◎	3	10	10	10	10	10	3	10	10
	4	10	10	10	10	10	10	1	10
◎	5	10	5	10	10	4	3	1	10
	6	10	5	10	6	7	10	6	10
	7	10	10	10	10	10	10	6	10
	8	10	5	8	6	7	10	6	10
	9	10	9	8	10	10	10	10	10
	10	10	10	10	10	10	10	10	10



(30s): Quickly show the transition table which represents the automata above and, again, show which aspect corresponds to the image.
(style): Boring, go through quickly, say that while this is more useful, the previous one is more beautiful and fun to see.

Automata Regular and Regularity

Definition

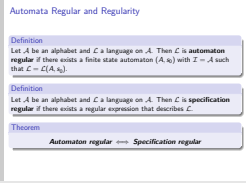
Let $\mathcal{A}$ be an alphabet and $\mathcal{L}$ a language on $\mathcal{A}$. Then $\mathcal{L}$ is **automaton regular** if there exists a finite state automaton (A, s_0) with $\mathcal{I} = \mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(A, s_0)$.

Definition

Let $\mathcal{A}$ be an alphabet and $\mathcal{L}$ a language on $\mathcal{A}$. Then $\mathcal{L}$ is **specification regular** if there exists a regular expression that describes $\mathcal{L}$.

Theorem

Automaton regular $\iff$ ***Specification regular***



(1min): Define automata regular, and then show that both specification regular and automata regular is equivalent.

(style): Read in a neutral tone, no gimmick here. Normal paced.

Equivalence to Regex

From what we have just told you above about Automaton regular and specification regular, can you guess what regex this automaton is equivalent to?

$$(A|a)a^*nn^*(y|i|j|h|gh)^*aa^*(h|gh)^*a^*r^*s$$

Believe it or not, the massive automaton shown above represents the example we gave you right at the start.

So what does this regex do you? This regex accepts any variation of the name Anna!

Equivalence to Regex

From what we have just told you above about Automaton regular and specification regular, can you guess what regex this automaton is equivalent to?

(A|a)a*nn*(y|i|j|h|gh)*aa*(h|gh)*a*r*s

Believe it or not, the massive automaton shown above represents the example we gave you right at the start.

So what does this regex do you? This regex accepts any variation of the name Anna!

(30s): Reveal that this is the equivalence of the initial regex!
(style): Lot of excitement!!!