COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 8 : Regular and NON-Reg. Languages

GNFA → Reg. Expression

CLAIM 1.65

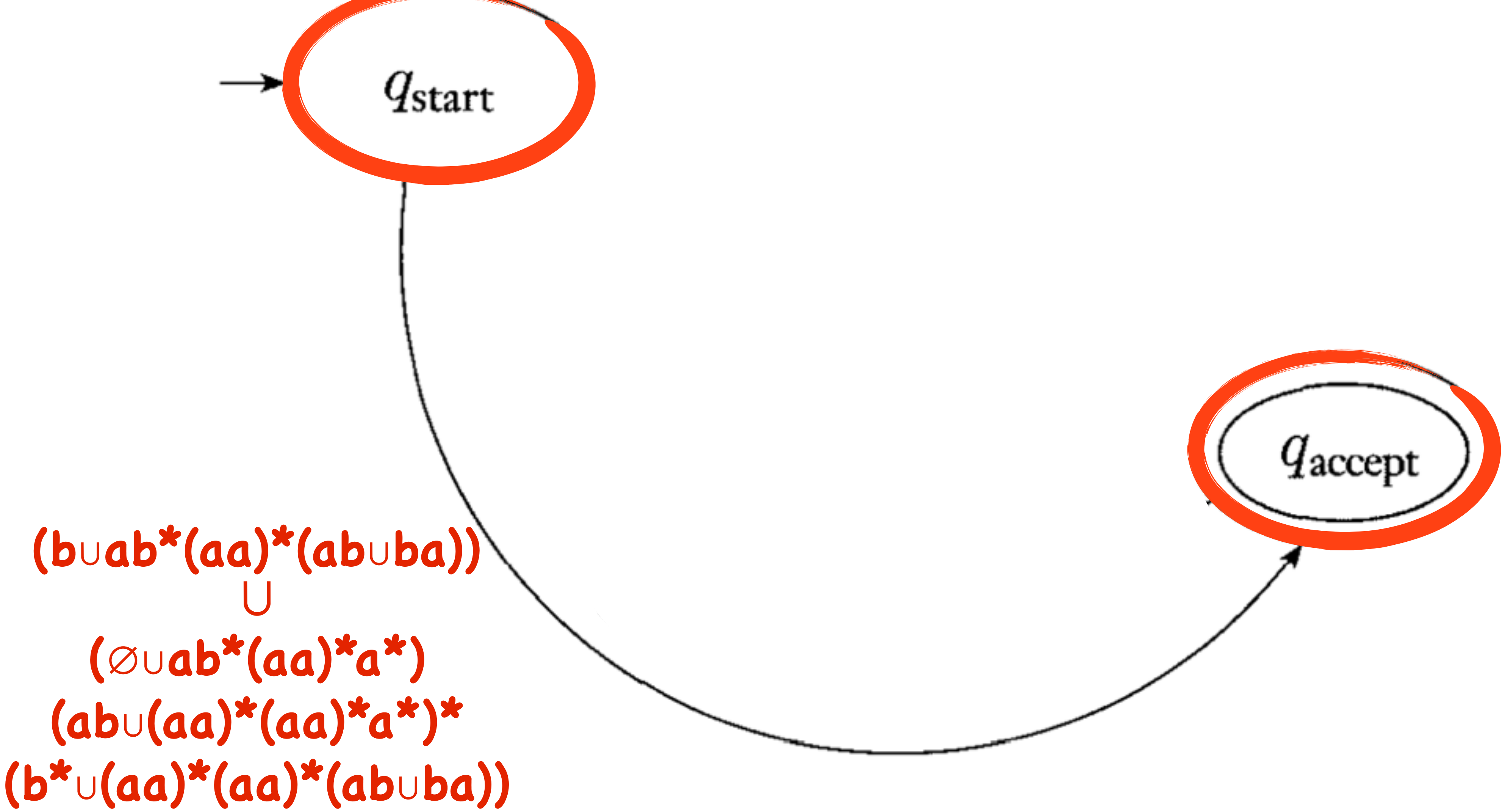
For any GNFA G, CONVERT(G) is equivalent to G.

We prove this claim by induction on k, the number of states of the GNFA.

"equivalent" means L(CONVERT(G)) = L(G)

GNFA → Reg. Expression

- Induction basis
- Let G be a GNFA with exactly k=2 states.
- Because of the special form of our GNFA, the two states are the start and accept states. The regular expression on the transition from q_{start} to q_{accept} generates the language accepted by this GNFA.



- Because of the special form of our GNFA, the two states are the start and accept states. The regular expression on the transition from q_{start} to q_{accept} generates the language accepted by this GNFA.

GNFA $\rightarrow$ Reg. Expression

- Induction basis
- Let G be a GNFA with exactly k=2 states.
- Because of the special form of our GNFA, the two states are the start and accept states. The regular expression on the transition from q_{start} to q_{accept} generates the language accepted by this GNFA.

GNFA $\rightarrow$ Reg. Expression

GNFA → Reg. Expression

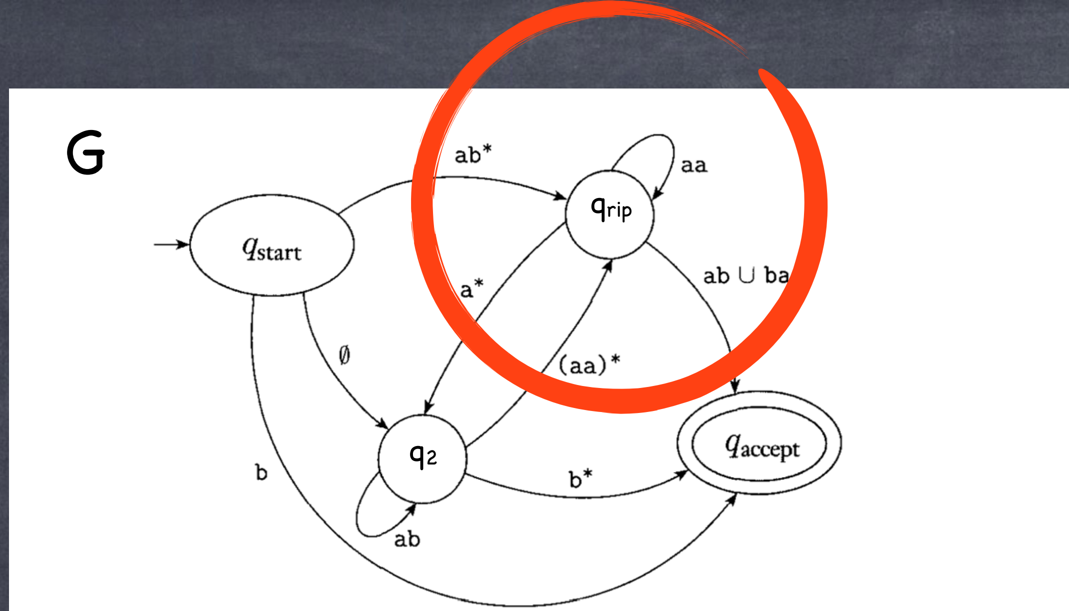
- <u>Induction step</u>

GNFA → Reg. Expression

- Induction step
- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).

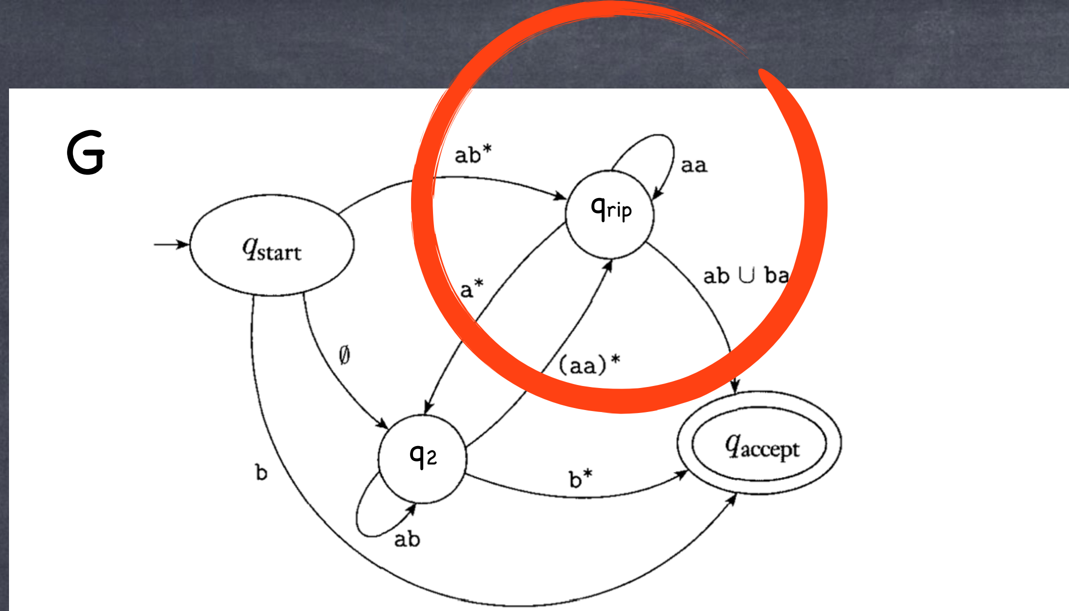
GNFA → Reg. Expression

- <u>Induction step</u>
- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).
- Since k>2 then there exists at least one state q_{rip} which is neither q_{start} nor q_{accept}.



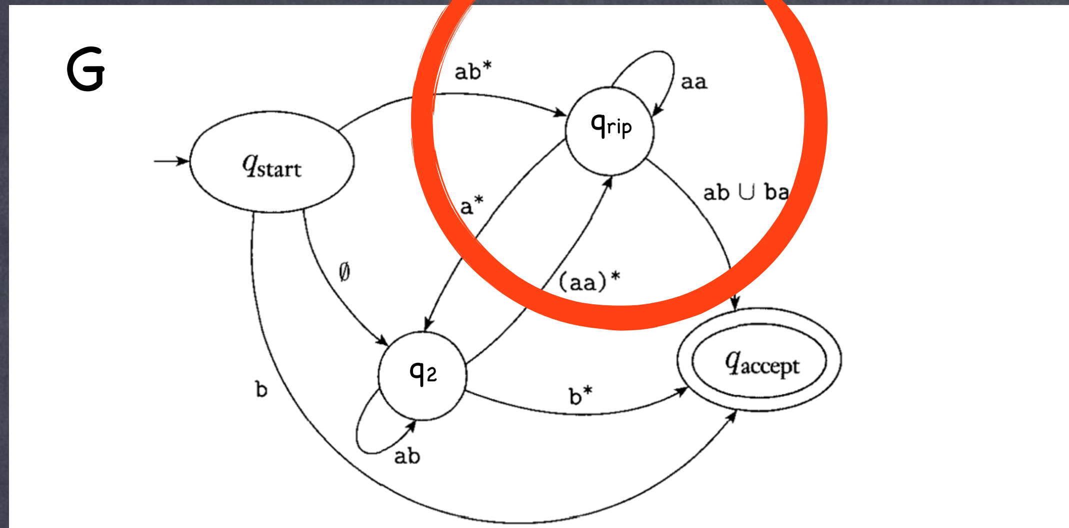
g. Expression

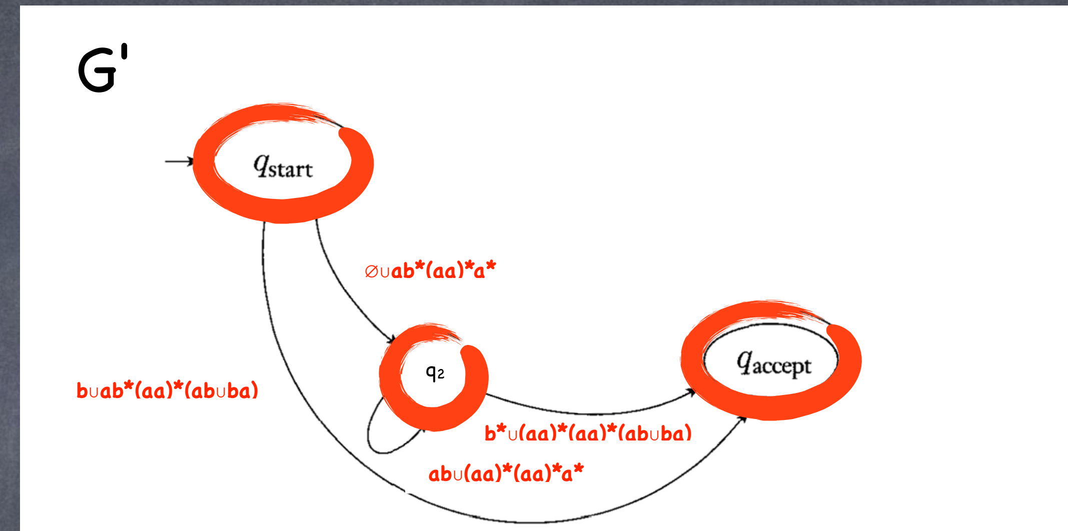
- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).
- Since k>2 then there exists at least one state q_{rip} which is neither q_{start} nor q_{accept}.



g. Expression

- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).
- Since k>2 then there exists at least one state q_{rip} which is neither q_{start} nor q_{accept}.
- Let G′ be, as in CONVERT, the GNFA obtained after ripping q_{rip} from G.



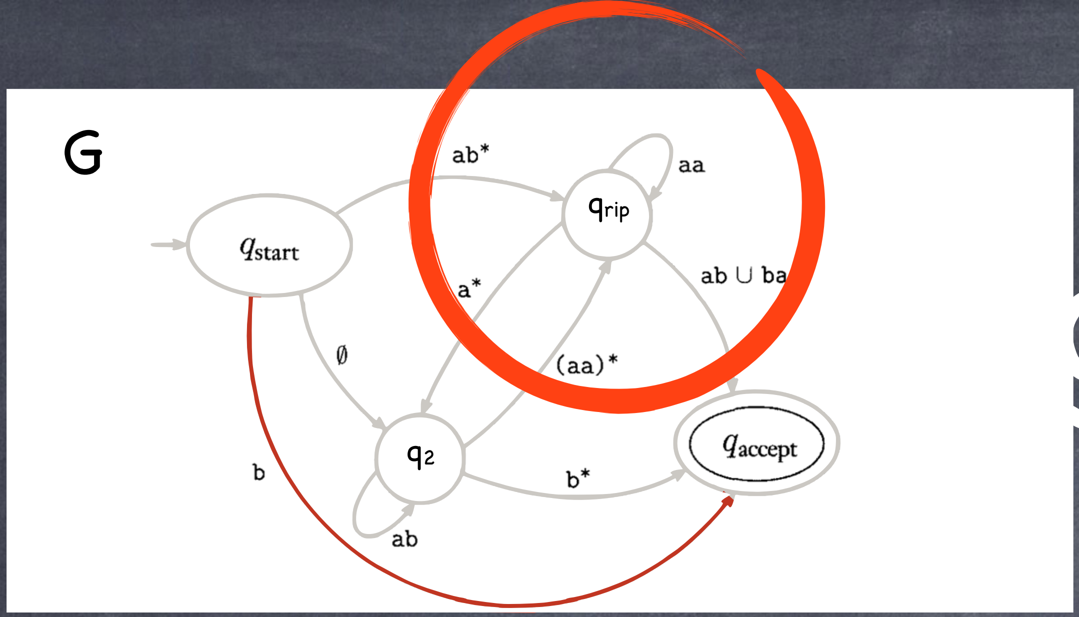


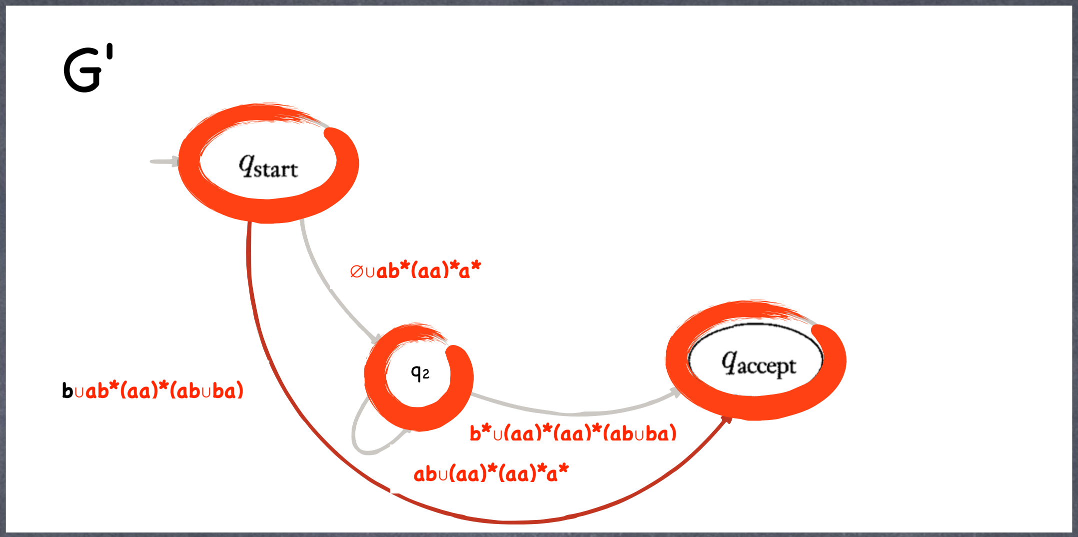
- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).
- Since k>2 then there exists at least one state q_{rip} which is neither q_{start} nor q_{accept}.
- Let G′ be, as in CONVERT, the GNFA obtained after ripping q_{rip} from G.

GNFA $\rightarrow$ Reg. Expression

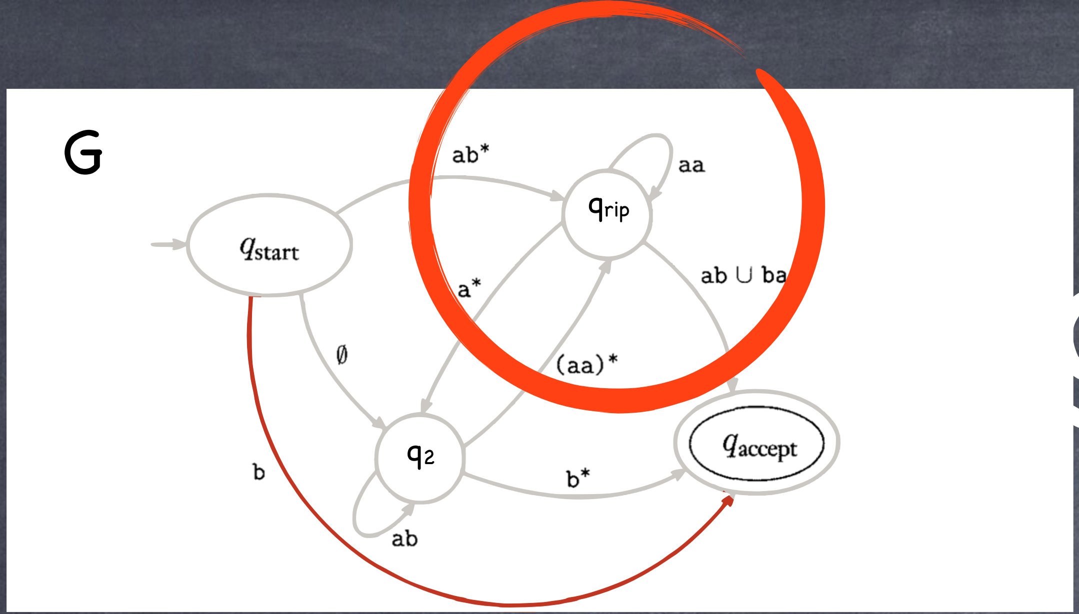
GNFA → Reg. Expression

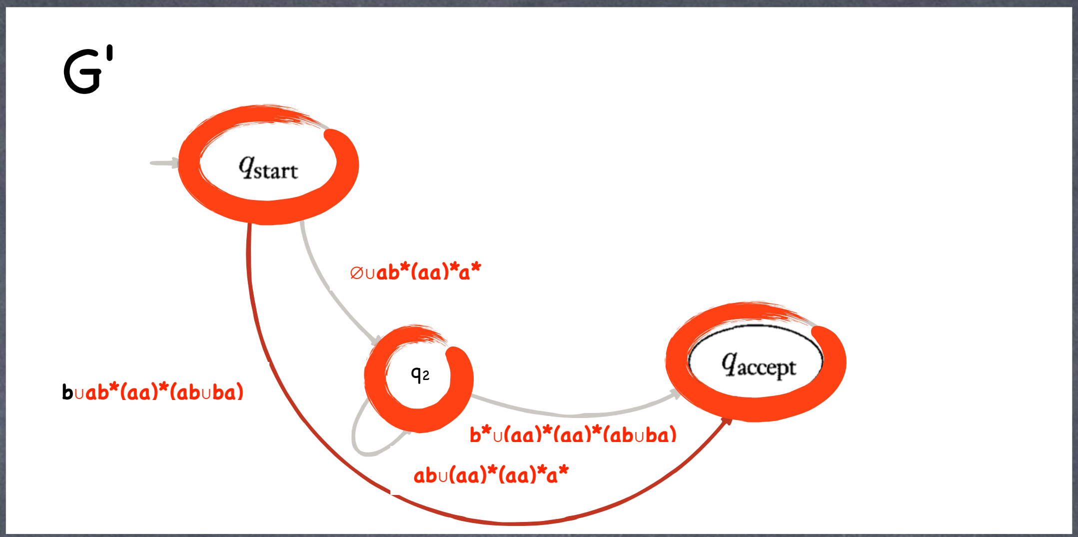
- Let w be a string accepted by G, $w \in L(G)$. Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w.





- Let w be a string accepted by G, w∈L(G). Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w.





- Let w be a string accepted by G, $w \in L(G)$. Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w.

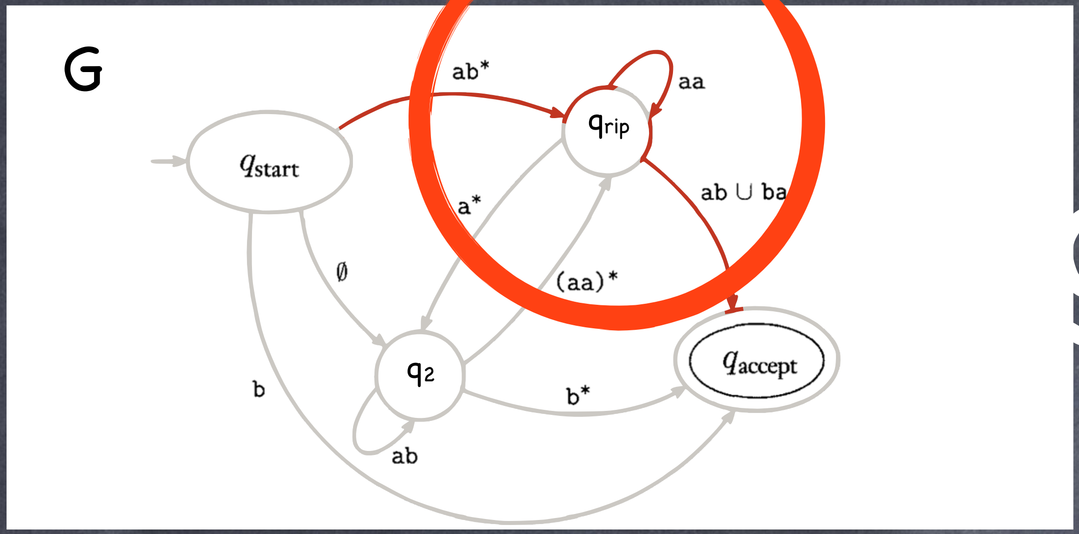
- If q_{rip} is not a state of the sequence, then the very same exact sequence will accept w in G′ because its transitions R_4 contain all those R_4 in G (except for q_{rip}) in a union with new possibilities related to ripping q_{rip}.

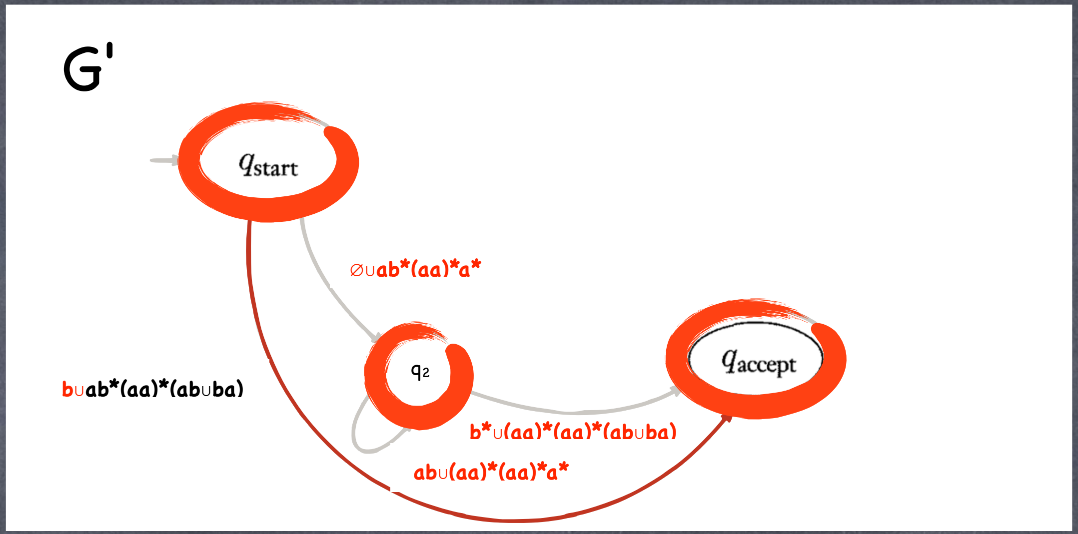
GNFA $\rightarrow$ Reg. Expression

- Let w be a string accepted by G, $w \in L(G)$. Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w.
- If q_{rip} is not a state of the sequence, then the very same exact sequence will accept w in G′ because its transitions R_4 contain all those R_4 in G (except for q_{rip}) in a union with new possibilities related to ripping q_{rip}.

GNFA → Reg. Expression

- If q_{rip} is a state of the sequence, then the same sequence (but with all q_{rip} removed) will accept w in G′. That's because any three elements in a row q_i,q_{rip},q_j ($q_i \neq q_{rip} \neq q_j$) in G's accepting sequence, will be processed identically through states q_i,q_j in G′. Remember that the transitions for q_i,q_j in G′ contain all those $R_1(R_2)^*R_3$ from G involving q_{rip} in a union with older possibilities (R_4). (we can deal with $q_i,q_{rip},\ldots,q_{rip},q_j$ similarly.)





- If q_{rip} is a state of the sequence, then the same sequence (but with all q_{rip} removed) will accept w in G′. That's because any three elements in a row q_i, q_{rip}, q_j ($q_i \neq q_{rip} \neq q_j$) in G's accepting sequence, will be processed identically through states q_i, q_j in G′. Remember that the transitions for q_i, q_j in G′ contain all those $R_1(R_2)^*R_3$ from G involving q_{rip} in a union with older possibilities (R_4). (we can deal with $q_i, q_{rip}, \ldots, q_{rip}, q_j$ similarly.)

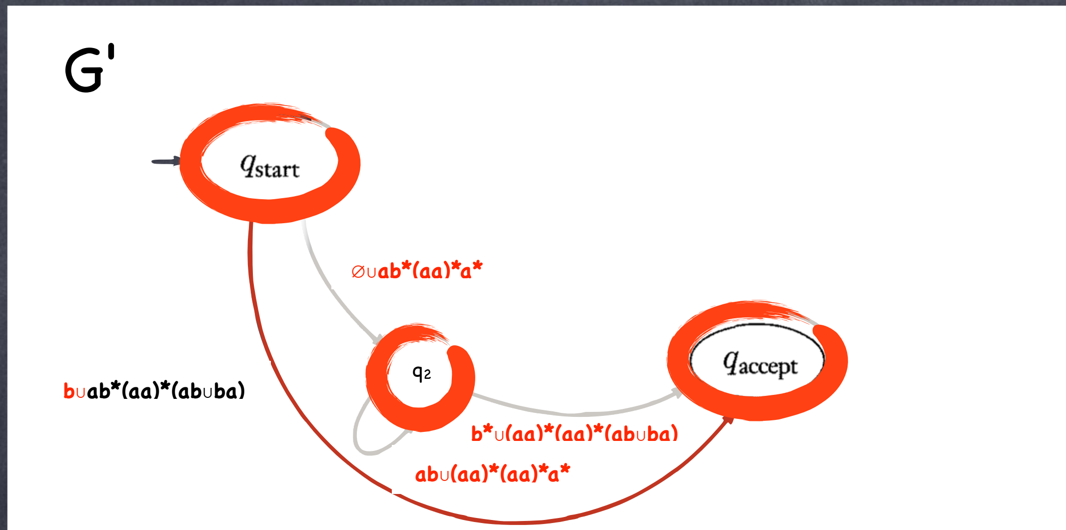
GNFA $\rightarrow$ Reg. Expression

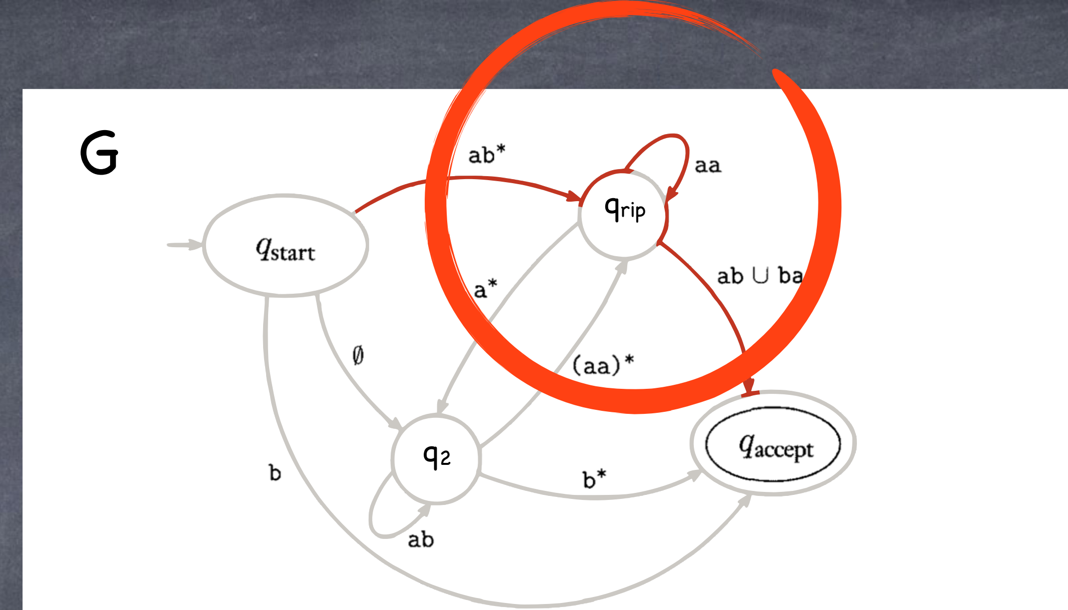
- If q_{rip} is a state of the sequence, then the same sequence (but with all q_{rip} removed) will accept w in G′. That's because any three elements in a row q_i, q_{rip}, q_j ($q_i \neq q_{rip} \neq q_j$) in G's accepting sequence, will be processed identically through states q_i, q_j in G′. Remember that the transitions for q_i, q_j in G′ contain all those $R_1(R_2)^*R_3$ from G involving q_{rip} in a union with older possibilities (R_4). (we can deal with $q_i, q_{rip}, \ldots, q_{rip}, q_j$ similarly.)

GNFA → Reg. Expression

- This proved "if $w \in L(G)$ then $w \in L(G')$". We should also prove "if $w \in L(G')$ then $w \in L(G)$".
- Let w be a string accepted by G', i.e. $w \in L(G')$. Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w. Consider any two consecutive states q_i, q_{i+1}. The same portion of w is processed in G in either part of the union, $R_1(R_2)^*R_3$ or R_4, along the transition between q_i and q_{i+1}.

eg.





- This proved "if w∈L(G) then w∈L(G′)". We should also prove "if w∈L(G′) then w∈L(G)".

- Let w be a string accepted by G′, i.e. w∈L(G′). Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w. Consider any two consecutive states q_i, q_{i+1}. The same portion of w is processed in G in either part of the union, $R_1(R_2)^*R_3$ or R_4, along the transition between q_i and q_{i+1}.

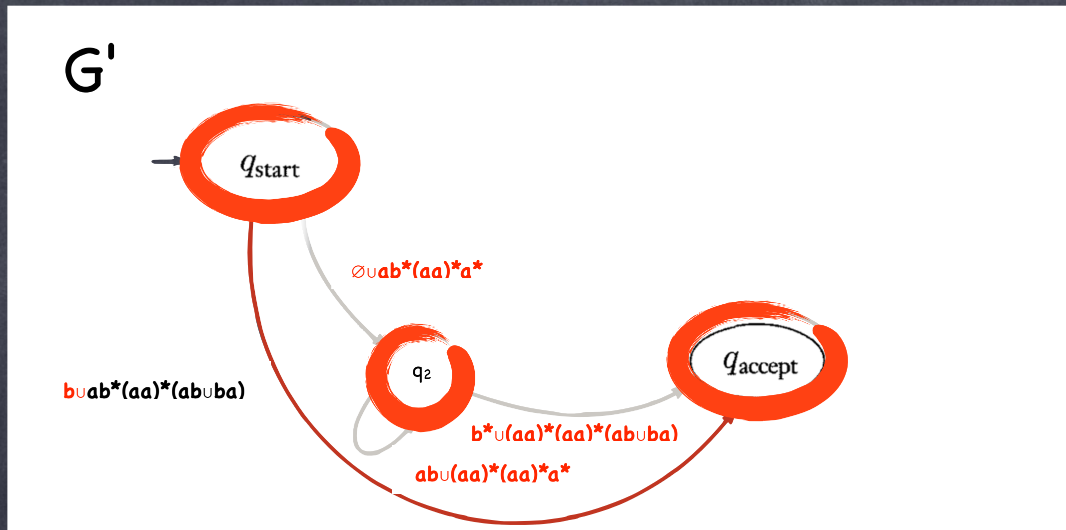
GNFA $\rightarrow$ Reg. Expression

- This proved "if $w \in L(G)$ then $w \in L(G')$". We should also prove "if $w \in L(G')$ then $w \in L(G)$".
- Let w be a string accepted by G', i.e. $w \in L(G')$. Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w. Consider any two consecutive states q_i, q_{i+1}. The same portion of w is processed in G in either part of the union, $R_1(R_2)^*R_3$ or R_4, along the transition between q_i and q_{i+1}.

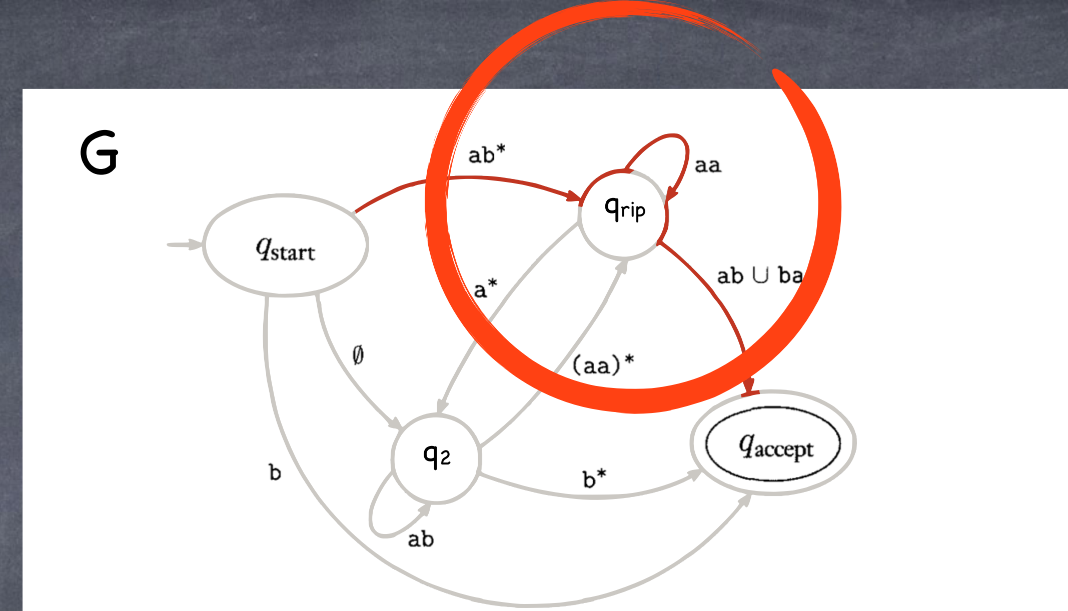
GNFA $\rightarrow$ Reg. Expression

- If the portion of w is generated by R_4 in G′ then it is also generated by R_4 in G. If the portion of w is generated by $R_1(R_2)^*R_3$ in G′ then there exists an m such that it is generated by $R_1(R_2)^mR_3$ and it is also generated in G by R_1, going through q_{rip} m times via R_2 and finally R_3. Thus q_i,q_{i+1} is replaced by $q_i,q_{rip},\ldots,q_{rip},q_{i+1}$.

- We conclude that if $w \in L(G')$ then $w \in L(G)$.



eg.



- If the portion of w is generated by R_4 in G′ then it is also generated by R_4 in G. If the portion of w is generated by $R_1(R_2)^*R_3$ in G′ then there exists an m such that it is generated by $R_1(R_2)^mR_3$ and it is also generated in G by R_1, going through q_{rip} m times via R_2 and finally R_3. Thus q_i,q_{i+1} is replaced by $q_i,q_{rip},\ldots,q_{rip},q_{i+1}$.

- We conclude that if $w \in L(G')$ then $w \in L(G)$.

GNFA → Reg. Expression

- If the portion of w is generated by R_4 in G′ then it is also generated by R_4 in G. If the portion of w is generated by $R_1(R_2)^*R_3$ in G′ then there exists an m such that it is generated by $R_1(R_2)^mR_3$ and it is also generated in G by R_1, going through q_{rip} m times via R_2 and finally R_3. Thus q_i,q_{i+1} is replaced by $q_i,q_{rip},\ldots,q_{rip},q_{i+1}$.
- We conclude that if $w \in L(G')$ then $w \in L(G)$.

GNFA → Reg. Expression

- Combining both statements we get L(G′)=L(G).
- By induction hypothesis L(G′)=L(CONVERT(G′)) because G′ contains k-1 states. By construction, CONVERT(G)=CONVERT(G′). Therefore

  L(G)=L(CONVERT(G))=L(CONVERT(G′))=L(G′).

QED

DFA → GNFA → Reg. Exp.

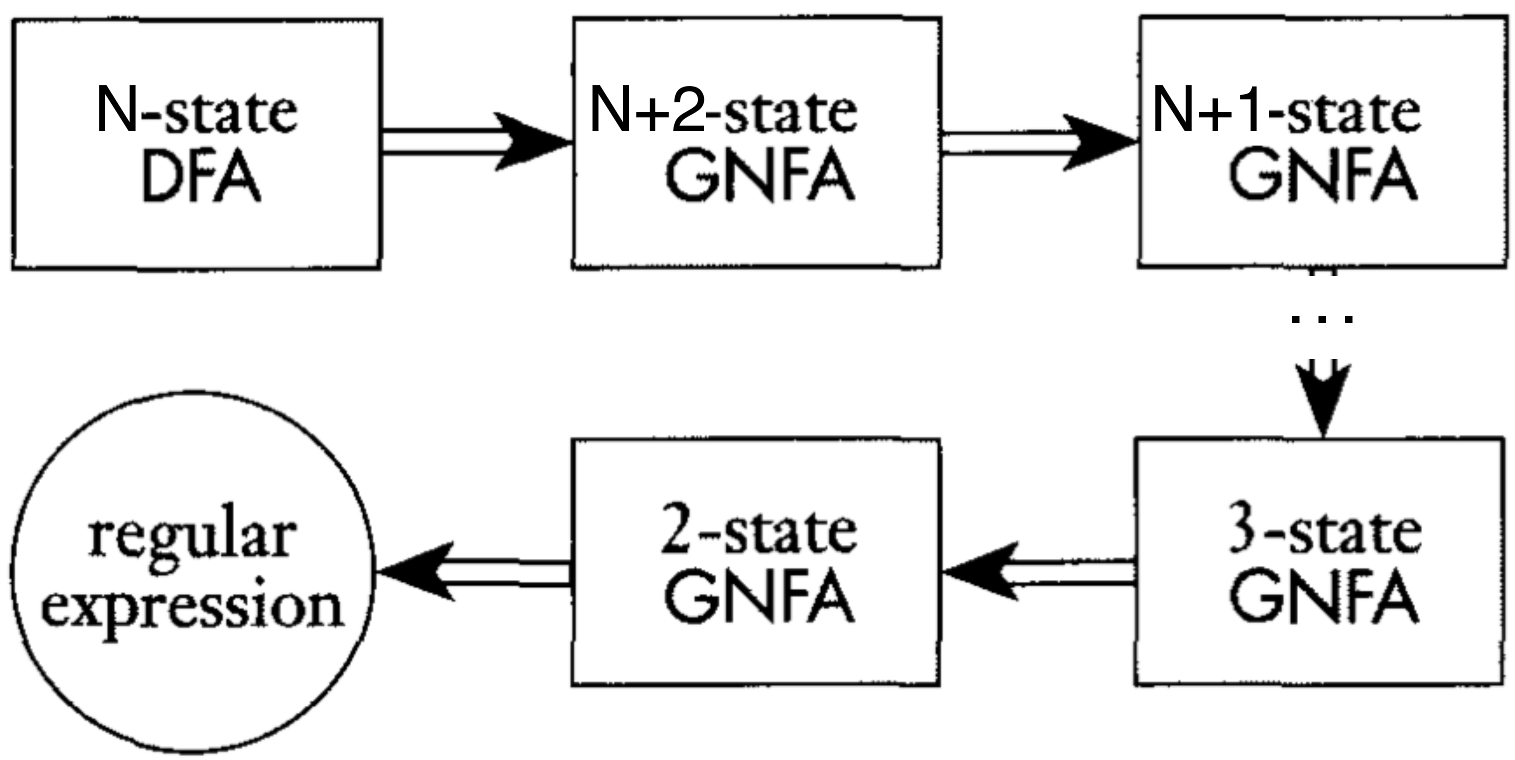
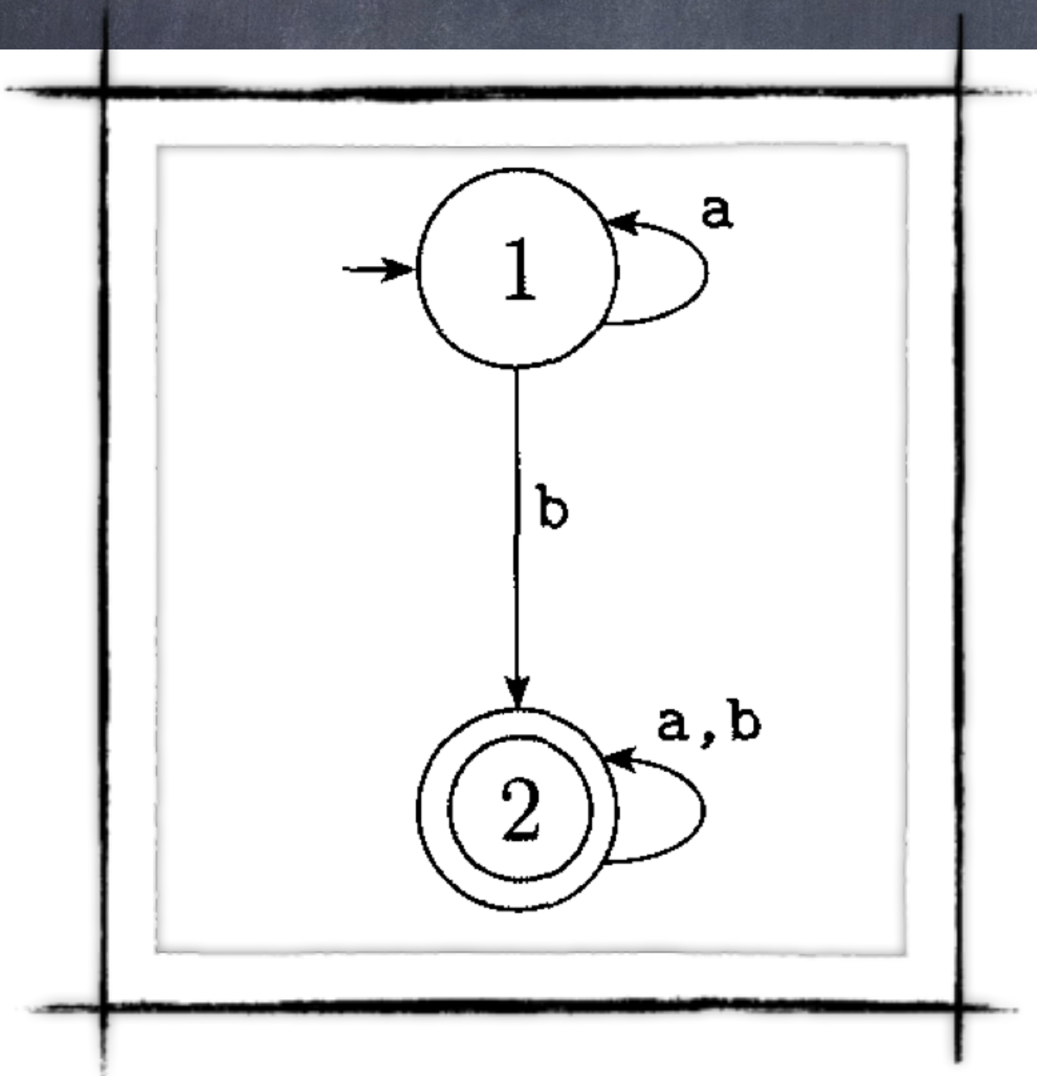
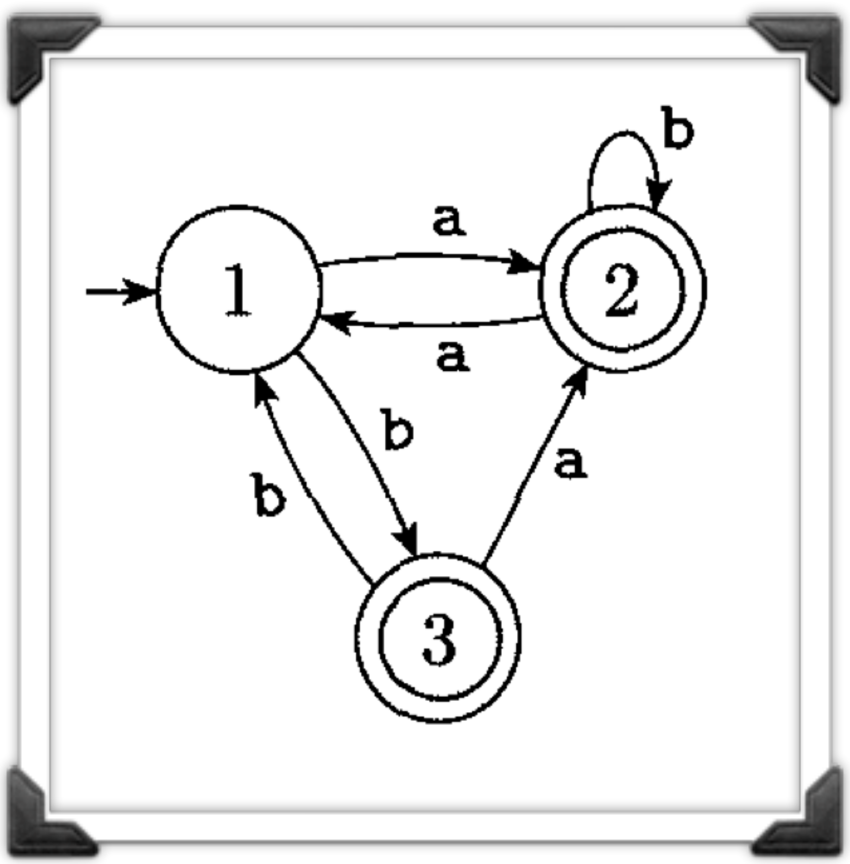


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

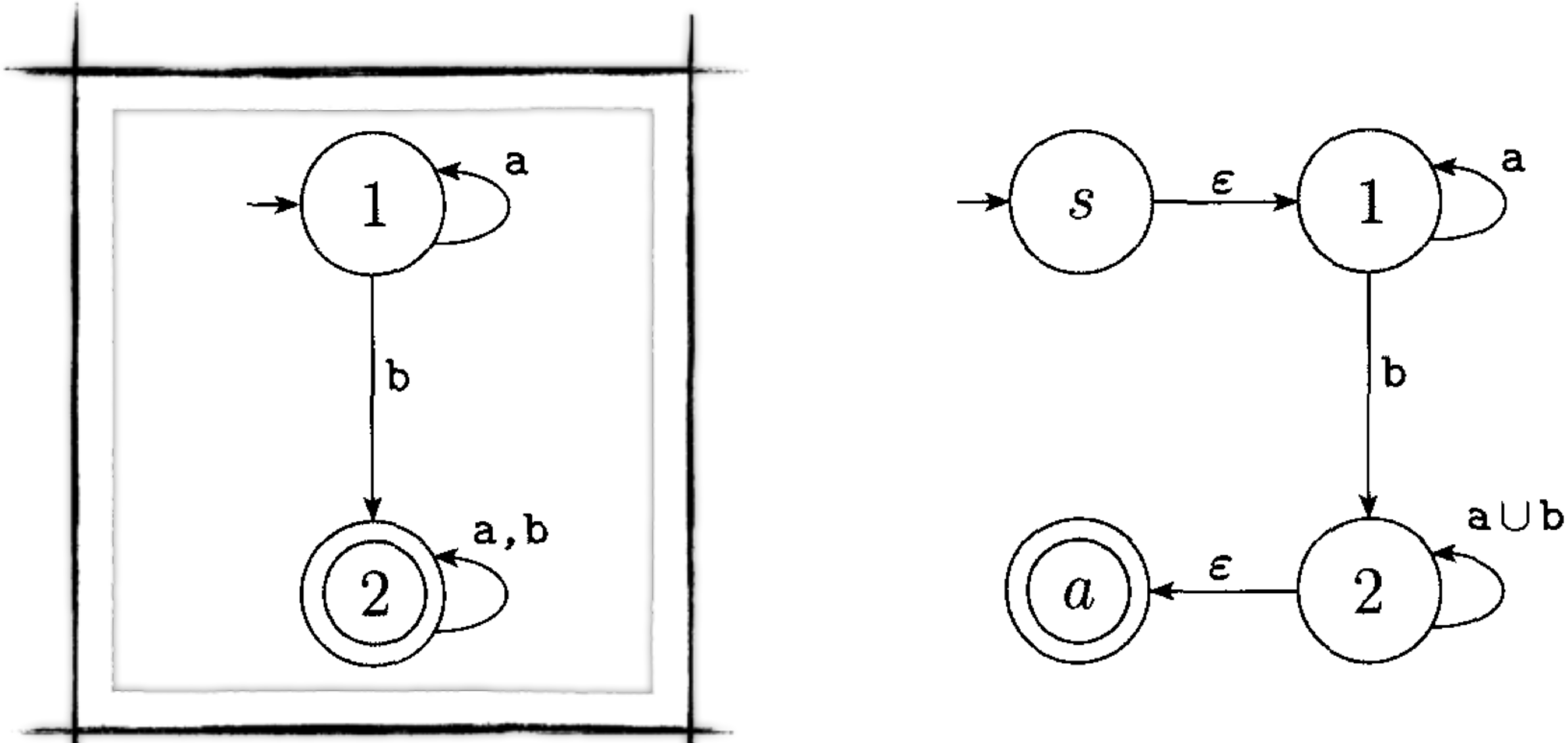
DFA → GNFA → Reg. Exp.

- Two examples



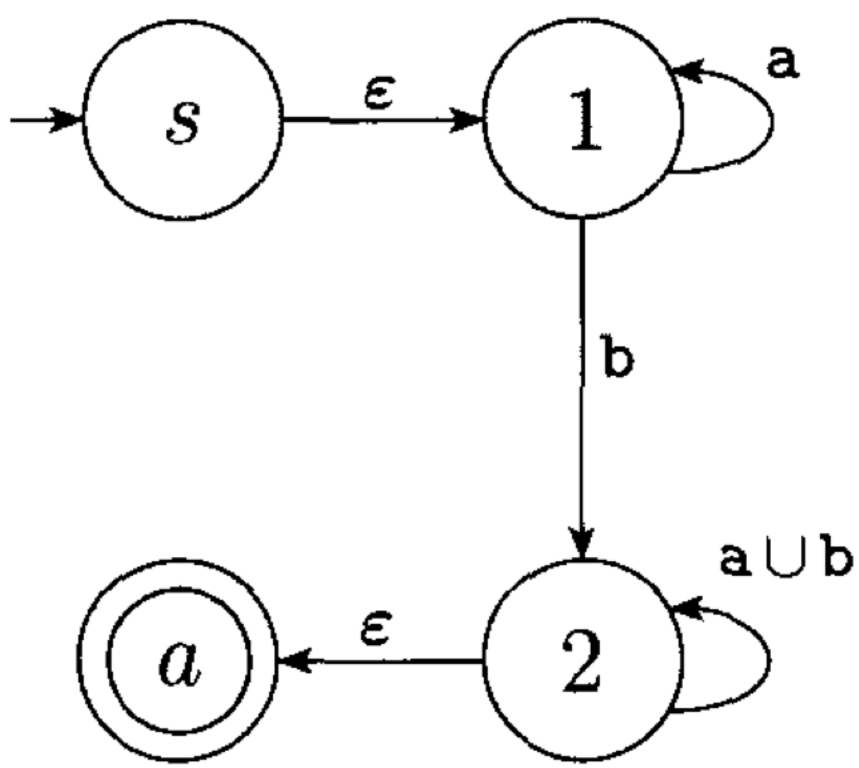


DFA → GNFA → Reg. Exp.

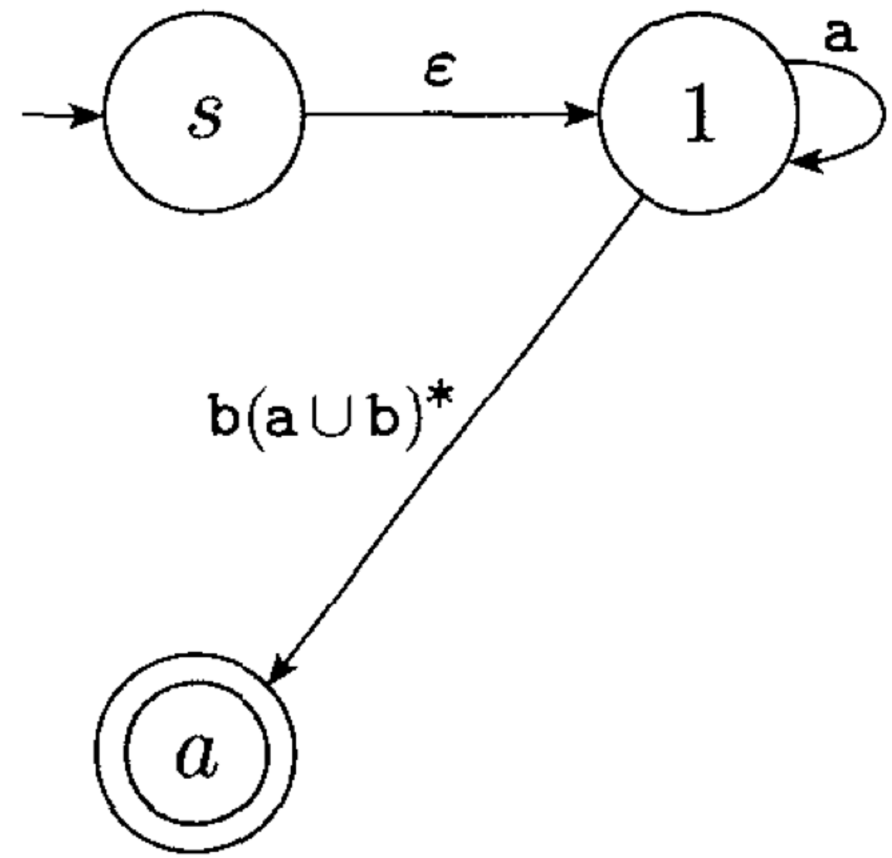


(a) (b)

DFA → GNFA → Reg. Exp.

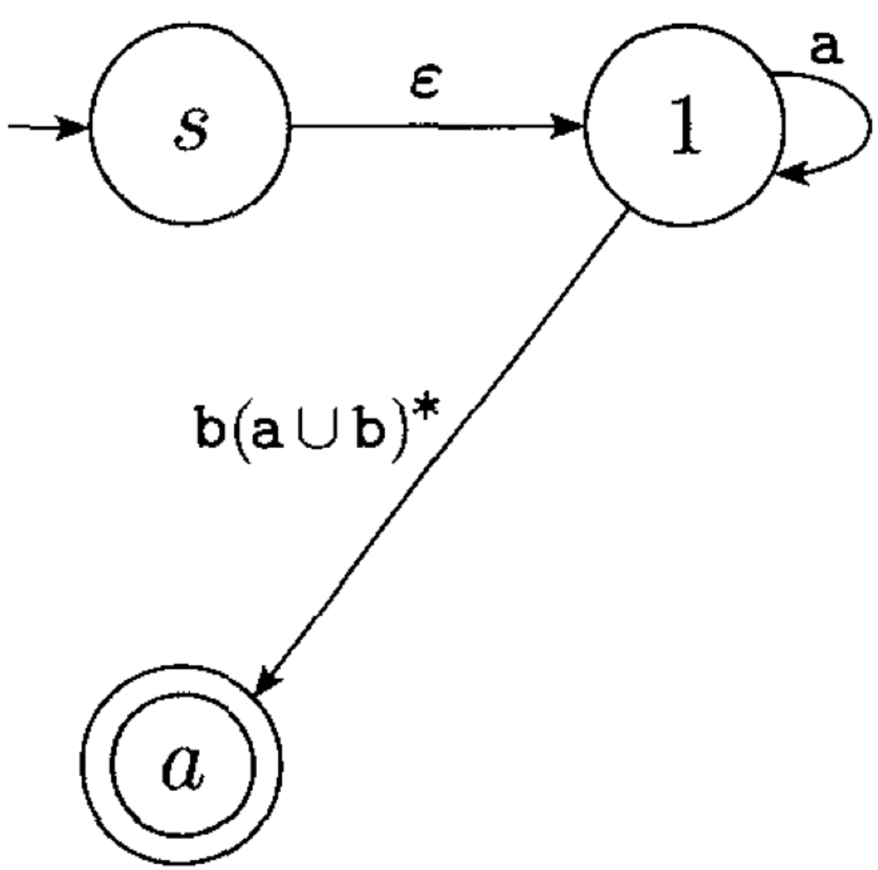


(b)

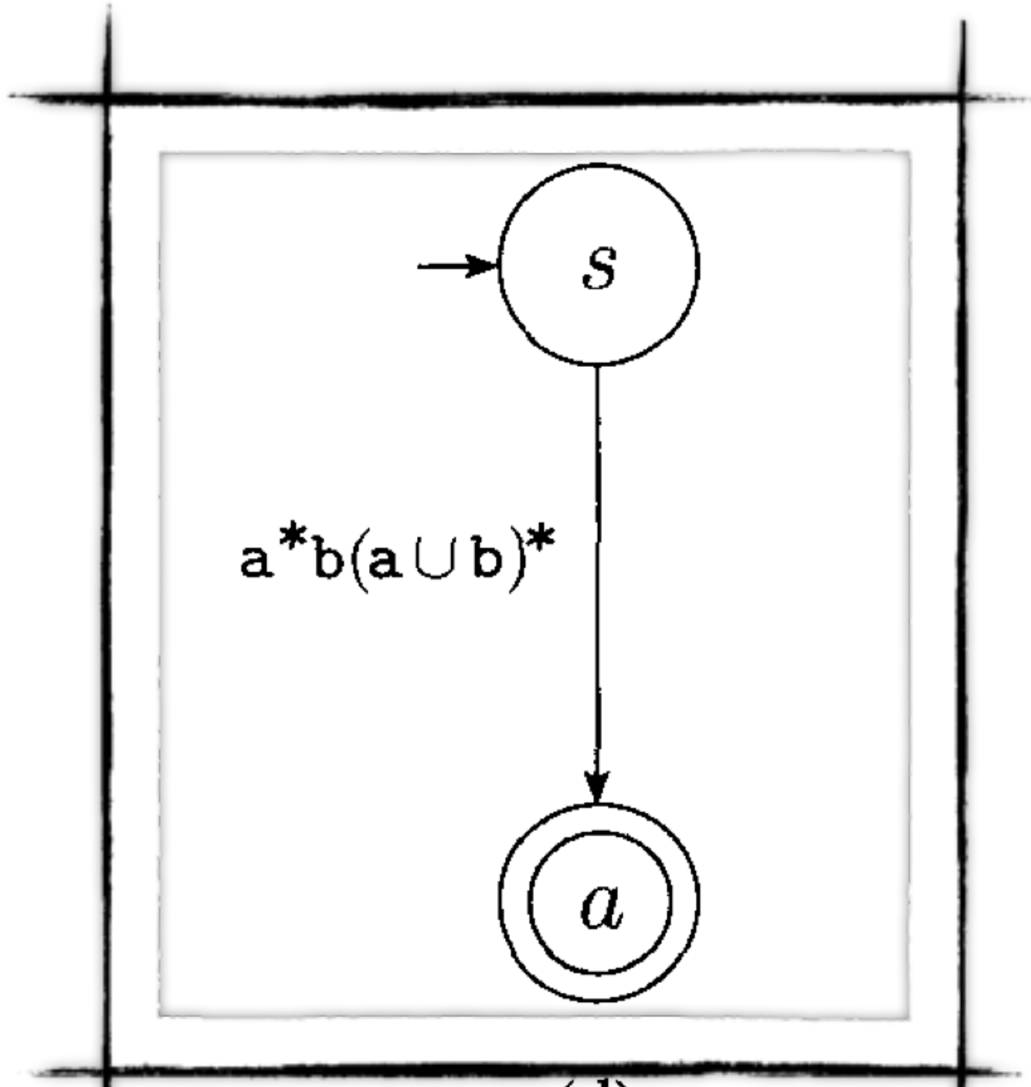


(c)

DFA → GNFA → Reg. Exp.

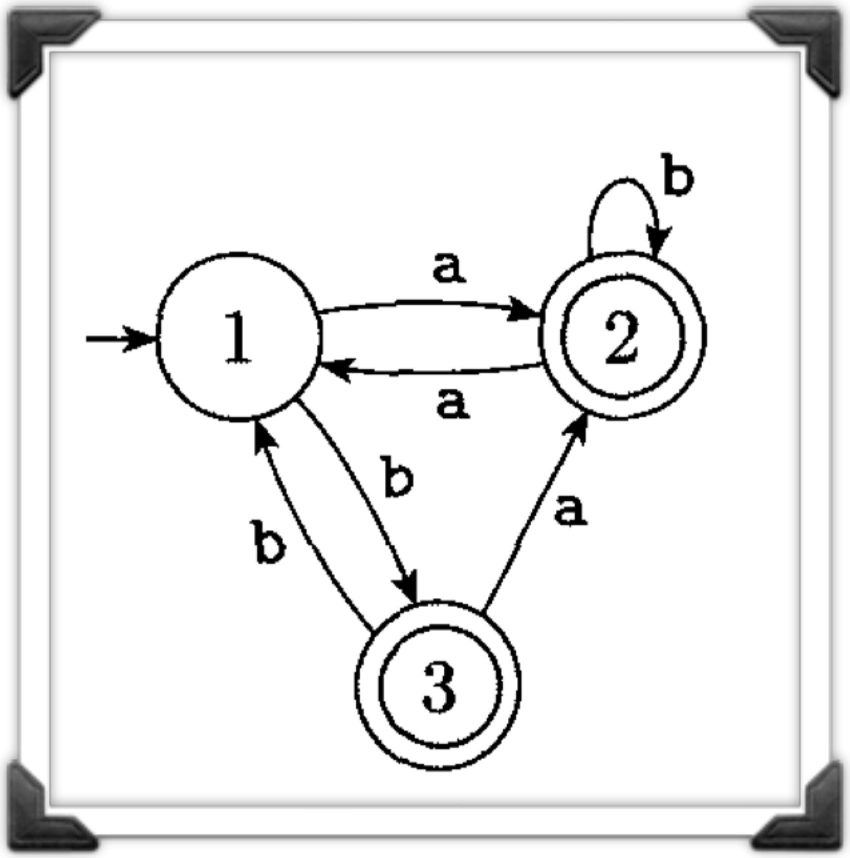


(c)

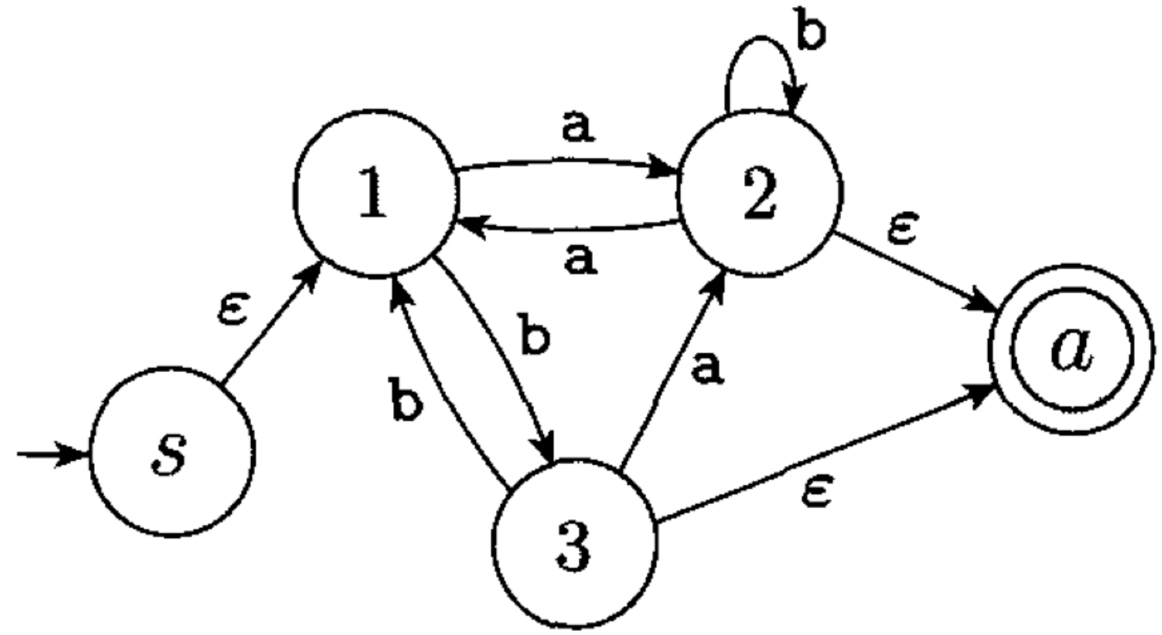


(d)

DFA → GNFA → Reg. Exp.

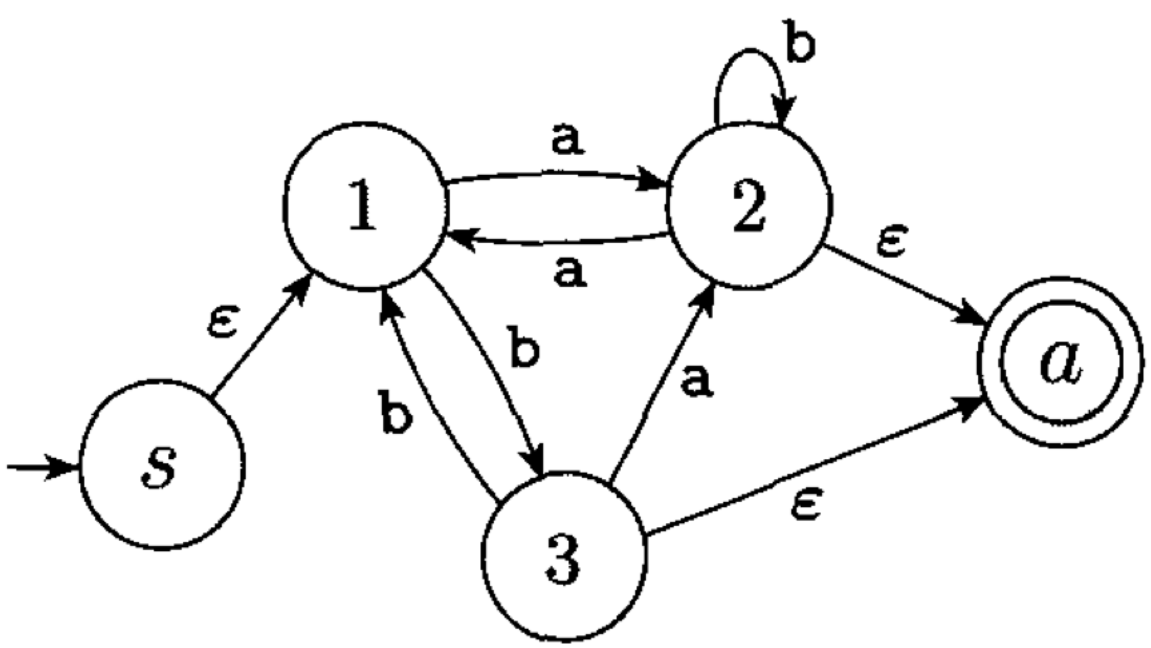


(a)

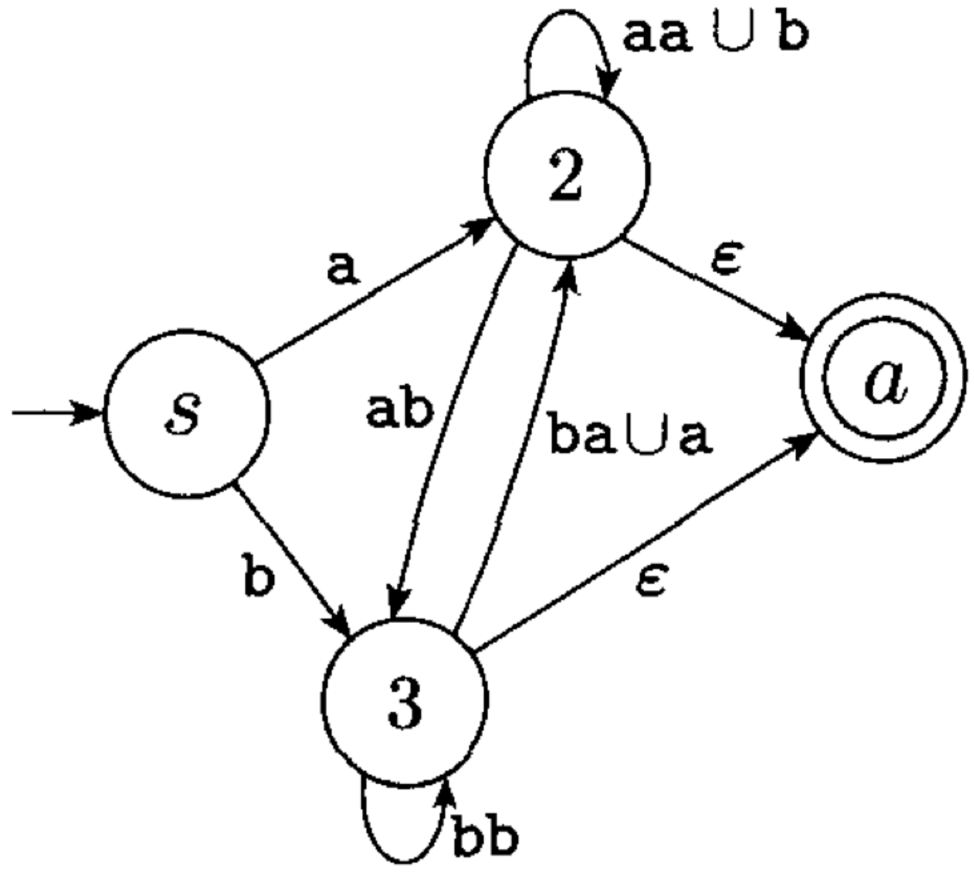


(b)

DFA → GNFA → Reg. Exp.

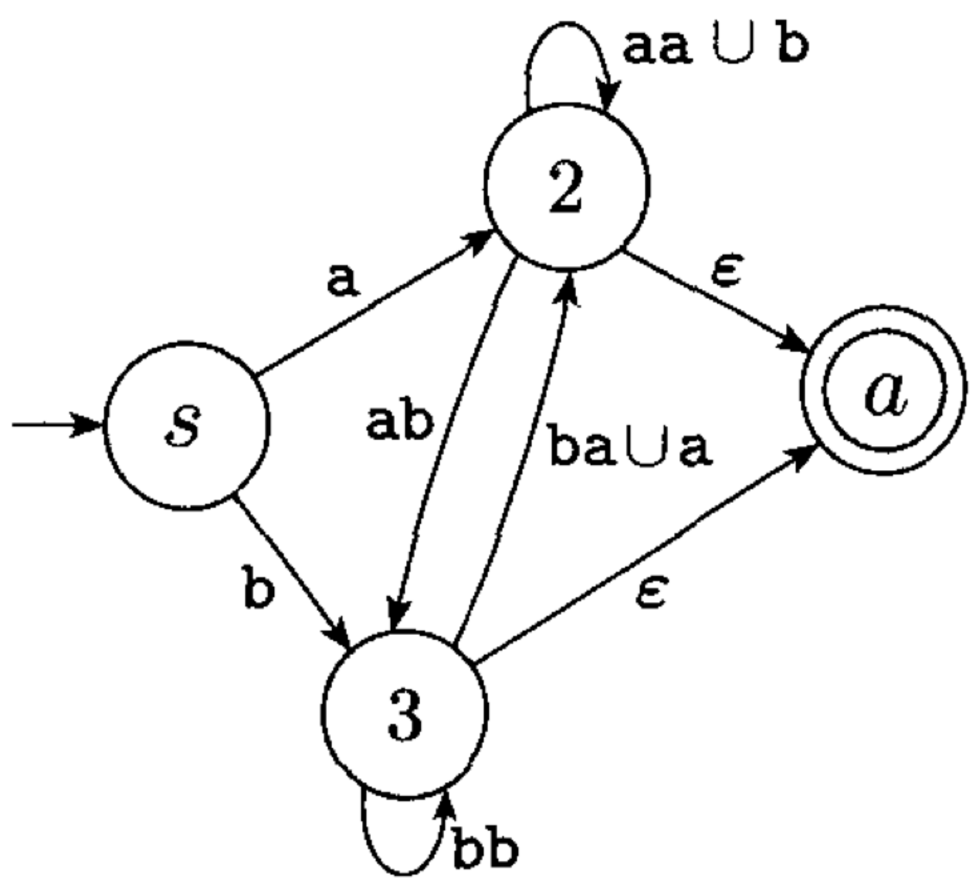


(b)

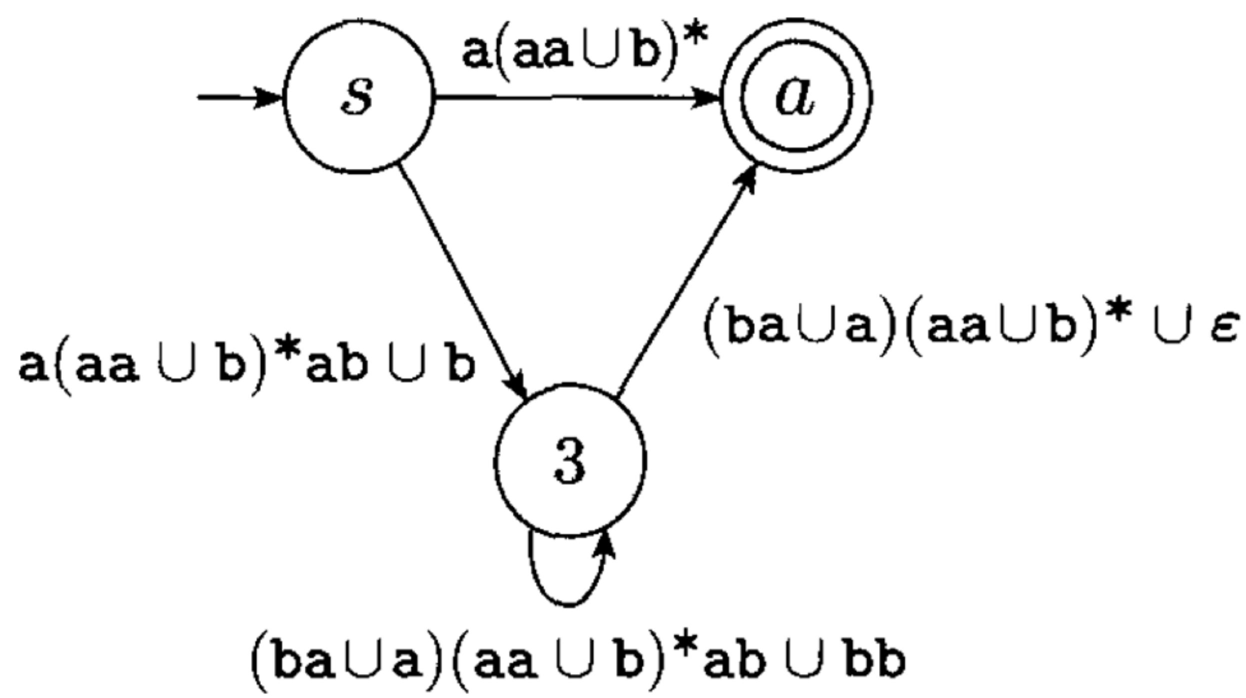


(c)

DFA → GNFA → Reg. Exp.

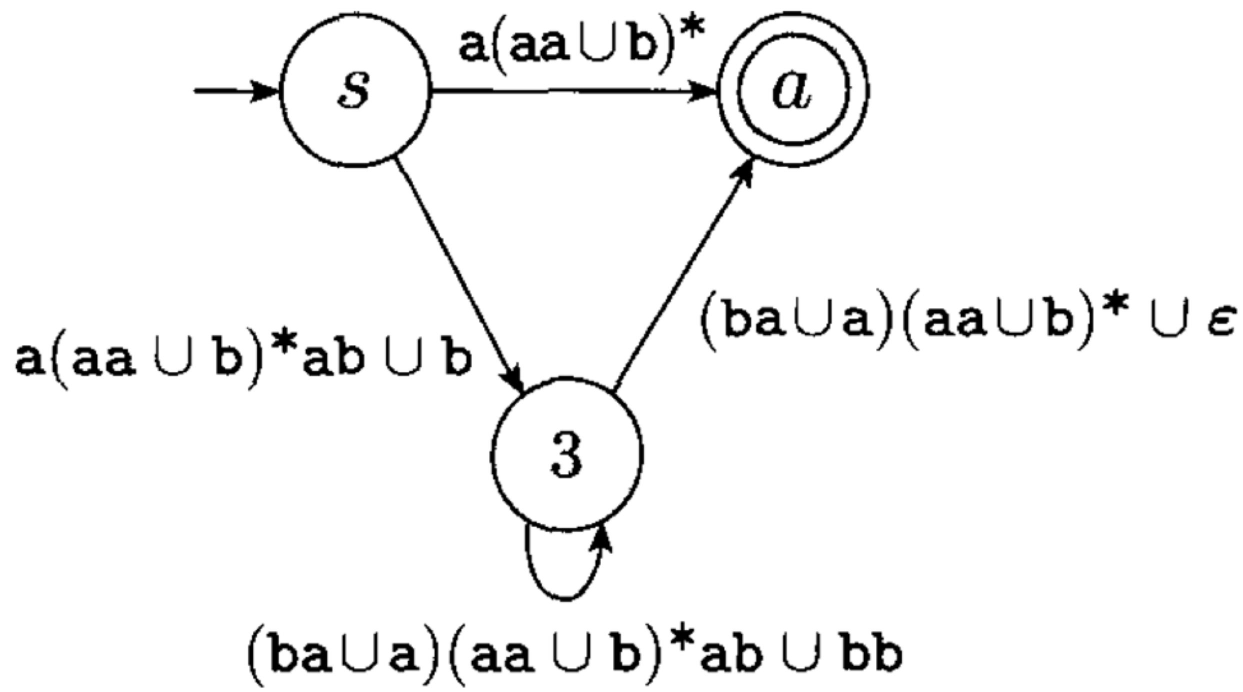


(c)

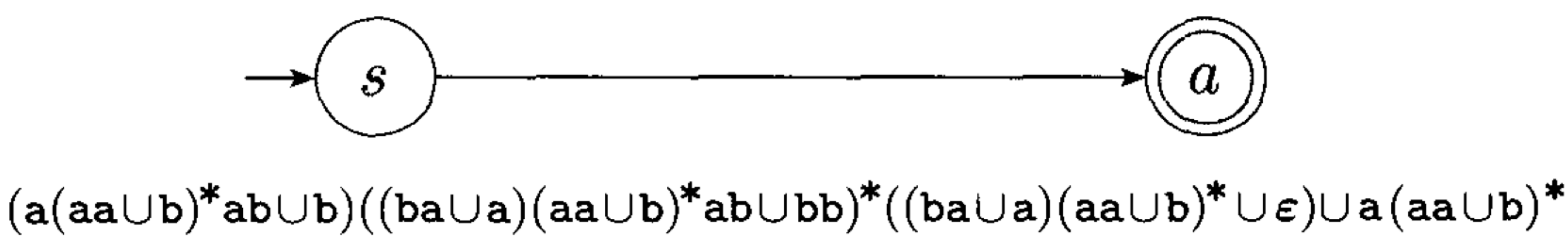


(d)

DFA → GNFA → Reg. Exp.

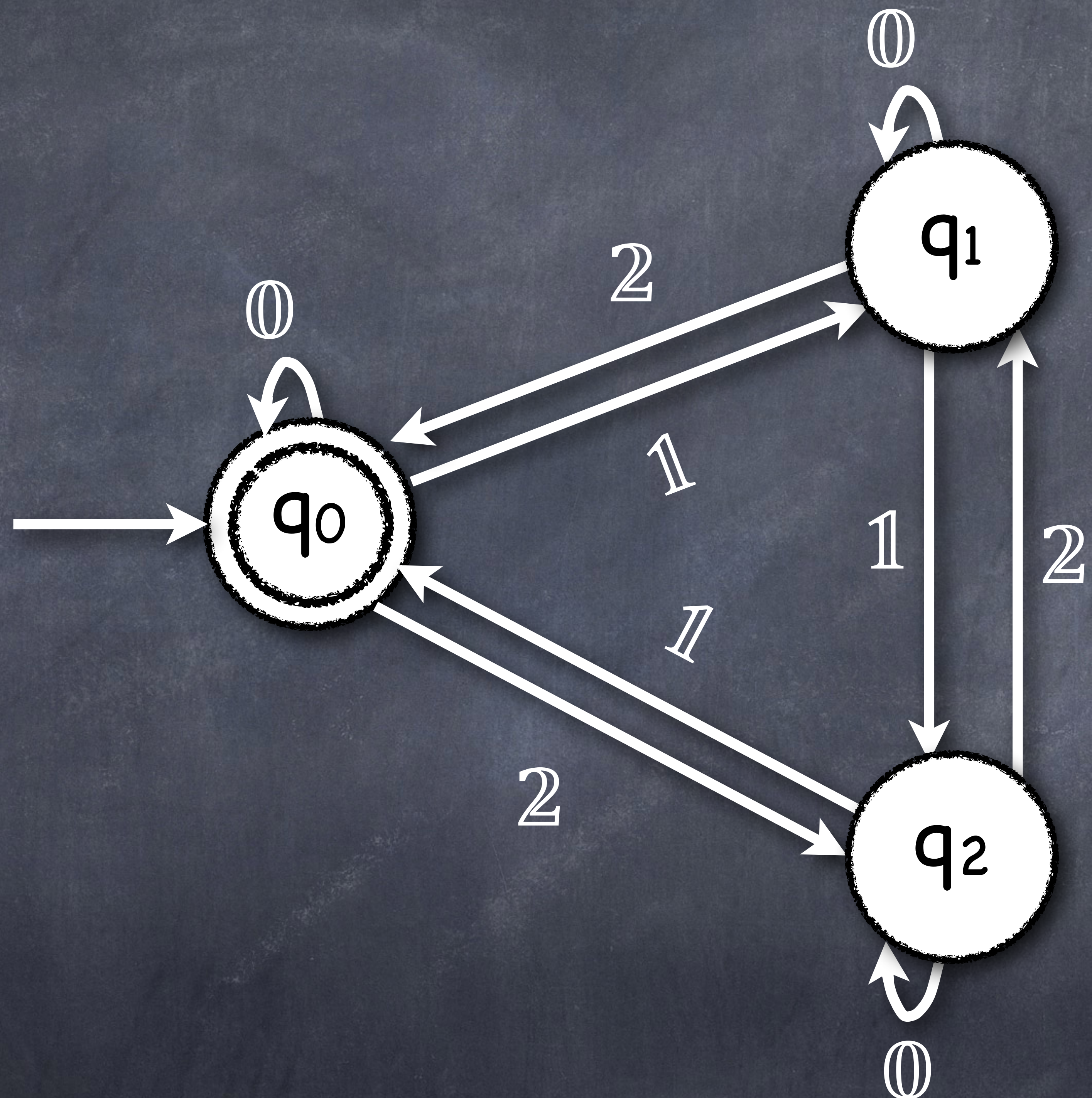


(d)



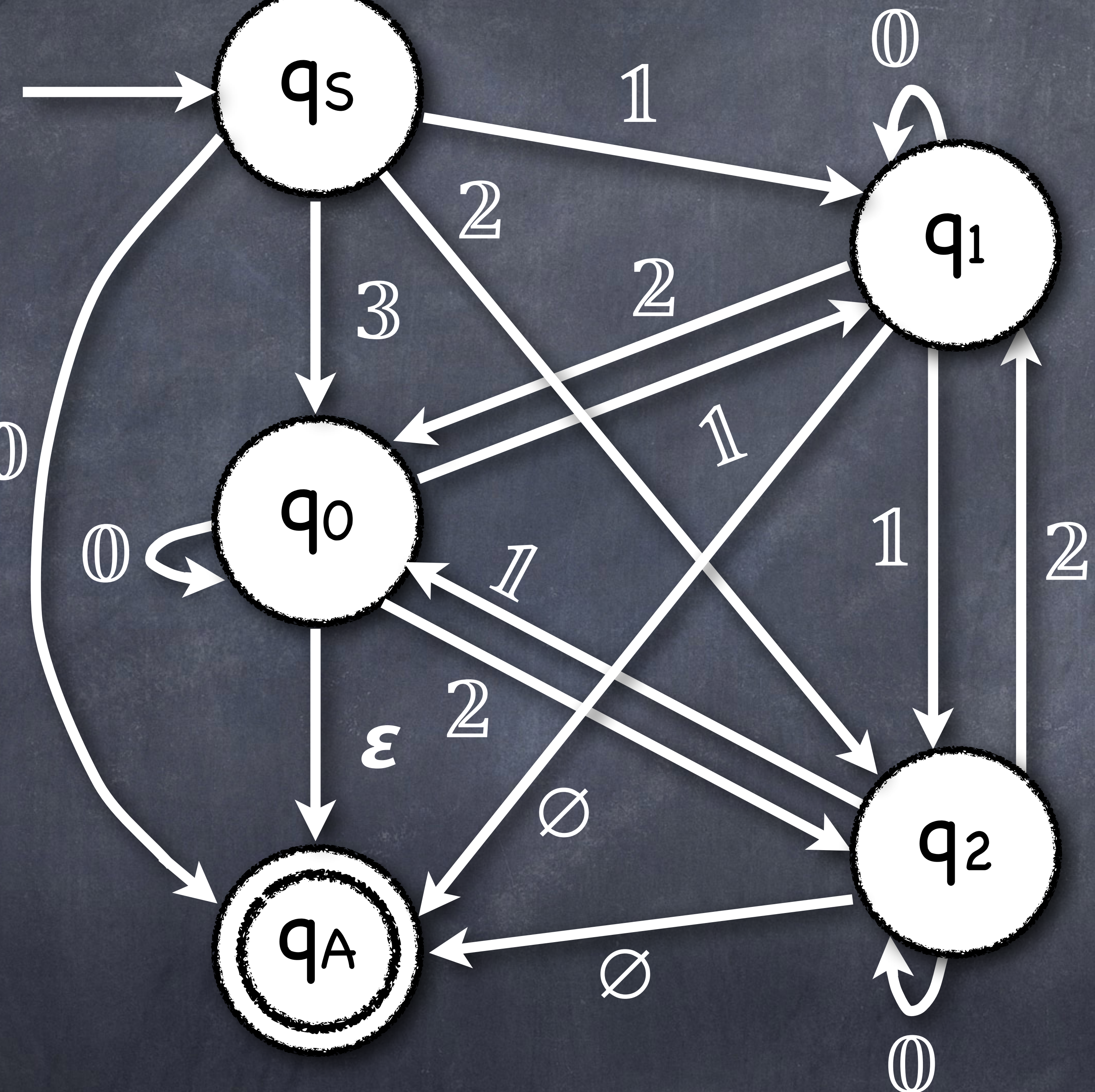
(e)

Multiples of 3 (base 10)



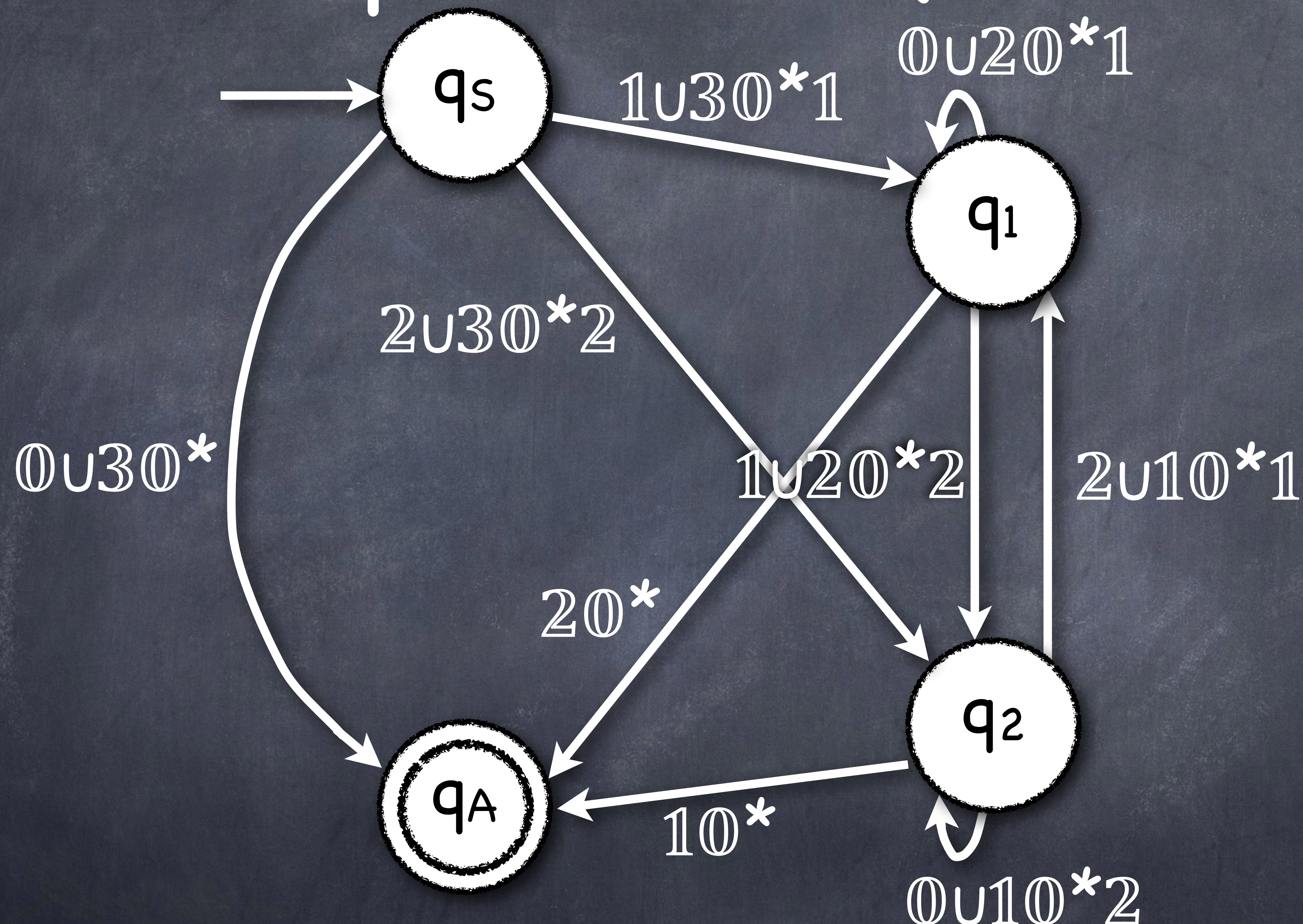
𝟘=0∪3∪6∪9,   𝟙=1∪4∪7,   𝟚=2∪5∪8

Multiples of 3 (base 10)



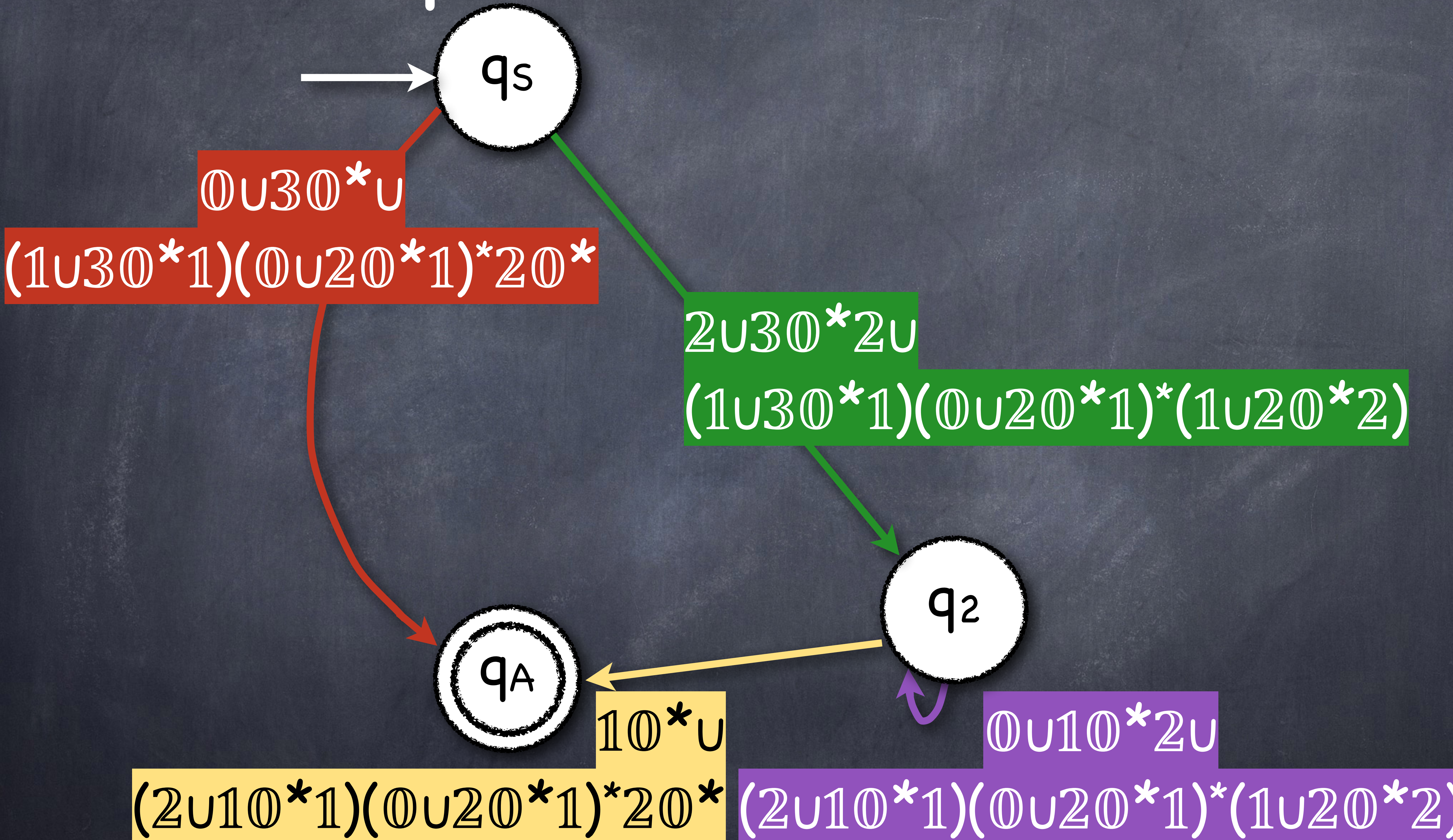
$\mathbb{0}=0\cup\mathbb{3}$, $\mathbb{3}=3\cup6\cup9$, $\mathbb{1}=1\cup4\cup7$, $\mathbb{2}=2\cup5\cup8$

Multiples of 3 (base 10)



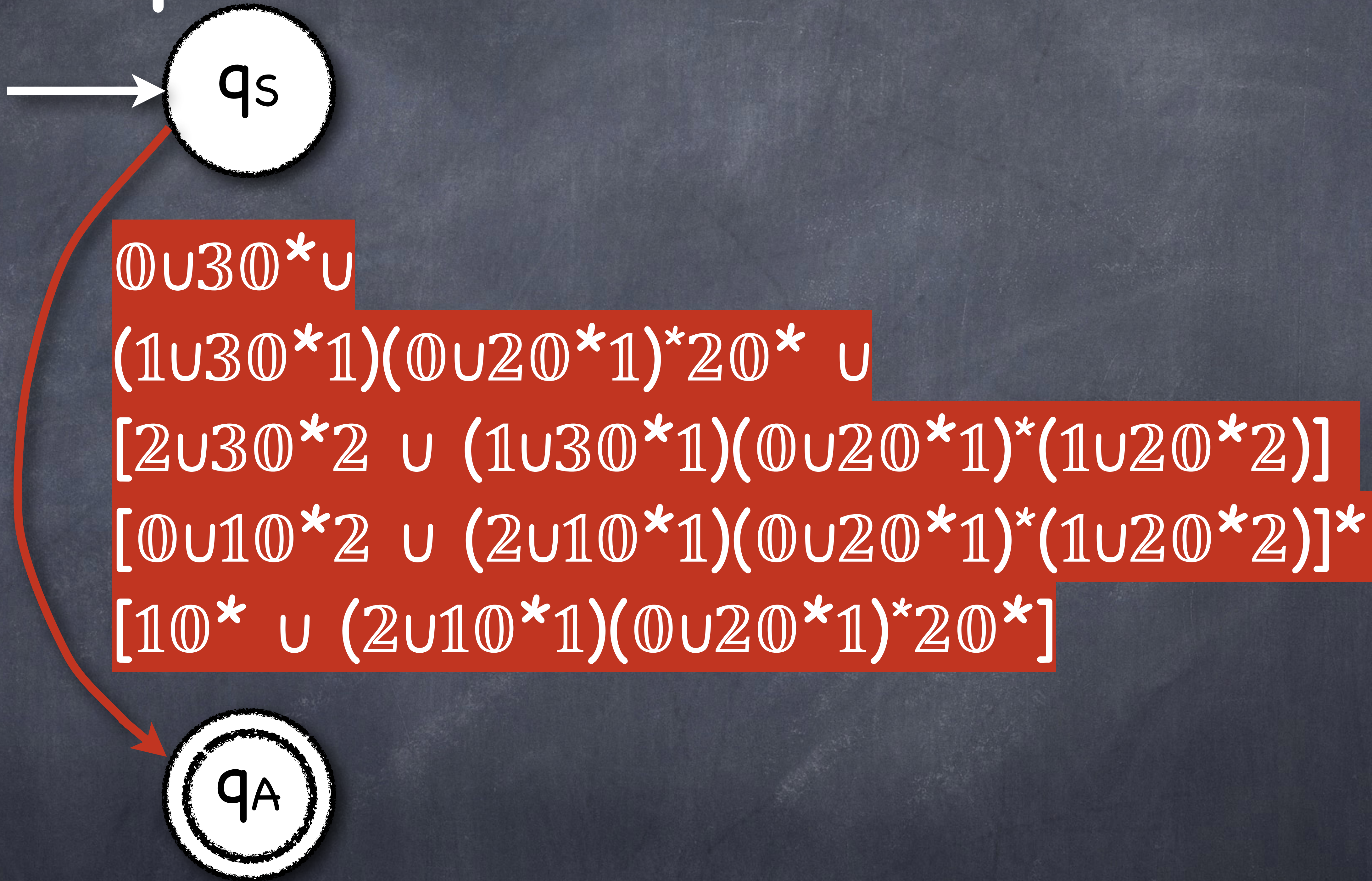
$\mathbb{0}=0\cup 3$, $\mathbb{3}=3\cup 6\cup 9$, $\mathbb{1}=1\cup 4\cup 7$, $\mathbb{2}=2\cup 5\cup 8$

Multiples of 3 (base 10)



𝟘=0∪3, 𝟛=3∪6∪9, 𝟙=1∪4∪7, 𝟚=2∪5∪8

Multiples of 3 (base 10)



$\mathbb{0}=0\cup 3$, $\mathbb{3}=3\cup 6\cup 9$, $\mathbb{1}=1\cup 4\cup 7$, $\mathbb{2}=2\cup 5\cup 8$

Multiples of 3 (base 10)

$\mathbb{3} = 3 \cup 6 \cup 9,$

$\mathbb{0} = 0 \cup \mathbb{3},$

$\mathbb{1} = 1 \cup 4 \cup 7,$

$\mathbb{2} = 2 \cup 5 \cup 8$

$$\mathbb{0} \;\cup\; \mathbb{3}\mathbb{0}^* \;\cup$$
$$(\mathbb{1} \cup \mathbb{3}\mathbb{0}^*\mathbb{1})(\mathbb{0} \cup \mathbb{2}\mathbb{0}^*\mathbb{1})^*\,\mathbb{2}\mathbb{0}^* \;\cup$$
$$[\mathbb{2} \cup \mathbb{3}\mathbb{0}^*\mathbb{2} \cup (\mathbb{1} \cup \mathbb{3}\mathbb{0}^*\mathbb{1})(\mathbb{0} \cup \mathbb{2}\mathbb{0}^*\mathbb{1})^*(\mathbb{1} \cup \mathbb{2}\mathbb{0}^*\mathbb{2})]$$
$$[\mathbb{0} \cup \mathbb{1}\mathbb{0}^*\mathbb{2} \cup (\mathbb{2} \cup \mathbb{1}\mathbb{0}^*\mathbb{1})(\mathbb{0} \cup \mathbb{2}\mathbb{0}^*\mathbb{1})^*(\mathbb{1} \cup \mathbb{2}\mathbb{0}^*\mathbb{2})]^*$$
$$[\mathbb{1}\mathbb{0}^* \cup (\mathbb{2} \cup \mathbb{1}\mathbb{0}^*\mathbb{1})(\mathbb{0} \cup \mathbb{2}\mathbb{0}^*\mathbb{1})^*\,\mathbb{2}\mathbb{0}^*]$$

Application of the Myhill-Nerode Theorem

Application of the Myhill-Nerode Theorem

Given two regular expressions R and R′ we can find out whether they generate the same regular language or not :

Application of the Myhill-Nerode Theorem

Given two regular expressions R and R′ we can find out whether they generate the same regular language or not :

1. From R and R′, compute NFAs N and N′ accepting L(R) and L(R′) (Lemma 1.55).

Application of the Myhill-Nerode Theorem

Given two regular expressions R and R′ we can find out whether they generate the same regular language or not :

1. From R and R′, compute NFAs N and N′ accepting L(R) and L(R′) (Lemma 1.55).

2. Compute equivalent DFAs M and M′ (Thm 1.39).

Application of the Myhill-Nerode Theorem

Given two regular expressions R and R′ we can find out whether they generate the same regular language or not :

1. From R and R′, compute NFAs N and N′ accepting L(R) and L(R′) (Lemma 1.55).
2. Compute equivalent DFAs M and M′ (Thm 1.39).
3. Using part (b) of Myhill-Nerode we construct minimal DFAs W for M and W′ for M′.

Application of the Myhill-Nerode Theorem

Given two regular expressions R and R′ we can find out whether they generate the same regular language or not :

1. From R and R′, compute NFAs N and N′ accepting L(R) and L(R′) (Lemma 1.55).

2. Compute equivalent DFAs M and M′ (Thm 1.39).

3. Using part (b) of Myhill-Nerode we construct minimal DFAs W for M and W′ for M′.

4. L(R)=L(R′) iff W≈W′
   (≈ means "identical up to state renaming").

Regular and non-Regular Languages

footnote 3 page 46:

footnote 3 page 46:

- Let $M_A=(Q_A,\Sigma,\delta_A,q_{0A},F_A)$ be a DFA accepting L_A and $M_B=(Q_B,\Sigma,\delta_B,q_{0B},F_B)$ be a DFA accepting L_B.

footnote 3 page 46:

- Let $M_A=(Q_A,\Sigma,\delta_A,q_{0A},F_A)$ be a DFA accepting L_A and $M_B=(Q_B,\Sigma,\delta_B,q_{0B},F_B)$ be a DFA accepting L_B.

- Consider $M_U=(Q_A \times Q_B,\Sigma,\delta_U,(q_{0A},q_{0B}),F_U)$ where
  $\delta_U((q,q'),s) = (\delta_A(q,s), \delta_B(q',s))$ for all q,q',s
  and
  $F_U = (FA \times QB) \cup (QA \times FB)$.

footnote 3 page 46:

- Let $M_A=(Q_A,\Sigma,\delta_A,q_{0A},F_A)$ be a DFA accepting L_A and $M_B=(Q_B,\Sigma,\delta_B,q_{0B},F_B)$ be a DFA accepting L_B.
- Consider $M_U=(Q_A \times Q_B,\Sigma,\delta_U,(q_{0A},q_{0B}),F_U)$ where

  $\delta_U((q,q'),s) = (\delta_A(q,s), \delta_B(q',s))$ for all q,q',s

  and

  $F_U = (FA \times QB) \cup (QA \times FB)$.
- $L_U = L_A \cup L_B$.

footnote 3 page 46:

- Let $M_A=(Q_A,\Sigma,\delta_A,q_{0A},F_A)$ be a DFA accepting L_A and $M_B=(Q_B,\Sigma,\delta_B,q_{0B},F_B)$ be a DFA accepting L_B.
- Consider $M_U=(Q_A \times Q_B,\Sigma,\delta_U,(q_{0A},q_{0B}),F_U)$ where
  $$\delta_U((q,q'),s) = (\delta_A(q,s), \delta_B(q',s)) \text{ for all } q,q',s$$
  and
  $$F_U = (FA \times QB) \cup (QA \times FB).$$
- $L_U = L_A \cup L_B$.
- $F_U = F_A \times F_B$ would yield the **intersection** (and not the **union**) of L_A and L_B.
  This proves that the class of regular languages is also closed under intersection.

NON-Regular Languages

- $B = \{ 0^n1^n \mid n \geq 0 \}$
- $C = \{ w \mid w$ contains an equal number of 0's and 1's $\}$
- $D = \{ w \mid w$ contains an equal number of occurrences of 01 and 10 as sub-strings $\}$

NON-Regular Languages

NON-Regular

- B … 0 }
- C = { w | w contains an equal number of 0's and 1's }
- D = { w | w contains an equal number of occurrences of 01 and 10 as sub-strings }

NON-Regular Languages

- B [illegible] 0 }

NON-Regular

- C = { w | w c[illegible] equal number of 0's and 1's }

NON-Regular

- D = { w | w contains an equal number of occurrences of 01 and 10 as sub-strings }

NON-Regular Languages

- B … }

NON-Regular

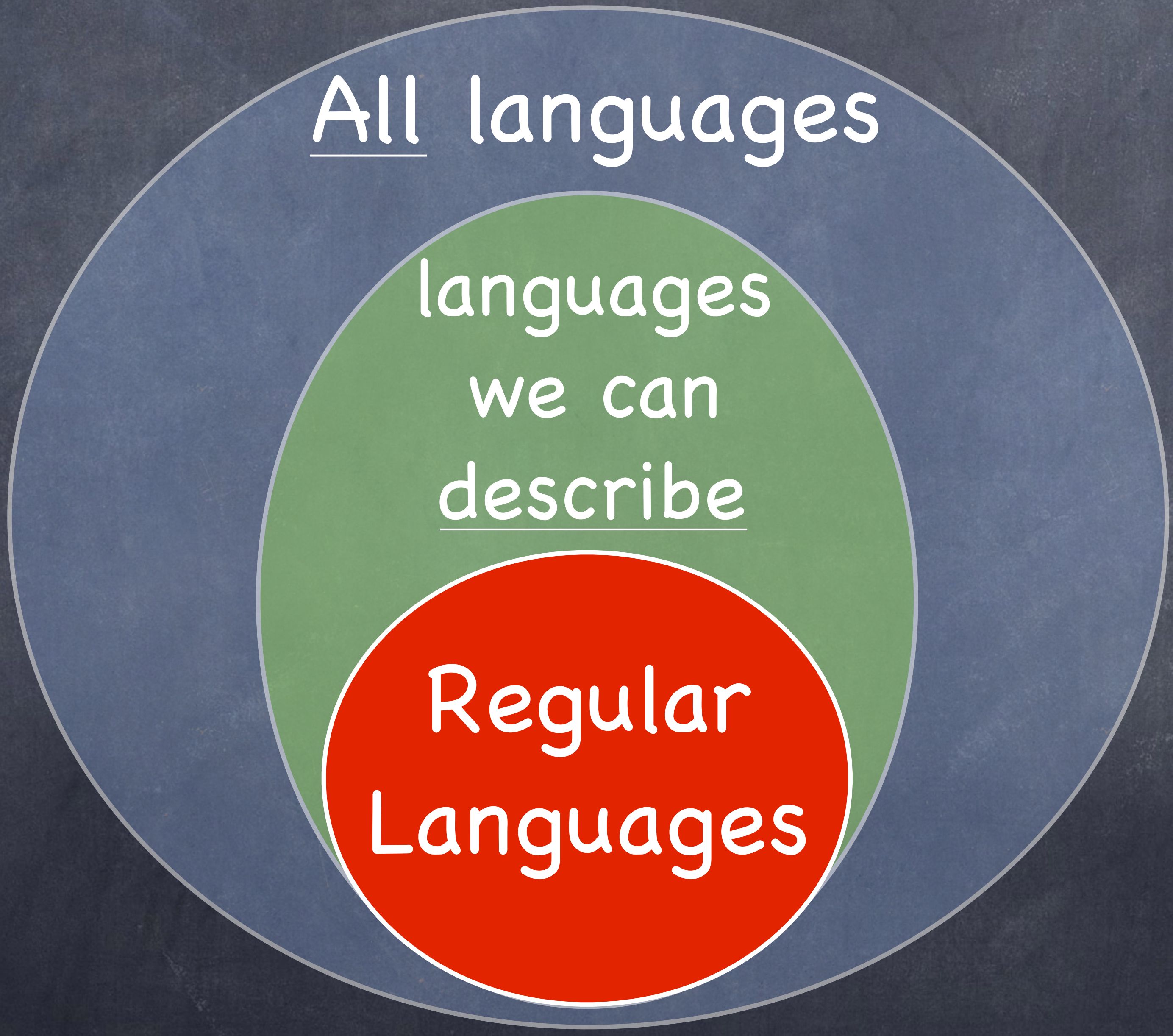
- C = { w | w c… equal number of 0's and 1's }

NON-Regular

- D = { w | w con… equal number of occurrences o… and 10 as sub-strings }

Regular

Computability Theory

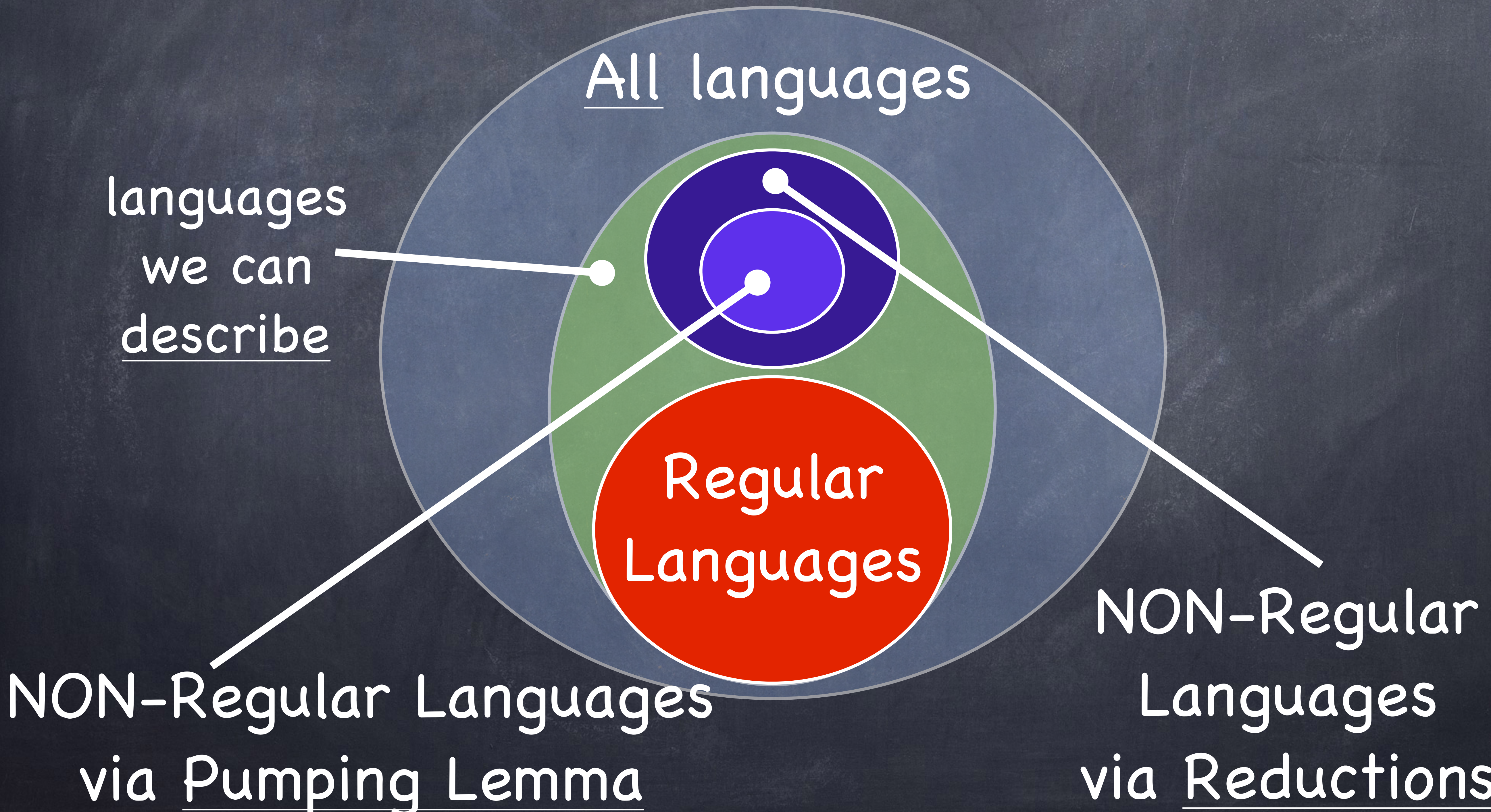


NON-Regular Languages

- Theorem: Some languages are not regular.

  Proof idea: all regular languages have certain properties. Some languages provably do not have one of these properties.

Computability Theory



Reductions

- If C is regular then so is B.
- Proof: Regular languages are closed under intersection (see footnote 3 page 46). Define $A = L(0^*1^*)$. Obviously A is regular. If C was regular then so would $C \cap A = B$.

  QED

- If B is NON-regular then so is C.

$B = \{ 0^n1^n \mid n \geq 0 \}$

$C = \{ w \mid w \text{ contains an equal number of 0's and 1's} \}$

Reductions

- If A is regular then so is A′.
- Regular laguages are closed under complement (see ex. 1.14), intersection, union, concatenation and star. If there exists R, a regular language, such that either $A^c=A'$, $A^*=A'$, $A\cap R=A'$, $A\cup R=A'$, $A\circ R=A'$ or any combinations of these operations then A′ is regular as long as A is.
- If A′ is NON-regular then so is A.

Simple Reductions

- If A^* is NON-regular then so is A.
- If A is NON-regular then so is A^c.
- If A is NON-regular then so is A^R.

Complex Reductions

- Let $A' = (A \cup R) \cap (A^c \cup R')$ (R,R′ regular)
- Let $A' = ((A^c \cap R) \cup (A^* \cap R')) \circ R''$ (R,R′,R″ regular)
- Let $A' = (A \circ R) \cap (A^c \circ R')$ (R,R′ regular)
- If A′ is NON-regular then so is A.

NON-Regular Languages

- Theorem: Some languages are not regular.

  Proof idea: all regular languages have certain properties. Some languages provably do not have one of these properties.

- Example: A property of all regular languages = the Pumping Lemma.

COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 8 : Regular and NON-Reg. Languages