Biçimsel Diller ve Otomata Teorisi

Sunu VI
Kleene Kuramı

İZZET FATİH ŞENTÜRK

Kleene's Theorem

- Any language that can be defined by regular expression, or finite automaton, or transition graph can be defined by all three methods

- Proved in 1956. The most important and fundamental result in the theory of finite automata

Kleene's Theorem

- We wish to show that the set of ZAPS, the set of ZEPS, and the set of ZIPS are all the same
- We need three parts
  - Part I, we shall show that all ZAPS are ZEPS
  - Part II, we shall show that all ZEPS are ZIPS
  - Part III, we shall show that all ZIPS are ZAPS
- $[\text{ZAPS} \subset \text{ZEPS} \subset \text{ZIPS} \subset \text{ZAPS}] \equiv [\text{ZAPS} = \text{ZEPS} = \text{ZIPS}]$

Proof

- Part 1: Every language that can be defined by a FA can also be defined by a TG
- Part 2: Every language that can be defined by a TG can also be defined by a RE
- Part 3: Every language that can be defined by a RE can also be defined by a FA

Proof – Part 1

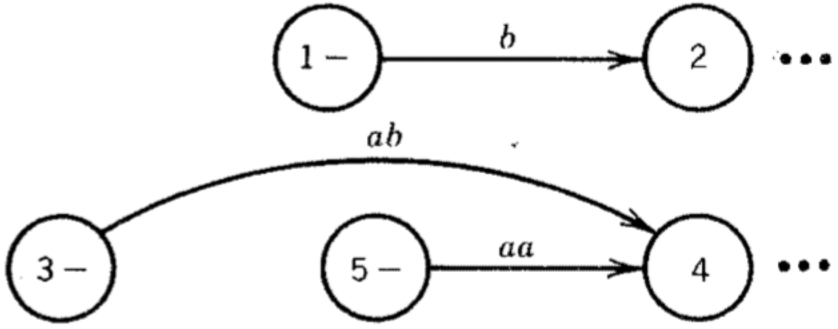
- The easiest part
- Every FA is itself already a TG
  - Any language that has been defined by a FA has already been defined by a TG

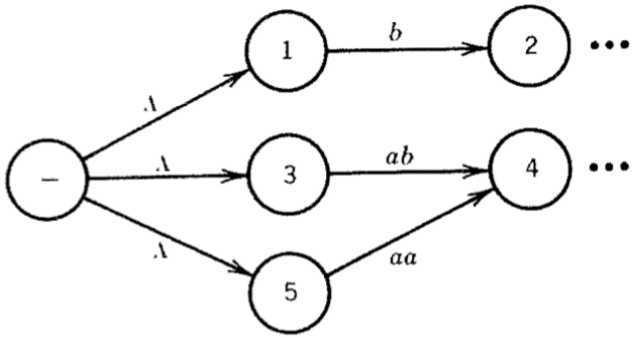
Proof – Part 2

- The proof will be by constructive algorithm
  - We present a procedure that starts out with a TG and ends up with a RE that defines the same language
- To be acceptable, any algorithm must satistfy two criteria
  - It must work for every TG
  - It must guarantee to finish its job in a finite time (number of steps)

Proof – Part 2

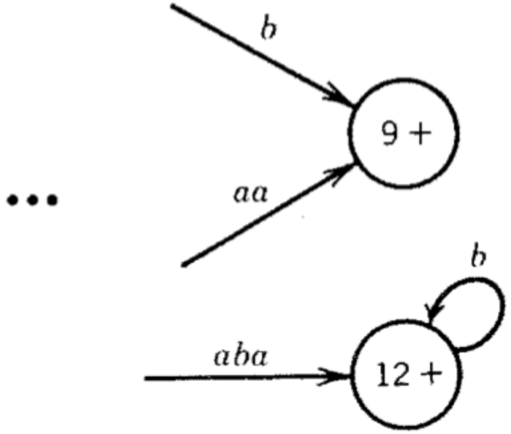
- Let us start with an abstract transition graph *T*
  - *T* may have many start states. We first want to simplify *T* so that it has only one start state
  - Introduce a new state that we label with a minus sign and that we connect to all the previous start states by edges labeled **Λ**
  - Drop the minus signs from the previous start states

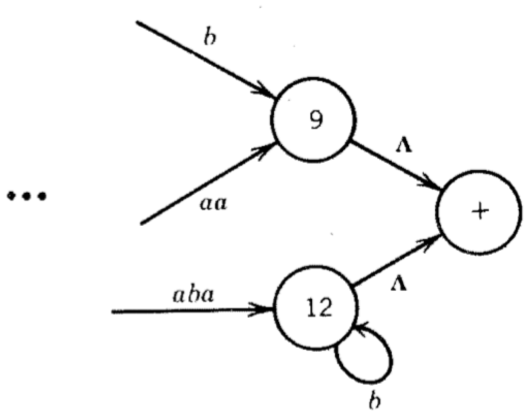




Proof – Part 2

- Another simplification we can make is to have a unique final state without changing the language it accepts
  - If *T* had no final states then it accepts no strings and has no language. We need produce no RE other than the null (φ)
  - If *T* has several final states, let us un-final them and introduce a new unique final state labeled with a plus sign. We draw edges from all former final states to the new one each labeled with **Λ**



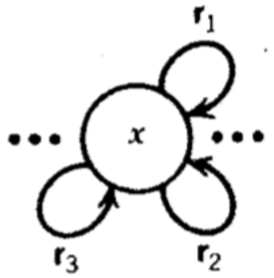


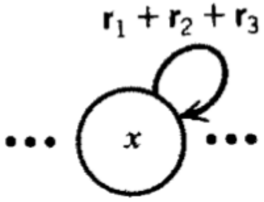
Proof – Part 2

- We shall require that the unique final state be a different state from the unique start state
  - If an old state used to have ± then both signs are removed from it to newly created states
- It should be noted that the new states does not affect the language *T* accepts
  - Any word accepted by *T* is also accepted by the new machine
  - Any word rejected by *T* is also rejected by the new machine

Proof – Part 2

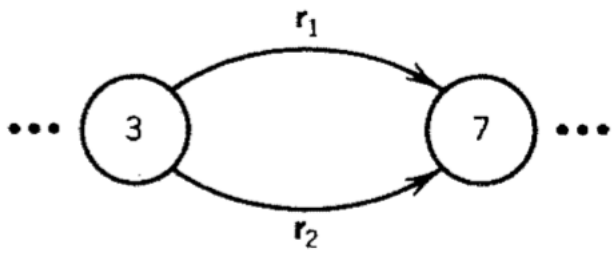
- We are now going to build piece by piece the RE that defines the same language as *T*
  - We will change *T* into a GTG
- Suppose that T has some state x inside it (no – or + state)
  - x has more than one loop
  - We can replace three loops by one loop labeled with a RE

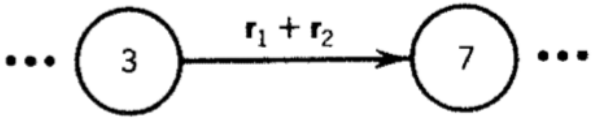




Proof – Part 2

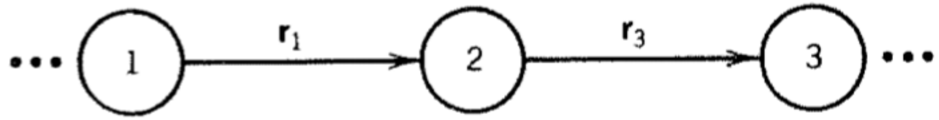
- Similarly, suppose that two states are connected by more than one edge going in the same direction
  - We can replace this with a single edge labeled with a RE

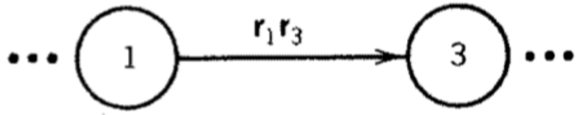




Bypass and State Elimination Operation

- If we have three states in a row connected by edges lebeled with REs (or simple strings)
  - We can eliminate the middleman and go directly from one outer state to the other by a new edge labeled with a RE that is **concatenation** of the previous labels

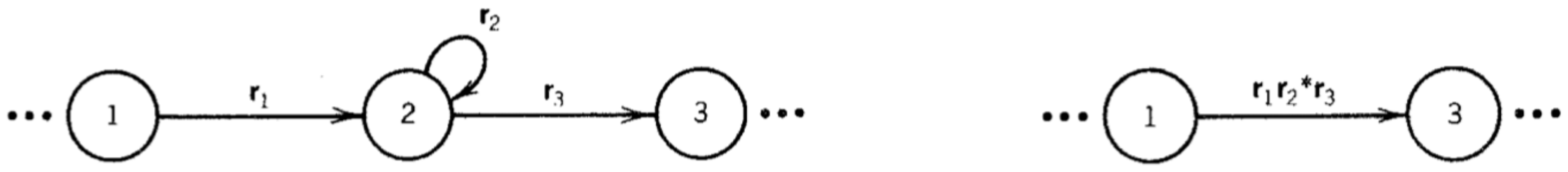




- We do not keep old edges from state 1 to state 2 and state 2 to state 3
  - Unless they are used in paths other than the ones from state 1 to state 3

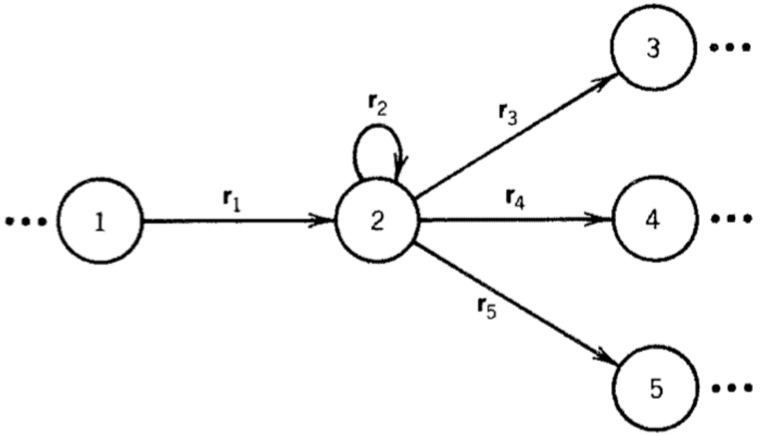
Bypass and State Elimination Operation

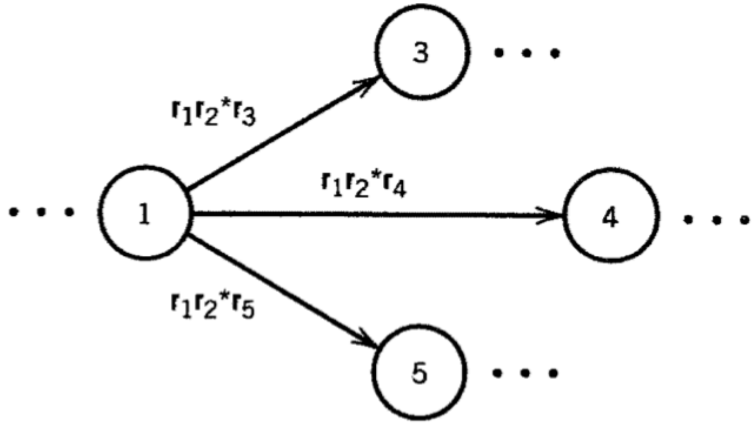
- We can do the bypass operation only as long as state 2 does not have a loop going back to itself
- If state 2 does have a loop, we must use this model



Bypass and State Elimination Operation

- If state 1 is connected to state 2 and state 2 is connected to more than one other state (e.g. states 3, 4, 5)
  - When we eliminate the edge from state 1 to state 2 we have to add edges that show how to go from state 1 to other states (e.g. states 3, 4, 5)



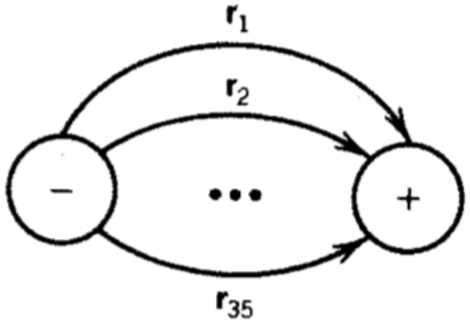


Bypass and State Elimination Operation

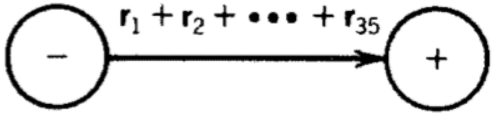
- Every state that leads into state 2 can be made to bypass state 2
  - If state 9 leads into state 2, we can eliminate the edge from state 9 to state 2 by adding edges from state 9 to states 3, 4, and 5 directly
  - We can repeat this operation until nothing leads into state 2
  - We can eliminate state 2 entirely because it will not be part of a path that accepts a language
- Without changing the set of words that *T* accepts, we have eliminated one of its states
  - We can repeat this process until we eliminate all the states from *T* except the unique start and final states

Proof – Part 2

- Finally, we will have this

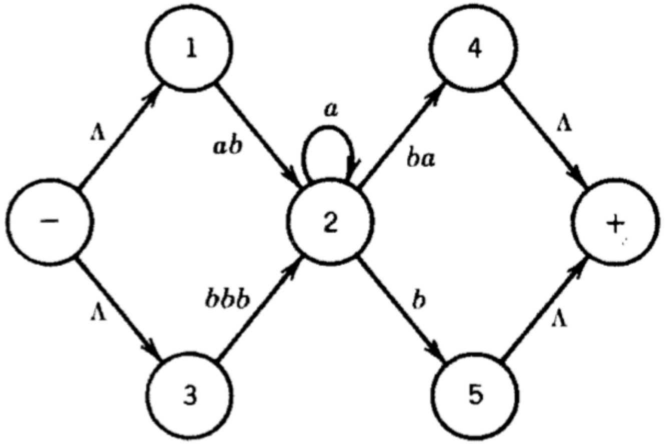


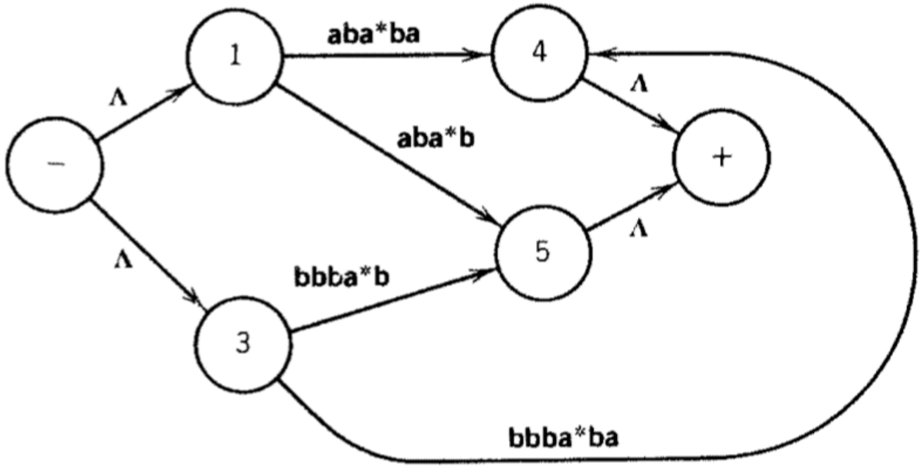
- We can then combine this once more to produce



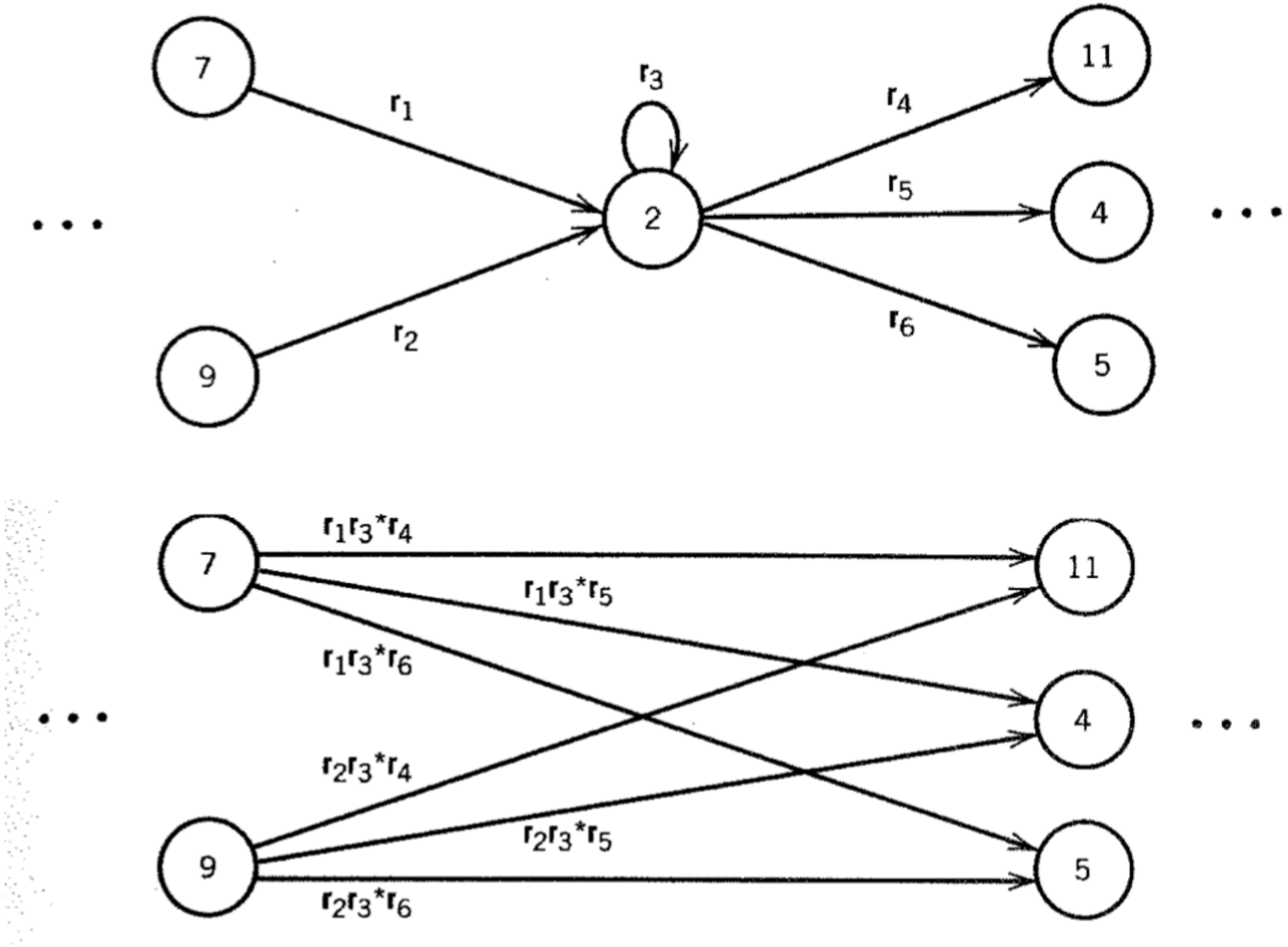
Example

- We can bypass state 2 by introducing a path from
  - State 1 to state 4 labeled aba*ba
  - State 1 to state 5 labeled aba*b
  - State 3 to state 4 labeled bbba*ba
  - State 3 to state 5 labeled bbba*b



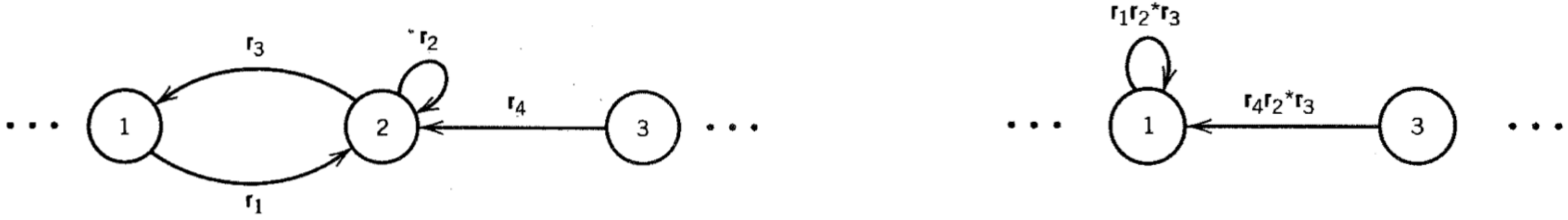


Example

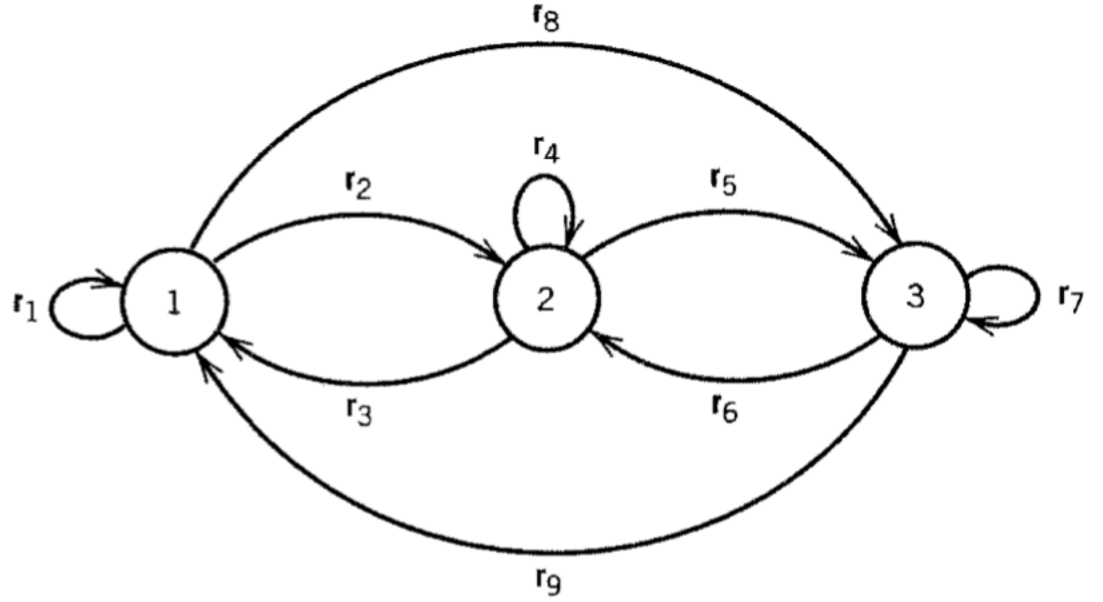

7
9
2
11
4
5
r1
r2
r3
r4
r5
r6
r1r3*r4
r1r3*r5
r1r3*r6
r2r3*r4
r2r3*r5
r2r3*r6

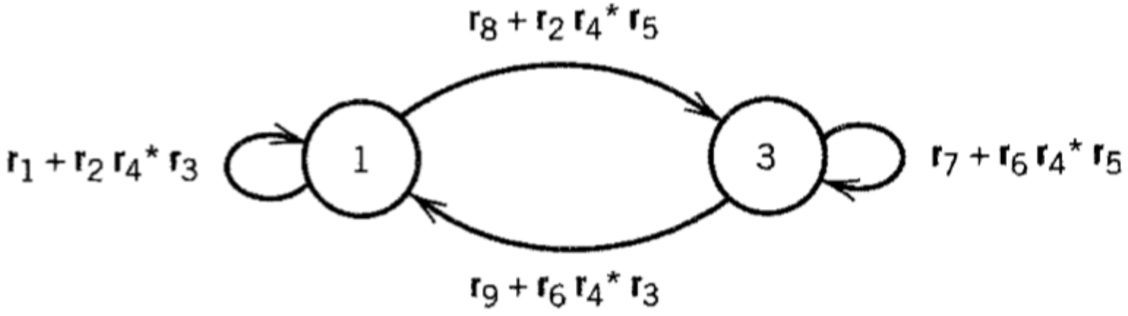
Example

- A special case that we must examine more carefully
- We want to eliminate state 2
  - One of the **source** states to the prospective bypassed state is <u>also</u> a **destination** state from that state



Example





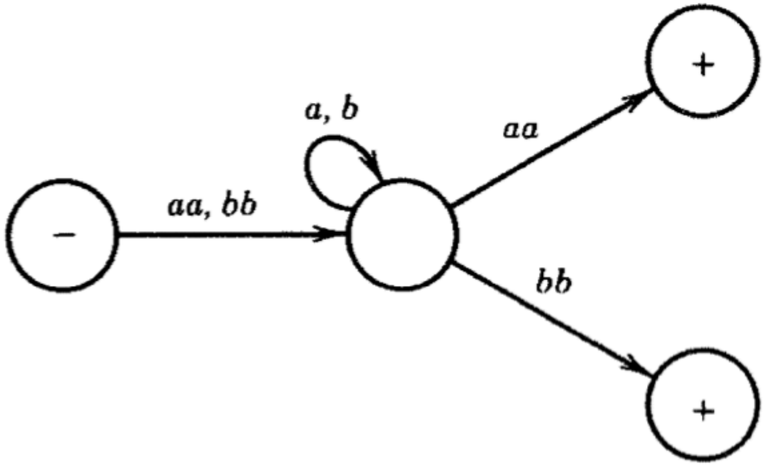
- Whenever we remove an edge or state, we must be sure that we have not destroyed any paths through T or created new paths

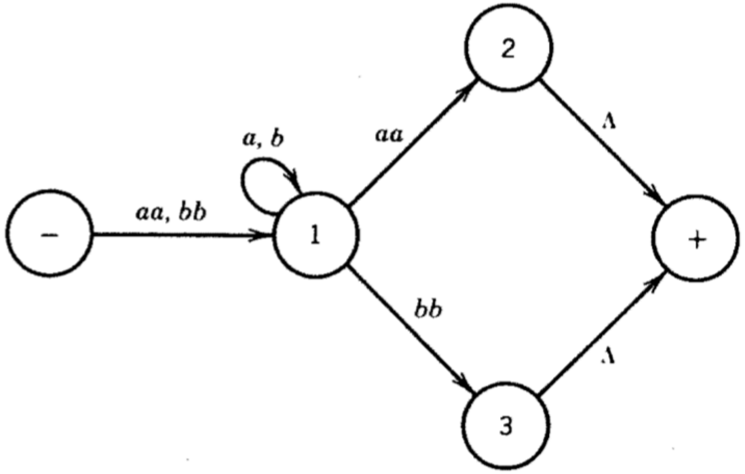
Proof – Part 2

- This algorithm terminates in a finite number of steps
  - T has only finitely many states and one state is eliminated with each iteration of the bypass procedure
- The other important observation is that the method works on all transition graphs
- Therefore, this algorithm provides a satisfactory proof that there is a RE for each TG

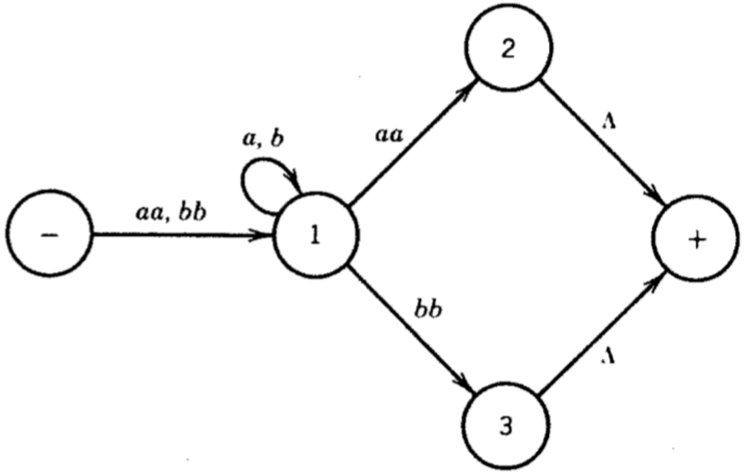
Example

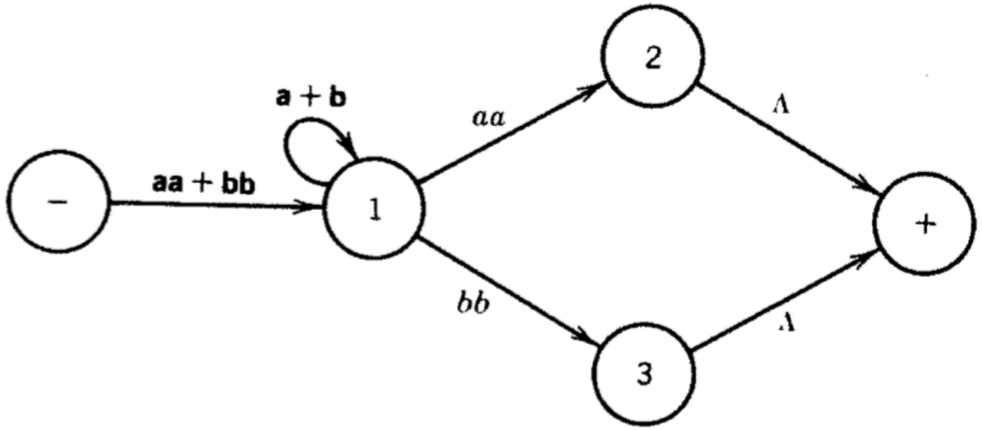
- The TG accepts all words that begin and end with double letters (having at least length 4)





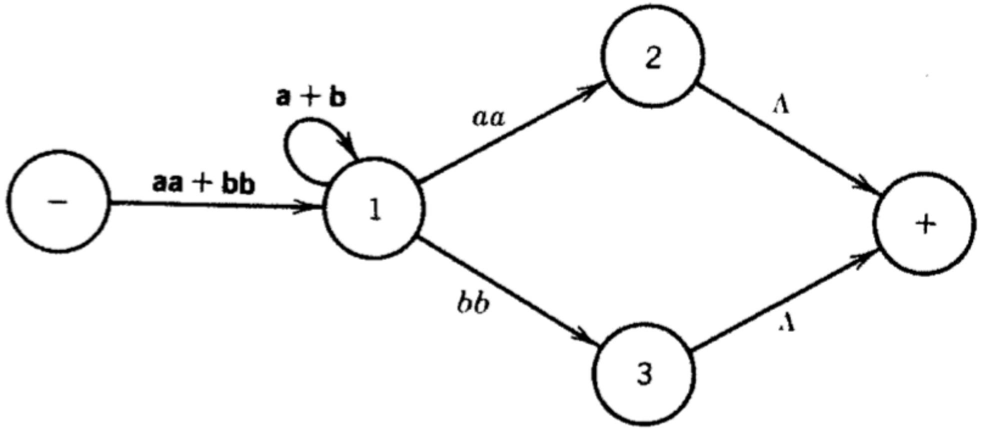
Example, continued

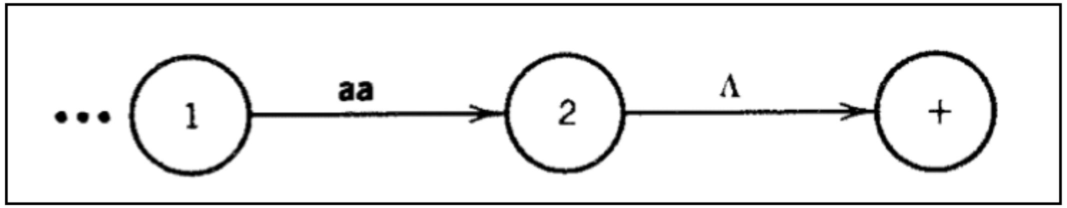


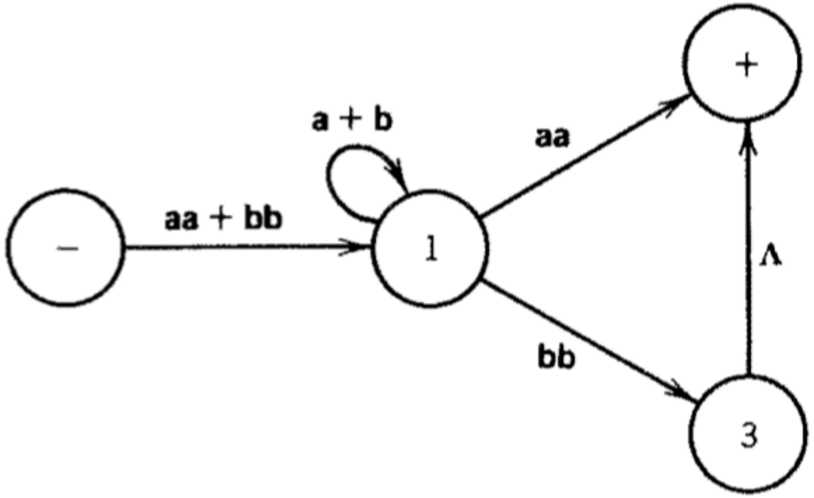


Example, continued

- The algorithm does not tell us which state of the TG we must bypass next. The order of elimination is left up to us
- Let us choose state 2 for elimination

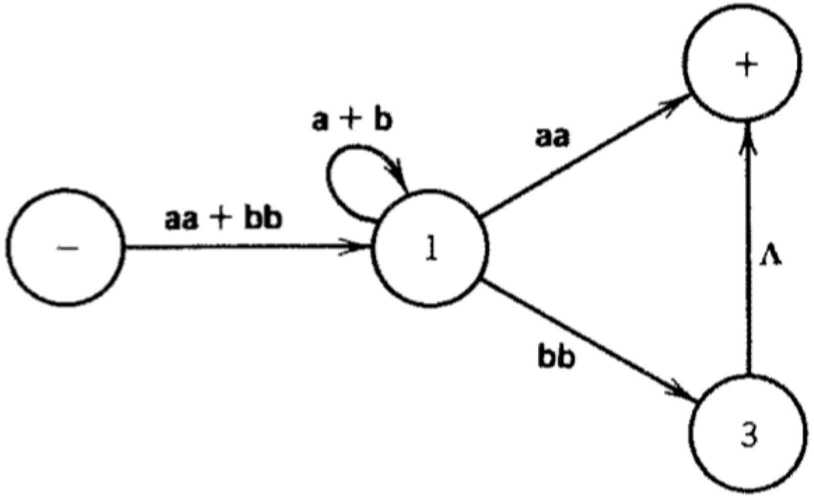


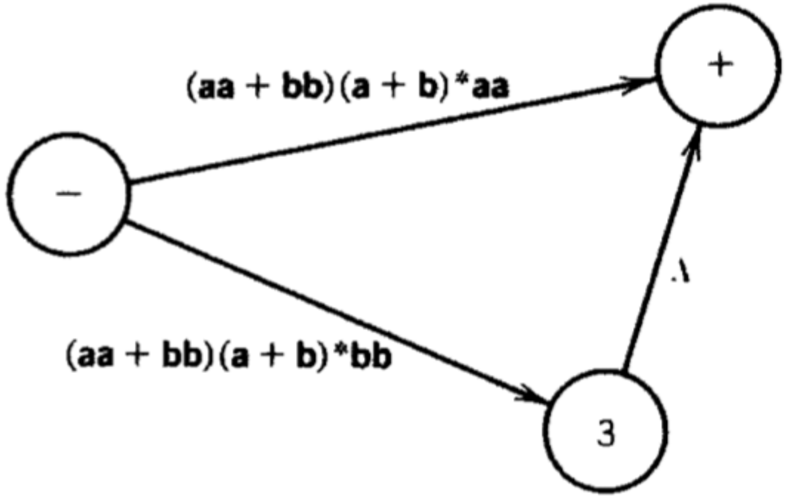




Example, continued

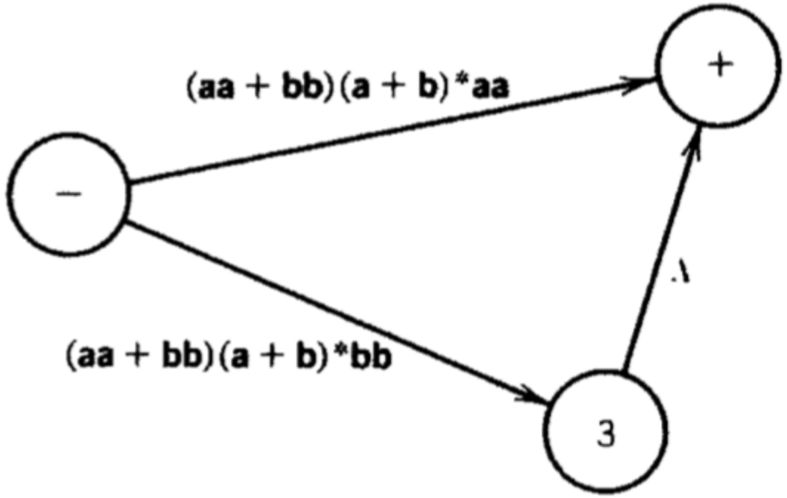
- Bypass state 1 next (before state 3)

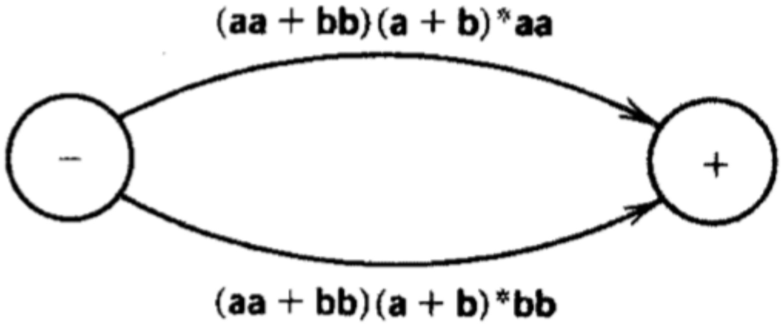




Example, continued

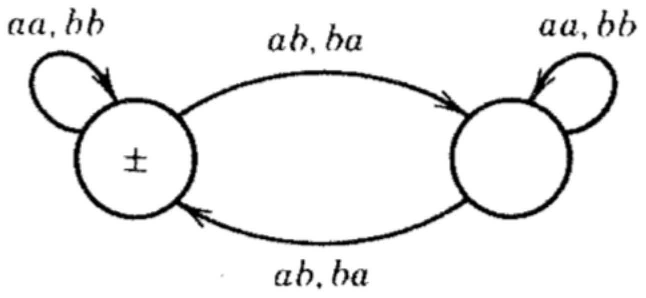
- Now we must eliminate state 3 (this is the only bypassable state left)



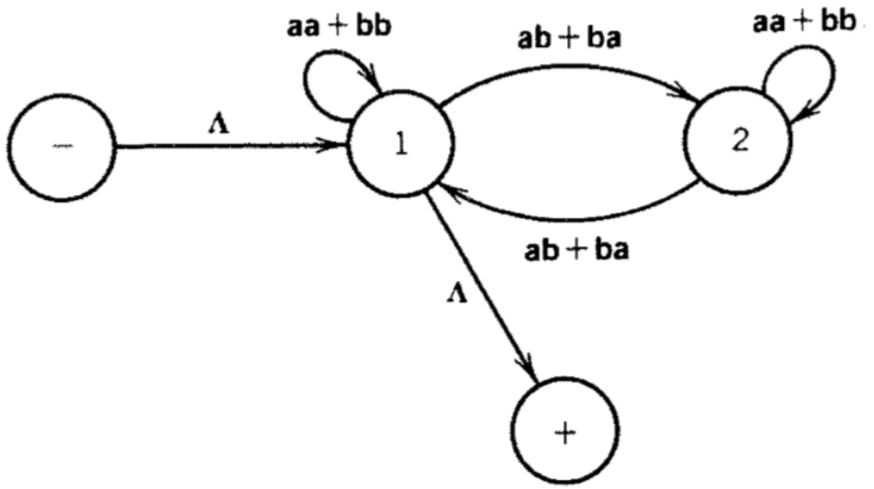


- This machine defines the same language as the RE
  - (**aa** + **bb**)(**a** + **b**)*(**aa**) + (**aa** + **bb**)(**a** + **b**)*(**bb**)

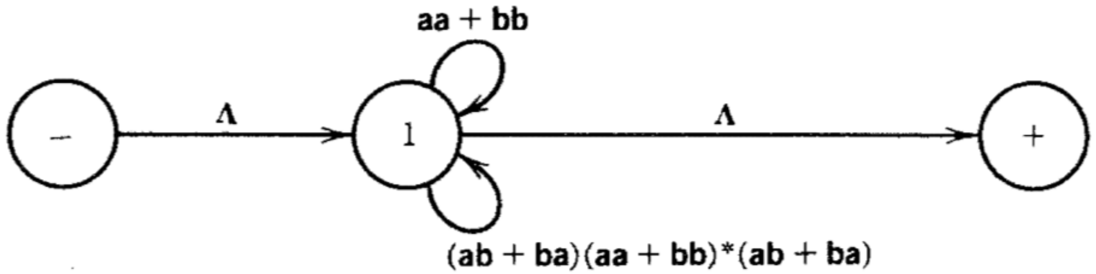
EVEN-EVEN



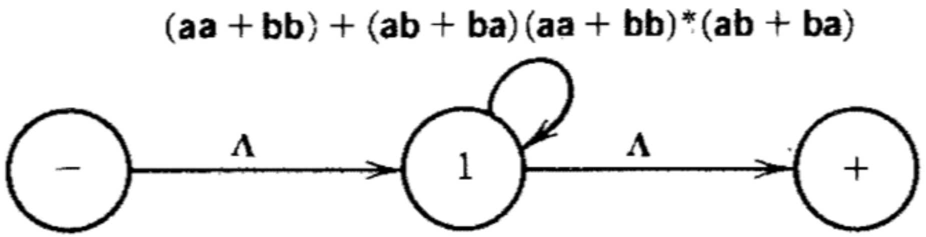
aa, bb
ab, ba
aa, bb
±
ab, ba



aa + bb
ab + ba
aa + bb
Λ
–
1
2
ab + ba
Λ
+

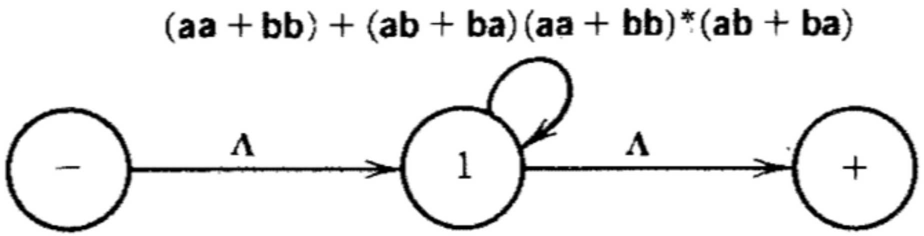


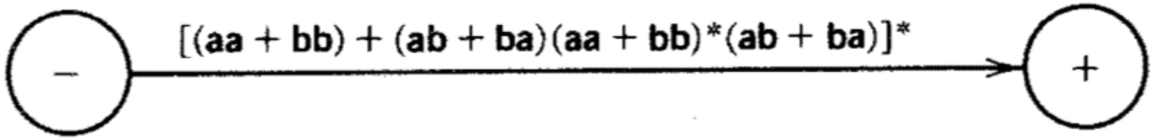
aa + bb
Λ
Λ
–
1
+
(ab + ba)(aa + bb)*(ab + ba)



(aa + bb) + (ab + ba)(aa + bb)*(ab + ba)
Λ
Λ
–
1
+

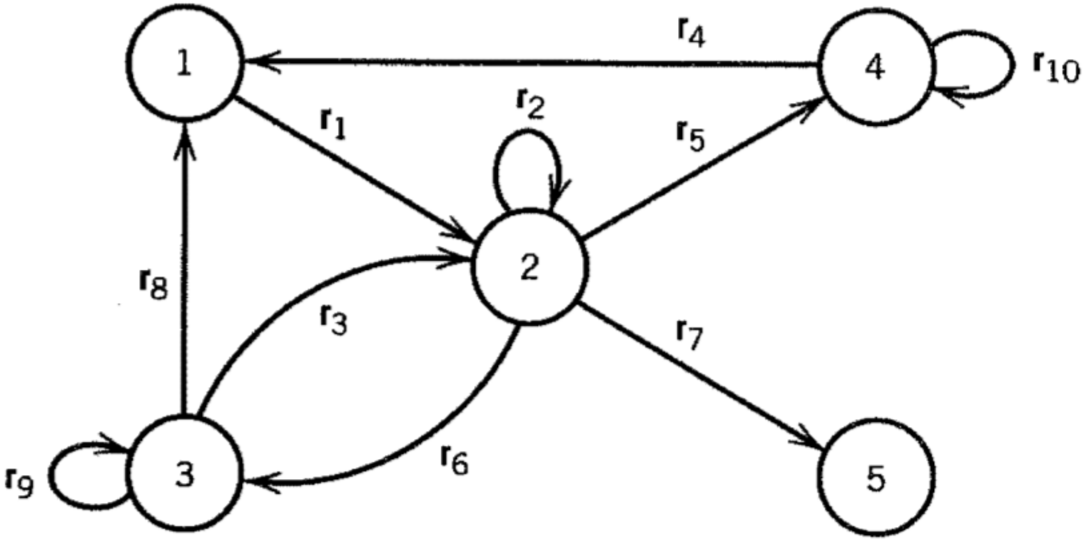
EVEN-EVEN





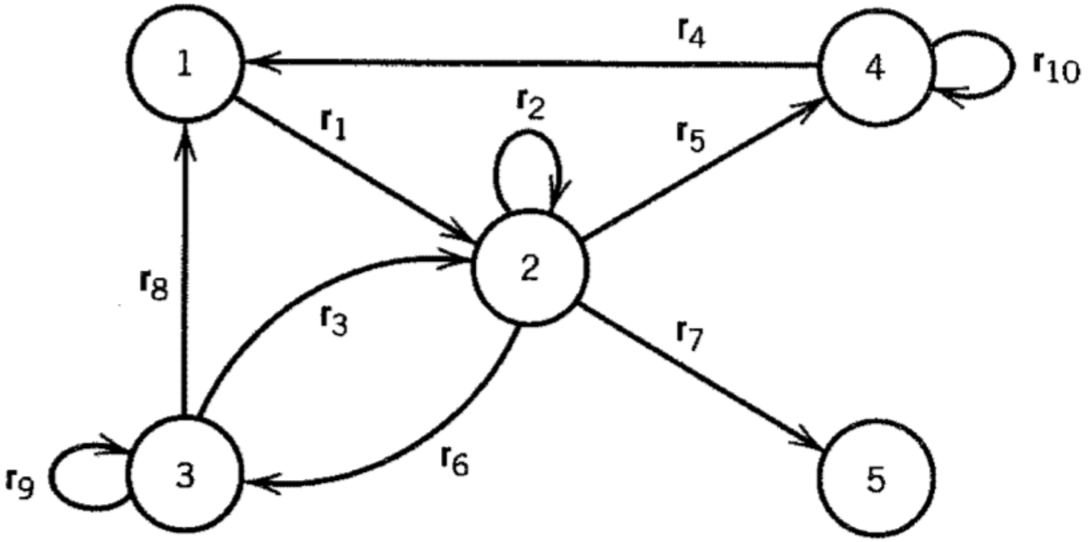
Example

- The next state to bypass is state 2
  - We need to introduce six new edges (including the loop at 3)

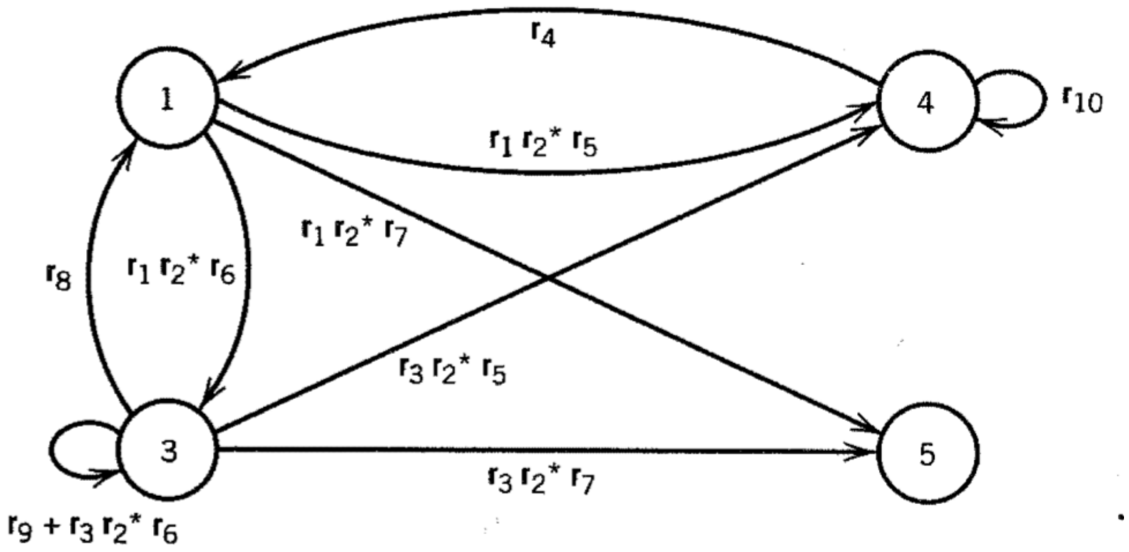


From	To	Labeled
1	3	$r_1r_2{}^*r_6$
1	4	$r_1r_2{}^*r_5$
1	5	$r_1r_2{}^*r_7$
3	3	$r_3r_2{}^*r_6$
3	4	$r_3r_2{}^*r_5$
3	5	$r_3r_2{}^*r_7$

Example



From	To	Labeled
1	3	$r_1r_2^*r_6$
1	4	$r_1r_2^*r_5$
1	5	$r_1r_2^*r_7$
3	3	$r_3r_2^*r_6$
3	4	$r_3r_2^*r_5$
3	5	$r_3r_2^*r_7$

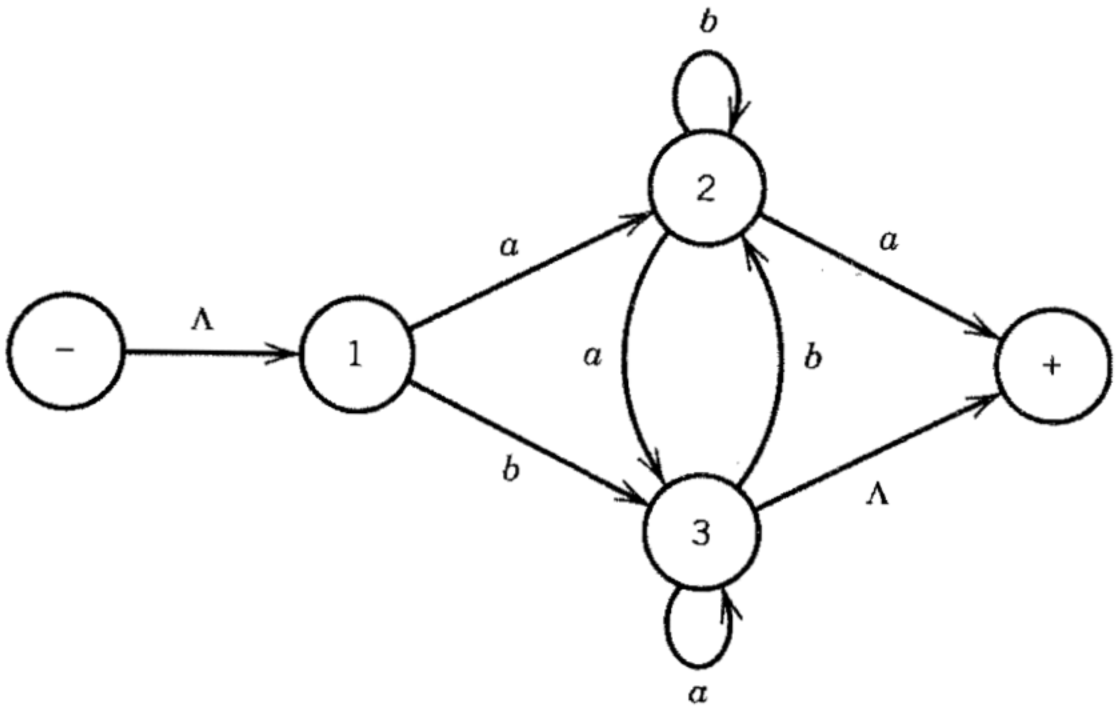


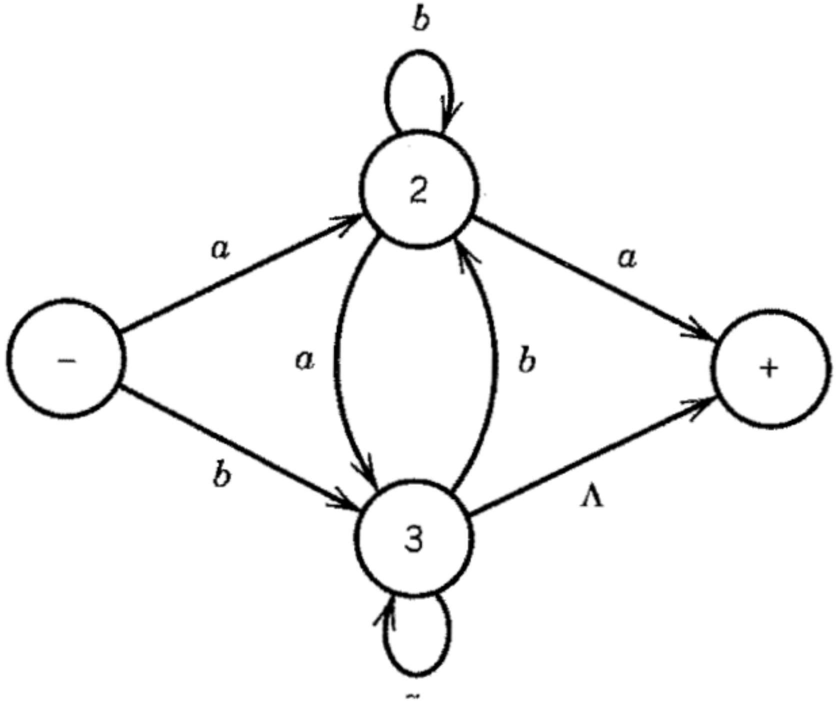
Algorithm

1. Create a unique, unenterable minus state and a unique, unleaveable plus state
2. One ob one, in any order, bypass and eliminate all the non – or + states in the TG. A state is bypassed by connecting each incoming edge with each outgoing edge. The label of each resultant edge is the concatenation of the label on the incoming edge with the label on the loop edge if there is one and the label on the outgoing edge
3. When two states are joined by more than one edge going in the same direction, unify them by adding their labels
4. When all that is left is one edge from – to +, the label on that edge is a RE that generates the same language as was recognized by the original machine

Example

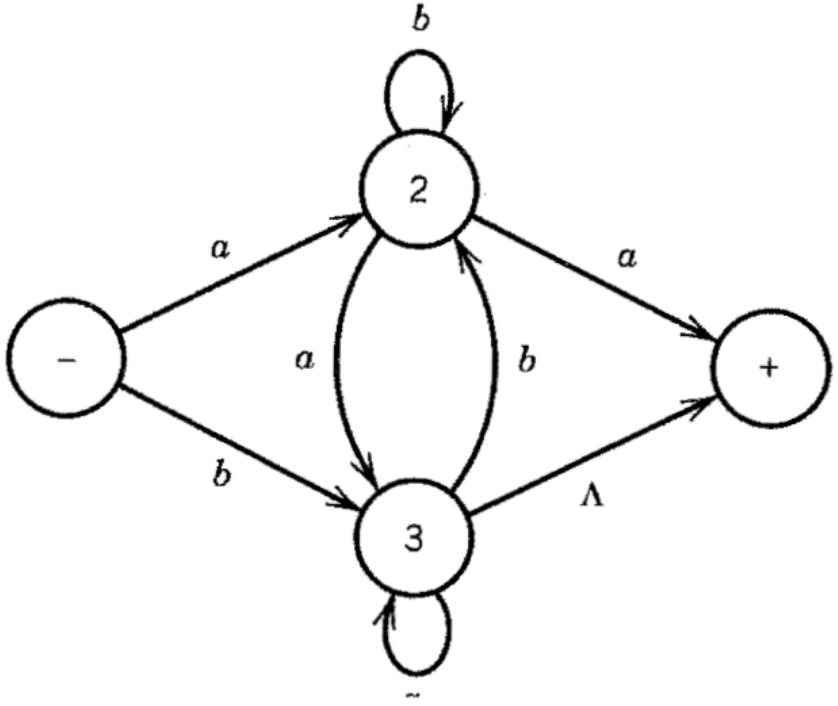
- Eliminate the states in the order 1, 2, 3

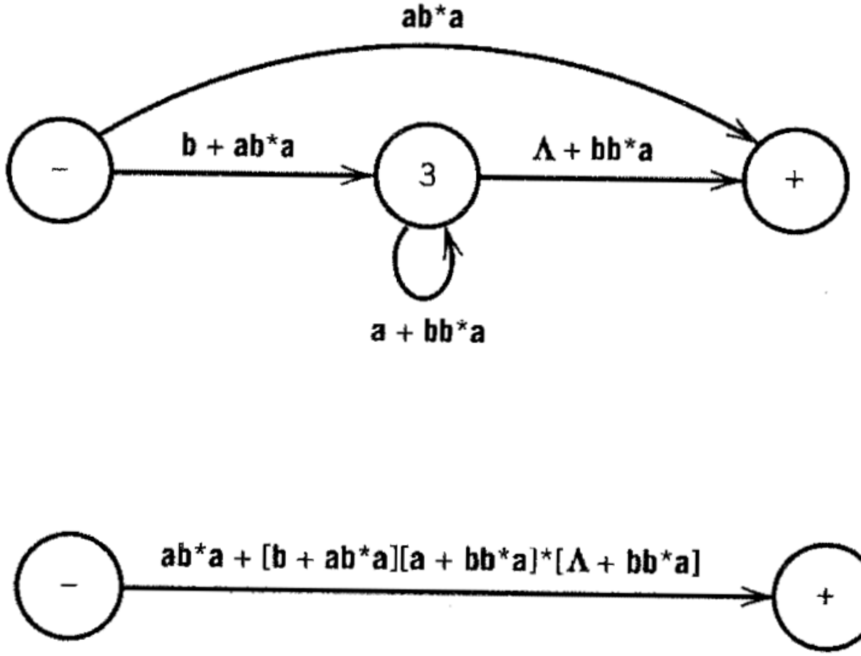




Example

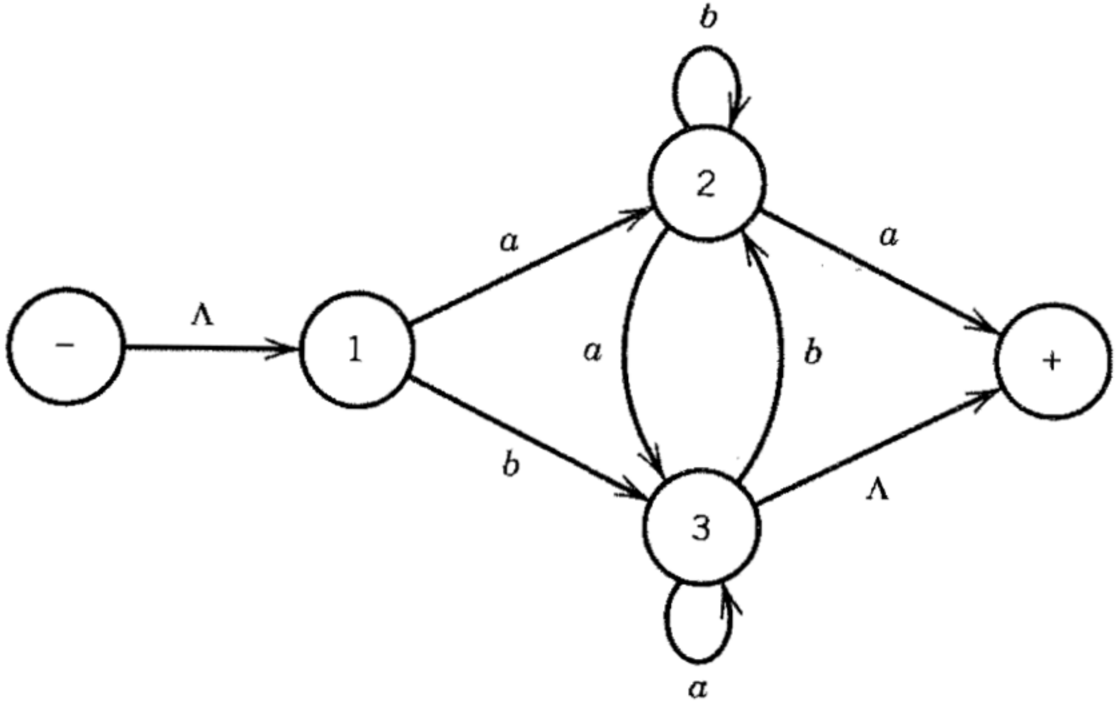
- Eliminate the states in the order 1, 2, 3

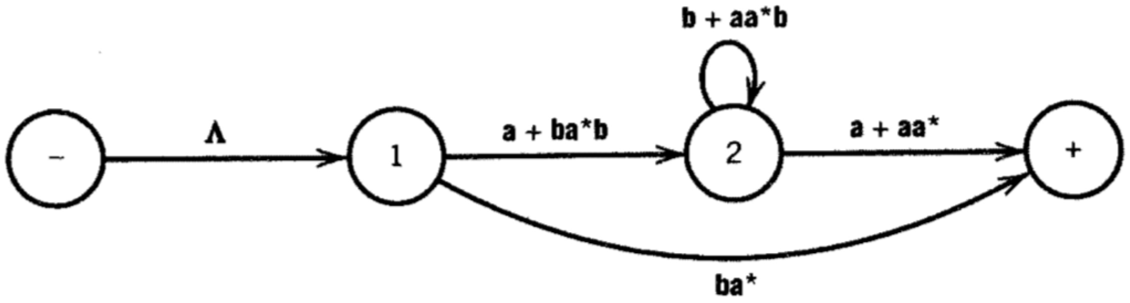




Example

- Eliminate the states in the order 3, 2, 1





Example

- Eliminate the states in the order 3, 2, 1