

COMP3630 / COMP6363

week 1: **Regular Expressions and Languages**

This Lecture Covers Chapter 3 of HMU: Regular Expressions and Languages

slides created by: Dirk Pattinson, based on material by Peter Hoefner and Rob van Glabbeek; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Introduction to regular expressions and regular languages
- Equivalence of classes of regular languages and languages accepted
- Algebraic laws of (abstract) regular expressions

Additional Reading: Chapter 3 of HMU.

Regular Expressions: Overview

- › So far: DFAs, NFAs were given a machine-like description
- › Regular expressions are user-friendly and declarative formulation
- › Regular expressions find extensive use.
 - › Searching/finding strings/pattern matching or conformance in text-formatting systems (e.g., UNIX `grep`, `egrep`, `fgrep`)
 - › Lexical analyzers (in compilers) use regular expressions to identify tokens (e.g., `Lex`, `Flex`)
 - › In Web forms to (structurally) validate entries (passwords, dates, email IDs)
- › A regular expression over an alphabet Σ is a string consisting of:
 - › symbols from Σ
 - › constants: \emptyset, ϵ
 - › operators: $+, *$
 - › parentheses: $(,)$
- › Each regular expression r denotes a language $L(r) \subseteq \Sigma^*$

Regular Expressions: Definition

› Regular expressions are defined inductively as follows:

› Basis:

B1 \emptyset and ϵ are regular expressions, with $L(\emptyset) = \emptyset$ and $L(\epsilon) = \{\epsilon\}$.

B2 For each $a \in \Sigma$, a is a regular expression with $L(a) = \{a\}$.

› Induction: If r and s are regular expressions, then:

I1 so is r^* with $L(r^*) = (L(r))^*$

e.g., $L(a^*) = (L(a))^* = \{a\}^* = \{\epsilon, a, aa, \dots\}$

I2 so is $r + s$ with $L(r + s) = L(r) \cup L(s)$

I3 so is rs with $L(rs) = L(r) \cdot L(s)$

(cf. Def. from day 1!)

e.g., $L(a^*b) = L(a^*) \cdot L(b) = \{\epsilon, a, aa, \dots\} \cdot \{b\} = \{b, ab, aab, \dots\}$

I4 so is (r) with $L((r)) = L(r)$.

› Only those generated by the above induction are regular.

› **Remark:** Some authors/texts use $|$ instead of $+$. HMU uses $+$.

› Precedence Rules:

$$(\cdot) > * > \cdot > +$$

where $>$ is 'binds more strongly than', and both $+$ and \cdot associate to the left.

Regular Expressions: Examples

- > $r = 0 + 11^*10$ is a regular expression
 - > with brackets that indicate precedence: $r = 0 + (1(1^*)10)$
 - > with more brackets indicating associativity: $r = 0 + ((1(1^*))1)0$
- > Computing the language:

$$\begin{aligned}
 L(r) &= L(0) \cup L(11^*10) \\
 &= \{0\} \cup L(1) \cdot L(1^*) \cdot L(1) \cdot L(0) \\
 &= \{0\} \cup \{1\} \cdot \{1\}^* \cdot \{1\} \cdot \{0\} \\
 &= \{0\} \cup \{1\} \cdot \{1^n \mid n \geq 0\} \cdot \{1\} \cdot \{0\} \\
 &= \{1^i0 \mid i \neq 1\}
 \end{aligned}$$

- > Q: What's a regular expression that describes alternating sequences of 0s and 1s?

Regular Languages: Some Basic Properties

Theorem 3.2.1

Let $w \in \Sigma^*$. Then $\{w\}$ is regular.

Proof of Theorem 3.2.1

- > $\{w\}$ being regular means there is a regular expression r with $L(r) = \{w\}$.
 Proof by induction on the length of w . For $w = \epsilon$, $\{w\} = \{\epsilon\} = L(\epsilon)$. For w of the form $w's$, we have (by induction) r s.t. $\{w'\} = L(r)$ so that $\{w\} = \{w's\} = L(rs)$.

Theorem 3.2.2

Let L_1 and L_2 be regular languages. Then, L_1^* , $L_1 \cup L_2$ and L_1L_2 are also regular.

Proof of Theorem 3.2.2

By definition of $L(r^*)$, $L(r + s)$ and $L(rs)$.

- > **Corollary 1:** The class of regular languages is closed under finite union and concatenation, i.e., if L_1, \dots, L_k are regular languages for any $k \in \mathbb{N}$, then $L_1 \cup \dots \cup L_k$ and $L_1 \cdots L_k$ are also regular languages.
- > **Corollary 2:** Any finite language is regular.

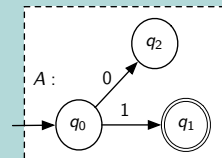
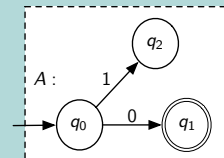
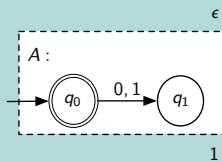
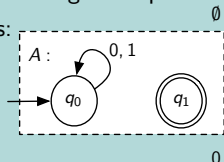
DFAs and Regular Languages

Theorem 3.2.3

For every regular language M , there exists a DFA A such that $M = L(A)$.

Proof of Theorem 3.2.3

- WLOG, let $\Sigma = \{0, 1\}$. Let M be a regular language. Then, $M = L(E)$ for some regular expression E .
- For each regular expression, we will devise an ϵ -NFA.
- Basis:



Note that these automata could be made smaller:

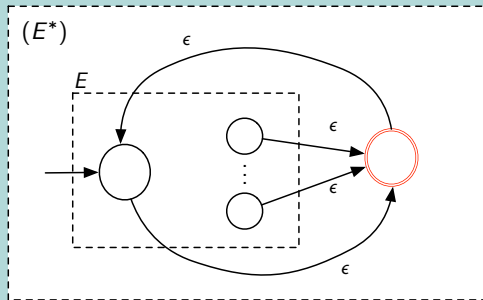
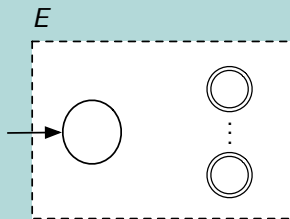
\emptyset/ϵ only keep initial state and no transitions since runs with non-existent transitions fail.

$0/1$ q_2 can be removed since runs with non-existent transitions fail.

DFAs and Regular Languages

Proof of Theorem 3.2.3 (Cont'd)

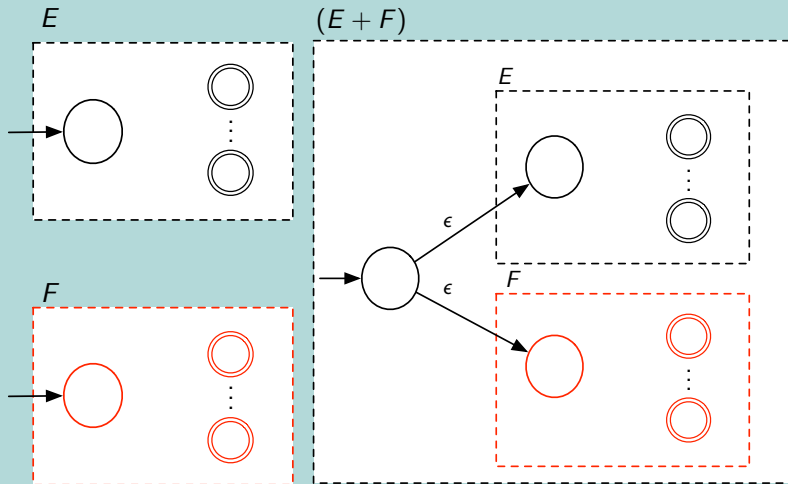
> Induction E^* :



DFA's and Regular Languages

Proof of Theorem 3.2.3 (Cont'd)

> Induction $E + F$:

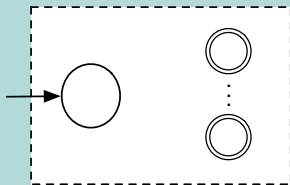


DFA's and Regular Languages

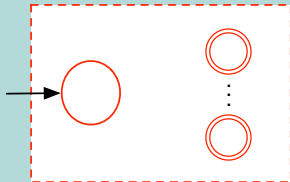
Proof of Theorem 3.2.1 (Cont'd)

> Induction I3':

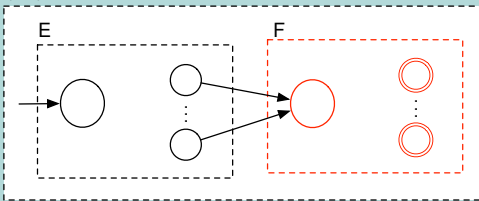
E



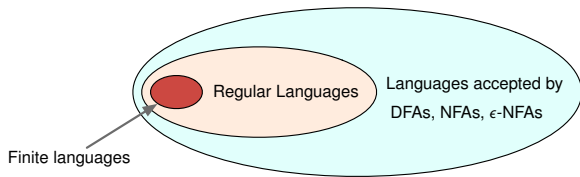
F



(EF)



So Far...



- › Is the inclusion strict?
- › Are there languages accepted by DFAs that are not regular?

DFA's and Regular Languages

Theorem 3.2.4

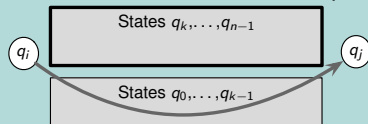
For every DFA A , there is a regular expression E such that $L(A) = L(E)$.

Proof of Theorem 3.2.4

- > Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ be given.
- > Let us rename the states so that $Q = \{q_0, q_1, q_2, \dots, q_{n-1}\}$.
- > For any string $s_1 \dots s_k \in L(A)$, there is a path

$$q_0 \xrightarrow{s_1} q_{i_1} \xrightarrow{s_2} q_{i_2} \cdots \xrightarrow{s_k} q_{i_k} \in F$$

- > **Define:** $R(i, j, k)$ be the set of all input strings that move the internal state of A from q_i to q_j using paths whose intermediate nodes comprise only of q_ℓ , $\ell < k$.



- > Idea: prove that (a) each $R(i, j, k)$ is regular, and (b) $L(A)$ is a union of $R(i, j, k)$'s.

DFAs and Regular Languages

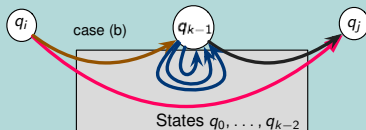
Proof of Theorem 3.2.4 (Cont'd)

- > Note that $L(A) = \bigcup_{j:q_j \in F} R(0, j, n)$. (i.e., paths that start in q_0 and end in an accepting state with intermediate nodes q_0, q_1, \dots, q_{n-1} (all nodes))
- > $L(A)$ will be regular if each $R(i, j, k)$ to be regular. We now proceed by induction to show that each $R(i, j, k)$ is regular.
- > **Basis:** Consider $R(i, j, 0)$ for $i, j \in \{0, 1, \dots, n-1\}$.
 - > $R(i, j, 0)$ consists of strings whose corresponding paths start in q_i and end in q_j with intermediate nodes $q_\ell, \ell < 0$.
 - \Rightarrow No intermediate nodes
 - $\Rightarrow R(i, j, 0)$ contains strings that change state q_i to q_j directly
 - $\Rightarrow R(i, j, 0) \subseteq \{\epsilon\} \cup \Sigma$
 - $\Rightarrow R(i, j, 0)$ is a regular language [Corollary 2]
- > **Induction:** Let $R(i, j, \ell)$ be regular for $i, j \in \{0, \dots, n-1\}$ and $0 \leq \ell < k$. Consider $R(i, j, k)$ for $i, j \in \{0, \dots, n-1\}$.

DFAs and Regular Languages

Proof of Theorem 3.2.4 (Cont'd)

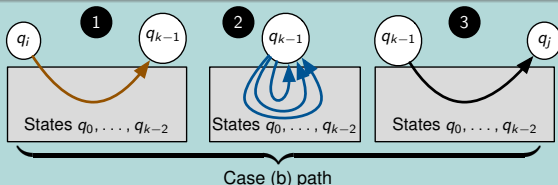
- > The strings in $R(i, j, k)$ correspond to paths whose intermediate nodes belong to $\{q_0, \dots, q_{k-1}\}$.
- > Partition $R(i, j, k)$ as follows:
 - Case (a): Strings whose paths **do not have** q_{k-1} as an intermediate node.
 - Case (b): Strings whose paths **do pass through** q_{k-1} as an intermediate node.



- > $R(i, j, k) = \{\text{Case (a) strings}\} \cup \{\text{Case (b) strings}\}$.
- > Case (a) Strings are exactly those in $R(i, j, k - 1)$
- > Hence, $R(i, j, k) = R(i, j, k - 1) \cup \{\text{Case (b) strings}\}$.

DFAs and Regular Languages

Proof of Theorem 3.2.4 (Cont'd)



› Each case (b) string is the concatenation of 3 strings:

1. A string that changes the state from q_i to q_{k-1} through a path whose intermediate nodes are q_0, \dots, q_{k-2} , i.e., $R(i, k-1, k-1)$
2. A finite concatenation of strings, each of which take q_{k-1} back to q_{k-1} via paths that use only q_0, \dots, q_{k-2} as intermediate nodes. i.e., i.e., $R(k-1, k-1, k-1)^*$
3. A string that takes q_{k-1} back to q_j via a path that uses only q_0, \dots, q_{k-2} as intermediate nodes, i.e., i.e., $R(k-1, j, k-1)$

Thus,

$$R(i, j, k) = R(i, j, k-1) \cup [R(i, k-1, k-1)R(k-1, k-1, k-1)^*R(k-1, j, k-1)]$$

› From Thm 3.2.2, it follows that $R(i, j, k)$ is regular for any i, j, k . Thus, $L(A)$ is regular.

Equivalence of Languages

- › The following are indeed equivalent:
 - › The class of regular languages
 - › The class of languages accepted by DFAs
 - › The class of languages accepted by NFAs
 - › The class of languages accepted by ϵ -NFAs

Properties of Regular Languages

- › Regular languages are closed under finite union, concatenation, and Kleene-* operation. (Theorem 3.2.2)
- › They are *also* closed under:
 - › *Complementation*:
Given DFA $A = (Q, \Sigma, \delta, q_0, F)$, DFA $A' = (Q, \Sigma, \delta, q_0, F^c)$ accepts $L(A)^c$.
 - › *Intersection*:
De Morgan's Law: $R_1 \cap R_2 = (R_1^c \cup R_2^c)^c$

(Where $F^c = Q \setminus F$ and L_Σ^c (for some language L over Σ) is $\Sigma^* \setminus L_\Sigma$)

Abstract Regular Expressions

- › We can also define **abstract** regular expressions over languages over Σ .
- › Let \mathcal{V} be a set of **variables** (which will be interpreted as languages)
- › Use the induction definition for regular languages replacing B2 alone by:
 - B2 M is an (abstract) regular expression for every $M \in \mathcal{V}$
- › **Remark:** Even though \mathcal{V} could be infinite, every regular expression consists only of finitely many variables.
- › Unlike **concrete** regular expressions (such as 1^* , $0 + 1$), **abstract** regular expressions (such as M^* , $M + N$) don't stand for a **unique** language.
- › However, we can **evaluate** abstract regular expressions by **assigning** any languages to variables, and inductively interpreting:
 - › Variable* \rightarrow Kleene-* closure of its language
 - › Sum of variables \rightarrow union of the languages assigned to them
 - › Concatenation of variables \rightarrow concatenation of their the languages
- › We can introduce a notion of equality of (abstract) regular expression:

Abstract regular expressions $E_1 = E_2 \Leftrightarrow$ For any assignment of languages to the variables contained in E_1, E_2 , their evaluations equal (i.e., $L(E_1) = L(E_2)$)

Algebraic Laws of Abstract Regular Expressions

- > Commutativity: $L + M = M + L$ (Union is commutative)
 $LM \neq ML$ (Concatenation is not commutative)
- > Associativity: $(L + M) + N = L + (M + N)$ (Union is associative)
 $(LM)N = L(MN)$ (Concatenation is associative)
- > Identity: $\emptyset + L = L + \emptyset = L$ (\emptyset is the identity element for $+$)
 $\epsilon L = L\epsilon = L$ (ϵ is the identity element for concatenation)
- > Annihilator: $\emptyset L = L\emptyset = \emptyset$
- > Idempotent: $L + L = L$
- > Distributive: $L(M + N) = LM + LN$
 $(M + N)L = ML + NL$
- > Kleene *: $(L^*)^* = L^*$; $\emptyset^* = \epsilon$; $\epsilon^* = \epsilon$.