

Regular expressions and Kleene's theorem

Informatics 2A: Lecture 5

Alex Simpson

School of Informatics
University of Edinburgh
als@inf.ed.ac.uk

25 September, 2014

- 1 More closure properties of regular languages
 - Operations on languages
 - ϵ -NFAs
 - Closure under concatenation and Kleene star
- 2 Regular expressions
 - Regular expressions
 - From regular expressions to regular languages
- 3 Kleene's theorem and Kleene algebra
 - Kleene's theorem
 - Kleene algebra
 - From DFAs to regular expressions

Recap of Lecture 4

- Regular languages are **closed under** union, intersection and complement.
- These closure properties are proved using explicit constructions on finite automata (sometimes using NFAs, sometimes DFAs).
- Every regular language has a unique **minimum** DFA that recognises it.
- An algorithm for **minimizing** a DFA.

Concatenation

We write $L_1.L_2$ for the **concatenation** of languages L_1 and L_2 , defined by:

$$L_1.L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

For example, if $L_1 = \{aaa\}$ and $L_2 = \{b, c\}$ then $L_1.L_2$ is the language $\{aaab, aaac\}$.

Later we will prove the following closure property.

If L_1 and L_2 are regular languages then so is $L_1.L_2$.

Kleene star

We write L^* for the **Kleene star** of the language L , defined by:

$$L^* = \{\epsilon\} \cup L \cup L.L \cup L.L.L \cup \dots$$

For example, if $L_3 = \{aaa, b\}$ then L_3^* contains strings like $aaaaaa$, $bbbbbb$, $baaaaaabbbaaa$, etc.

More precisely, L_3^* contains all strings over $\{a, b\}$ in which the letter a always appears in sequences of length some multiple of 3

Later we will prove the following closure property.

If L is a regular language then so is L^ .*

Self-assessment question

Consider the language over the alphabet $\{a, b, c\}$

$$L = \{x \mid x \text{ starts with } a \text{ and ends with } c\}$$

Which of the following strings are valid for the language $L.L$?

- ① *abcabc* Ans: yes
- ② *acacac* Ans: yes
- ③ *abcbcac* Ans: yes
- ④ *abcbacbc* Ans: no

Self-assessment question

Consider the (same) language over the alphabet $\{a, b, c\}$

$$L = \{x \mid x \text{ starts with } a \text{ and ends with } c\}$$

Which of the following strings are valid for the language L^* ?

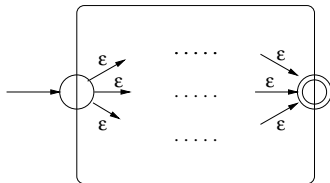
- ① ϵ Ans: yes
- ② $acaca$ Ans: no
- ③ $abc bc$ Ans: yes
- ④ $acacacacac$ Ans: yes

NFAs with ϵ -transitions

We can vary the definition of NFA by also allowing transitions labelled with the special symbol ϵ (*not* a symbol in Σ).

The automaton may (but doesn't have to) perform a spontaneous ϵ -transition at any time, without reading an input symbol.

This is quite convenient: for instance, we can turn any NFA into an ϵ -NFA with just **one start state** and **one accepting state**:



(Add ϵ -transitions from new start state to each state in S , and from each state in F to new accepting state.)

Equivalence to ordinary NFAs

Allowing ϵ -transitions is just a convenience: it doesn't fundamentally change the power of NFAs.

If $N = (Q, \Delta, S, F)$ is an ϵ -NFA, we can convert N to an ordinary NFA with the same associated language, by simply 'expanding' Δ and S to allow for silent ϵ -transitions.

To achieve this, perform the following steps on N .

- For every pair of transitions $q \xrightarrow{a} q'$ (where $a \in \Sigma$) and $q' \xrightarrow{\epsilon} q''$, add a new transition $q \xrightarrow{a} q''$.
- For every transition $q \xrightarrow{\epsilon} q'$, where q is a start state, make q' a start state too.

Repeat the two steps above until no further new transitions or new start states can be added.

Finally, remove all ϵ -transitions from the ϵ -NFA resulting from the above process. This produces the desired NFA.

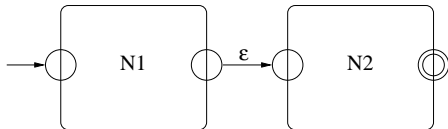
Closure under concatenation

We use ϵ -NFAs to show, as promised, that regular languages are closed under the **concatenation** operation:

$$L_1.L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

If L_1, L_2 are any regular languages, choose ϵ -NFAs N_1, N_2 that define them. As noted earlier, we can pick N_1 and N_2 to have just one start state and one accepting state.

Now hook up N_1 and N_2 like this:



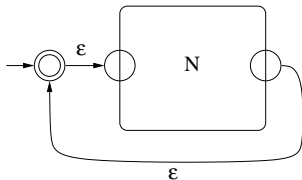
Clearly, this NFA corresponds to the language $L_1.L_2$.

Closure under Kleene star

Similarly, we can now show that regular languages are closed under the **Kleene star** operation:

$$L^* = \{\epsilon\} \cup L \cup L.L \cup L.L.L \cup \dots$$

For suppose L is represented by an ϵ -NFA N with one start state and one accepting state. Consider the following ϵ -NFA:



Clearly, this ϵ -NFA corresponds to the language L^* .

Regular expressions

We've been looking at ways of specifying regular languages via machines (often presented as **pictures**). But it's very useful for applications to have more **textual** ways of defining languages.

A **regular expression** is a written mathematical expression that defines a language over a given alphabet Σ .

- The **basic** regular expressions are

$$\emptyset \quad \epsilon \quad a \quad (\text{for } a \in \Sigma)$$

- From these, more complicated regular expressions can be built up by (repeatedly) applying the two binary operations $+$, \cdot and the unary operation $*$. Example: $(a \cdot b + \epsilon)^* + a$

We use brackets to indicate precedence. In the absence of brackets, $*$ binds more tightly than \cdot , which itself binds more tightly than $+$.

$$\text{So } a + b \cdot a^* \text{ means } a + (b \cdot (a^*))$$

Also the dot is often omitted: ab means $a \cdot b$

How do regular expressions define languages?

A regular expression is itself just a **written expression**. However, every regular expression α over Σ can be seen as **defining** an actual **language** $\mathcal{L}(\alpha) \subseteq \Sigma^*$ in the following way.

- $\mathcal{L}(\emptyset) = \emptyset, \quad \mathcal{L}(\epsilon) = \{\epsilon\}, \quad \mathcal{L}(a) = \{a\}.$
- $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$
- $\mathcal{L}(\alpha.\beta) = \mathcal{L}(\alpha) . \mathcal{L}(\beta)$
- $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$

Example: $a + ba^*$ defines the language $\{a, b, ba, baa, baaa, \dots\}$.

The languages defined by \emptyset, ϵ, a are obviously **regular**.

What's more, we've seen that regular languages are **closed under** union, concatenation and Kleene star.

This means **every regular expression defines a regular language**.
(Formal proof by induction on the size of the regular expression.)

Self-assessment question

Consider (again) the language

$$\{x \in \{0, 1\}^* \mid x \text{ contains an even number of 0's}\}$$

Which of the following regular expressions define the above language?

- ① $(1^*01^*01^*)^*$ **Ans:** no — 1 does not match expression
- ② $(1^*01^*0)^*1^*$ **Ans:** yes
- ③ $1^*(01^*0)^*1^*$ **Ans:** no — 00100 does not match expression
- ④ $(1 + 01^*0)^*$ **Ans:** yes

Kleene's theorem

We've seen that every regular expression defines a regular language.

The major goal of the lecture is to show the converse: **every regular language can be defined by a regular expression**. For this purpose, we introduce **Kleene algebra**: the algebra of regular expressions.

The equivalence between regular languages and expressions is:

Kleene's theorem

DFAs and regular expressions give rise to exactly the same class of languages (the regular languages).

As we've already seen, NFAs (with or without ϵ -transitions) also give rise to this class of languages.

So the evidence is mounting that the class of regular languages is mathematically a very 'natural' class to consider.

Kleene algebra

Regular expressions give a **textual** way of specifying regular languages. This is useful e.g. for communicating regular languages to a computer.

Another benefit: regular expressions can be manipulated using algebraic laws (**Kleene algebra**). For example:

$$\begin{array}{ll}
 \alpha + (\beta + \gamma) &= (\alpha + \beta) + \gamma & \alpha + \beta &= \beta + \alpha \\
 \alpha + \emptyset &= \alpha & \alpha + \alpha &= \alpha \\
 \alpha(\beta\gamma) &= (\alpha\beta)\gamma & \epsilon\alpha &= \alpha\epsilon = \alpha \\
 \alpha(\beta + \gamma) &= \alpha\beta + \alpha\gamma & (\alpha + \beta)\gamma &= \alpha\gamma + \beta\gamma \\
 \emptyset\alpha &= \alpha\emptyset = \emptyset & \epsilon + \alpha\alpha^* &= \epsilon + \alpha^*\alpha = \alpha^*
 \end{array}$$

Often these can be used to **simplify** regular expressions down to more pleasant ones.

Other reasoning principles

Let's write $\alpha \leq \beta$ to mean $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$ (or equivalently $\alpha + \beta = \beta$). Then

$$\alpha\gamma + \beta \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma$$

$$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma$$

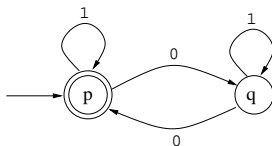
Arden's rule: Given an equation of the form $X = \alpha X + \beta$, its smallest solution is $X = \alpha^*\beta$.

What's more, if $\epsilon \notin \mathcal{L}(\alpha)$, this is the *only* solution.

Beautiful fact: The rules on this slide and the last form a **complete** set of reasoning principles, in the sense that if $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$, then ' $\alpha = \beta$ ' is provable using these rules. (Beyond scope of Inf2A.)

DFAs to regular expressions

We use an example to show how to convert a DFA to an equivalent regular expression.



For each state r , let the variable X_r stand for the set of strings that take us from r to an accepting state. Then we can write some simultaneous equations:

$$X_p = 1X_p + 0X_q + \epsilon$$

$$X_q = 1X_q + 0X_p$$

Where do the equations come from?

Consider:

$$X_p = 1X_p + 0X_q + \epsilon$$

This asserts the following.

Any string that takes us from p to an accepting state is:

- a 1 followed by a string that takes us from p to an accepting state; or
- a 0 followed by a string that takes us from q to an accepting state; or
- the empty string.

Note that the empty string is included because p is an accepting state.

Solving the equations

We solve the equations by eliminating one variable at a time:

$$X_q = 1^*0X_p \quad \text{by Arden's rule}$$

$$\begin{aligned} \text{So } X_p &= 1X_p + 01^*0X_p + \epsilon \\ &= (1 + 01^*0)X_p + \epsilon \end{aligned}$$

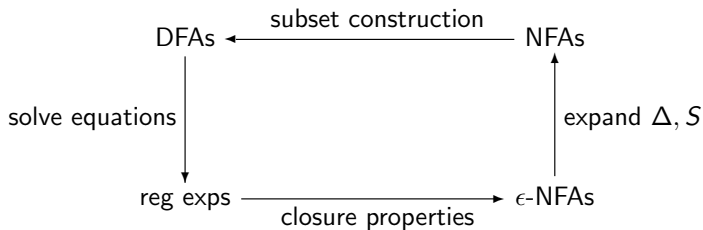
$$\text{So } X_p = (1 + 01^*0)^* \quad \text{by Arden's rule}$$

Since the start state is p , the resulting regular expression for X_p is the one we are seeking. Thus the language recognised by the automaton is:

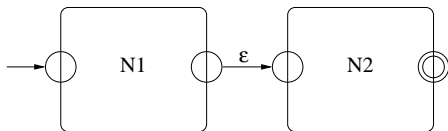
$$(1 + 01^*0)^*$$

The method we have illustrated here, in fact, works for arbitrary NFAs (without ϵ -transitions).

Theory of regular languages: overview



End-of-lecture question



Suppose the above ϵ -NFA defining concatenation is modified by **identifying** the final state of N_1 with the start state of N_2 (and removing the then-redundant ϵ -transition linking the two states).

- 1 Find a pair of ϵ -NFAs, N_1 and N_2 , each with a single start state and single accepting state, for which the modified construction **does not** recognise $\mathcal{L}(N_1).\mathcal{L}(N_2)$.
- 2 Show that if N_1 has no loops from the accepting state back to itself, then the modified ϵ -NFA does recognise $\mathcal{L}(N_1).\mathcal{L}(N_2)$.
- 3 Which construction of an ϵ -NFA in this lecture violates the assumption above about N_1 ?

Reading

Relevant reading:

- Regular expressions: Kozen chapters 7,8; J & M chapter 2.1. (Both texts actually discuss more general 'patterns' — see next lecture.)
- From regular expressions to NFAs: Kozen chapter 8; J & M chapter 2.3.
- Kleene algebra: Kozen chapter 9.
- From NFAs to regular expressions: Kozen chapter 9.

Next time: Some applications of all this theory.

- [Pattern matching](#)
- [Lexical analysis](#)

Appendix: (non-examinable) proof of Kleene's theorem

Given an NFA $N = (Q, \Delta, S, F)$ (without ϵ -transitions), we'll show how to define a regular expression defining the same language as N .

In fact, to build this up, we'll construct a **three-dimensional array** of regular expressions α_{uv}^X : one for every $u \in Q, v \in Q, X \subseteq Q$.

Informally, α_{uv}^X will define the set of *strings that get us from u to v allowing only intermediate states in X* .

We shall build suitable regular expressions $\alpha_{u,v}^X$ by working our way from smaller to larger sets X .

Eventually, the language defined by N will be given by the **sum** (+) of the languages α_{sf}^Q for all states $s \in S$ and $f \in F$.

Construction of α_{uv}^X

Here's how the regular expressions α_{uv}^X are built up.

- If $X = \emptyset$, let a_1, \dots, a_k be all the symbols a such that $(u, a, v) \in \Delta$. Two subcases:
 - If $u \neq v$, take $\alpha_{uv}^{\emptyset} = a_1 + \dots + a_k$
 - If $u = v$, take $\alpha_{uv}^{\emptyset} = (a_1 + \dots + a_k) + \epsilon$

Convention: if $k = 0$, take ' $a_1 + \dots + a_k$ ' to mean \emptyset .

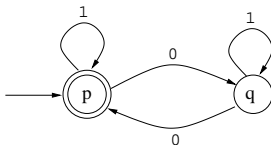
- If $X \neq \emptyset$, choose any $q \in X$, let $Y = X - \{q\}$, and define

$$\alpha_{uv}^X = \alpha_{uv}^Y + \alpha_{uq}^Y (\alpha_{qq}^Y)^* \alpha_{qv}^Y$$

Applying these rules repeatedly gives us $\alpha_{u,v}^X$ for every u, v, X .

NFAs to regular expressions: tiny example

Let's revisit our old friend:



Here p is the only start state and the only accepting state.
 By the rules on the previous slide:

$$\alpha_{p,p}^{\{p,q\}} = \alpha_{p,p}^{\{p\}} + \alpha_{p,q}^{\{p\}} (\alpha_{q,q}^{\{p\}})^* \alpha_{q,p}^{\{p\}}$$

Now by inspection (or by the rules again), we have

$$\begin{aligned} \alpha_{p,p}^{\{p\}} &= 1^* & \alpha_{p,q}^{\{p\}} &= 1^*0 \\ \alpha_{q,q}^{\{p\}} &= 1 + 01^*0 & \alpha_{q,p}^{\{p\}} &= 01^* \end{aligned}$$

So the required regular expression is

$$1^* + 1^*0(1 + 01^*0)^*01^* \quad (\text{A bit messy!})$$