Theory of Computation Notes: Regular Expressions

Arjun Chandrasekhar

1 Regular expressions

Definition 1.1. A **regular expression** is an an expression describing a language. The regular expression describes what strings are part of a language by describing rules for sequentially constructing strings in the language.

Formally, let Σ be an alphabet. We say R is a regular expression if R is

1. a, for some $a \in \Sigma$
2. ϵ
3. $\emptyset$
4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*) where R_1 is a regular exrpression

In items 1 and 2 the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively. In item 3, thre regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the Kleene star of the language R_1, respectively.

Some remarks:

- Do not confuse ϵ and $\emptyset$. ϵ represents a language with the single string ϵ. $\emptyset$ represents a language with no strings.
- This definition may seem circular, because we define one regular expression in terms of other ones. However, we are defining a regular expression in terms of two *smaller* ones. This is a valid **inductive definition**.
- Parentheses are optional. If they are omitted, the order of operations (in descending order of precedence): star, concatenation, then union
- Concatenation is often omitted. For example, $1 \circ 1 \circ 1$ will often be written as 111
- For convenience, we let $R^+ = RR*$. Note that $R^* = R^+ \cup \epsilon$
- We let R^k be shorthand for the concatenation of k copies of R with each other, i.e. $R^4 = RRRR$
- We have already seen one regular expression: Σ^*, i.e. the language containing all strings that can be formed with the alphabet Σ
- We write $L(R)$ to denote the *language of* R, or the set of strings generated by R

Example 1.1. In each of the following instances $\Sigma = \{0, 1\}$

1. $0^*10^* = \{\text{w} \mid \text{w contains a single 1}\}$
2. $(\Sigma\Sigma)^* = \{\text{w} \mid \text{w is a string of even length}\}$
3. $(0 \cup \epsilon)(1 \cup \epsilon) = \{\epsilon, 0, 1, 01\}$

Example 1.2. Some regular expression identities:

1. $R \cup \emptyset = R$ - adding the empty language to R will not change it
2. $R \circ \epsilon = R$ - concatenating the empty string will not change R
3. $R \circ \emptyset = \emptyset$ - to concatenate two regular expressions, we need a string from the first regex and a string from the second one. In this case, it is impossible to produce a string from $\emptyset$, so it is impossible to concatenate R with $\emptyset$. Thus, the regular expression generates no strings - in other words, $\emptyset$

2 Equivalence of regular expressions and finite automata

Theorem 2.1 (Kleene's Theorem)**.** A language is regular if and only if some regular expression describes it.

Lemma 2.1. If a language is described by a regular expression, then it is regular.

Proof. Given any regular expression R, we show how to convert it into an NFA N. We consider the six cases in the formal definition of regular expressions. We will prove it by induction.

Base Case

1. Suppose $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$ and the following NFA reognizes $L(R)$

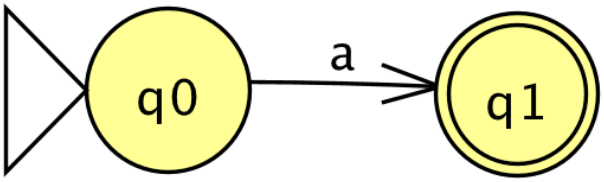

Figure 1: **An NFA for** $a \in \Sigma$. If the machine tries to read any string other than 'a ', the computation will either finish in q_0 or die.

2. Suppose $R = \{\epsilon\}$ then $L(R) = \{\epsilon\}$, and the following NFA recognizes $L(R)$

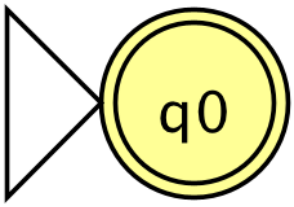

Figure 2: **An NFA for** ϵ. The NFA will halt and accept on the empty string by the computation will die if it tries to read any characters.

3. Suppose $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$

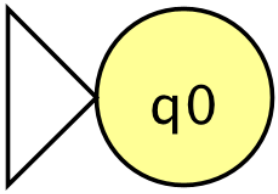

Figure 3: **An NFA for** $\emptyset$. The machine will reject the empty string, and the computation will die if it tries to read any characters.

Inductive case: Assume that every regular expression of length $\leq n$ can be converted to an NFA. Let R be a regular expression with length $n + 1$

1. Suppose $R = R_1 \cup R_2$. Then R_1 and R_2 must have lengths $\leq n$, so by our inductive hypothesis there exist NFAs N_1 and N_2 which recognize R_1 and R_2, respectively. We have previously shown that it is possible to construct an NFA N such that $L(N) = L(N_1) \cup L(N_2) = L(R_1) \cup L(R_2) = R$.

2. Suppose $R = R_1 \circ R_2$. Then R_1 and R_2 must have lengths $\leq n$, so by our inductive hypothesis there exist NFAs N_1 and N_2 which recognize R_1 and R_2, respectively. We have previously shown that it is possible to construct an NFA N such that $L(N) = L(N_1) \circ L(N_2) = L(R_1) \circ L(R_2) = R$.

3. Suppose $R = R_1^*$. Then R_1 must have length $\leq n$, so by our inductive hypothesis there exists an NFA N_1 which recognizes R_1. We have previously shown that it is possible to construct an NFA N such that $L(N) = L(N_1)^* = L(R_1)^* = R$.

□

Example 2.1. Figure 4 shows how ton create an NFA for the regex $(ab \cup b)^*$. Note that ab has the implied concatenation $a \circ b$.

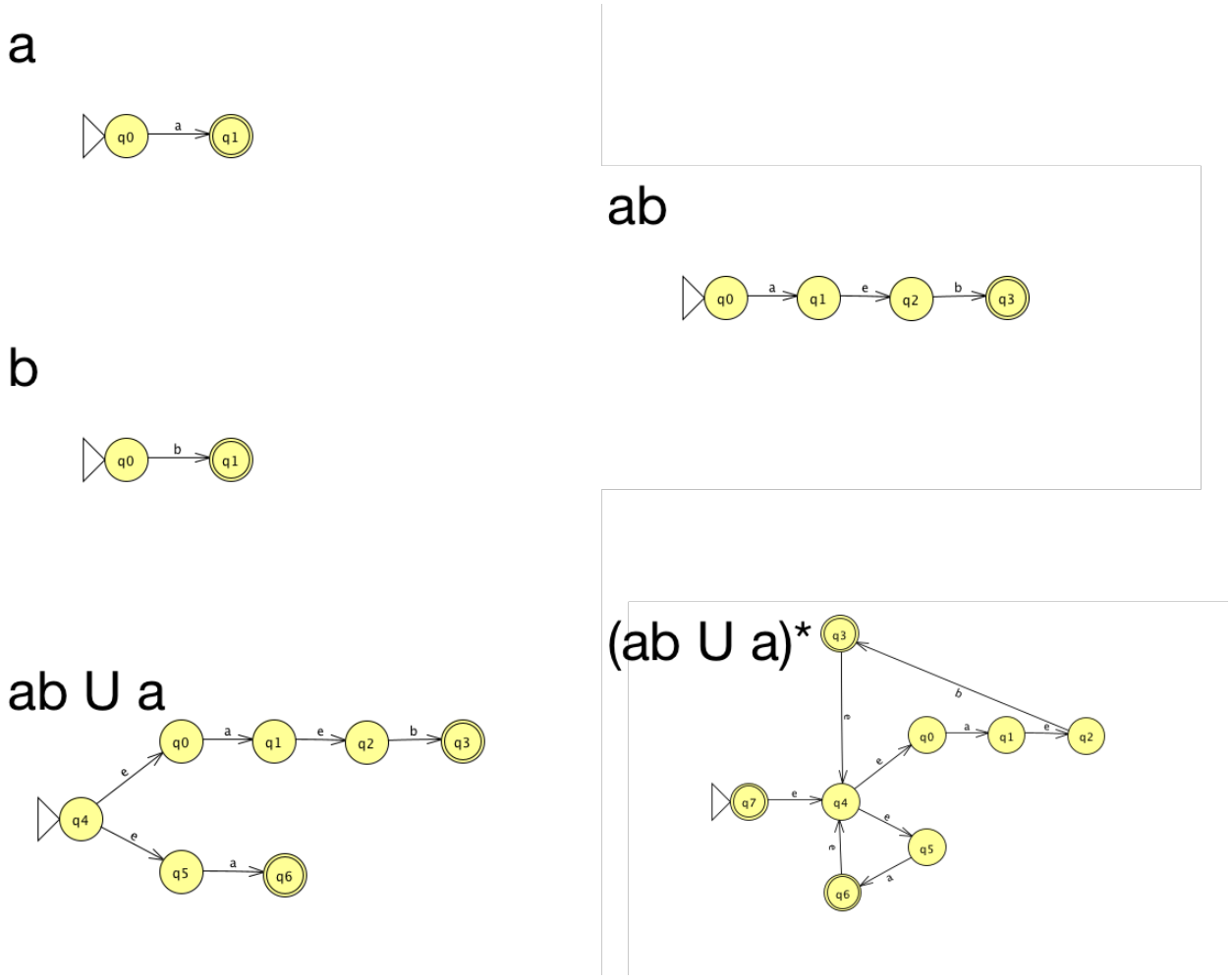

Figure 4: **Converting a regex to an NFA**. First we construct NFAs for the languages $\{a\}$ and $\{b\}$, and then combine them via concatenation, union, and Kleene star.

Lemma 2.2. If a language is regular, then it is described by a regular expression.

Proof idea: We need to show that if a language A is regular, a regular expression describes it. Because A is regular, it is accepted by a DFA D. We describe a procedure for converting D into an equivalent regular expression.

We break this procedure into two parts, using a new type of finite automaton called a **generalized nondeterministic finite automaton** (GNFA). First we show how to convert a DFA into a GNFA, and then a GNFA into a regex.

GNFAs are simply NFAs, except the transitions can contain a full regular expression (rather than a single character). Rather than reading a single character at a time, a GNFA can read an entire chunk of input. I may take a transition if it can read a chunk of the input that is in the language of the regex for that transition.

For convenience, we require that GNFAs always be in a special form that meet the following requirements:

- The start state q_s has transition arrows going to every other state, but no arrows coming in from any other state.
- There is only a single accept state, q_F, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state, and also from each state to itself.

Figure 5 shows an example of a GNFA.

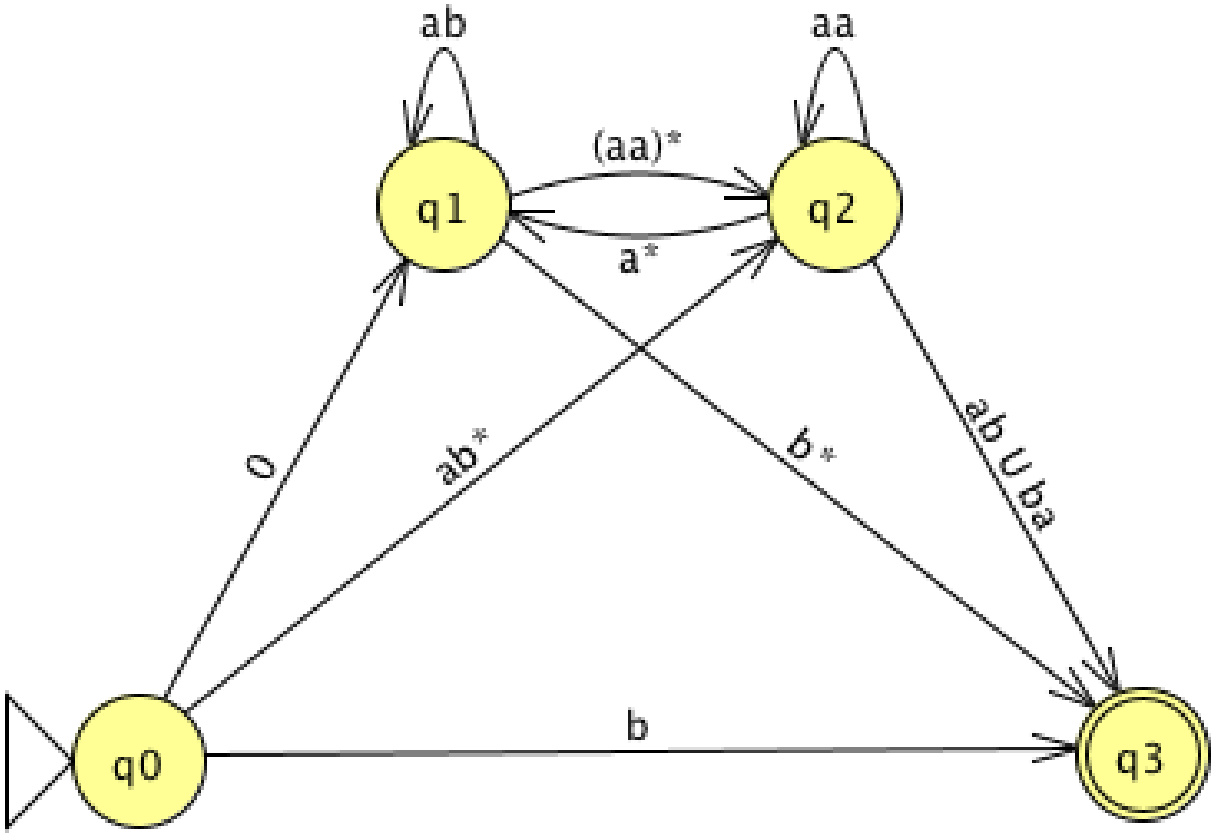

Figure 5: **Example GNFA**.

We can easily convert at DFA into the GNFA. We add a new start state with an ϵ arrow to the old start state, and a new accept state with ϵ arrows from the old accept states. If there are multiple arrows going between the same two states in the same direction, we replace them with one arrow whose label is the union of all the the previous labels. We add an arrow labelled $\emptyset$ between states with no arrows.

Proof. Now we show how to convert a DFA to a regular expression. First, we convert the DFA to a GNFA. We then proceed to iteratively 'rip 'away one state at a time, until we have only two states left, at which point it is trivial to conver the GNFA to a regular expression. We will formalize this using induction. Let G be a GNFA. Because G is in the special form, it must have at least 2 states, q_s and q_F

Base case: Suppose G has two states. There is a single arrow going from q_s to q_F. The label of this arrow, R, is the equivalent regular expression.

Inductive case: Assume every GNFA with $\leq k$ states can be converted to a regular expression. We show how to convert a GNFA G with $k+1$ states to a regular expression. We will select a state q_{rip}, 'rip 'it out of the machine, and repair the machine so that it still recognizes the same language. Any state other than the start or accept state will work; because $k > 2$ such a state must exist.

After removing q_{rip} we repair the machine by altering the regular expressions that label each of the remaining arrows. The new labels compensate for the absence of q_{rip} by adding back the lost computations. The new label going from state q_i to q_j is a regular expression that describes all strings that would have taken the machine from q_i to q_j, either directly, or via q_{rip}.

In the old machine, suppose

- $q_i \to q_{\text{rip}}$ has label R_1
- $q_{\text{rip}} \to q_{\text{rip}}$ has label R_2
- $q_{\text{rip}} \to q_j$ has label R_3
- $q_i \to q_j$ has label R_4

There are two ways to get from q_i to q_j:

- Go from $q_i \to q_{\text{rip}}$ using R_1; loop from q_{rip} back to itself any number of times R_2; and go from $q_{\text{rip}} \to q_j$ using R_3. The regular expression for this is $R_1 \circ R_2^* \circ R_3$
- Go directly from $q_i \to q_j$ using R_4

To reflect this, after ripping away q_{rip}, we update the transition $q_i \to q_j$ to have the label $(R_1 \circ R_2^* \circ R_3) \cup R_4$. Figure 6 illustrates the process of ripping away a state.

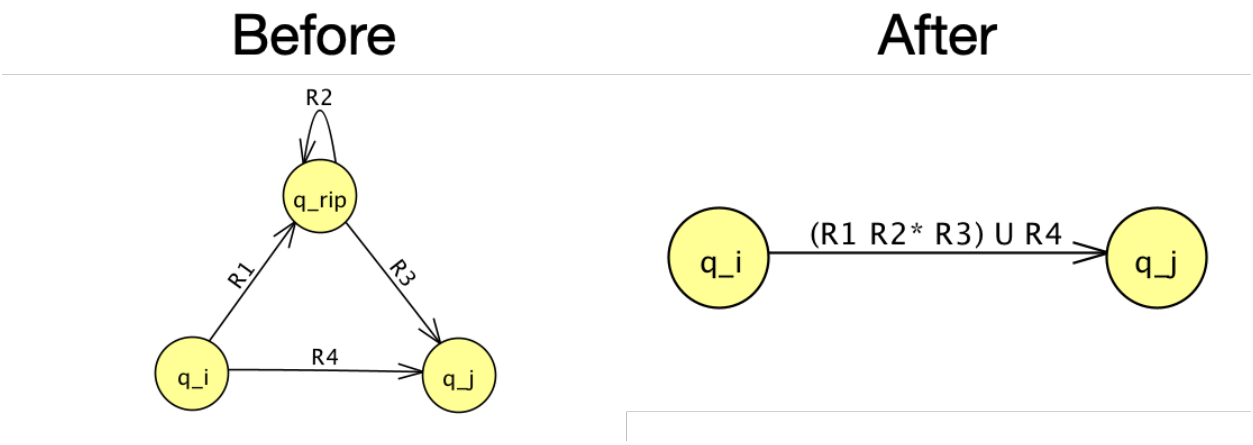

Figure 6: **Ripping away a state**.

Upon ripping away q_{rip}, we are left with a GNFA G' that accepts the same strings as G. Moreover, G' has k states. By our inductive hypothesis, G' can be converted to a regex R. Because R is a regex that is equivalent to G', it must be equivalent to G.

For an even more formal proof that utilizes the formal definition of a GNFA, see lemma 1.60 in Sipser. □

Example 2.2. Figure 7 shows an example of converting a DFA to a regular expression. To avoid cluttering the figure, all $\emptyset$ arrows are omitted.

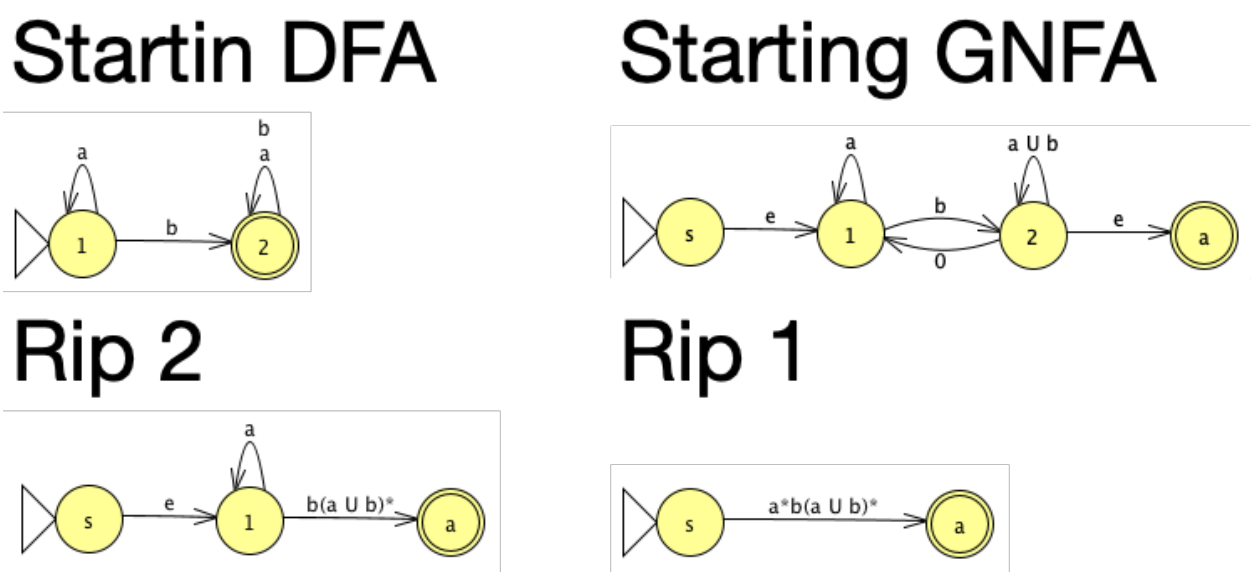

Figure 7: **Converting a DFA to a regular expression.** We start with the DFA, and convert it to a GNFA in the special form. We iteratively rip away states until we only have two states left; the remaining regex is the regex corresponding to the original DFA. In the starting GNFA, the "0" transition from 2 to 1 is an $\emptyset$ transition