Regular Expressions

Every language is a set of strings.
We have defined 3 regular operations on languages: $\cup$, $\circ$, $*$
Regular languages are closed under regular operations.
We can build complex regular languages from simpler ones using regular operations.

Examples: $\{0\} \cup \{1\}$ $\qquad$ $\{0,1\} \circ \{0\}^*$

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

Shorthand: Since we're concerned with languages (not strings) we write 0 instead of $\{0\}$

So $\{0\} \cup \{1\} = 0 \cup 1$

$\{0,1\} \circ \{0\}^* = (0 \cup 1) \circ 0^* = (0 \cup 1)0^*$

$(\{0\} \cup \{1\})^* \circ \{1\} \circ (\{0\} \cup \{1\}) = (0 \cup 1)^* 1 (0 \cup 1)$

Defining Regular Expressions Recursively

Given an alphabet Σ, a regular expression R over Σ is:

1. $a, \quad a \in \Sigma$ represents the language $\{a\}$
2. ϵ $\{\epsilon\}$
3. ϕ $\{\}$
4. $(R_1 \cup R_2)$, where R_1, R_2 are both regular expressions
5. $(R_1 \circ R_2)$
6. $(R_1)^*$

Rules to remember:

$R \circ \phi = \phi$

$R \circ \epsilon = R$

$\phi^* = \{\epsilon\}$

$R \cup \phi = R$

$R \cup \epsilon = R \; iff \; \epsilon \in R$

Sample Regular Expressions

Let $\Sigma = \{0,1\}$.

$L_1 = \{3rd\ last\ bit\ is\ 1\}$

$\Sigma^* 1 \Sigma\Sigma$

$L_2 = \{strings\ in\ which\ every\ 0\ is\ followed\ by\ at\ least\ one\ 1\}$

$1^*(011^*)^* \quad = \quad 1^*(01^+)^*$

$L_3 = \{strings\ starting\ with\ 0\ and\ followed\ by\ any\ number\ of\ 1s\ OR\ strings\ with\ no\ 0\}$

$01^* \cup 1^* \quad = \quad (0 \cup \epsilon)1^*$

$L_4 = \{strings\ with\ lengths\ that\ are\ multiples\ of\ 3\}$

$(\Sigma\Sigma\Sigma)^*$

Regular Expressions and FSAs

Theorem 1.54 A language is regular if and only if some regular expression describes it.

Proof in two parts:

- Given a regular expression, construct an FSA for it.
- Given an FSA, construct a regular expression for it.

How do regular expressions relate to regex?

Initially, regex was defined as regular expressions.

As more sophisticated string matching algorithms evolved from the theory, regex evolved.

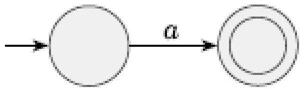
From Regular Expressions to NFA

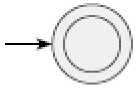
Part I: Given a regular expression R, construct a DFA (or an NFA) for it.

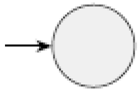
Proof by induction:

Base Cases: $R = a\ (a \in \Sigma)$ $\quad R = \epsilon$ $\quad R = \phi$

NFAs:



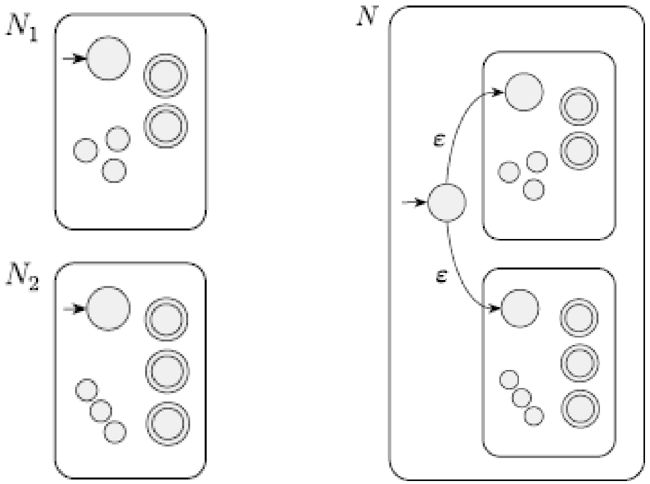




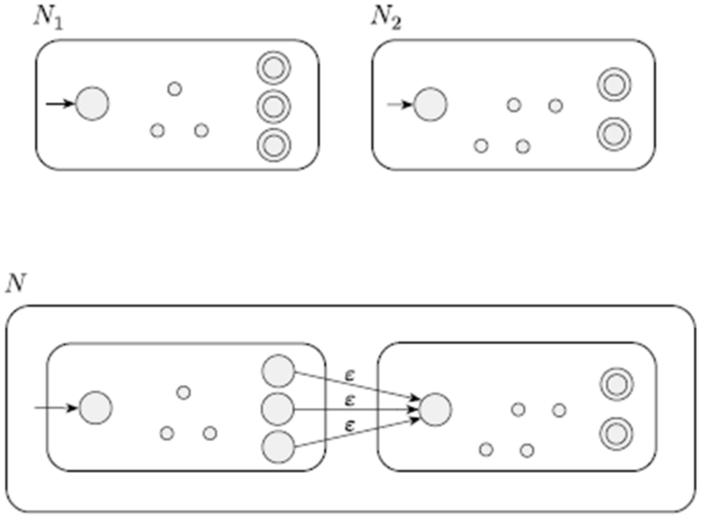
Inductive Step: $R = R_1 \cup R_2$ $\quad R = R_1 \circ R_2$ $\quad R = (R_1)^*$

NFAs We already did these!

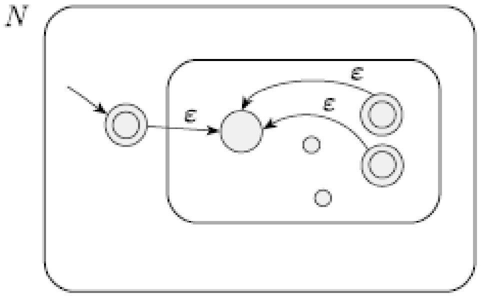
Recursively Designed NFAs for REs



NFA for $A \cup B$



NFA for $A \circ B$

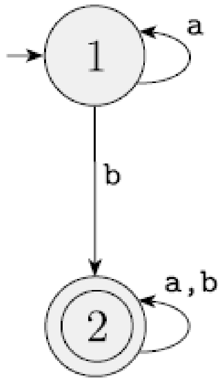


NFA for A^*

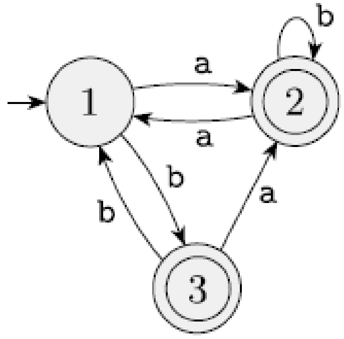
From DFA to Regular Expressions

Part II: Given a DFA M describe $L(M)$ as a regular expression.

Examples:



$$a^*b(a \cup b)^*$$



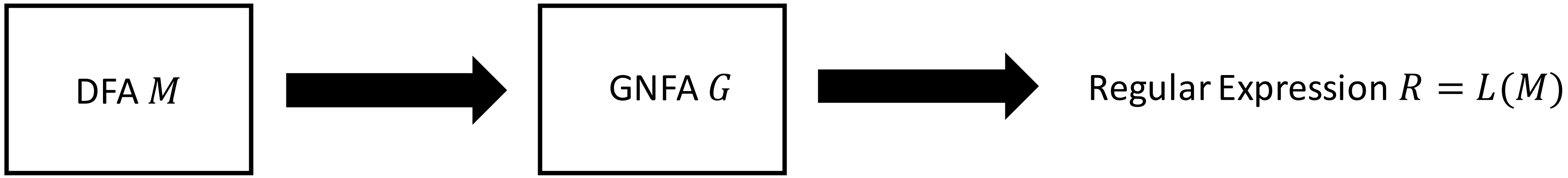
$$(a(aa \cup b)^*ab \cup b)((ba \cup a)(aa \cup b)^*ab \cup bb)^*((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup a(aa \cup b)^*$$

Is there a systematic way to convert DFAs?

Converting DFA to REs Systematically

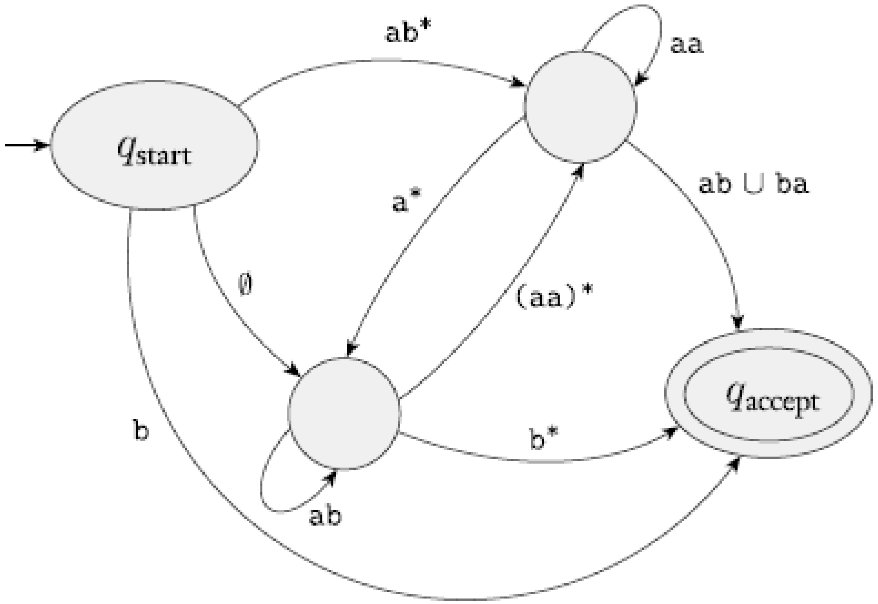
Part II: Given a DFA M describe $L(M)$ as a regular expression .

Proof Outline:



Step 1: Convert the DFA into a Generalized NFA
Step 2: Derive the regular expression from the GNFA

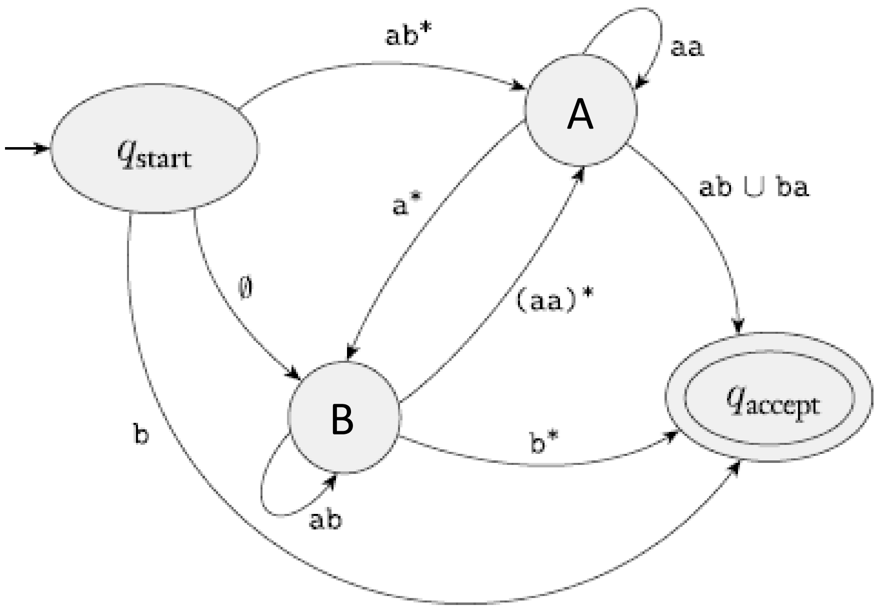
Generalized NFAs



Like an NFA but:

Each transition labeled with a regular expression

Generalized NFAs

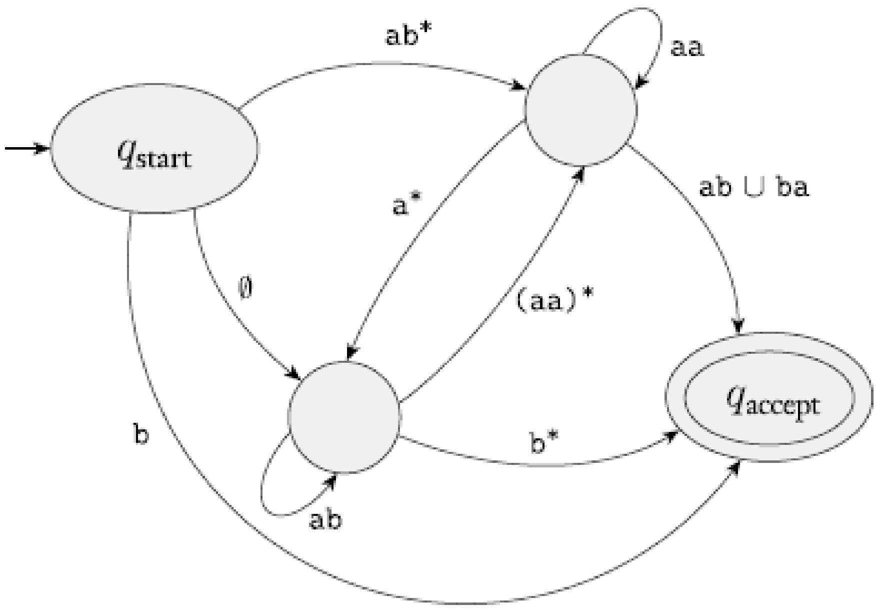


$abbbbaaba$
abb
bb
aab

How does a GNFA computation proceed?

1. Take a transition by matching a block of input symbols to the RE label.
2. Accept input if in accept state and entire input has been processed.

Generalized NFAs

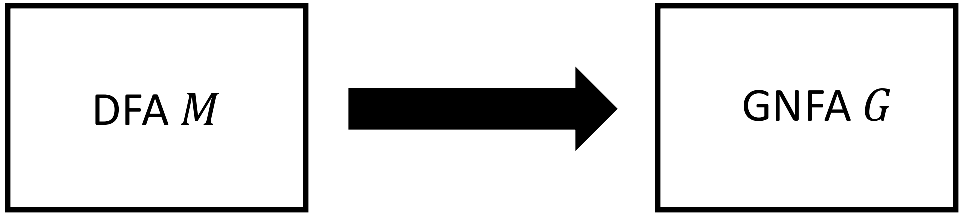


Like an NFA but:

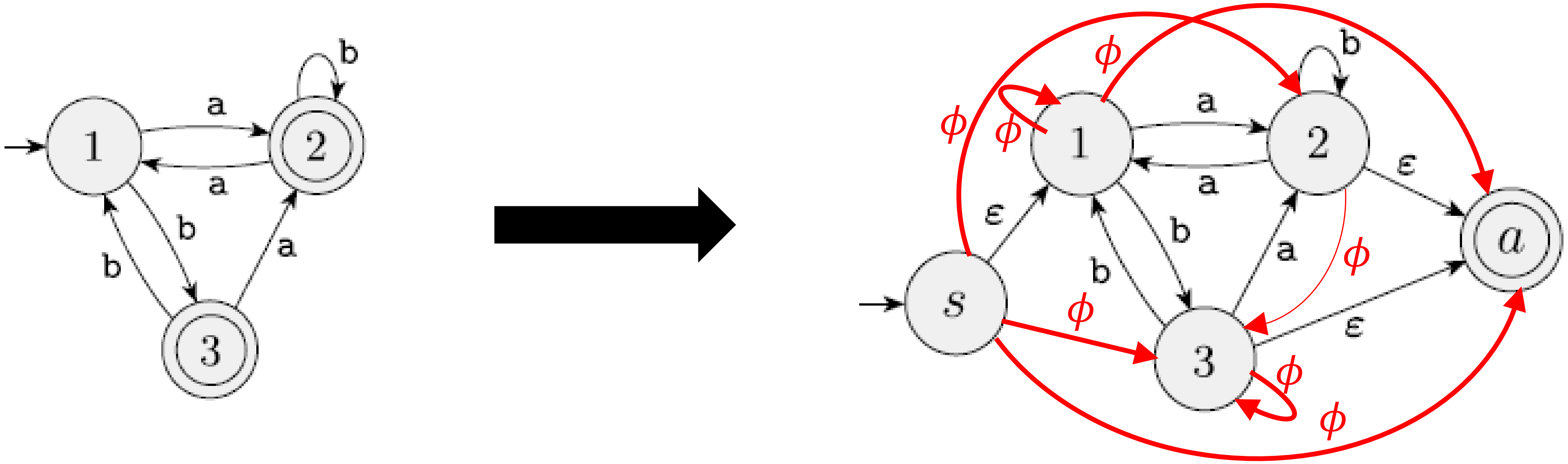
1. Start state:
   a. No incoming transition
   b. Outgoing transitions to every state
2. Accept state:
   a. No outgoing transition
   b. Incoming transitions from every state
   c. Distinct from start state
3. All other states:
   a. Outgoing transition to every state (except start), including itself.
4. Transitions labeled with regular expressions.

DFA to GNFA Conversion

Step 1: Convert the DFA into a Generalized NFA

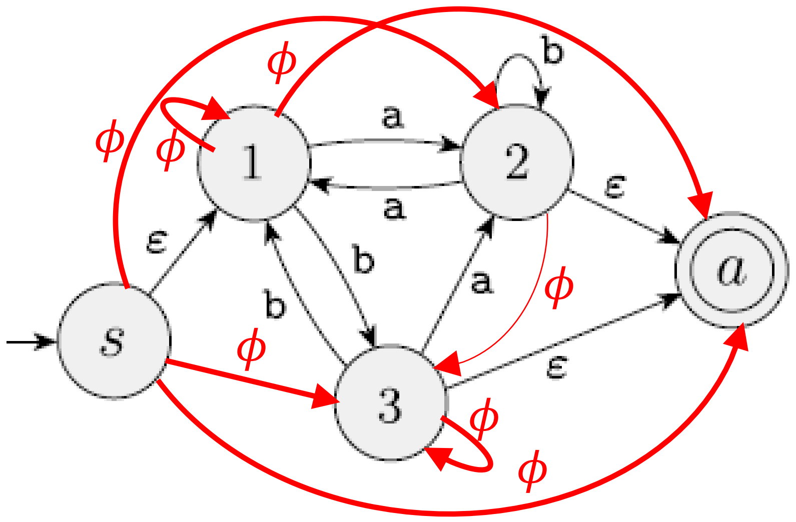


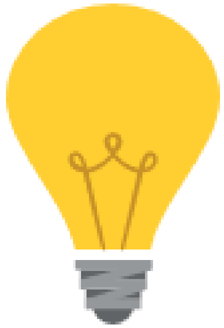
Create a new start state with outgoing ϵ-transition to the start state of M
Create a new accept state with incoming ϵ-transitions from every accept state of M
If there are multiple labels on any transition, replace it with the union of the labels



Deriving the RE from the GNFA

Step 2: Derive the regular expression from the GNFA





Manipulate labels so that the label on (s, a) describes all strings accepted by the GNFA!

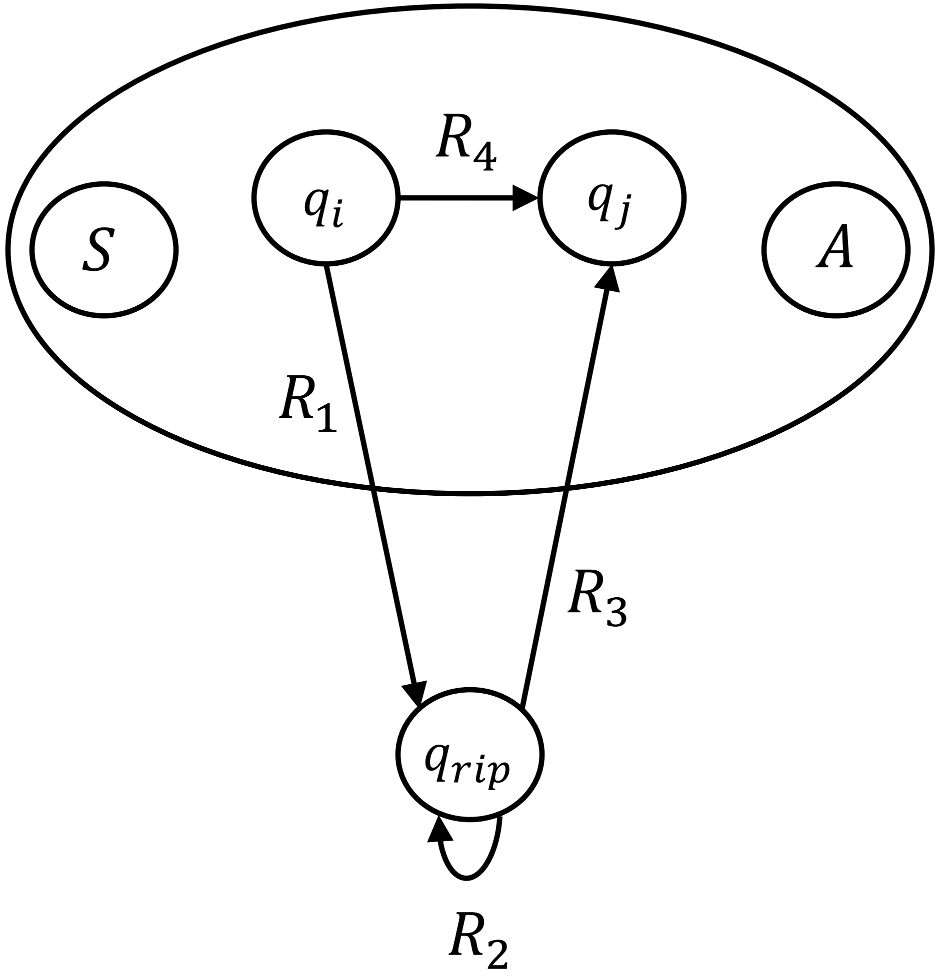
Eliminate states one-at-a-time:
updating labels at each step

HOW EXACTLY?

When only two states (start and accept) are left:
the only label is the RE we seek!

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

Removing a state of the GNFA



Ignore the rest of the states and transitions

To remove state q_{rip}

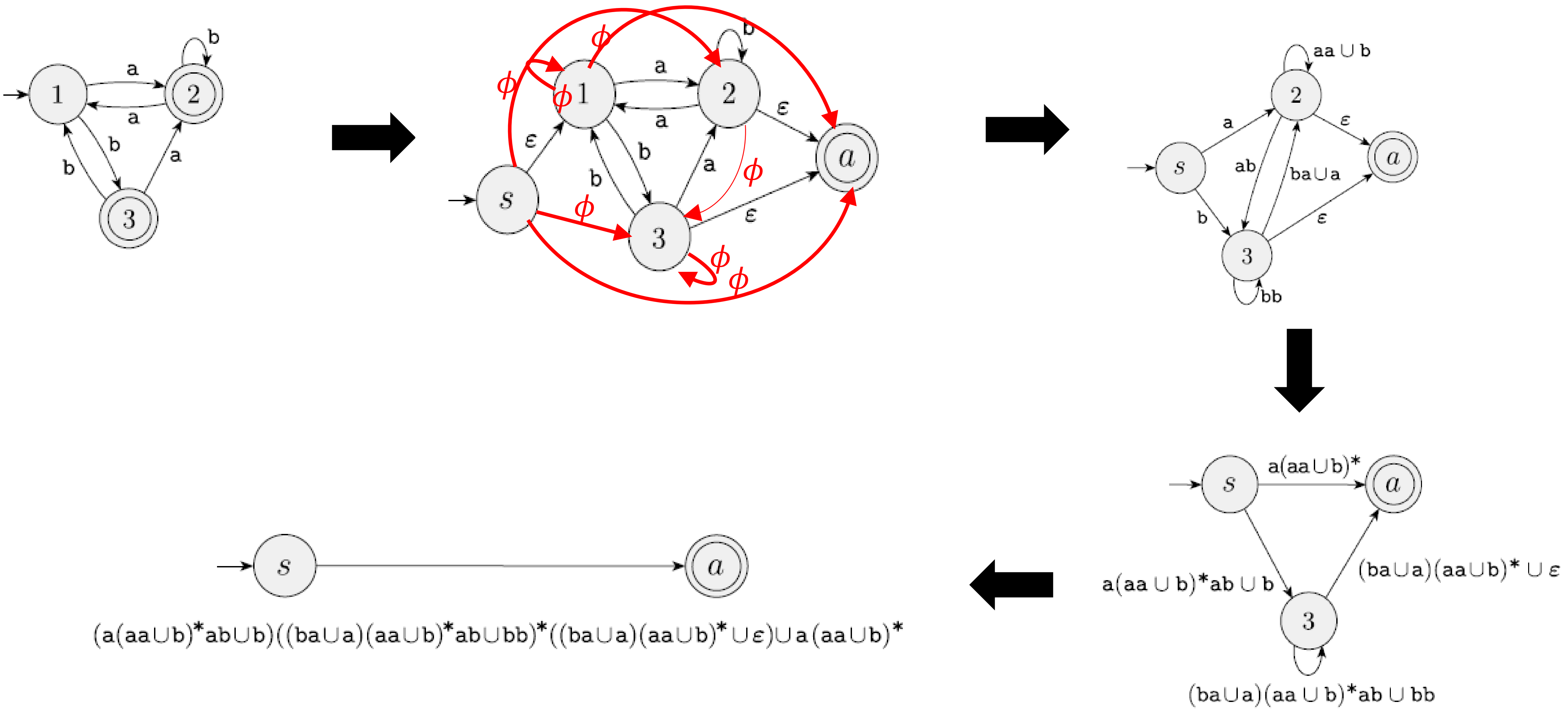
For every pair (q_i, q_j):

replace R_4 by $R_4 \cup R_1 R_2^* R_3$

consider all ways – old and using the bypass to go from q_i to q_j

Must also separately consider (q_j, q_i)

The Full Conversion



Formal Definitions

$G = (Q, \Sigma, \delta, q_{start}, q_{accept})$

$\delta: (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}$ (the set of REs over Σ)

G accepts input string $w \in \Sigma^*$ if

$w = w_1 w_2 \dots w_k$, $w_i \in \Sigma^*$ (the input can be divided into blocks)

and $\exists q_0, q_1, \dots, q_k$ (and there is a path)

1. $q_0 = q_{start}$ (from the start state)
2. $q_k = q_{accept}$ (and ending in the accept state)
3. $w_i \in L(R_i),\ R_i = \delta(q_{i-1}, q_i)$ (and each block matches the RE on the transition)

The Conversion Algorithm

CONVERT(G): $\quad (G = (Q, \Sigma, \delta, q_{start}, q_{accept})$ is the GNFA we start with)

1. Let k be the number of states of G.
2. If $k = 2$: return the label on transition (q_{start}, q_{accept})
3. If $k > 2$: select state $q_{rip} \in Q - \{q_{start}, q_{accept}\}$

   Define $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$

   where $Q' = Q - \{q_{rip}\}$

   and $\forall q_i \in Q - \{q_{accept}\}, \forall q_j \in Q - \{q_{start}\}$:

   if $R_1 = \delta(q_i, q_{rip}), R_2 = \delta(q_{rip}, q_{rip}), R_3 = \delta(q_{rip}, q_j), R_4 = \delta(q_i, q_j)$

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

   New label contains the previous label.
4. Return CONVERT(G')

G and CONVERT(G) are Equivalent

CLAIM: The RE CONVERT(G) describes $L(G)$.

Proof Sketch: By Induction on the number of states.

Base Case: $\boldsymbol{k = 2}$ All strings accepted state in G match the RE returned by CONVERT(G)

Inductive Hypothesis: Claim is true for all GNFAs with $k - 1$ states.

Inductive Step: (from $G\ to\ G'$):

if G accepts w:

Case 1: q_{rip} is not on an accepting path. The same path exists in G' because the new label in G' contains the old ones in G.

Case 2: q_{rip} appears on an accepting path in G: $\ldots q_i q_{rip} q_{rip} q_{rip} q_j \ldots \ldots q_l q_{rip} q_k \ldots \ldots$

G' contains $\ldots q_i q_j \ldots \ldots q_l q_k \ldots \ldots$ which accepts w (use the bypass)

If G' accepts w:

every transition in G' is either a transition in G or a path in G. So an accepting path in G' implies an accepting path in G.

So, $L(G) = L(G')$

By the Inductive Hypothesis the RE CONVERT(G') describes $L(G')$ and, therefore, $L(G)$.

The Story Thus Far

- DFA, NFA, Regular Expressions are all equivalent
  - They accept/describe the same set of languages
  - We can convert any one to any other
- Regular Expressions are useful in defining patterns
- NFAs are useful in designing automata
- DFAs are a useful concept for designing pattern matching algorithms (and a whole lot more).
- There are NFAs for which equivalent DFAs have exponentially more states.

What's Next?

Is every language regular?

If not, how can we prove that a language is not regular?