بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

Equivalence with Finite Automata

Regular expressions and finite automata are equivalent in their descriptive power. This fact is surprising because finite automata and regular expressions superficially appear to be rather different. However, any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa. Recall that a regular language is one that is recognized by some finite automaton.

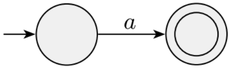
The two concepts are essentially the same; for every regular language there is a regular expression, and for every regular expression there is a regular language. We will show this in two parts.

Theorem: A language is regular if and only if some regular expression describes it. This theorem has two directions. We state and prove each direction as a separate lemma.

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

PROOF Let's convert R into an NFA N. We consider the six cases in the formal definition of regular expressions.

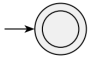
1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.



Note that this machine fits the definition of an NFA but not that of a DFA because it has some states with no exiting arrow for each possible input symbol. Of course, we could have presented an equivalent DFA here; but an NFA is all we need for now, and it is easier to describe.

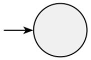
Formally, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where we describe δ by saying that $\delta(q_1, a) = \{q_2\}$ and that $\delta(r, b) = \emptyset$ for $r \neq q_1$ or $b \neq a$.

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



Formally, $N = \big(\{q_1\}, \Sigma, \delta, q_1, \{q_1\}\big)$, where $\delta(r, b) = \emptyset$ for any r and b.

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



Formally, $N = \big(\{q\}, \Sigma, \delta, q, \emptyset\big)$, where $\delta(r, b) = \emptyset$ for any r and b.

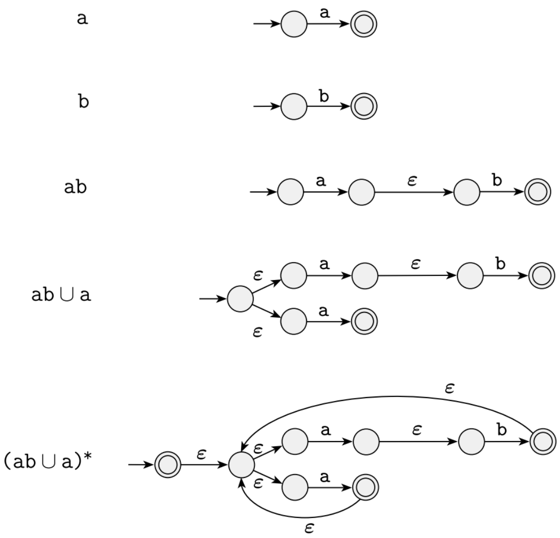
4. $R = R_1 \cup R_2$.

5. $R = R_1 \circ R_2$.

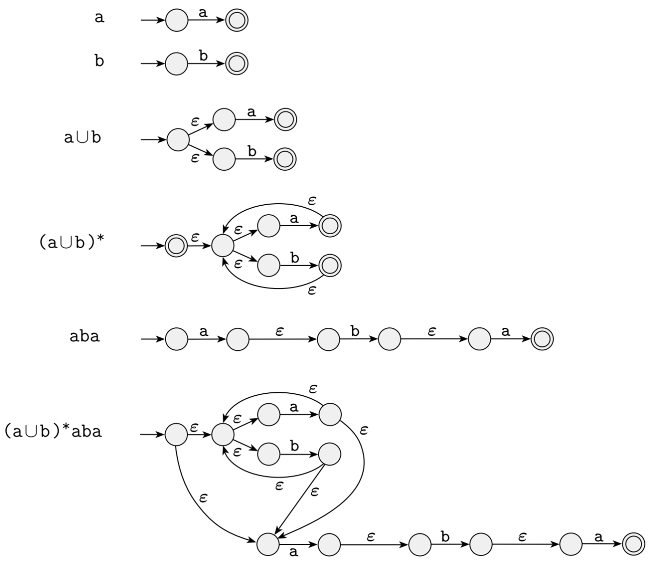
6. $R = R_1^*$.

For the last three cases, we use the constructions given in the proofs that the class of regular languages is closed under the regular operations. In other words, we construct the NFA for R from the NFAs for R_1 and R_2 (or just R_1 in case 6) and the appropriate closure construction.

Example: Building an NFA from the regular expression $(ab \cup a)^*$



Example: Building an NFA from the regular expression $(a \cup b)^*aba$



Exercise:* *Convert the following regular expression to an NFA:

$$(abab)^* \cup (aaa^* \cup b)^*.$$

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

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. We describe a procedure for converting DFAs into equivalent regular expressions.

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 DFAs into GNFAs, and then GNFAs into regular expressions.

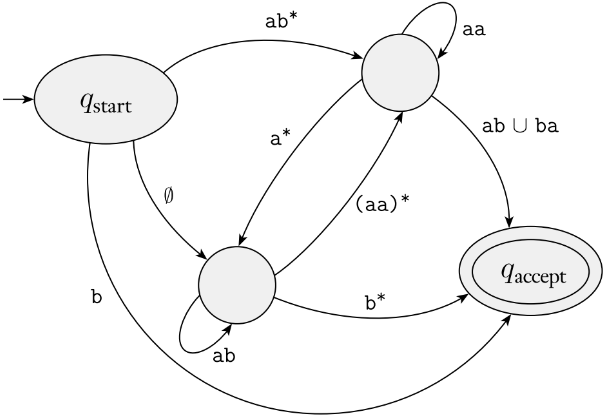
GNFAs or Generalized Transition Graphs (GTGs)

☞ *Generalized nondeterministic finite automata 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 from the input, not necessarily just one symbol at a time as in an ordinary NFA. The GNFA moves along a transition arrow connecting two states by reading a block of symbols from the input, which themselves constitute a string described by the regular expression on that arrow.*

☞ *A GNFA is nondeterministic and so may have several different ways to process the same input string.*

☞ *It accepts its input if its processing can cause the GNFA to be in an accept state at the end of the input.*


q_{start}
q_{accept}
ab*
aa
a*
ab ∪ ba
∅
(aa)*
b
b*
ab

For convenience, we require that GNFAs always have a special form that meets the following conditions.

☞ *The start state has transition arrows going to every other state but no arrows coming in from any other state.*

☞ *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. 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.*

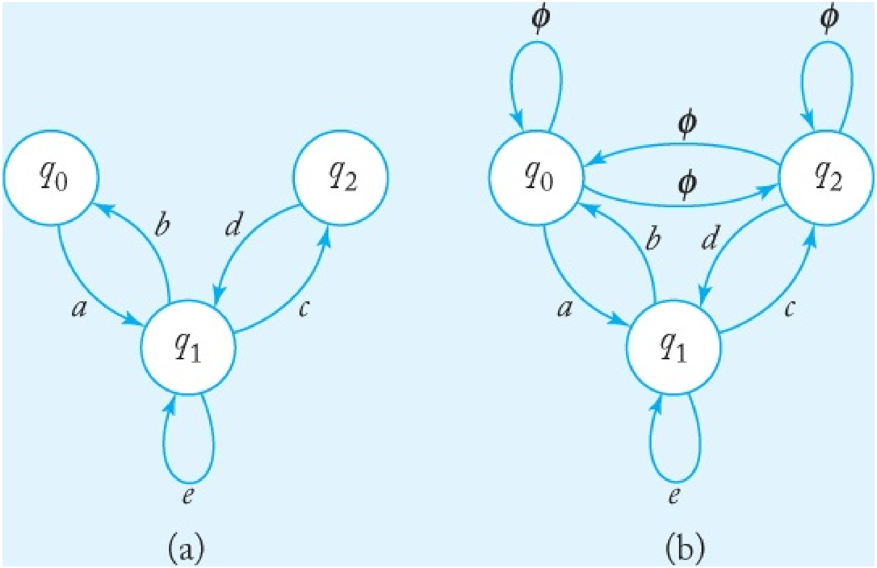
We can easily convert a DFA into a GNFA in the special form.

☞ *We simply **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 any arrows have multiple labels (or if there are multiple arrows going between the same two states in the same direction), we replace each with a **single arrow** whose **label** is the **union of the previous labels.***

☞ *Finally, we add arrows labeled $\varnothing$ between states that had no arrows. This last step won't change the language recognized because a transition labeled with $\varnothing$ can never be used.*

From here on we assume that all GNFAs are in the special form.


q_0
q_1
q_2
a
b
c
d
e
ϕ
(a)
(b)

Now we show how to convert a GNFA into a regular expression.

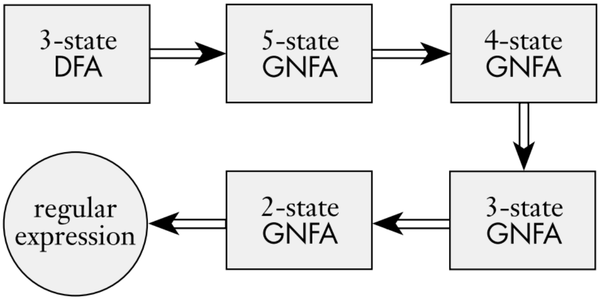
☞ *Say that the GNFA has k states. Then, because a GNFA must have a start and an accept state and they must be different from each other, we know that $k \geq 2$.*

☞ *If $k > 2$, we construct an equivalent GNFA with $k - 1$ states. This step can be repeated on the new GNFA until it is reduced to two states.*

☞ *If $k = 2$, the GNFA has a single arrow that goes from the start state to the accept state. The label of this arrow is the equivalent regular expression.*

Typical stages in converting a DFA to a regular expression

For example, the stages in converting a DFA with three states to an equivalent regular expression are shown in the following figure.



The crucial step is constructing an equivalent GNFA with one fewer state when $k > 2$.

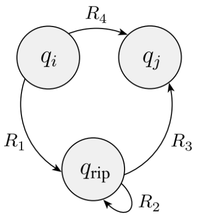
☞ *We do so by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized.*

☞ *Any state will do, provided that it is not the start or accept state. We are guaranteed that such a state will exist because $k > 2$.*

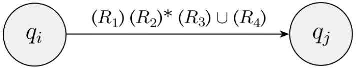
☞ *Let's call the removed state q_{rip}. After removing q_{rip} we repair the machine by altering the regular expressions that label each of the remaining arrows. The new labels compensate for the absence of q_{rip} by adding back the lost computations.*

☞ *The new label going from a state q_i to a state q_j is a regular expression that describes all strings that would take the machine from q_i to q_j either directly or via q_{rip}.*

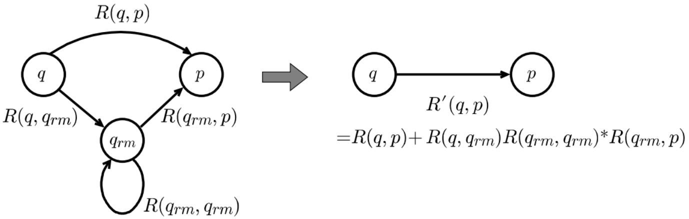
Constructing an equivalent GNFA with one fewer state



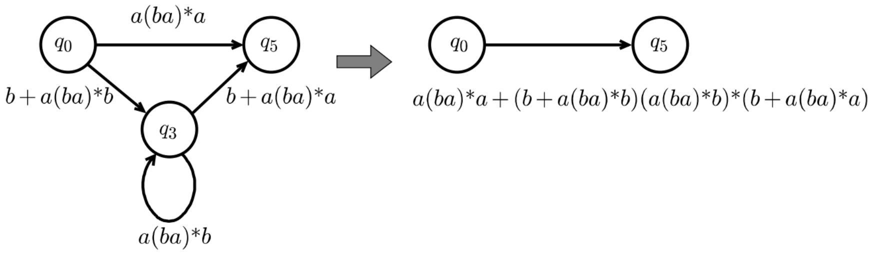
before



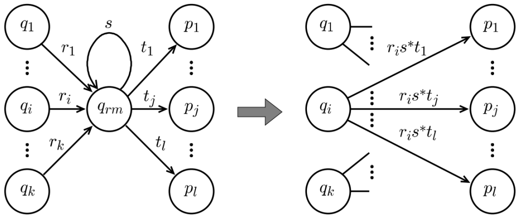
after



مثال



q_0
q_5
q_3
$a(ba)^*a$
$b+a(ba)^*b$
$b+a(ba)^*a$
$a(ba)^*b$
q_0
q_5
$a(ba)^*a+(b+a(ba)^*b)(a(ba)^*b)^*(b+a(ba)^*a)$



q_1
q_i
q_k
q_{rm}
p_1
p_j
p_l
r_1
r_i
r_k
s
t_1
t_j
t_l
$r_is^*t_1$
$r_is^*t_j$
$r_is^*t_l$

In the old machine, if

1. q_i goes to q_{rip} with an arrow labeled R_1,

2. q_{rip} goes to itself with an arrow labeled R_2,

3. q_{rip} goes to q_j with an arrow labeled R_3, and

4. q_i goes to q_j with an arrow labeled R_4, then in the new machine, the arrow from q_i to q_j gets the label $(R_1)(R_2)^*(R_3) \cup (R_4)$.

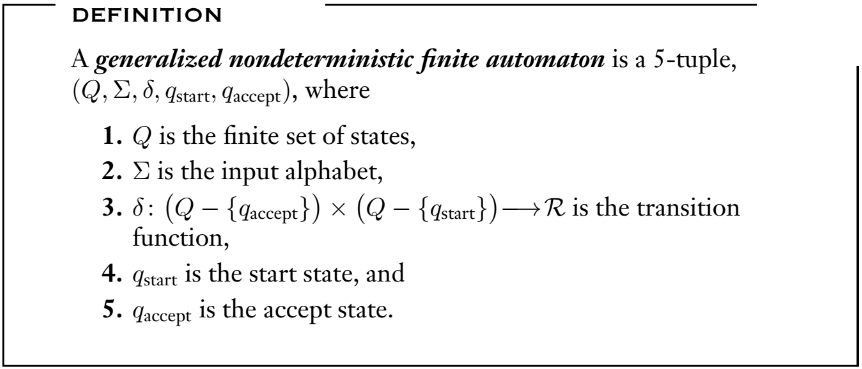
We make this change for each arrow going from any state q_i to any state q_j, including the case where $q_i = q_j$. The new machine recognizes the original language.

Let's now carry out this idea formally.

We formally define the new type of automaton introduced. A GNFA is similar to a nondeterministic finite automaton except for the transition function, which has the form

$$\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \mapsto \mathcal{R}.$$

The symbol $\mathcal{R}$ is the collection of all regular expressions over the alphabet Σ, and q_{start} and q_{accept} are the start and accept states. If $\delta(q_i, q_j) = R$, the arrow from state q_i to state q_j has the regular expression R as its label. The domain of the transition function is $(Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\})$ because an arrow connects every state to every other state, except that no arrows are coming from q_{accept} or going to q_{start}.



DEFINITION

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 states,
2. Σ is the input alphabet,
3. $\delta\colon (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

تمرین: تابع انتقال یک ***GNFA*** را با تابع انتقال یک ***DFA*** یا یک ***NFA*** مقایسه کنید.

مفهوم پذیرش برای یک GNFA

A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1 w_2 \cdots w_k$, where each w_i is in Σ^* and a sequence of states $q_0, q_1, \ldots, q_k$ exists such that

1. *$q_0 = q_{\text{start}}$ is the start state,*
2. *$q_k = q_{\text{accept}}$ is the accept state, and*
3. *for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$; in other words, R_i is the expression on the arrow from q_{i-1} to q_i.*

تمرین: تعریف رسمی فوق برای پذیرش یک رشته توسط یک GNFA را با تعریف رسمی مفهوم پذیرش برای DFAها و NFAها مقایسه کنید.

پروسهٔ کاستن از تعداد استیت‌ها در یک GNFA

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.
   Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

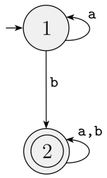
   $$Q' = Q - \{q_{\text{rip}}\},$$

   and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$, let

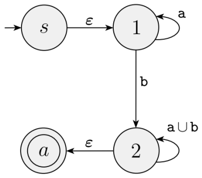
   $$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

   for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Compute CONVERT(G') and return this value.

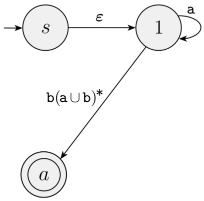
***Example:** To avoid cluttering up the figure, we do not draw the arrows labeled $\emptyset$, even though they are present.*



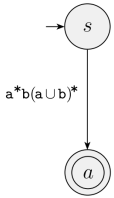
(a)



(b)

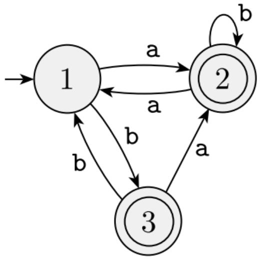


(c)

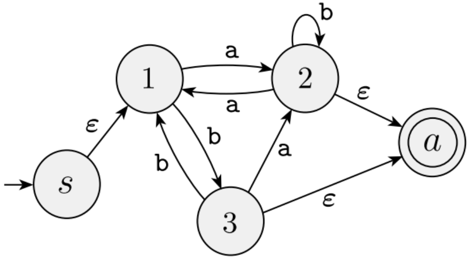


(d)

Example: *In this example, we begin with a three-state DFA. The steps in the conversion are shown in the following figure.*

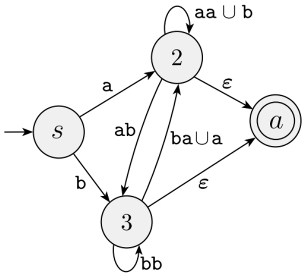


(a)

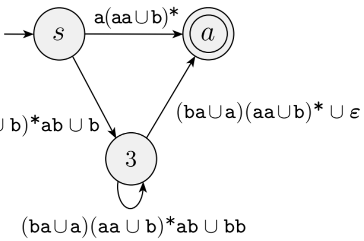


(b)

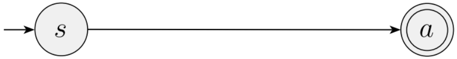
ادامهٔ مثال



(c)



(d)



$(a(aa\cup b)^*ab\cup b)((ba\cup a)(aa\cup b)^*ab\cup bb)^*((ba\cup a)(aa\cup b)^*\cup \varepsilon)\cup a(aa\cup b)^*$

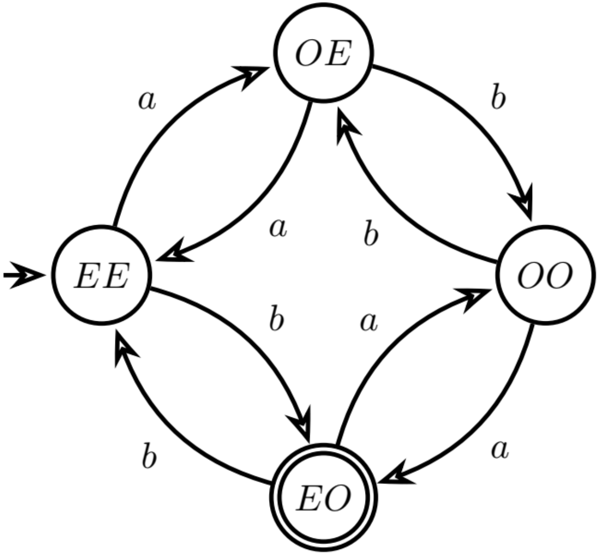
(e)

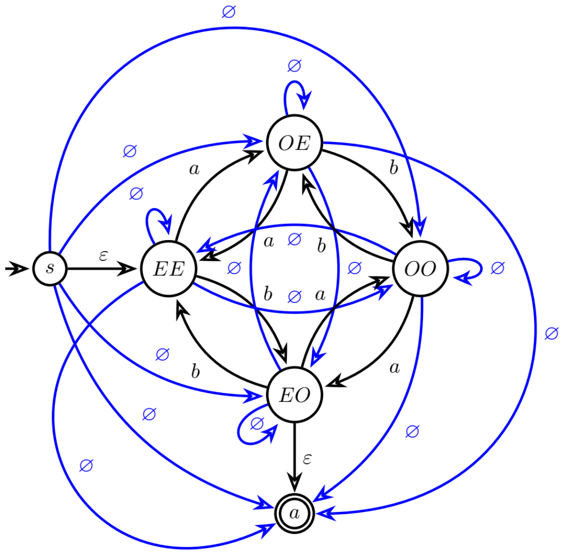
***Example*: Find a regular expression for the language**

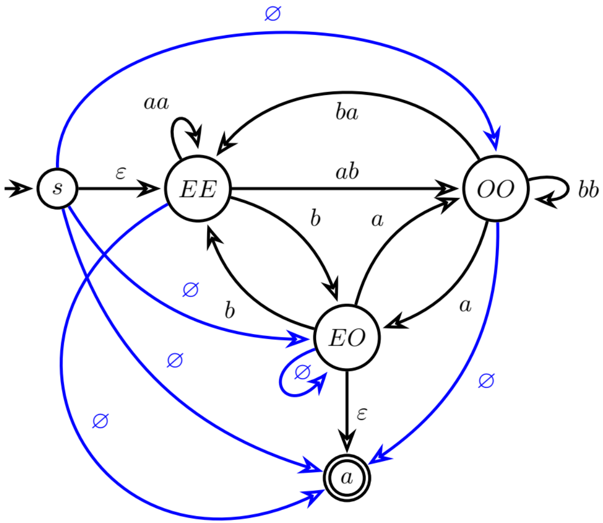
$$L = \{w \in \{a,b\}^* : n_a(w) \textbf{ is even and } n_b(w) \textbf{ is odd}\}.$$

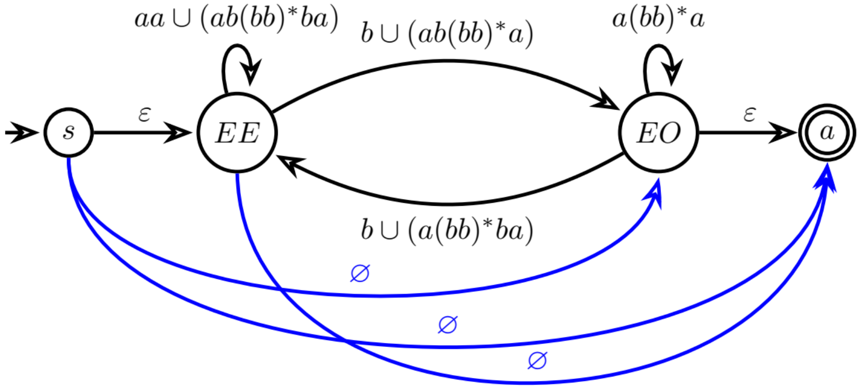
An attempt to construct a regular expression directly from this description leads to all kinds of difficulties. On the other hand, finding an NFA for it is easy as long as we use vertex labeling effectively.

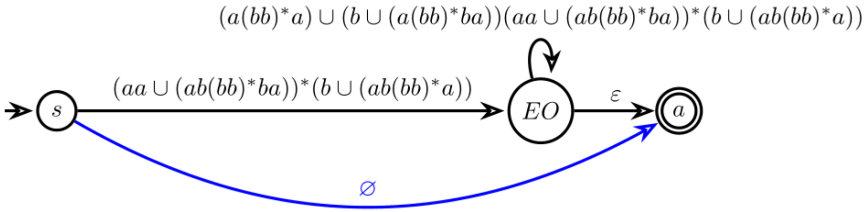
We label the vertices with EE to denote an even number of a's and b's, with OE to denote an odd number of a's and an even number of b's, and so on.

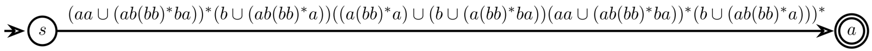

OE
EE
OO
EO
a
b
a
b
b
a
b
a


s
EE
OE
OO
EO
a
ε
a
b
b
a
a
b
ε
∅


∅
aa
ba
s
ε
EE
ab
OO
bb
b
a
∅
b
a
EO
∅
∅
∅
ε
∅
a


$aa \cup (ab(bb)^*ba)$
$b \cup (ab(bb)^*a)$
$a(bb)^*a$
s
ε
EE
EO
ε
a
$b \cup (a(bb)^*ba)$
$\emptyset$
$\emptyset$
$\emptyset$


$(a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a))$
$(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a))$
s
EO
ε
a
$\varnothing$


$(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a))((a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a)))^*$
s
a

$$R = (aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a))$$
$$((a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a)))^*$$