Regular expressions and finite automata

Bakhadyr Khoussainov

Computer Science Department, The University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

Regular expressions are used in many programming languages and tools. They can be used in finding and extracting patterns in texts and programs. For example, using regular expressions we can find lines in texts that contain specific words, characters and letters. We can replace, change, clean and reformat sections in html and xml documents, and more generally, in semi-structured data. Using regular expressions, we can also specify and validate forms of data such as passwords, e-mail addresses, user IDs, etc. The aim of this section is to introduce regular expressions and study their relationship with finite automata. In particular, we will describe methods that convert regular expressions to finite automata, and finite automata to regular expressions.

1 Regular expressions

Every arithmetic expression without an integer variable can be *evaluated*. The value of such expressions are integers. For example, the value of the arithmetic expression $(((1 + 7) - (25 - 19)) \times 2)$ is 4. In this section, we show how to built expressions called regular expressions. These expressions can also be evaluated. The values of regular expressions are languages. Here is now our definition of regular expressions.

Definition 1. *Let Σ be an alphabet. We define* **regular expressions** *of the alphabet Σ by induction:*

Base Case: *The following are regular expressions: each letter σ of Σ, and the symbols $\emptyset$ and λ.*

Inductive step: *Assume that r_1 and r_2 are regular expressions that have been defined. Then each of the following is a regular expression:*

1. $(r_1 + r_2)$.
2. $(r_1 \cdot r_2)$.
3. $(r_1^\star)$.

Thus, every regular expression is a string that is built using the rules above. For example, for the alphabet $\{a, b\}$, the following are regular expressions: a, b, $((a^\star)+(b^\star))$, $(((a^\star)+(b^\star))\cdot(a\cdot b))$, $(((a^\star) + (b^\star)) + \lambda)$. To represent regular expressions, we use symbols r, s, t, etc.

Every regular expression r can be evaluated. The value of the regular expression r, written as $L(r)$, is the language defined below. We call the language $L(r)$ the language of r. Since r is defined inductively, the language $L(r)$ is also defined inductively. The reader needs to recall the concatenation and the star operations on languages introduced in Lecture 11.

Definition 2. *For the regular expression r, the language $L(r)$ is defined as follows:*

Base Case:

1. *If $r = a$, where $a \in \Sigma$, then $L(r) = \{a\}$.*
2. *If $r = \emptyset$ then $L(r) = \emptyset$.*
3. *If $r = \lambda$ then $L(r) = \{\lambda\}$.*

Inductive step: *Assume that the languages $L(r_1)$ and $L(r_2)$, for regular expressions r_1 and r_2, have been defined. Then:*

1. *$L(r_1 + r_2) = L(r_1) \cup L(r_2)$.*
2. *$L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.*
3. *$L(r_1^\star) = (L(r_1)^\star$.*

Note the following. In part 2) of the inductive step we use the symbol $\cdot$ in two different ways. On the left side of the equality, the symbol $\cdot$ is used as a symbol. On the right side of the equality, the symbol $\cdot$ is used as the concatenation operation on the languages. Similarly, in part 3) of the inductive step, the symbol $\star$ is used as a symbol and as an operation.

Now we give several examples of languages $L(r)$ defined by regular expressions r. In all these examples, the alphabet is $\{a, b\}$.

1. $L(a \cdot b + b) = \{ab, b\}$.
2. $L(b^\star) = \{\lambda, b, bb, bbb, bbbb, \ldots\}$.
3. $L(a^\star \cdot b^\star) = \{a^n b^m \mid n \geq 0 \text{ and } m \geq 0\}$.
4. $L((a + b)^\star \cdot (a \cdot b \cdot a) \cdot (a + b)^\star) = \{w \mid w \text{ contains the substring } aba\}$.
5. $L(\emptyset^\star) = \{\lambda\}$.
6. $L(a^\star \cdot a \cdot (bbbaaa) \cdot b \cdot b^\star) = \{w \mid w$ there exist nonempty strings u and v such that $w = ubbbaaav$, where u consists of a's only, and v consists of bs only $\}$.

Strictly speaking, in our examples above we did not follow the syntax of writing the regular expressions. For example, by $a \cdot b + b$ we meant the regular expression $((a \cdot b) + b)$. However, we follow the general rules by omitting parenthesis in order not to make our notation cumbersome. The general rules say that $\cdot$ is of higher precedence then $+$, and $\star$ is higher precedence than $\cdot$. Thus, for example $a \cdot b + a \cdot b^\star$ represents $((a \cdot b) + (a \cdot (b^\star))$.

Definition 3. *We say that a language $L \subseteq \Sigma^\star$ is* **regular** *if there exists a regular expression r such that $L = L(r)$. In this case, we also say that r* **represents** *the language L.*

Note that if r represents L then $r + \emptyset$, $r + \emptyset + \emptyset$, $\ldots$ all represent L. Therefore every regular language has infinitely many regular expressions representing it. Now observe the following. We have two quite different ways of describing languages. The first uses finite automata, and the second uses regular expressions. In the next two sections we show that these two methods of describing languages are equivalent.

2 From regular expressions to finite automata

The goal of this section is to provide a method of converting regular expressions r into automata recognizing $L(r)$. Here is the main goal of this section. We want to show that

Every regular language is FA recognizable.

Let r be a regular expression. We want to show that $L(r)$ is FA recognizable. In order to show that we just follow the inductive definition given for the language $L(r)$ from the previous section and show that $L(r)$ is FA recognizable. We use FA with silent moves.

Base Case: Assume that r is defined in the base case. Then $L(r)$ is either $\{a\sigma\}$ for some $\sigma \in \Sigma$ or $\emptyset$ or $\{\lambda\}$. Clearly in each case $L(r)$ is FA recognizable.

Inductive step: In this step there are three cases to consider.

Case 1. Assume that r is of the form $(r_1 + r_2)$. We now use the inductive hypothesis applied to r_1 and r_2. By the hypothesis, the languages $L(r_1)$ and $L(r_2)$ are FA recognizable (and hence DFA recognizable). The language $L(r_1 + r_2)$ is by definition $L(r_1) \cup L(r_2)$. From the previous lectures, we know that the union of FA recognizable languages is again FA recognizable. Hence $L(r)$ is FA recognizable.

Case 2. Assume that r is of the form $(r_1 \cdot r_2)$. By the inductive hypothesis, the languages $L(r_1)$ and $L(r_2)$ are FA recognizable. The language $L(r_1 \cdot r_2)$ is by definition $L(r_1) \cdot L(r_2)$. We want to show that $L(r_1) \cdot L(r_2)$ is FA recognizable.

Let $\mathcal{A}_1 = (S_1, I_1, T_1, F_1)$ and $\mathcal{A}_2 = (S_2, I_2, T_2, F_2)$ be finite automata recognizing $L(r_1)$ and $L(r_2)$, respectively. Both automata are NFA that may have silent moves. We may assume that the state sets S_1 and S_2 have no states in common. Using these two automata we want to build a finite automaton recognizing $L(r_1) \cdot L(r_2)$.

We now use our method of pretending to be a machine to accept strings in $L(r_1) \cdot L(r_2)$. Suppose w is an input string. We first pretend that we are the machine $\mathcal{A}_1$ and run $\mathcal{A}_1$ on w. If we never reach a final state of $\mathcal{A}_1$ then we reject w. Assume, that we have reached a final state after processing some prefix of w. In this case we need to nondeterministically choose one of the following actions:

1. Continue on running $\mathcal{A}_1$ on w.
2. Make a silent transition to the initial state of $\mathcal{A}_2$ and start simulating $\mathcal{A}_2$.

This observation suggests that we can put the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ together as follows. We keep the states and transition tables of both machine. We declare the initial states of $\mathcal{A}_1$ to be the initial states, and the final states of $\mathcal{A}_2$ to be the final states of our machine. Finally, we add the λ-transitions from the final states of $\mathcal{A}_1$ to the initial states of $\mathcal{A}_2$. Based on this description, we formally define the automaton $\mathcal{A} = (S, I, T, F)$ recognizing $L(r_1) \cdot L(r_2)$ as follows:

1. $S = S_1 \cup S_2$.
2. $I = I_1$.

3. On input (s, σ), where $\sigma \in \Sigma \cup \{\lambda\}$, the transition function T is defined according to the following rules.
   (a) If $s \in S_1$ and s is not a final state then $T(s, \sigma) = T_1(s, \sigma)$.
   (b) If $s \in S_1$, s is a final state, and $\sigma \neq \lambda$ then $T(s, \sigma) = T_1(s, \sigma)$.
   (c) If $s \in S_1$, s is a final state, and $\sigma = \lambda$ then $T(s, \sigma) = T_1(s, \sigma) \cup I_2$.
   (d) If $s \in S_2$ then $T(s, \sigma) = T_2(s, \sigma)$.
4. $F = F_2$.

It is not hard to check that $\mathcal{A}$ recognizes $L(r_1) \cdot L(r_2)$.

Case 3. Assume that r is of the form $(r_1)^\star$. By the inductive hypothesis, the language $L(r_1)$ is FA recognizable. The language $L(r_1^\star)$ is by definition $L(r_1)^\star$. We want to show that $L(r_1)^\star$ is FA recognizable.

Let $\mathcal{A}_1 = (S_1, q_1, T_1, F_1)$ be a finite automaton recognizing $L(r_1)$. We want to construct a finite automaton that recognizes the language $L(r_1)^\star$. As above, we use our method of pretending to be a machine that accepts strings in $L(r_1)^\star$. For an input string w, if it is the empty string then we accept it as the star of every language contains λ. Otherwise we simulate $\mathcal{A}_1$ and read w. Every time we reach a final state of $\mathcal{A}_1$ we have to make a nondeterministic choice. We either continue on running $\mathcal{A}_1$ or make a silent move to one of the initial states of $\mathcal{A}_1$. Thus, we construct our automaton $\mathcal{A}$ recognizing $L(r_1)^\star$ as follows. We keep all the states of $\mathcal{A}_1$; we keep all the transitions of $\mathcal{A}_1$; we add λ-transitions from each final state of $\mathcal{A}_1$ to every initial state of $\mathcal{A}_1$; we add one new initial state q_{new} with no outgoing transitions and declare it to be a final state (this is needed to accept the empty string). Here is a more formal definition of the automaton $\mathcal{A} = (S, I, T, F)$ recognizing $L(r_1)^\star$:

1. $S = S_1 \cup \{q_{new}\}$, where q_{new} is a state not in S_1.
2. $I = I_1 \cup \{q_{new}\}$.
3. On input (s, σ), where $\sigma \in \Sigma \cup \{\lambda\}$, the transition function T is defined according to the following rules.
   (a) If $s \in S_1$ and s is not a final state then $T(s, \sigma) = T_1(s, \sigma)$.
   (b) If $s \in S_1$, s is a final state, and $\sigma \neq \lambda$ then $T(s, \sigma) = T_1(s, \sigma)$.
   (c) If $s \in S_1$, s is a final state, and $\sigma = \lambda$ then $T(s, \sigma) = T_1(s, \sigma) \cup I_1$.
   (d) If $s = q_{new}$ then $T(s, \sigma) = \emptyset$.
4. $F = F_1 \cup \{q_{new}\}$.

Now it is not hard to check that $\mathcal{A}$ recognizes $L(r_1)^\star$.

3 Generalized finite automata

The goal of this section is to show how to convert finite automata $\mathcal{A}$ to regular expressions r such that $L(\mathcal{A}) = L(r)$. There are several methods of such conversions. Here we provide one method based on an elegant generalization of finite automata.

One can generalize finite automata in many ways. One generalization can be obtained by assuming that a machine processes an input not just once but may move its head back and forth as many times as needed. This produces the Turing machine model of computations. Another generalization of finite automata is obtained by providing an extra memory location for a given finite automaton. When the automaton reads an input of length n, the memory location can store at most n amount of information. This type of machines are called pushdown automata. In this section we explain machines called generalized finite automata that process input strings block by block instead of symbol by symbol.

Informally, a generalized finite automaton is the same as a finite automaton apart from the fact that the labels of the transitions are now regular expressions. Thus, a generalized finite automaton has a finite set of states, some states are initial states, and some are final. In addition, there are edges between the states so that the edges are labeled with regular expressions. Clearly, every NFA is a generalized finite automata because the labels of all of its edges are symbols from the alphabet or the empty string λ. All these labels are regular expressions.

A generalized finite automaton works as follows. The automaton reads a block of symbols from the input and not necessarily just one input signal, then makes a transition along an edge. The block of symbols read must belong to the language described by one of the regular expressions that label the edge. Of course, the automaton is nondeterministic so it has several ways to process the block. A sequence of blocks is now accepted by the automaton if the automaton can reach a final state by processing the sequence. We now give a formal definition:

Definition 4. *A* **generalized nondeterministic finite automaton***, written as GNFA, is a* 5*-tuple* $\mathcal{A} = (S, I, T, F, \Sigma)$*, where*

1. S *is a finite nonempty set called the* **set of states***.*
2. I *is a nonempty subset of* S *called the* **set of initial states***.*
3. T *is a function that labels transitions with regular expressions.*
4. F *is a subset of* S *called the* **set of accepting states***.*

We say that a GNFA $\mathcal{A}$ **accepts** a string $w \in \Sigma^\star$ if for some $w_1,\ldots,w_n \in \Sigma^\star$ we have the following properties:

1. $w = w_1 \ldots w_n$.
2. There exists a sequence $s_1, s_2, \ldots s_{n+1}$ of states of $\mathcal{A}$ such that:
   (a) s_1 is an initial state,
   (b) s_{n+1} is a final state, and
   (c) For every i, $1 \le i \le n$, the word w_i belongs to the language $L(r)$, where the transition from s_i and s_{i+1} has r as a label.

The language recognized by the NGFA $\mathcal{A}$ is the set of all strings accepted by $\mathcal{A}$. We use the same notation as for finite automata:

$$L(\mathcal{A}) = \{w \in \Sigma^\star \mid \mathcal{A} \text{ accepts } w\}.$$

Our goal now is to show that it is possible to reduce the number of states to 2 of the generalized automaton $\mathcal{A}$ without changing the language recognized. In order to achieve this goal we need the following definition:

Definition 5. *A GNFA $\mathcal{A}$ is* **reduced** *if the following properties hold:*

1. *$\mathcal{A}$ has exactly one start state q_0.*
2. *There exists exactly one accepting state f not equal to the start state.*
3. *For any state s there is no transition from s to s_0.*
4. *For any state s there is no transition from f to s.*
5. *For each pair s, s' of states not in $\{s_0, f\}$ there exist edges from s to s'.*
6. *Every transition has exactly one regular expression labeling it.*

The fact below shows that we can restrict ourselves to consider reduced GNFA only.

For any GNFA $\mathcal{A}$ there exists a reduced GNFA $\mathcal{B}$ such that the automata $\mathcal{A}$ and $\mathcal{B}$ accept the same language.

We change the automaton $\mathcal{A}$ as follows. We add a new initial state q_0 with the λ-transitions to all original initial states. We add a new final state f with the λ-transitions from all original final states to f. Declare old initial and final states to be neither initial nor final. Thus, now we have exactly one initial state and one final state. This satisfies the conditions 1), 2) and 3) needed for a GNFA to be reduced. These changes do not alter the language recognized. If a transition has several labels, say the labels are $r_1, \ldots, r_k$, then we label the transition with a single regular expression which is the sum $(r_1 + \ldots + r_k)$. Finally, for all $s, s' \notin \{s_0, f\}$, if there is no edge from s to s', then we simply add an edge from s to s' and label the edge with $\emptyset$. The resulting GNFA is equivalent to the original one. This proves the lemma.

Now our goal is to show the following:

For every GNFA $\mathcal{A}$ there exists a reduced GNFA $\mathcal{B}$ with exactly two states such that $L(\mathcal{A}) = L(\mathcal{B})$.

We can assume that the automaton $\mathcal{A}$ is reduced by the fact above. The idea of our proof is this. We select a state s in $\mathcal{A}$ which is neither initial nor final. We remove the state and then repair the "damage" done by relabeling and adding some transitions to the original automaton $\mathcal{A}$. These new labeled transitions recover all the runs lost when the state s is removed. The new automaton has fewer states than the original automaton and accepts the same language. Continue this procedure until two states, the initial state and the final state, are left. The automaton thus constructed will have two states and will then be the desired one.

Now we give a formal construction, called *Reduce*$(\mathcal{A})$, of the desired automaton. Assume that $\mathcal{A}$ has exactly n states. Now proceed as follows:

1. If $n = 2$, then $\mathcal{A}$ is the desired one.
2. Suppose that $n > 2$. Take a state s distinct from both the initial state and the final state. Construct a generalized nondeterministic finite automaton $\mathcal{A}_1 = (S_1, I_1, T_1, F_1)$ according to the following rules:
   (a) $S_1 = S \setminus \{s\}$.
   (b) $I_1 = I$, $F_1 = F$.

(c) For all $s_1, s_2 \in S$ that are distinct from s such that $T(s_1, s) = r_1$, $T(s, s) = r_2$, $T(s, s_2) = r_3$, $T(s_1, s_2) = r_4$ label the edge from s_1 to s_2 with the regular expression $r_1 \cdot r_2^\star r_3 + r_4$. Thus, $T_1(s_1, s_2) = r_1 \cdot r_2^\star \cdot r_3 + r_4$.

The automaton $\mathcal{A}_1$ constructed has the following properties. The number of states of $\mathcal{A}_1$ is $n - 1$. The automaton $\mathcal{A}_1$ is still reduced. The automaton $\mathcal{A}_1$ recognizes exactly the same language recognized by $\mathcal{A}$. Indeed, if $\mathcal{A}$ makes a transition without going through the state s then the transition is also a transition of $\mathcal{A}_1$. If $\mathcal{A}$ makes a transition from state s_1 to s then from s to s_2, where s_1 and s_2 are distinct from s, then these two transitions of $\mathcal{A}$ can be replaced by one transition from s_1 to s_2 in $\mathcal{A}_1$ due to the definition of T_1. Similarly, any transition in $\mathcal{A}_1$ can be simulated by transitions in $\mathcal{A}$.

In order to complete the proof of the theorem, we continue the procedure above by replacing $\mathcal{A}$ with $\mathcal{A}_1$. This continues on until the two states, the initial and final states, are left. The two state automaton $\mathcal{B}$ thus obtained is then the desired one.

Corollary 1. *Every FA recognizable language L is regular.*

Indeed, let $\mathcal{A}$ be an NFA that recognizes L. This automaton is also a GNFA. By the theorem above there exists a reduced two state GNFA $\mathcal{B} = (S, I, T, F)$ such that $L(\mathcal{B}) = L$. Let s_0 be the initial state and f be the final state of $\mathcal{B}$. Let $r = T(s_0, f)$. Then $L = L(r)$. The corollary is proved.

Thus, we have proved the following result known as the Kleene's theorem.

Theorem 1 (The Kleene's theorem). *A language L is FA recognizable if and only if L is regular.*

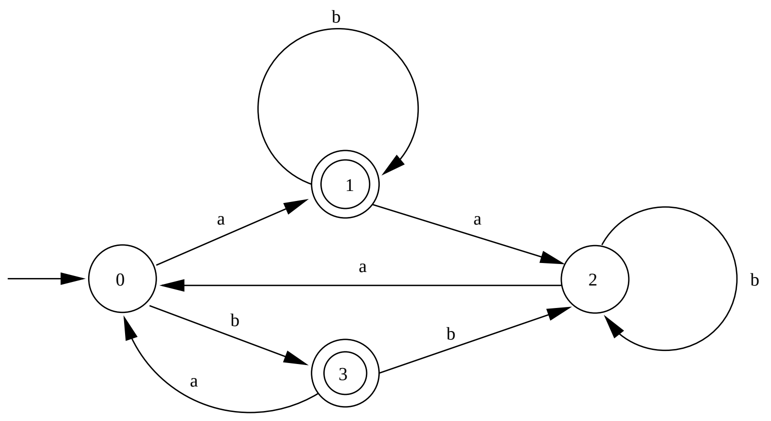
4 Exercise

1. Design an algorithm that checks if a string r is a regular expression of the alphabet $\{a, b\}$. Implement the algorithm.
2. Prove that every finite language is regular.
3. Write down regular expressions representing the following languages over the alphabet $\{a, b\}$:
   (a) $\{w1 \mid w \in \Sigma^\star\}$.
   (b) $\{\lambda\}$.
   (c) $\{w \mid w \in \Sigma^\star$ and $w \neq \lambda\}$.
   (d) $\{uaabv \mid u, v \in \Sigma^\star\}$.
   (e) $\{w \mid w$ ends with $000\}$.
   (f) $\{w \mid w$ contains a substring aub such that u has length $6\}$.
   (g) $\{w \mid w$ constains substring $abab\}$.
   (h) $\{w \mid w$ ends with an a and no a occurs between any occurrences of $b\}$.
   (i) $\{w \mid w$ has a on the 9th position from the right of $w\}$.
4. For each of the following regular expressions r construct finite automata recognizing the language $L(r)$:

(a) a.
(b) $a^\star$.
(c) $(a^\star + b^\star)$.
(d) $(a \cdot b)$.
(e) $(a \cdot b)^\star$.
(f) $(a^\star + b^\star) \cdot (a \cdot b)^\star$.

Your construction should follow the proof of Theorem **??**.

5. Consider the automaton $\mathcal{B}$ in Figure below. Do the following:
   (a) Transform the automaton to the reduced automaton $\mathcal{A}$.
   (b) Apply the *Reduced*($\mathcal{A}$) algorithm by removing states 0, 1, 2, and 3 turn by turn. Draw transition diagrams of the resulting generalized nondeterministic finite automata.


b
1
a
a
a
0
2
b
b
b
3
a