1 Regular Expressions (1.3)

We can use **regular expressions** (RegExp) to describe a language. An example of a regular expression is:

$$(\mathtt{0} \cup \mathtt{1})\mathtt{0}^*$$

To give a comparison, consider the arithmetic expression:

$$(5 + 4) \times 2$$

In an arithmetic expression, the value is a number; in our case above, we would get 18. In a regular expression, the value is a **language**; in our case above, we can break the expression into multiple parts:

- $\mathtt{0} \cup \mathtt{1}$: This is the same thing as saying $\{\mathtt{0}\} \cup \{\mathtt{1}\}$, so this segment is saying that its language is $\{\mathtt{0}, \mathtt{1}\}$.
- $\mathtt{0}^*$: This is the same thing as saying $\{\mathtt{0}\}^*$, so its value is the language consisting of all strings containing any numbers of $\mathtt{0}$s.

Putting it together, this regular expression recognizes any string which starts with $\mathtt{0}$ or $\mathtt{1}$ and ends with some number of $\mathtt{0}$s. Just like how the multiplication sign $\times$ is often implicitly written (that is, we can write $2(5 + 4)$ instead of $2 \times (5 + 4)$), the concatenation sign $\circ$ is also implicitly written. That is, $(\mathtt{0} \cup \mathtt{1})\mathtt{0}^*$ is the shorthand for $(\mathtt{0} \cup \mathtt{1}) \circ \mathtt{0}^*$.

1.1 Formal Definition of a Regular Expression

Definition 1.1: Regular Expression

We say that R is a **regular expression** if R is:

1. a for some a in the alphabet Σ,
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,
6. (R_1^*), where R_1 is a regular expressions,

In items 1 and 2, the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively. In item 3, the 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 star of the language R_1, respectively.

Remarks:

- Remember, ϵ and $\emptyset$ are not the same. ϵ is the same thing as $\{\epsilon\}$, i.e. the language containing only the empty string; however, $\emptyset$ represents the language that doesn't contain anything.
- In regular expressions, there is the notion of operator precedence. In our case, the star operation is done first, followed by concatenation, and finally union *unless* parentheses change the usual order.
- We may omit the $\circ$ notation for concatenation. For example, R_1R_2 is the same thing as $R_1 \circ R_2$.

Additionally, we define some more notation.

- Let R^+ be shorthand for RR^*. In other words, while R^* has all strings that are 0 or more concatenations of strings from R, the language R^+ has all strings that are **1** or more concatenations of strings from R. So, really, $R^+ \cup \epsilon = R^*$.

- We let R^k be shorthand for the concatenation of k R's with each other.

Finally, when we want to distinguish between a regular language R and the language it described, we write $L(R)$ to be the language of R.

1.1.1 Example: Regular Languages

Suppose $\epsilon = \{\texttt{0}, \texttt{1}\}$. Then, some examples of regular expressions are:

RegExp	Examples	Formal Description
$\texttt{0}^*\texttt{10}^*$	1, 01, 0100	$\{w \mid w$ contains a single $\texttt{1}\}$
$\Sigma^*\texttt{1}\Sigma^*$	1, 00101101	$\{w \mid w$ has at least one $\texttt{1}\}$
$\Sigma^*\texttt{001*}$	001, 0100101	$\{w \mid w$ contains the string 001 as a substring$\}$
$\texttt{1}^*(\texttt{01}^+)^*$	1010110111, 1110101	$\{w \mid$ every 0 in w is followed by at least one $\texttt{1}\}$
$\underbrace{(\Sigma\Sigma\ldots\Sigma\Sigma)^*}_{n \text{ times}}$		$\{w \mid$ the length of w is a multiple of $n\}$
$\texttt{01} \cup \texttt{10}$	10, 01	$\{\texttt{01}, \texttt{10}\}$
$\texttt{0}\Sigma^*\texttt{0} \cup \texttt{1}\Sigma^*\texttt{1} \cup \texttt{0} \cup \texttt{1}$	00, 11, 10101, 0, 1	$\{w \mid w$ starts and ends with the same symbol$\}$
$(\texttt{0} \cup \epsilon)\texttt{1}^*$	11111, 01, 0111	$\texttt{01}^* \cup \texttt{1}^*$
$(\texttt{0} \cup \epsilon)(\texttt{1} \cup \epsilon)$	01, 1, 0, ϵ	$\{\epsilon, \texttt{0}, \texttt{1}, \texttt{01}\}$
$\texttt{1}^*\emptyset$		$\emptyset$
$\emptyset^*$	ϵ	$\{\epsilon\}$

Remarks:

- Concatenating the empty set to any set yields the empty set.
- The star operation on the empty set produces the set containing only the empty string.

1.2 Identities

Let R be any regular expression. The following identities hold:

1. $R \cup \emptyset = R$. Adding the empty language to any other language will not change it.
2. $R \circ \epsilon = R$. Joining the empty string to any string will not change it.

As a warning, the following do not necessarily hold:

1. $R \cup \epsilon = R$. If $R = \texttt{0}$, then $L(R) = \{\texttt{0}\}$ but $L(R \cup \epsilon) = \{\texttt{0}, \epsilon\}$
2. $R \circ \emptyset = R$. If $R = \texttt{0}$, then $L(R) = \{\texttt{0}\}$ but $L(R \circ \emptyset) = \emptyset$.

1.3 Practical Applications of RegExp

Regular expressions have practical applications. One example is in the world of compilers for programming languages. In particular, elemental objects in a programming language, called **tokens**, such as variable names and constants, can be described with regular expression. Consider the following regular expression:

$$(\texttt{+} \cup \texttt{-} \cup \epsilon)(D^+ \cup D^+ . D^* \cup D^* . D^+)$$

Where $D = \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{8}, \texttt{9}\}$. This regular expression describes a numerical constant which may include a fractional part and/or a sign. For example, the following strings are valid:

- `3.1415926`
- `+2.`
- `-.15`

After we can describe the syntax of a programming language with a regular expression in terms of its tokens, we can generate a **lexical analyzer** which processes it.

1.4 Generalized Nondeterministic Finite Automaton

We now introduce a new type of finite automaton called a **generalized nondeterministic finite automaton**, also known as a GNFA. First, we briefly introduce what a GNFA is:

- GNFAs are simply nondeterministic finite automata wherein the transition arrows may have any *regular expressions* as labels, instead of only members of the alphabet or ϵ.
- The GNFA reads *blocks of symbols* form the input, not necessarily just one symbol at a time.
- The GNFA moves along a transition arrow connecting two states by reading a block of symbols from the input, which themselves constittue a string described by the regular expression on that arrow.
- GNFAs are nondeterministic, so there may be several different ways to process the same input string.

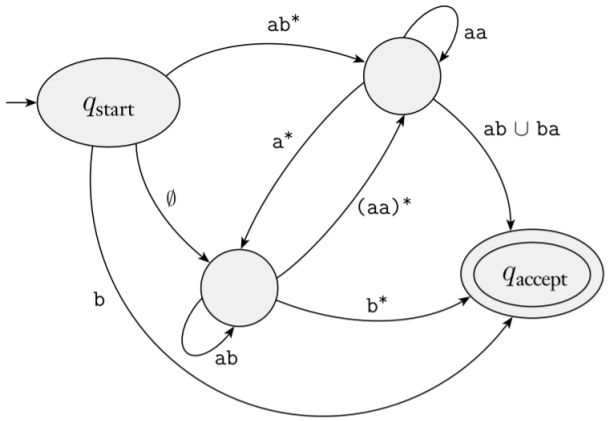


Figure: A generalized nondeterministic finite automaton.

We always require GNFAs to have a special form that meets the following conditions:

1. The start state has transition arrows going to every other state but no arrows coming in from any other state.
2. There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Additionally, the start state cannot be the accept state.
3. For all other states except the start/accept states, one arrow goes from every state to every other state and also from each state to itself.

1.4.1 DFA to GNFA

To convert a DFA to a GNFA, we do the following:

- We can add a new start state with a ϵ arrow to the old start state and a new accept state with ϵ from the old accept states.
- If any arrows have multiple labels, or if there are multiple arrows going between the same two states in the same direction, replace each with a single arrow whose label is the union of the previous labels.
- Finally, add arrows labeled $\emptyset$ between states that have no arrows.

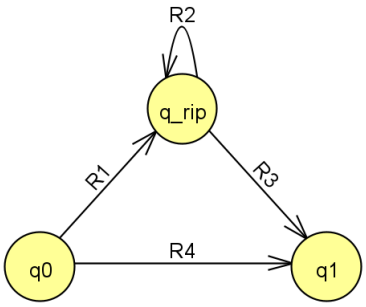
1.4.2 GNFA to Regular Expression

We now need to convert a GNFA to a regular expression. Say that a GNFA has k states. Then, because a GNFA must have a start and an accept state and they must be different from each other, we know that $k \geq 2$. If $k > 2$, we construct an equivalent GNFA with $k-1$ states. We continue to do this until the GNFA is reduced to two states. If $k = 2$, then the GNFA has a single arrow that goes from the start state to the accept state. The label of this arrow would then be the *equivalent regular expression*.

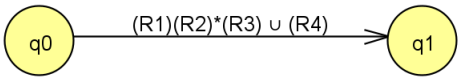
The most important step in this process is constructing an equivalent GNFA with one fewer state when $k > 2$. How can we do this? Well:

- Select a state that isn't the start or accept state, rip that state out of the machine, and then repairing what is left of the machine so the same language is still recognized. Call this state q_{rip}.
- After removing q_{rip}, we need to repair the machine by altering the regular expressions that label each of the remaining arrows. We use these new labels because they add back the lost computations (from ripping q_{rip}).

Consider the following GNFA:



If we remove q_{rip}, we get the following GNFA:



Essentially, in the old machine, if:

1. q_0 goes to q_{rip} with an arrow labeled R_1, and
2. q_{rip} goes to itself with an arrow labeled R_2, and
3. q_{rip} goes to q_1 with an arrow labeld R_3, and
4. q_0 goes to q_1 with an arrow labeled R_4

Then, in the new revised machine, the arrow from q_0 to q_1 gets the label

$$(R_1)(R_2)^*(R_3) \cup (R_4)$$

We can make this change for each arrow going from any state q_0 to any state q_1, including when $q_0 = q_1$.

1.4.3 Formal Definition

The formal definition of a GNFA is:

Definition 1.2: Generalized Nondeterministic Finite Automaton

A **generalized nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ where

1. Q is the finite set of tuples.
2. Σ is the input alphabet.
3. $\delta : (Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \mapsto \mathcal{R}$ is the transition function.
4. q_{start} is the start state.
5. q_{accept} is the accept state.

Remarks:

- Here, $\mathcal{R}$ is the collection of all regular expressions over the alphabet Σ.
- If $\delta(q_i, q_j) = R$, then the arrow from state q_i to state q_j has the regular expression R as its label.

1.4.4 Convert Algorithm

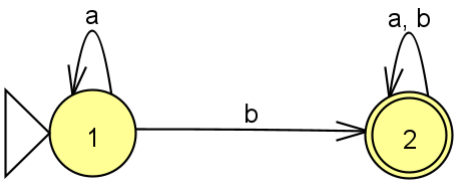
Suppose G is an GNFA. Then, the `CONVERT`(G) algorithm takes a GNFA and returns an equivalent regular expression. The algorithm works like so (Page 73):

`CONVERT`(G)

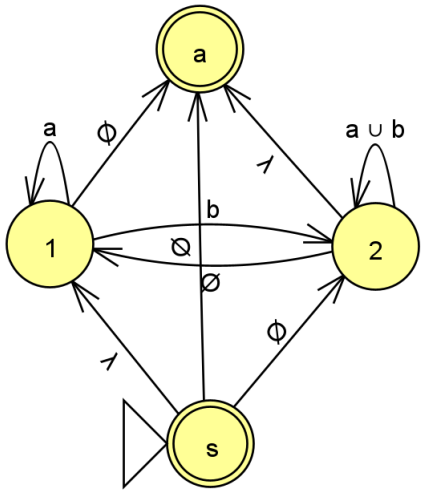
1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. So, return R.
3. Otherwise, $k > 2$ so we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept}. Let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ where $Q' = Q \setminus \{q_{\text{rip}}\}$ and, for any $q_i \in G' \setminus \{q_{\text{accept}}\}$ and $q_j \in Q' \setminus \{q_{\text{start}}\}$, let
$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$
Where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = (q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Compute `CONVERT`(G').

1.4.5 Example 1: DFA to Regular Expression

Suppose we wanted to convert the following DFA to a regular expression:



1. First, we need to convert this DFA to a GNFA. This would look like:



Here, we've made a few changes.

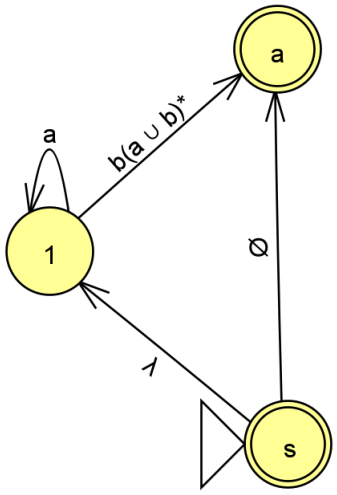
- First, we added two new states: s for the new *start* state and a for the new *accept* state. We have an arrow from s to 1 (the old start state) with ϵ as its label[1]. We also have an arrow from 2 (the old accept state) to a with ϵ as its label.
- Next, note that there was an arrow labeled a, b at state 2. We take the *union* of these two labels to get $a \cup b$. Thus, state 2 now has an arrow with $a \cup b$ instead of a, b. This is because the DFA's label represents two transitions, but a GNFA may only have a single transition going from a state to itself.
- Finally, we add several arrows with the labels being $\emptyset$:
  - 2 to 1 since every state needs to be able to transition to all non-start states.
  - 1 to a for the same reason as above.
  - s to 2 for the same reason as above.
  - s to a for the same reason as above.

2. Next, we pick one non-start/accept state as q_{rip}. We'll pick 2 for our case, so let $2 = q_{\text{rip}}$. We're going to make use of the `CONVERT` algorithm. So, we pick $q_i = 1$ and $q_j = a$. Then:

   - $\delta(q_i, q_{\text{rip}}) = R_1 = b$
   - $\delta(q_{\text{rip}}, q_{\text{rip}}) = R_2 = a \cup b$
   - $\delta(q_{\text{rip}}, q_j) = R_3 = \epsilon$
   - $\delta(q_i, q_j) = R_4 = \emptyset$

   Therefore, $\delta'(q_i, q_j) = (b)(a \cup b)^* \epsilon \cup \emptyset$. This simplifies to $\delta'(q_i, q_j) = (b)(a \cup b)^*$. So, the corresponding new state diagram is:

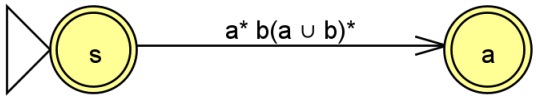
[1] The software used to create these state machines use λ instead of ϵ.



3. We do this process again. We pick our one non-start/accept state as $q_{\text{rip}} = 1$. By our algorithm again, let $q_i = s$ and $q_j = a$. Then:

   - $\delta(q_i, q_{\text{rip}}) = R_1 = \epsilon$
   - $\delta(q_{\text{rip}}, q_{\text{rip}}) = R_2 = a$
   - $\delta(q_{\text{rip}}, q_j) = R_3 = b(a \cup b)^*$
   - $\delta(q_i, q_j) = R_4 = \emptyset$

   Therefore, $\delta'(q_i, q_j) = (\epsilon)(a)^* b(a \cup b)^* \cup \emptyset$. This can be simplified to $\delta'(q_i, q_j) = (a)^* b(a \cup b)^*$. So, the corresponding new state diagram is:



   Thus, the regular expression corresponding to the given DFA is $\boxed{(a)^* b(a \cup b)^*}$

1.5 Regular Expressions and Regularity of Language

Theorem 1.1

A language is regular if and only if some regular expression describes it.

Proof. The proof is given by the two lemmas. □

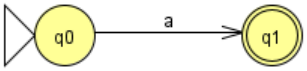
1.5.1 Regular Expression Implies Regularity

Lemma 1.1

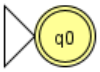
If a language is described by a regular expression, then it is regular.

Proof. Suppose we convert R into an NFA N. We then need to consider six cases as defined by the formal definition of regular expression.

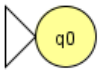
1. Let $R = a$ for some $a \in \Sigma$. Then, $L(R) = \{a\}$ and the following NFA recognizes $L(R)$:



2. Let $R = \epsilon$. Then, $L(R) = \{\epsilon\}$ and the following NFA recognizes $L(R)$:



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



4. $R = R_1 \cup R_2$
5. $R = R_1 \circ R_2$
6. $R = R_1^*$

Where the last three cases is given by a previous proof. □

1.5.2 Regularity Implies Regular Expression

Lemma 1.2

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

Proof. If the language is regular, then it is accepted by a DFA. From the above, we've given a sketch of how to convert a DFA to a regular expression. □