COURSE: THEORY OF AUTOMATA COMPUTATION

TOPICS TO BE COVERED

- Minimization Algorithm contd.
- Minimization Algorithm Example

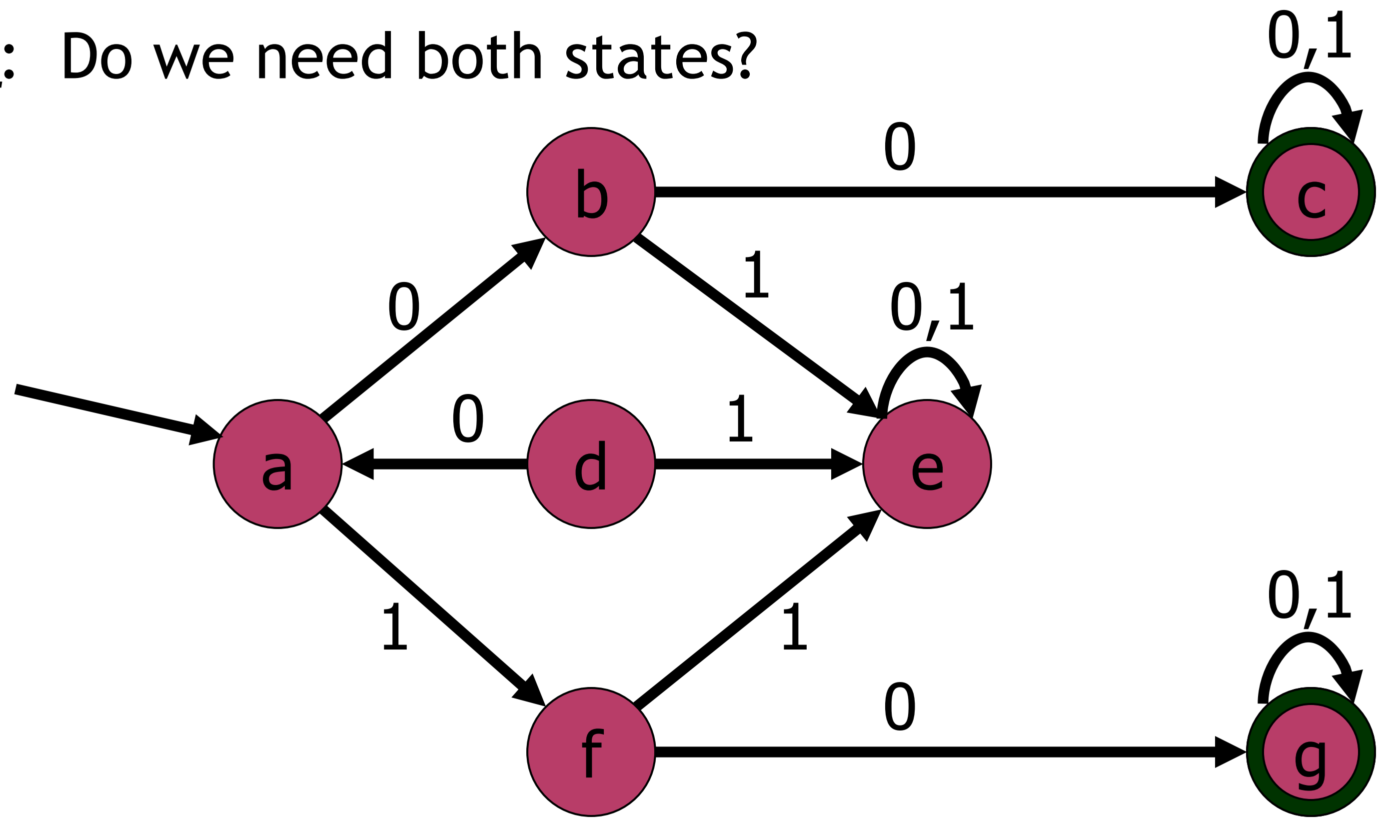
AGENDA

- Minimization Algorithm
  - Guarantees smallest possible DFA for a given regular language
  - Proof of this fact (*Time allowing*)
- Pumping Lemma
  - Gives a way of determining when certain languages are non-regular
  - A direct consequence of applying pigeonhole principle to automata (*Time allowing*)

EQUIVALENT STATES. EXAMPLE

Consider the accept states c and g. They are both sinks meaning that any string which ever reaches them is guaranteed to be accepted later.

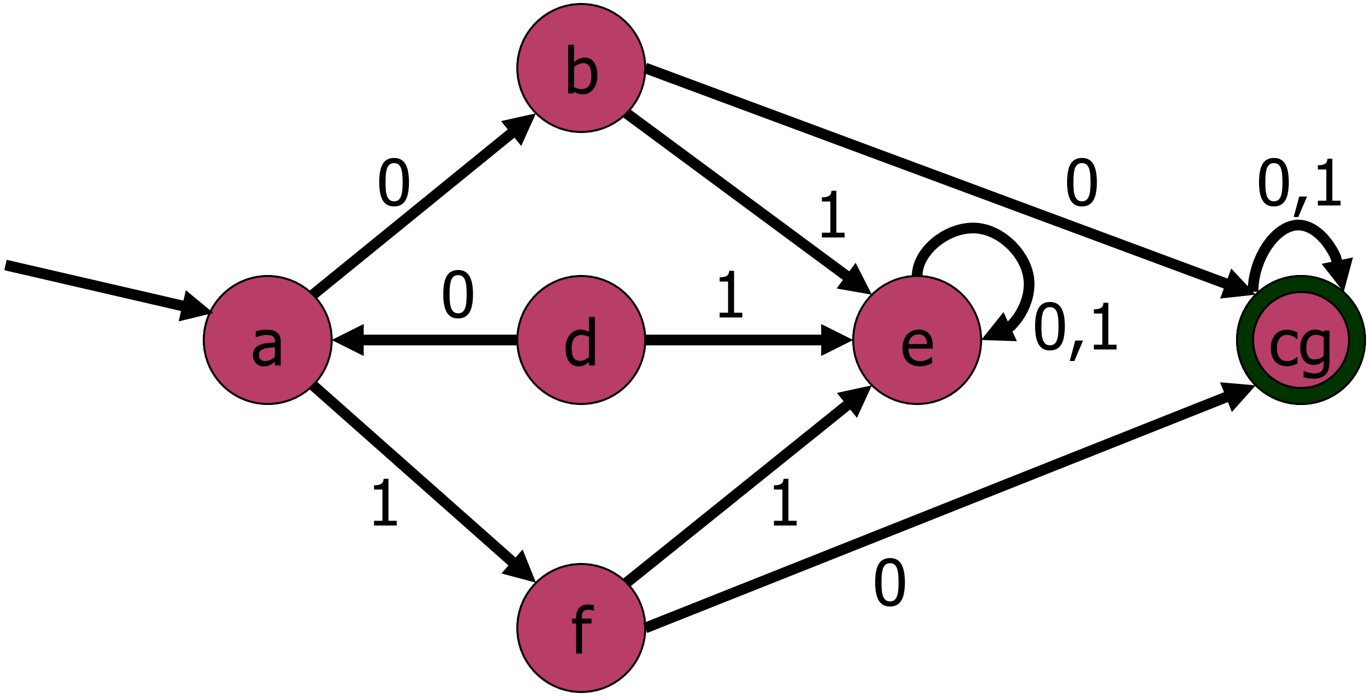
Q: Do we need both states?



EQUIVALENT STATES. EXAMPLE

A: No, they can be unified as illustrated below.

Q: Can any other states be unified because any subsequent string suffixes produce identical results?

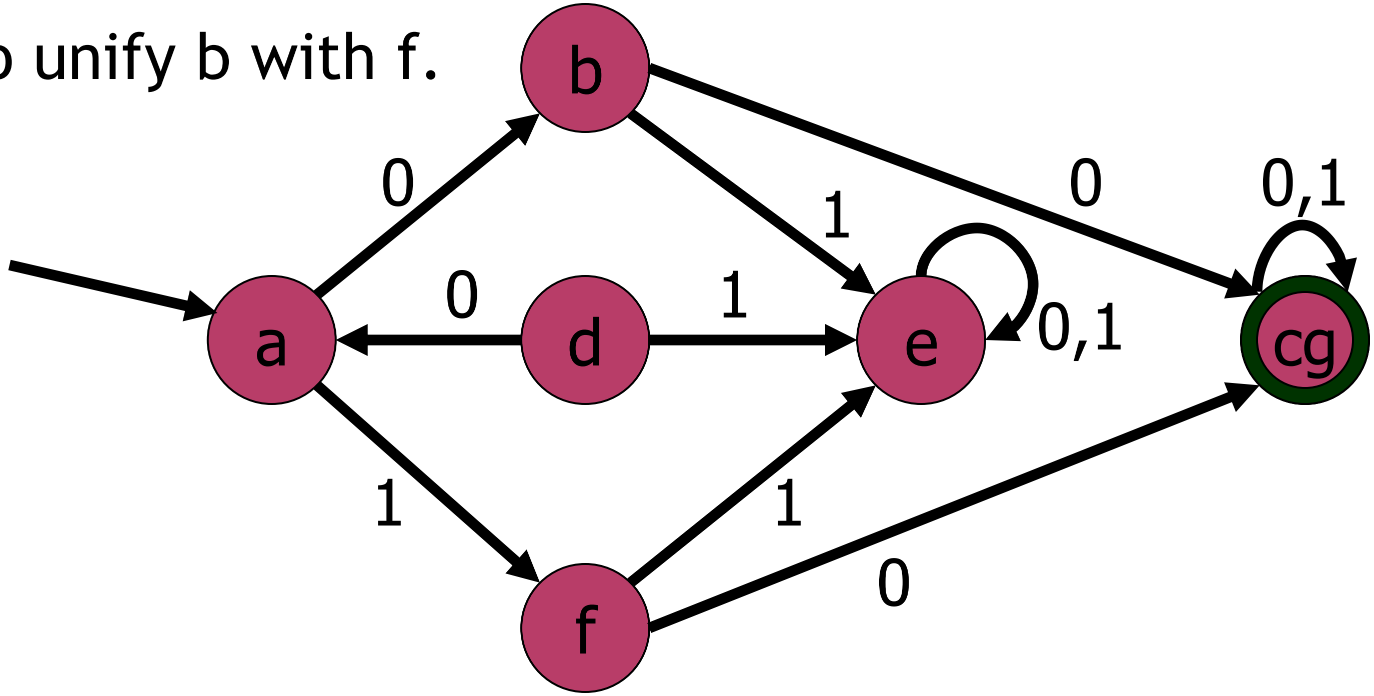


EQUIVALENT STATES. EXAMPLE

A: Yes, b and f. Notice that if you're in b or f then:

1. if string ends, reject in both cases
2. if next character is 0, forever accept in both cases
3. if next character is 1, forever reject in both cases

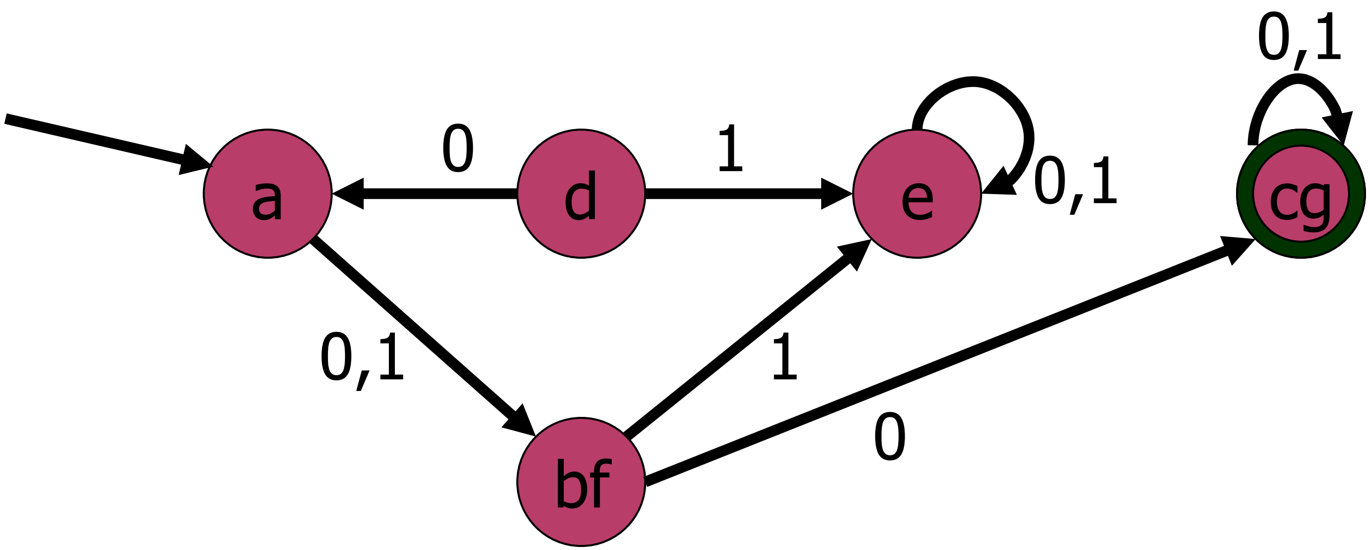
So unify b with f.



EQUIVALENT STATES. EXAMPLE

Intuitively two states are equivalent if all subsequent behavior from those states is the same.

Q: Come up with a formal characterization of state equivalence.



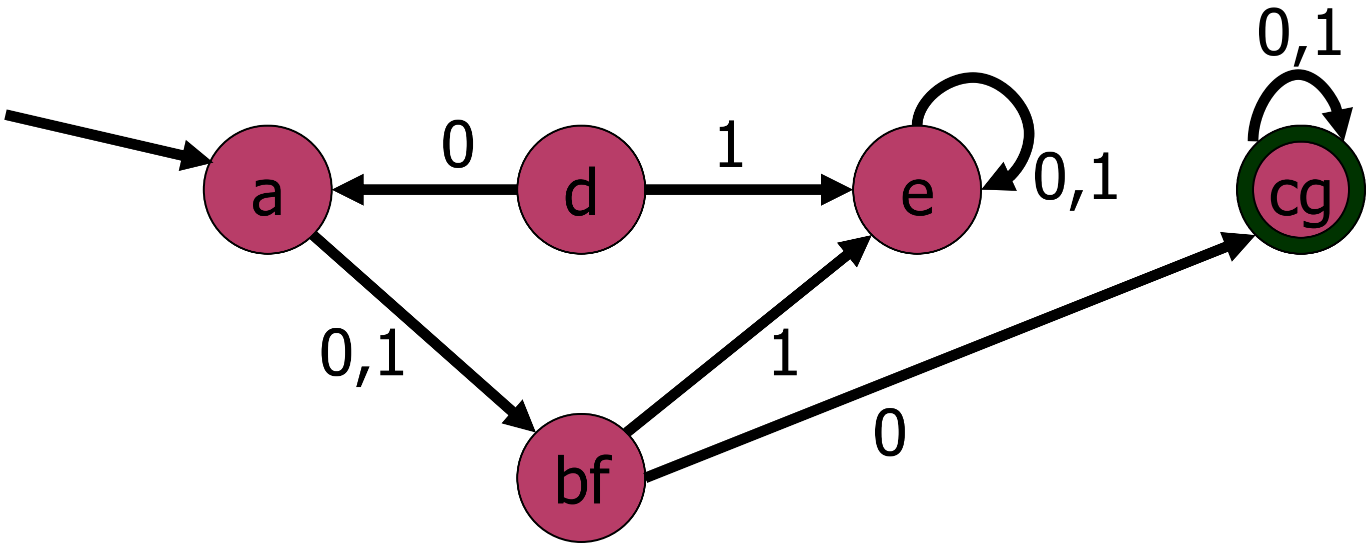
EQUIVALENT STATES. DEFINITION

DEF: Two states q and q' in a DFA $M = (Q, \Sigma, \delta, q_0, F)$ are said to be ***equivalent*** (or ***indistinguishable***) if for all strings $u \in \Sigma^*$, the states on which u ends on when read from q and q' are both accept, or both non-accept.

Equivalent states may be glued together without affecting M's behavior.

FINISHING THE EXAMPLE

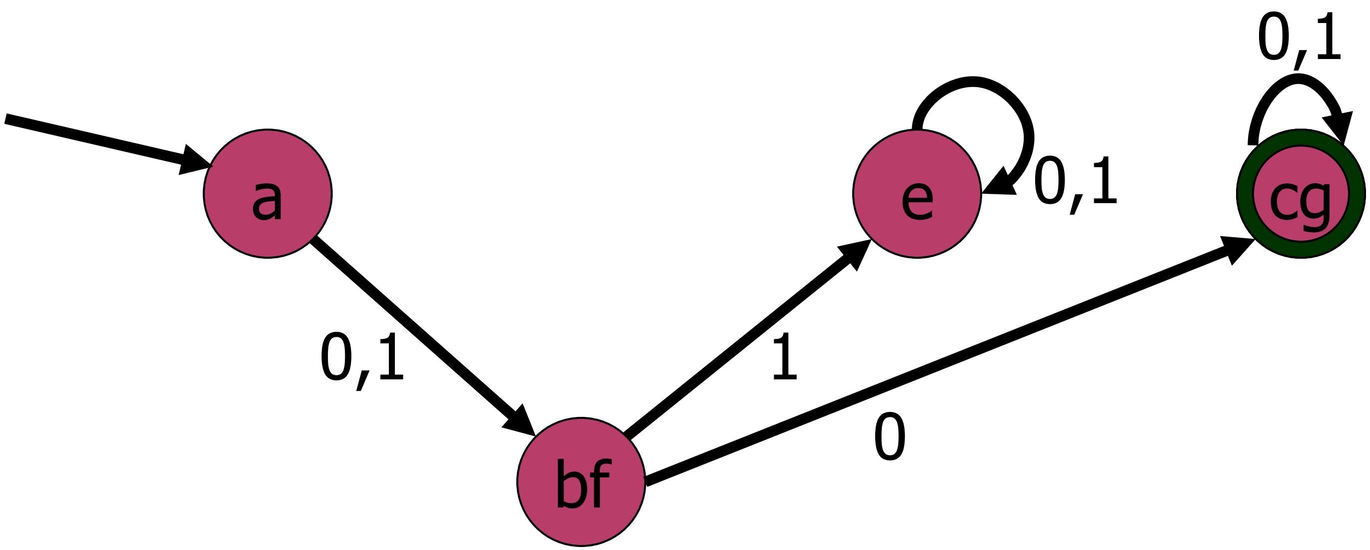
Q: Any other ways to simplify the automaton?



USELESS STATES

A: Get rid of d.

Getting rid of unreachable ***useless states*** doesn't affect the accepted language.



MINIMIZATION ALGORITHM. GOALS

DEF: An automaton is ***irreducible*** if
- it contains no useless states, and
- no two distinct states are equivalent.

The goal of minimization algorithm is to create irreducible automata from arbitrary ones. Later: remarkably, the algorithm actually produces smallest possible DFA for the given language, hence the name "minimization".

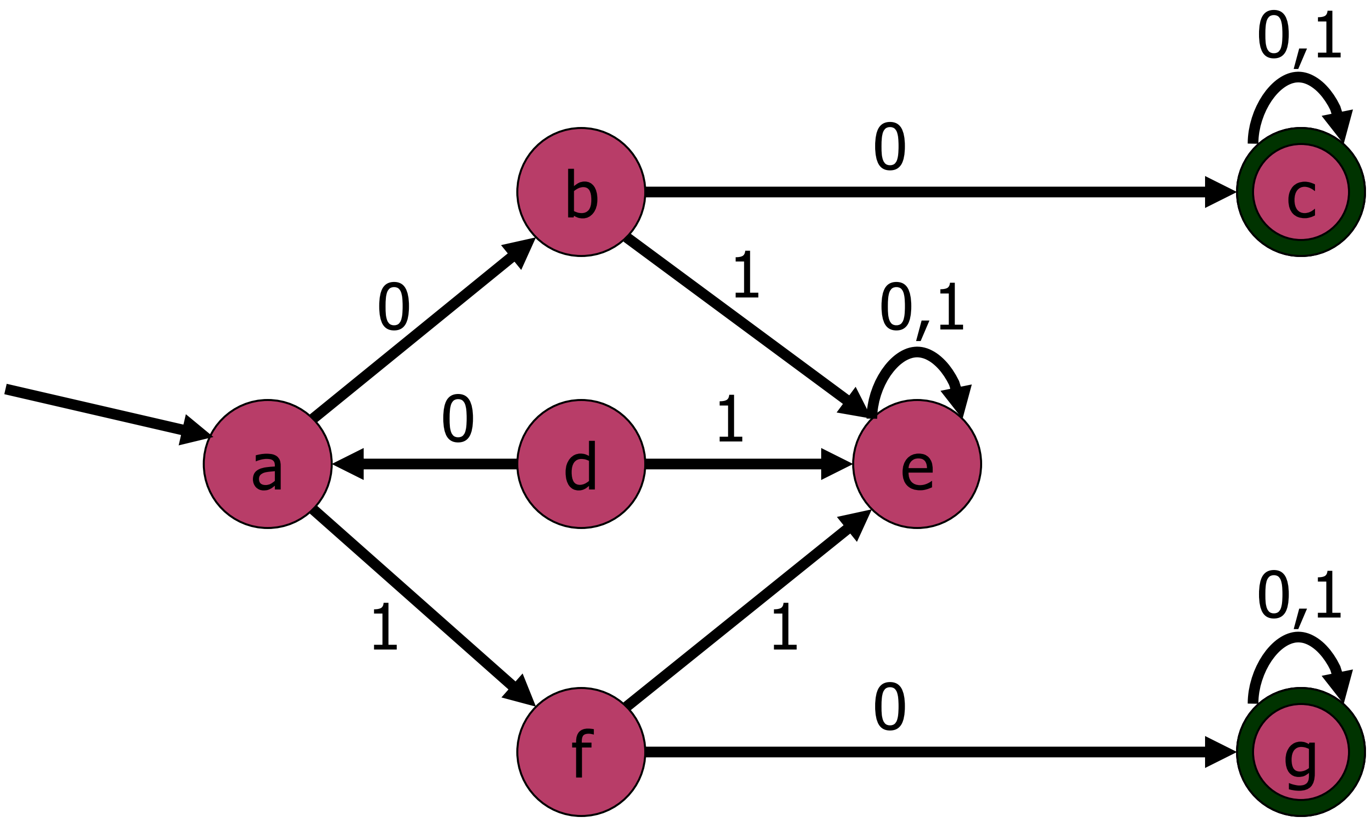
The minimization algorithm *reverses* previous example. Start with least possible number of states, and create new states when forced to.

Explain with a game:

THE GAME OF MINIMIZE

0. All useless players are disqualified.
1. Game proceeds in rounds.
2. Start with 2 teams: ACCEPT vs. REJECT.
3. Each round consists of sub-rounds –one sub-round per team.
4. Two members of a team are said to **agree** if for a given label, they want to pass the buck to same team. Otherwise, **disagree**.
5. During a sub-round, disagreeing members split off into new maximally agreeing teams.
6. If a round passes with no splits, STOP.

THE GAME OF MINIMIZE



MINIMIZATION ALGORITHM. (PARTITION REFINEMENT) CODE

```
DFA minimize(DFA (Q, Σ, δ, q0, F ) )
  remove any state q unreachable from q0
  Partition P = {F, Q - F }
  boolean Consistent = false
  while( Consistent == false )
    Consistent = true
    for(every Set S ∈ P, char a ∈ Σ, Set T ∈ P )
          Set temp = {q ∈T | δ(q,a) ∈S }
          if (temp != Ø  && temp != T )
            Consistent = false
            P = (P –T )∪{temp,T–temp}
  return defineMinimizor( (Q, Σ, δ, q0, F ), P )
```

MINIMIZATION ALGORITHM. (PARTITION REFINEMENT) CODE

DFA defineMinimizor

(DFA (Q, Σ, δ, q_0, F), Partition P)

Set $Q' = P$

State q'_0 = the set in P which contains q_0

$F' = \{ S \in P \mid S \subseteq F \}$

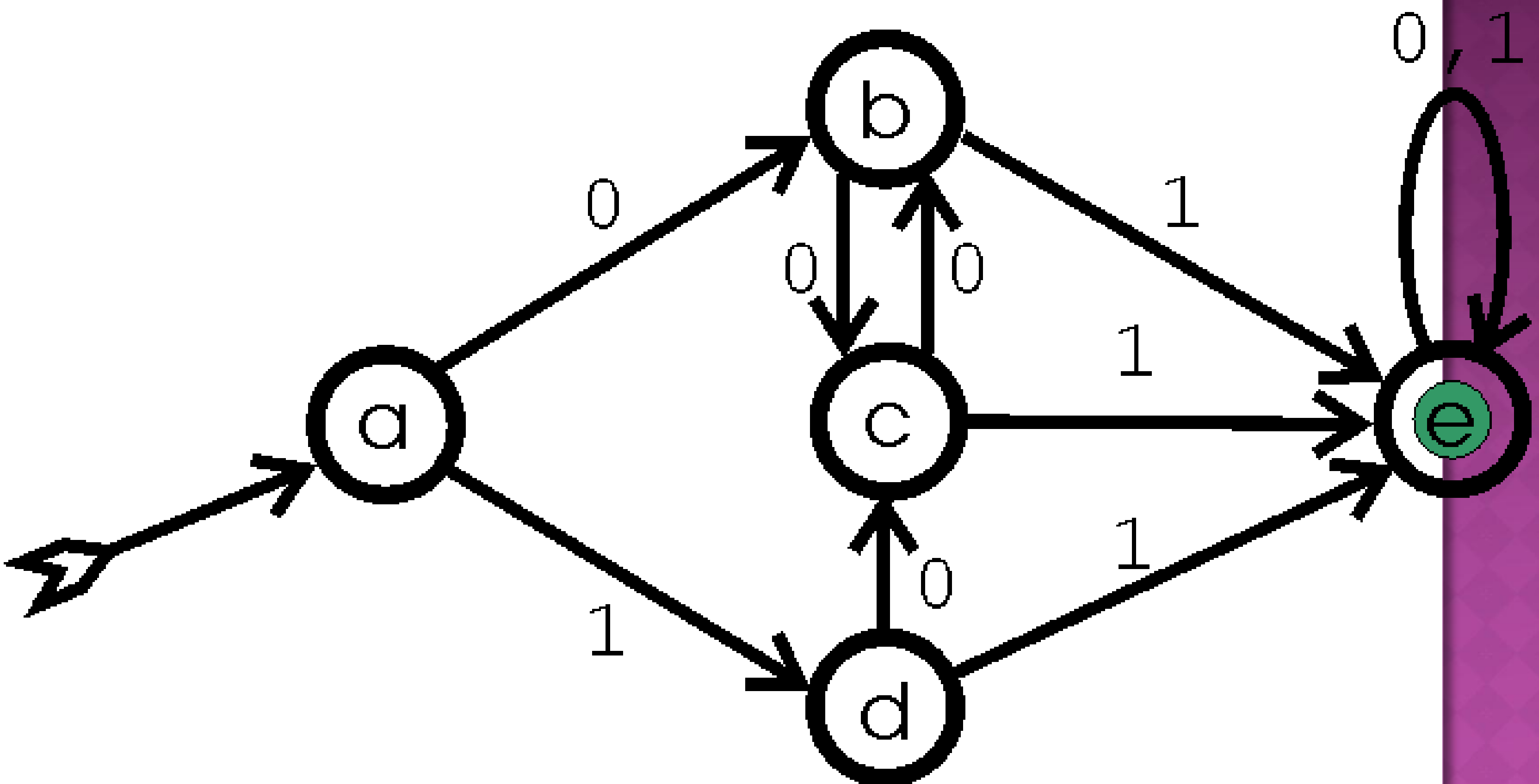
for (each $S \in P$, $a \in \Sigma$)

define $\delta'(S,a)$ = the set $T \in P$ which contains the states $\delta'(S,a)$

return (Q', Σ, δ', q'_0, F')

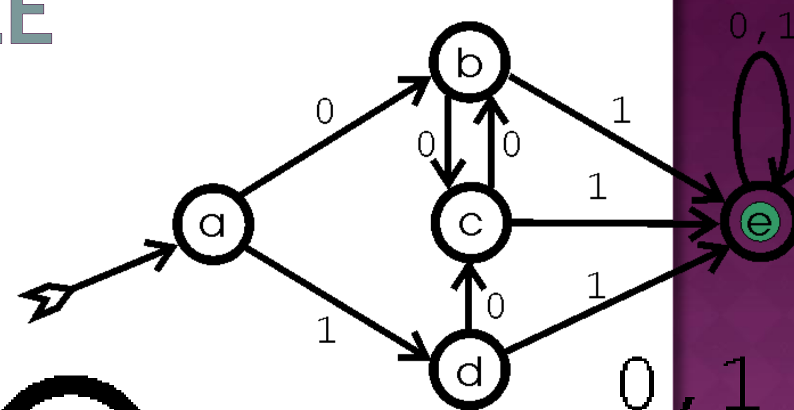
MINIMIZATION EXAMPLE

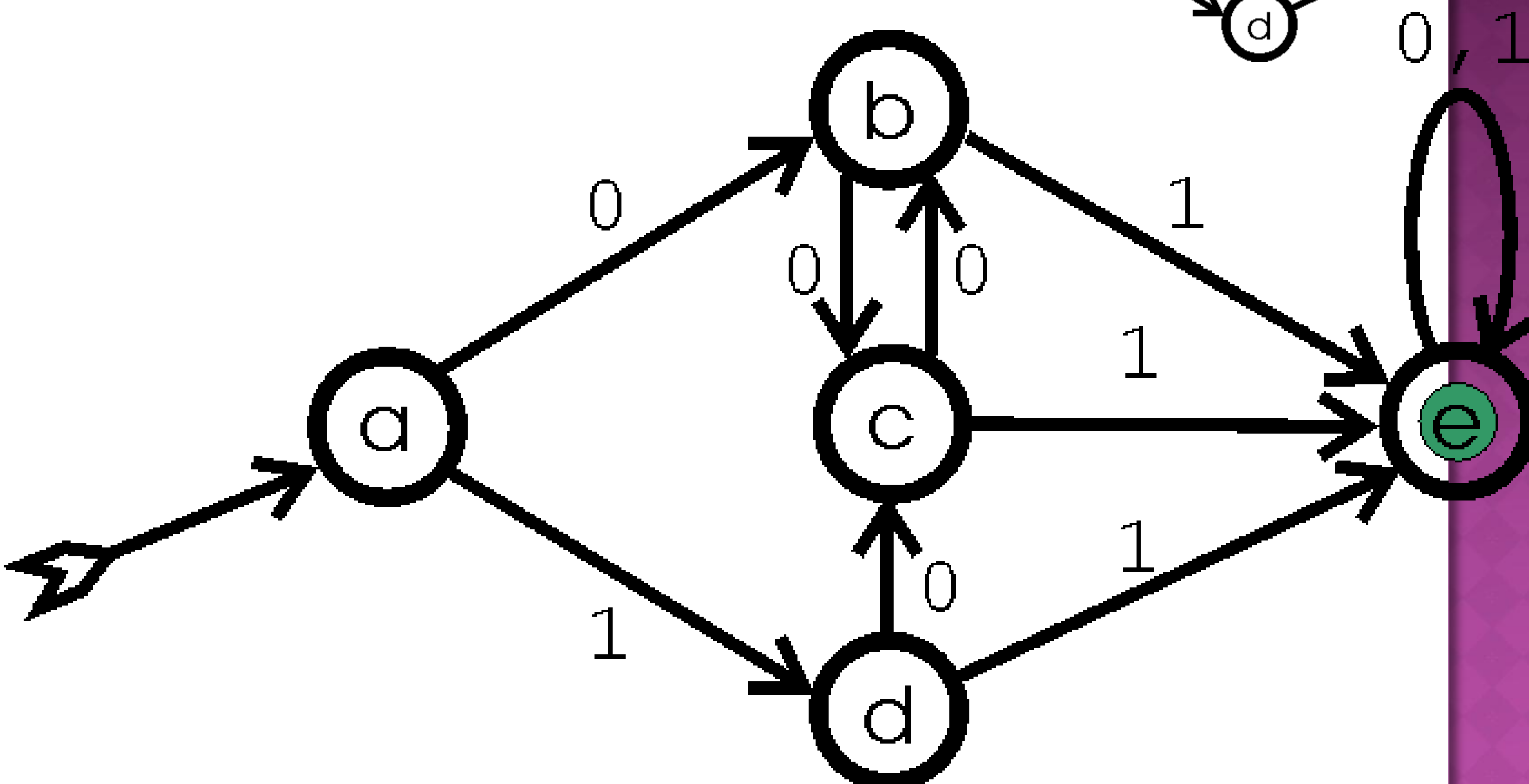
Start with a DFA



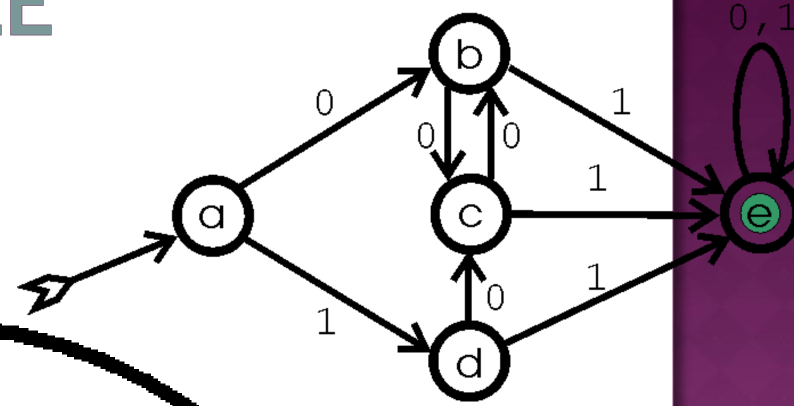
MINIMIZATION EXAMPLE

Miniature version →





MINIMIZATION EXAMPLE

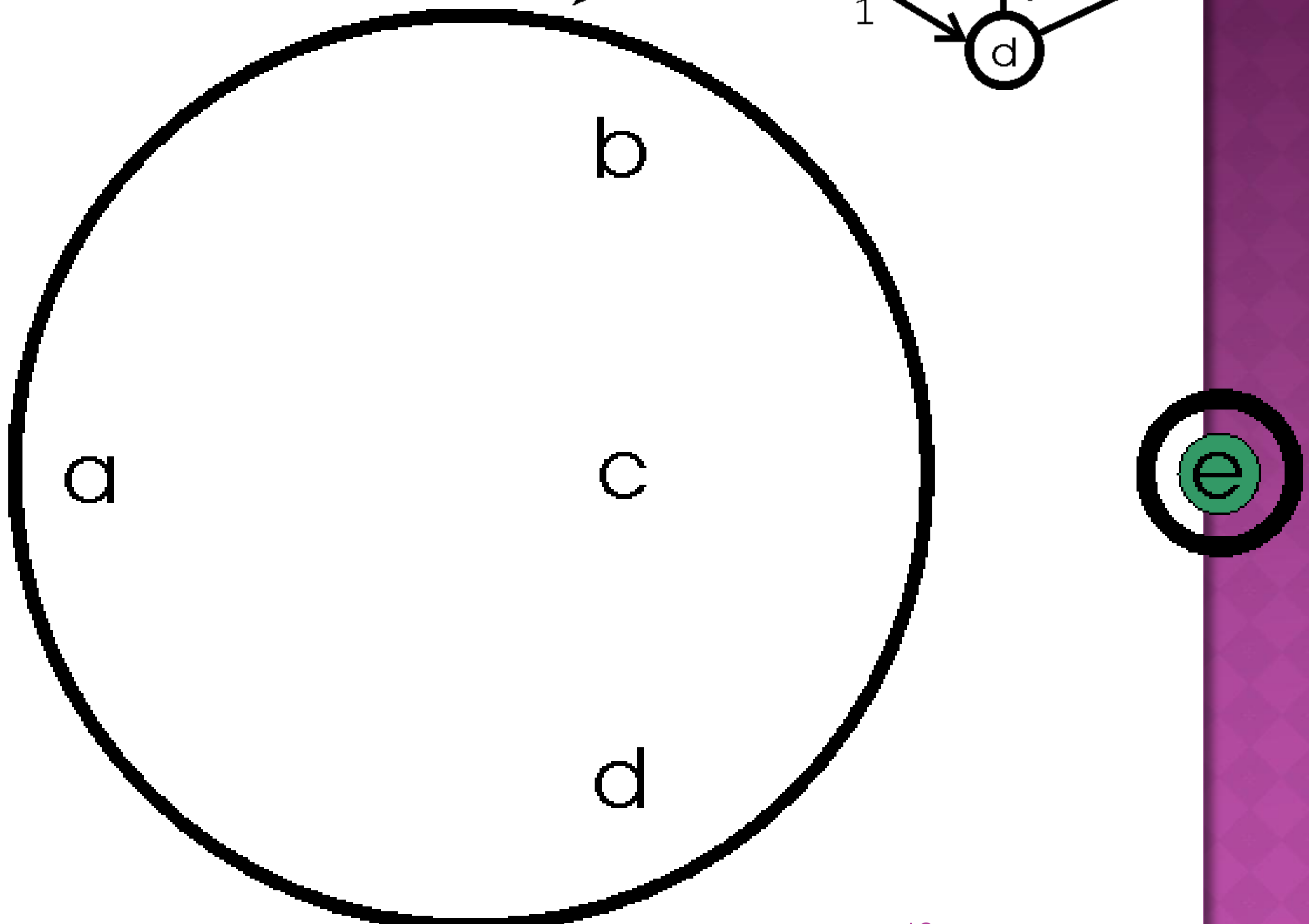


Split into two teams.

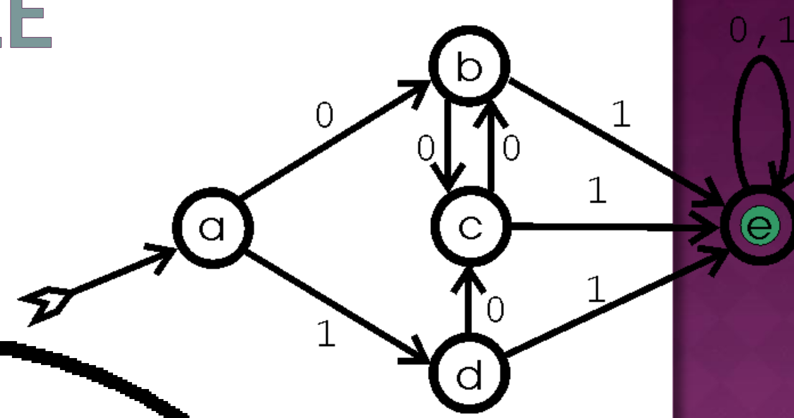
ACCEPT

vs.

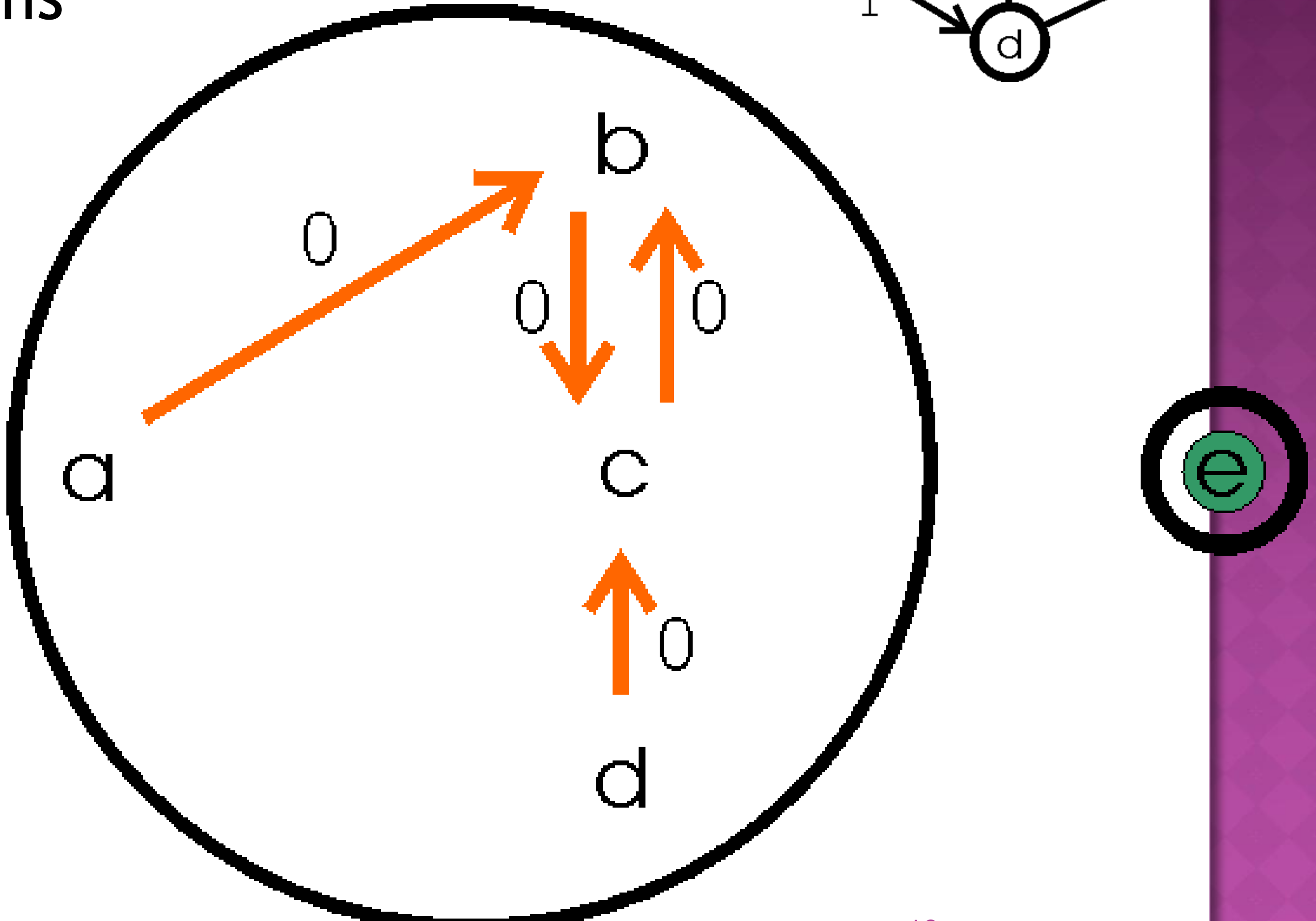
REJECT



MINIMIZATION EXAMPLE

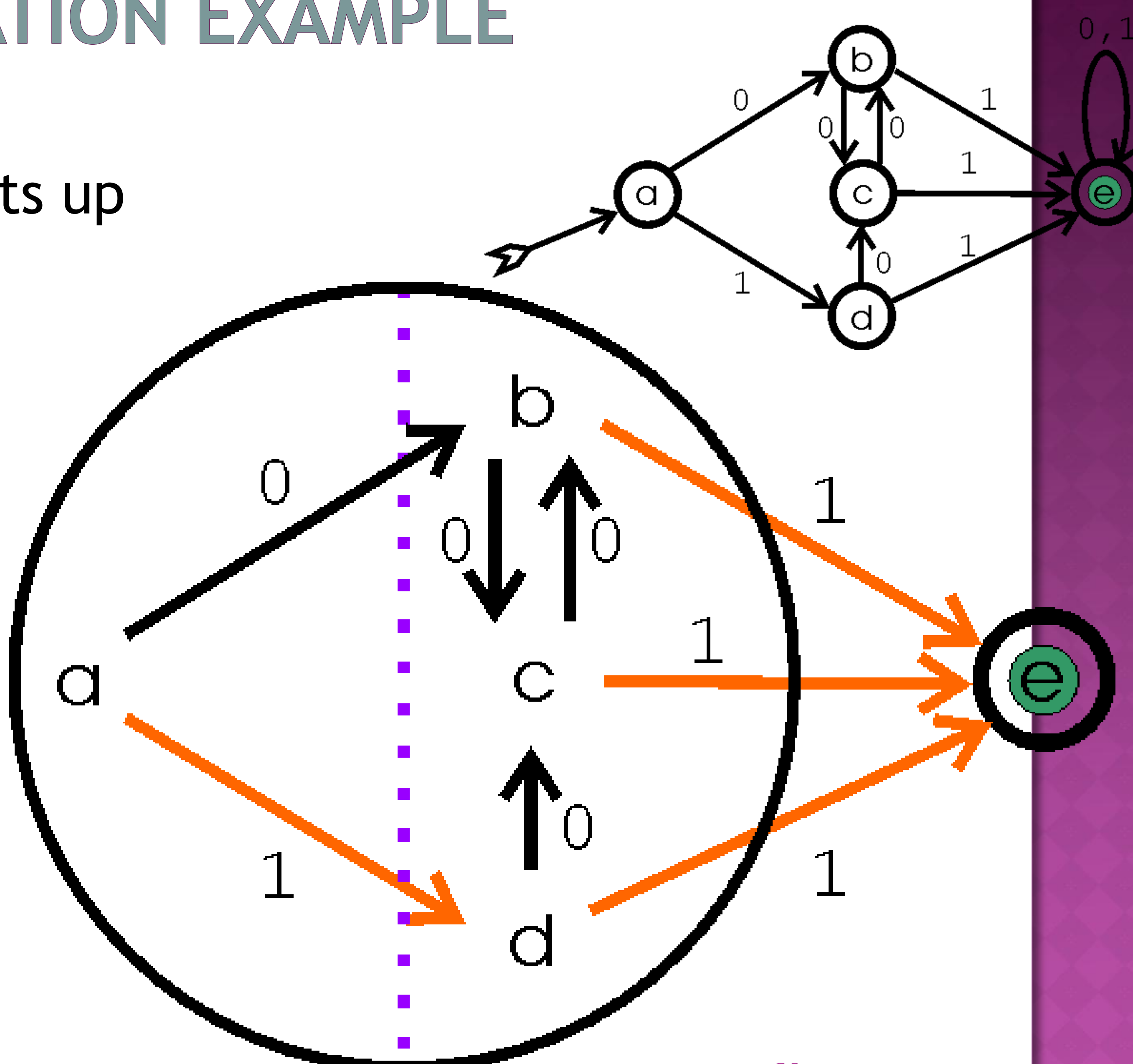


0-label doesn't split up any teams

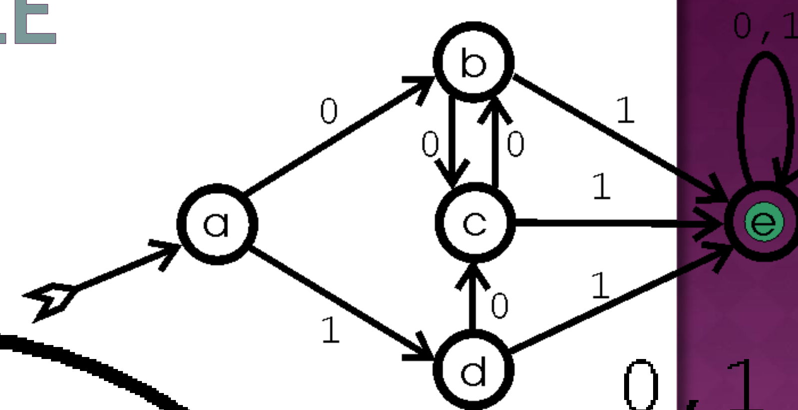


MINIMIZATION EXAMPLE

1-label splits up REJECT's

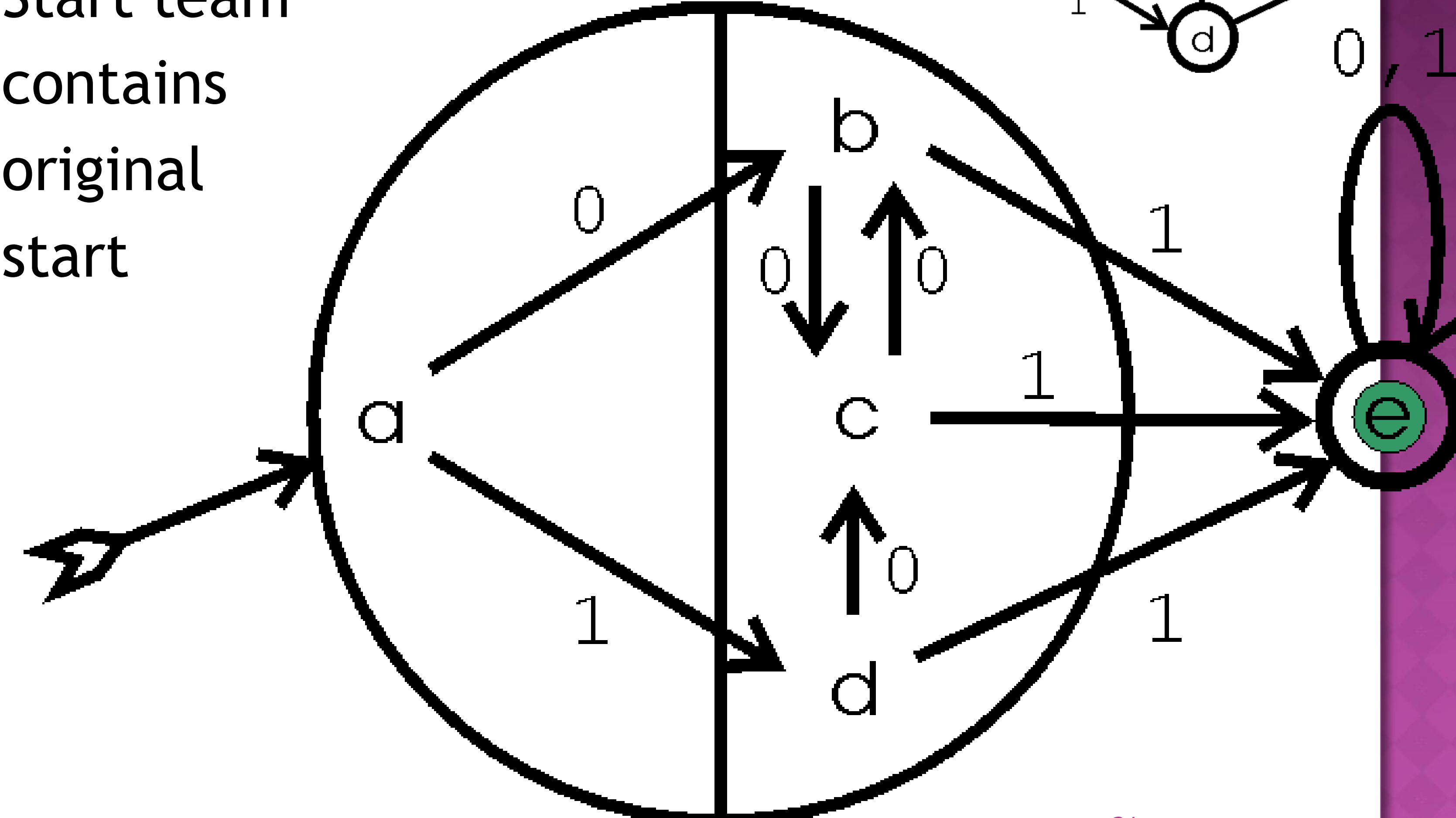


MINIMIZATION EXAMPLE



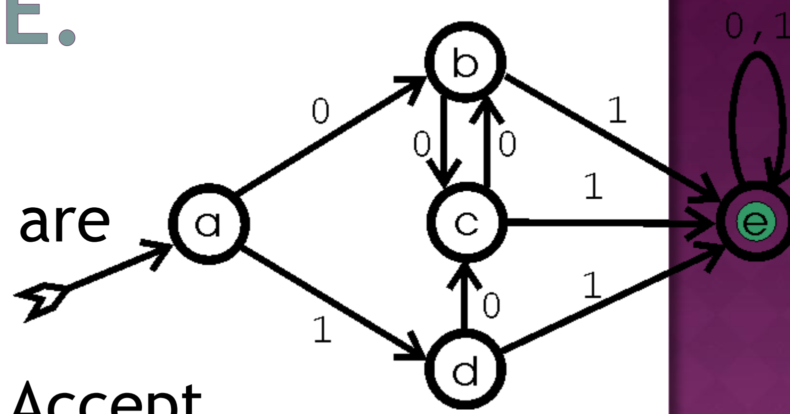
No further splits. HALT!

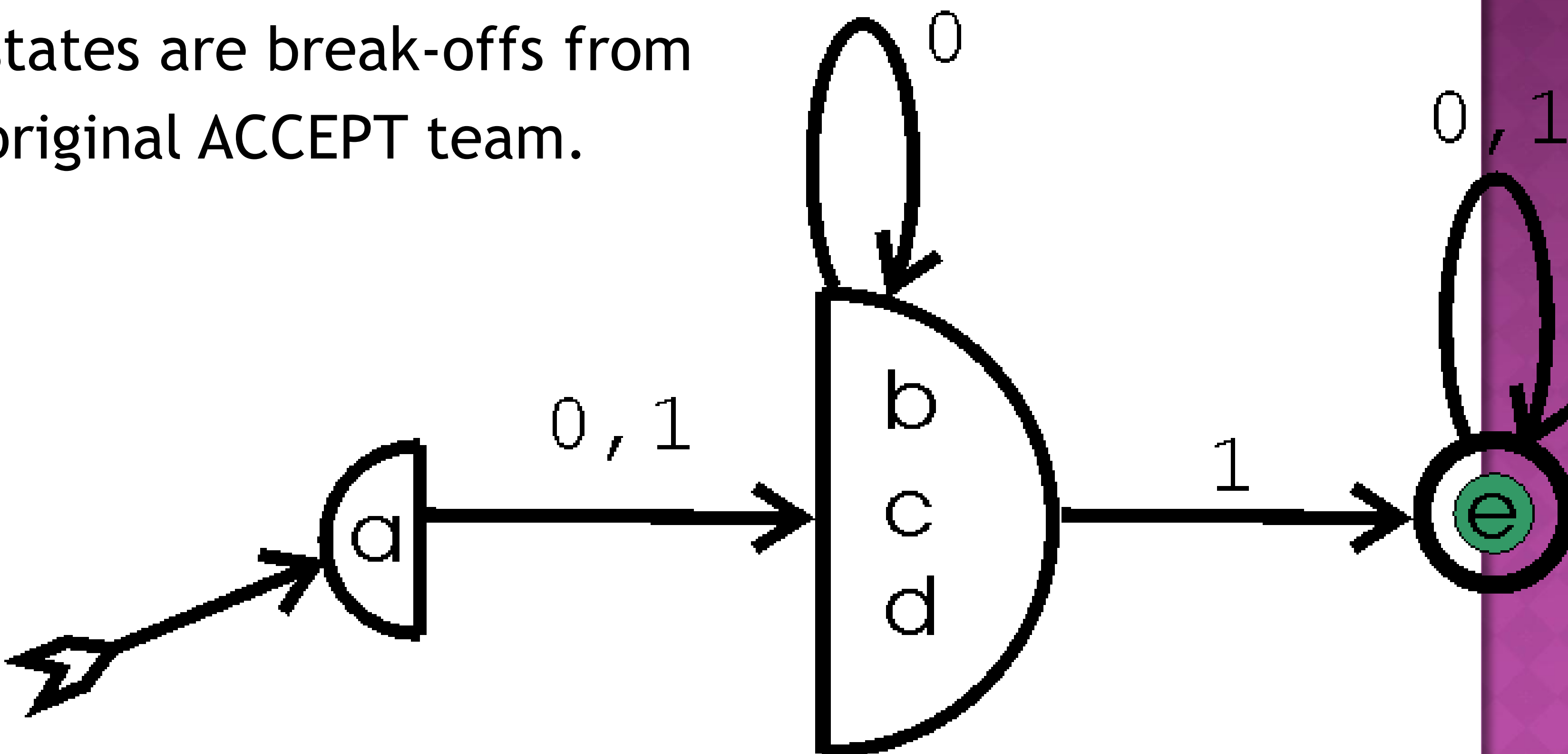
Start team contains original start



MINIMIZATION EXAMPLE. END RESULT

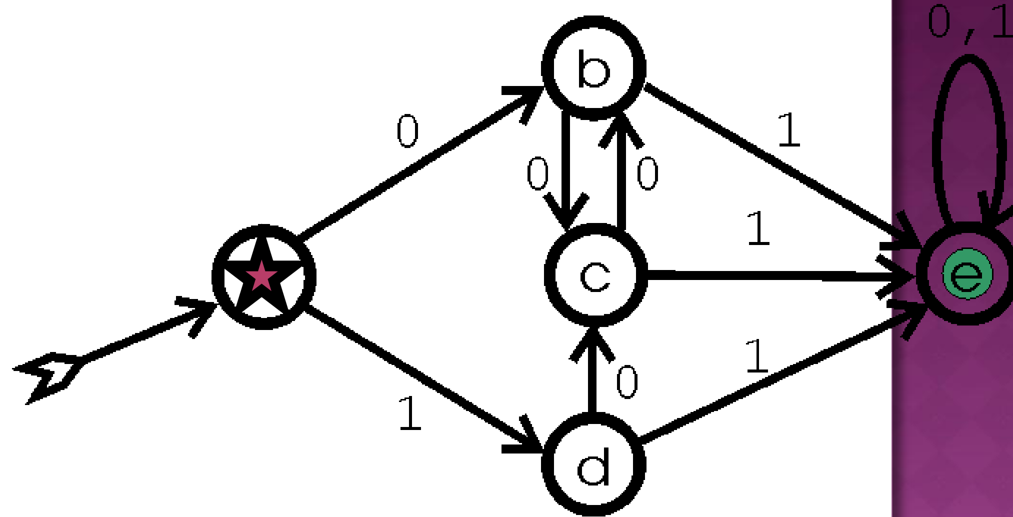
States of the minimal automata are remaining teams. Edges are consolidated across each team. Accept states are break-offs from original ACCEPT team.

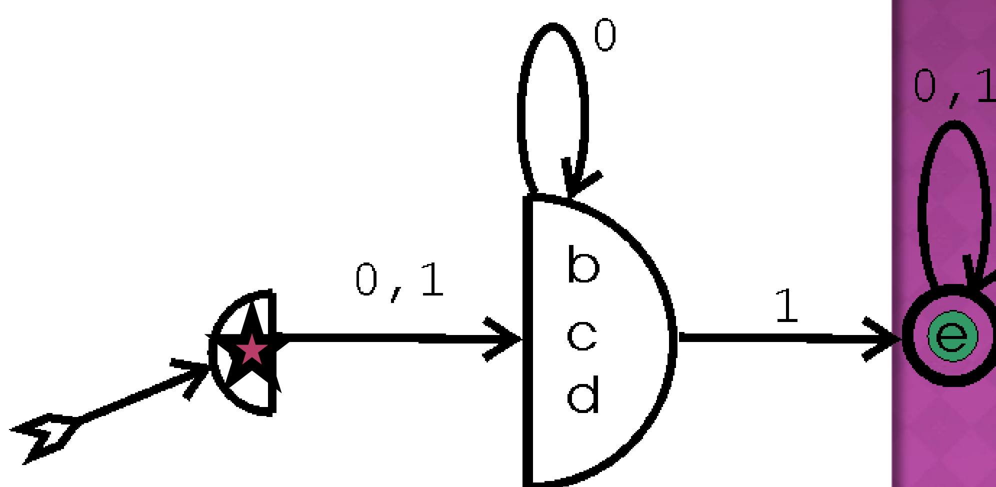




MINIMIZATION EXAMPLE. COMPARE

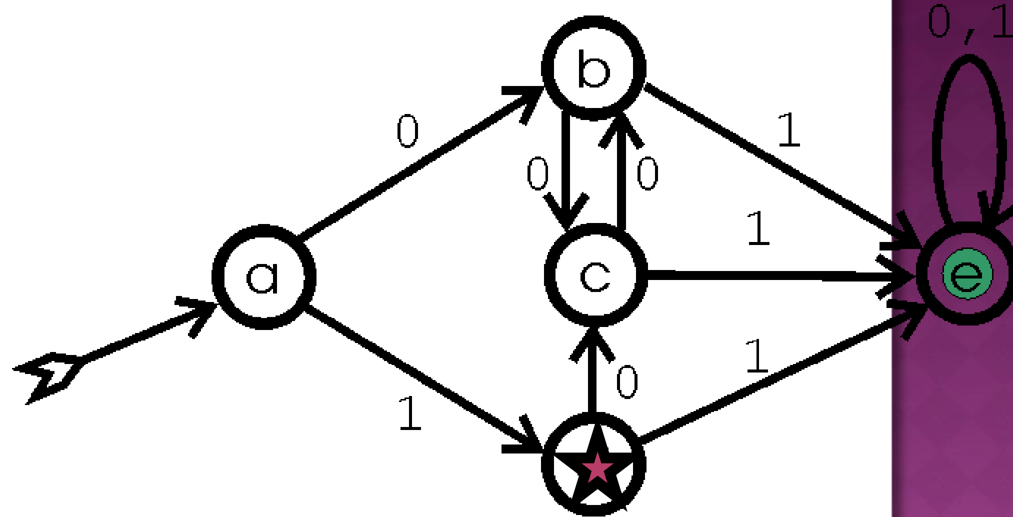
100100101

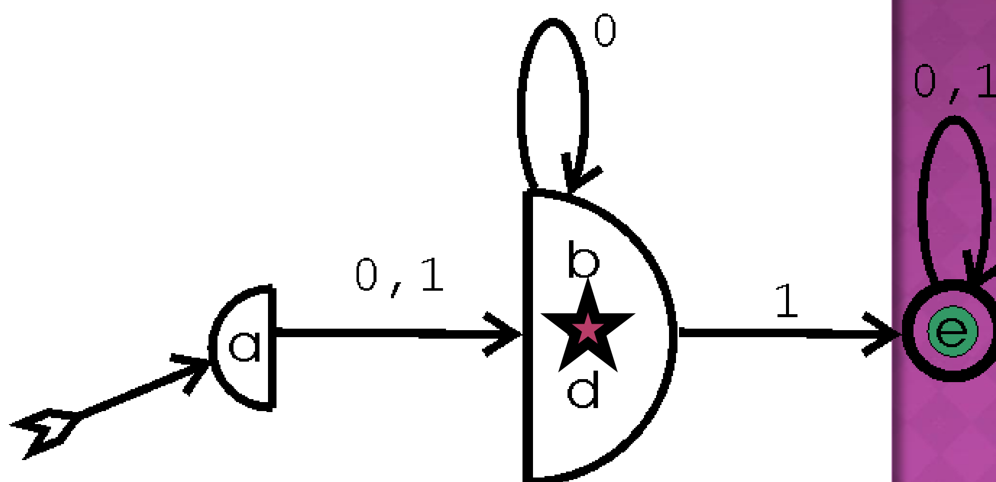




MINIMIZATION EXAMPLE. COMPARE

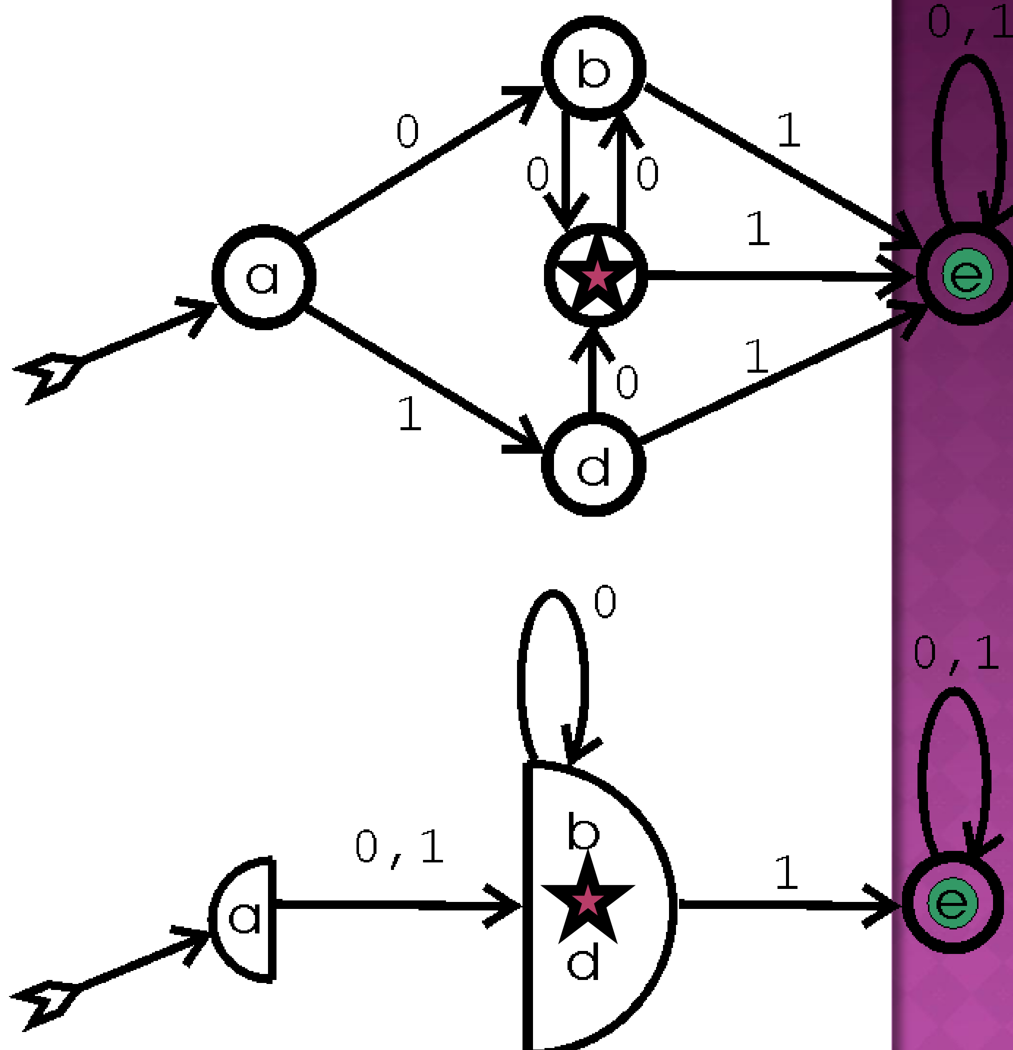
100100101





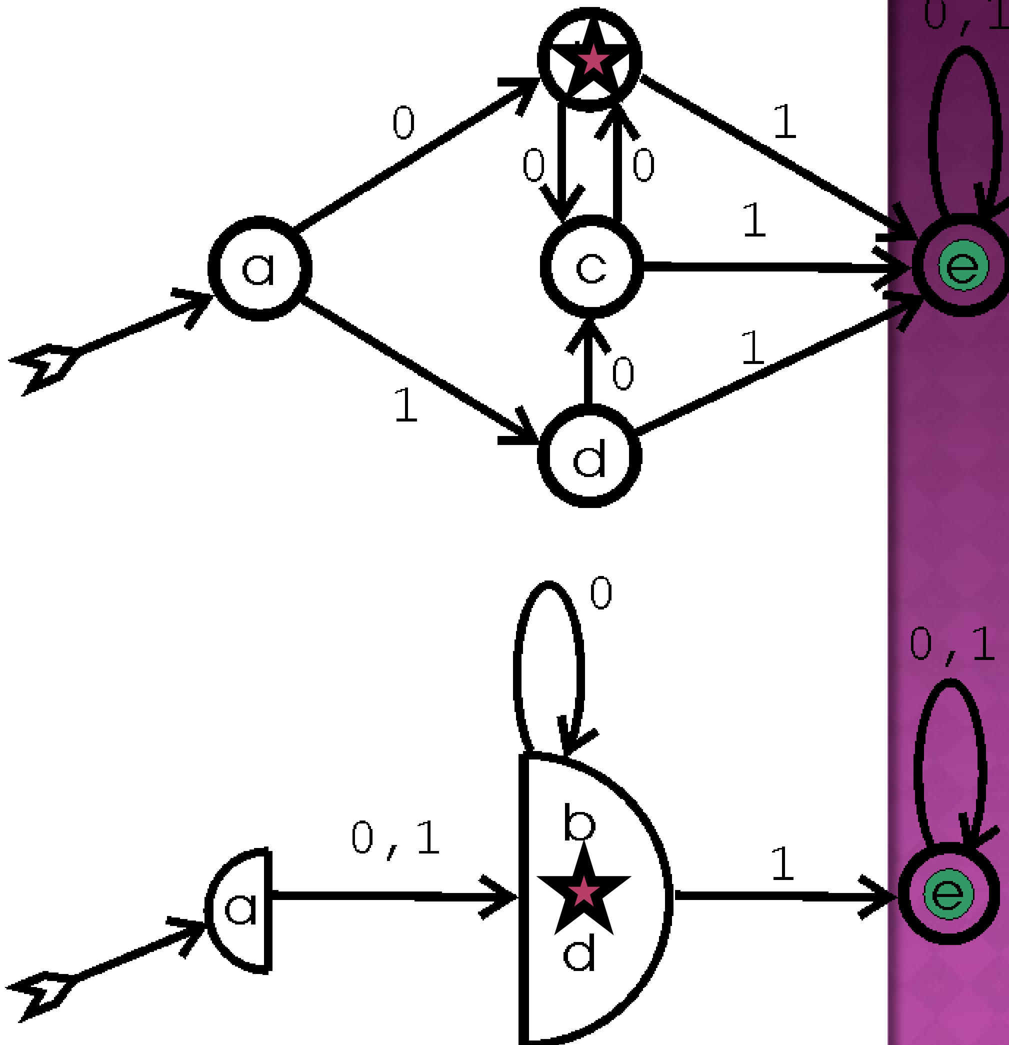
MINIMIZATION EXAMPLE. COMPARE

100100101



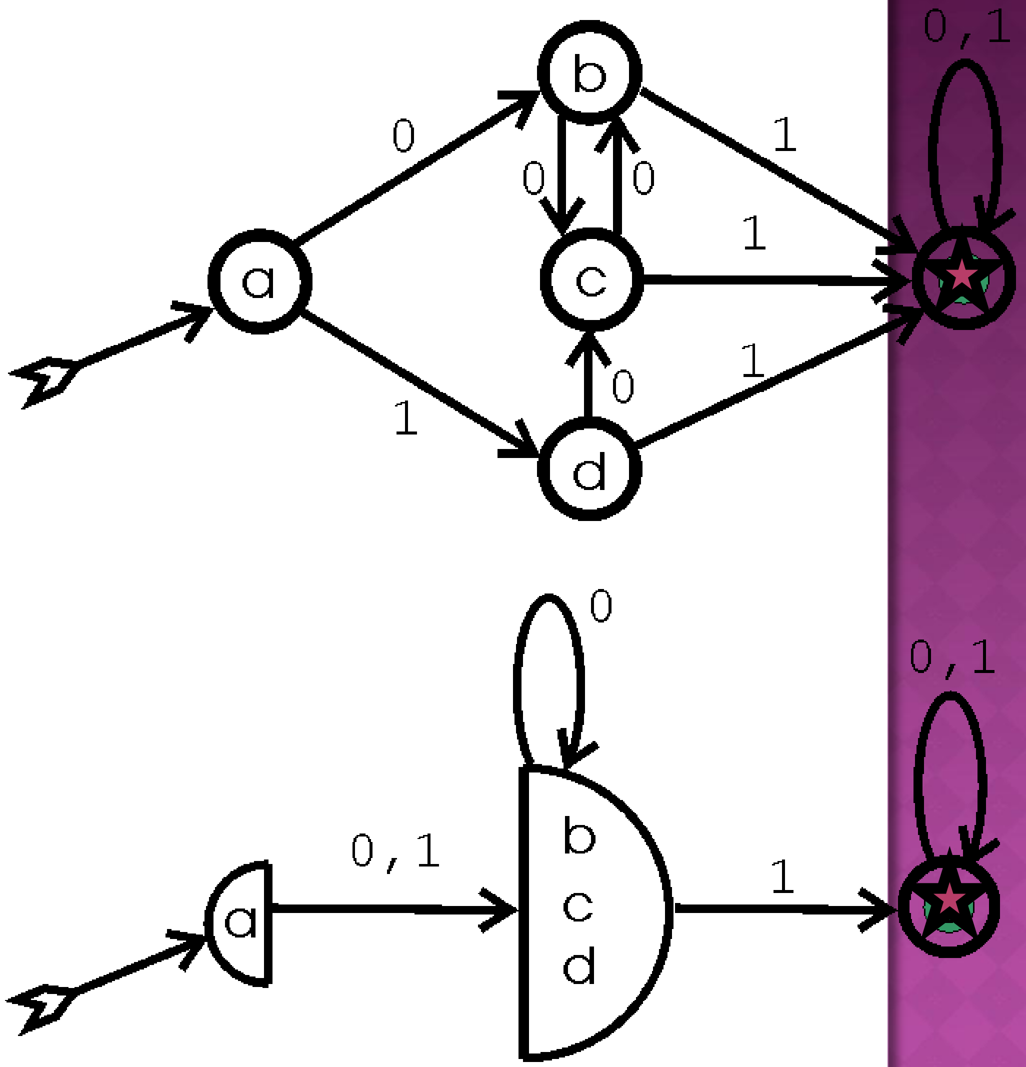
MINIMIZATION EXAMPLE. COMPARE

100100101



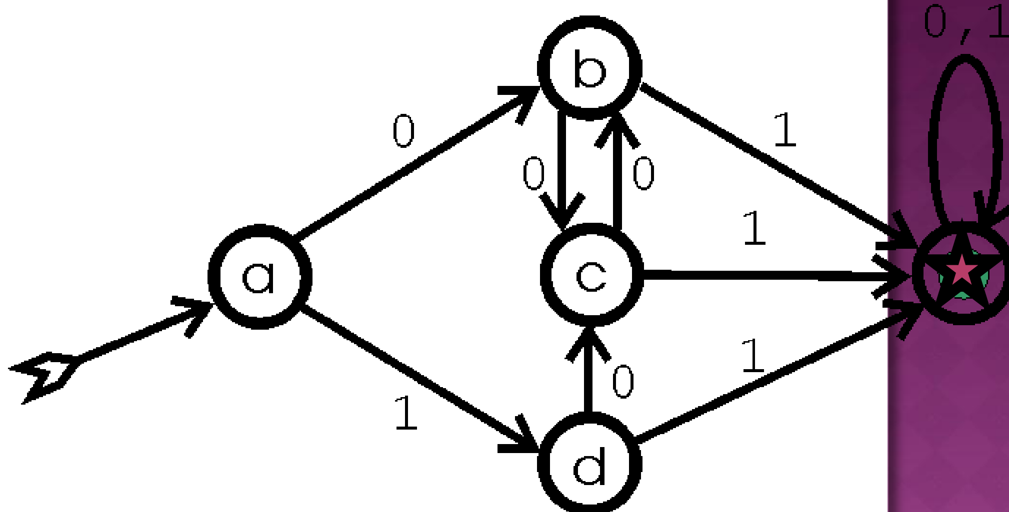
MINIMIZATION EXAMPLE. COMPARE

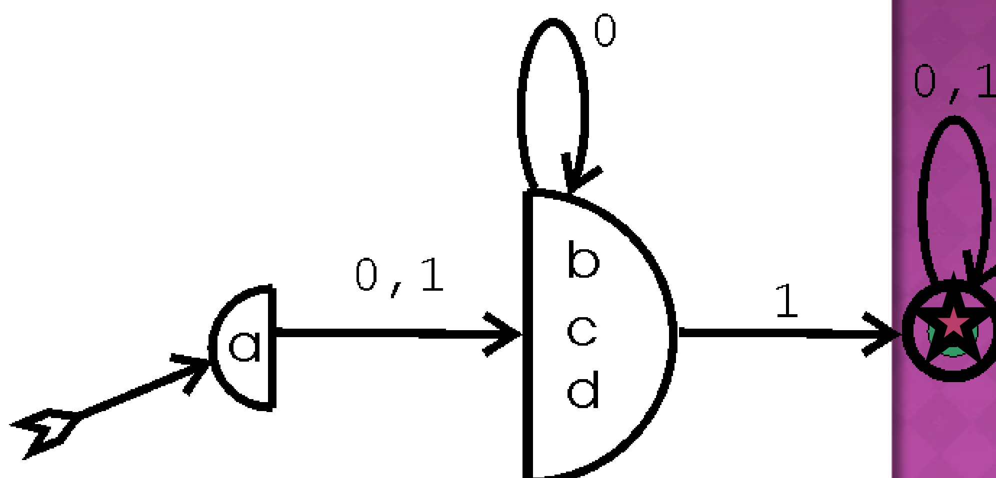
100100101



MINIMIZATION EXAMPLE. COMPARE

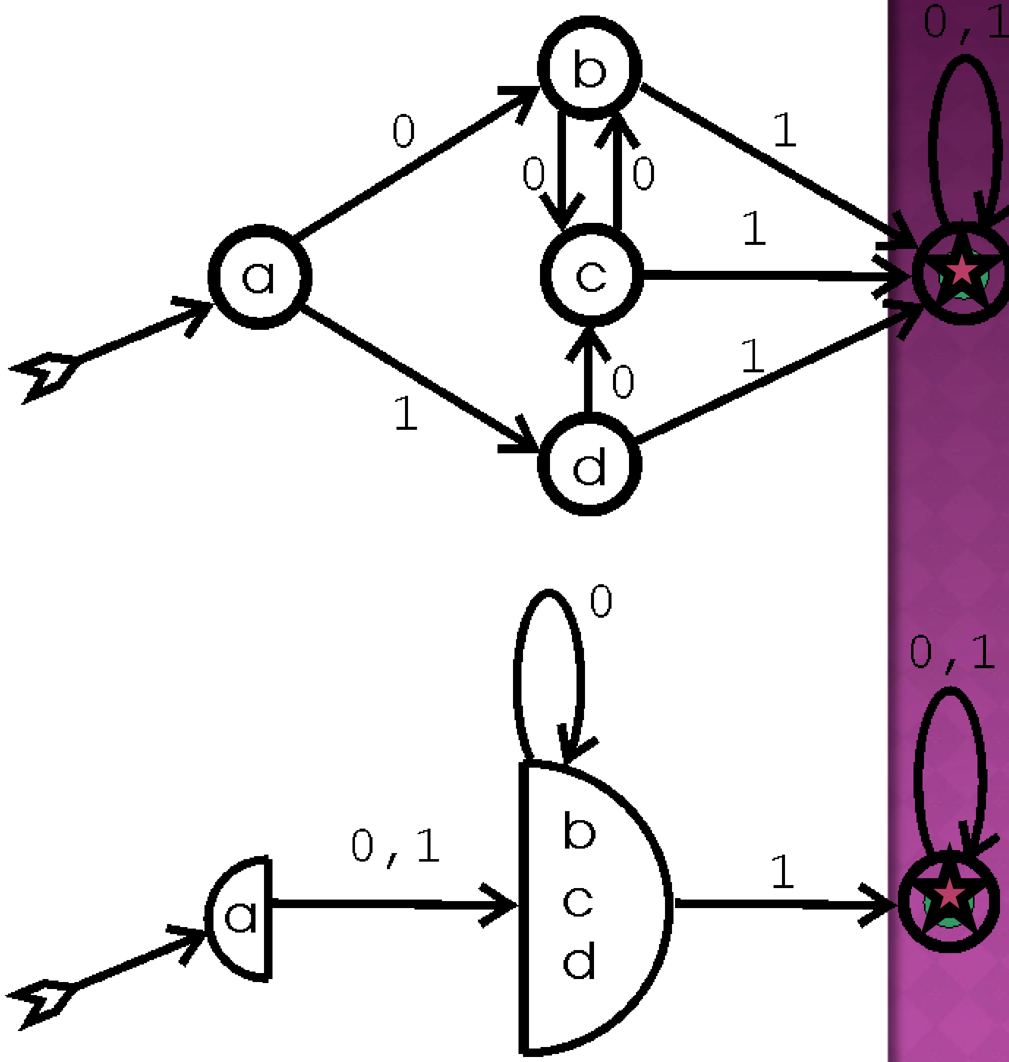
100100101





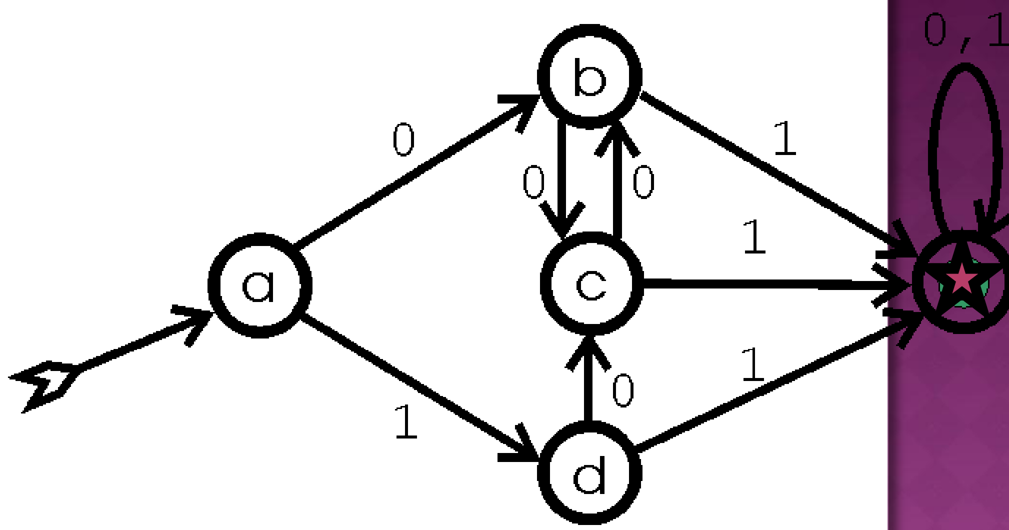
MINIMIZATION EXAMPLE. COMPARE

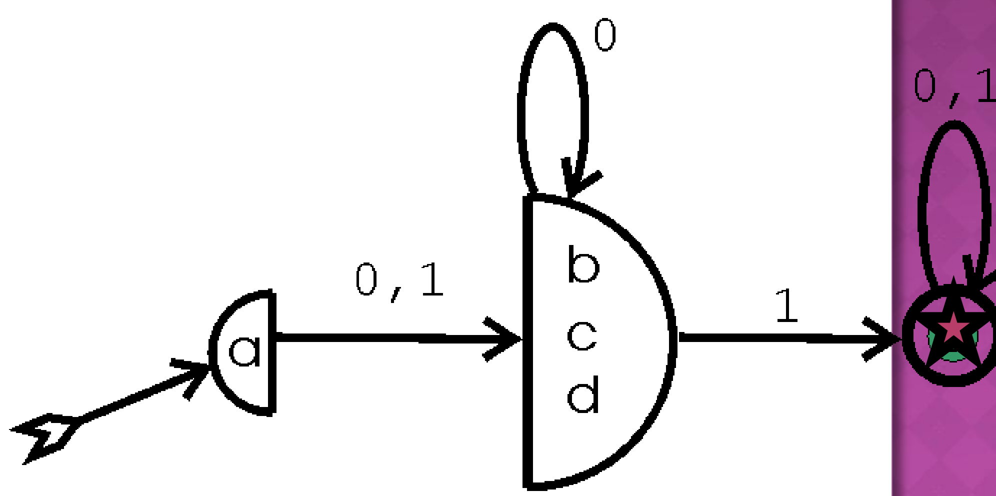
100100101



MINIMIZATION EXAMPLE. COMPARE

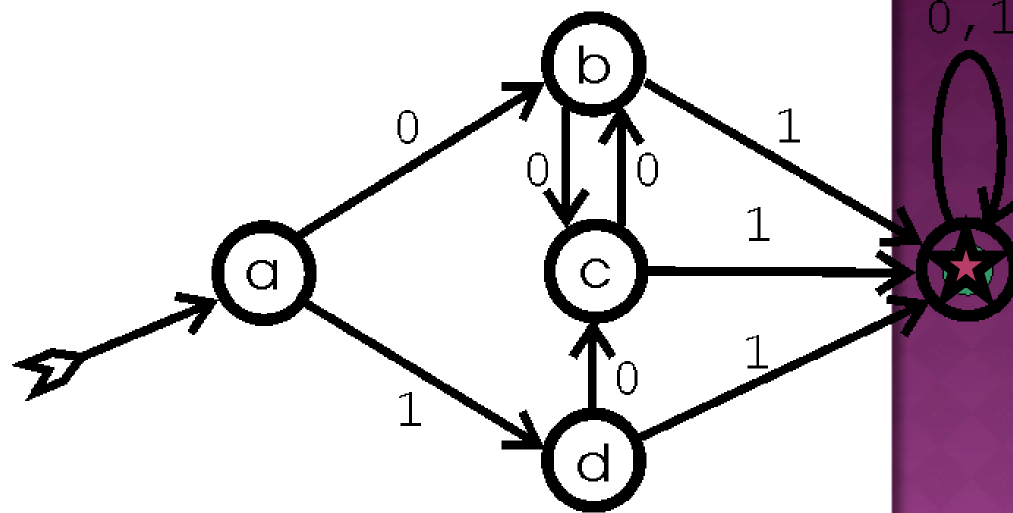
100100101

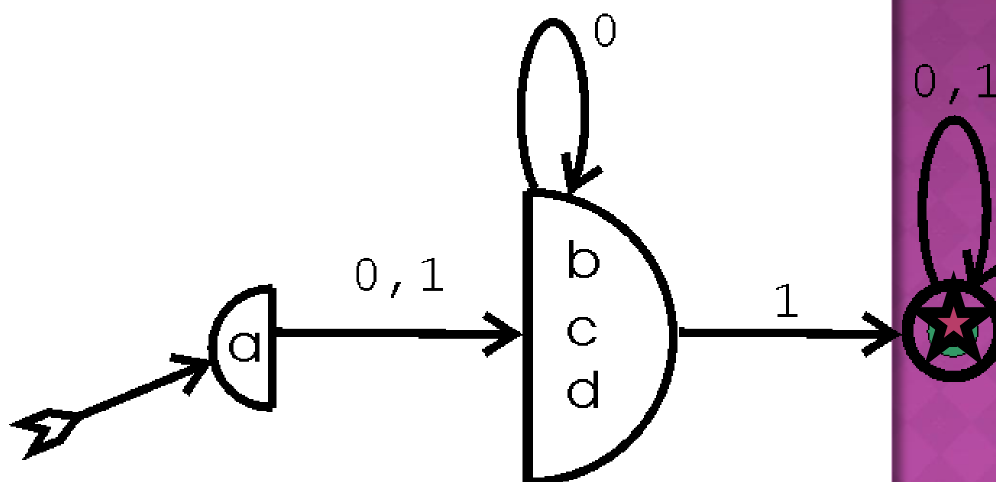




MINIMIZATION EXAMPLE. COMPARE

100100101

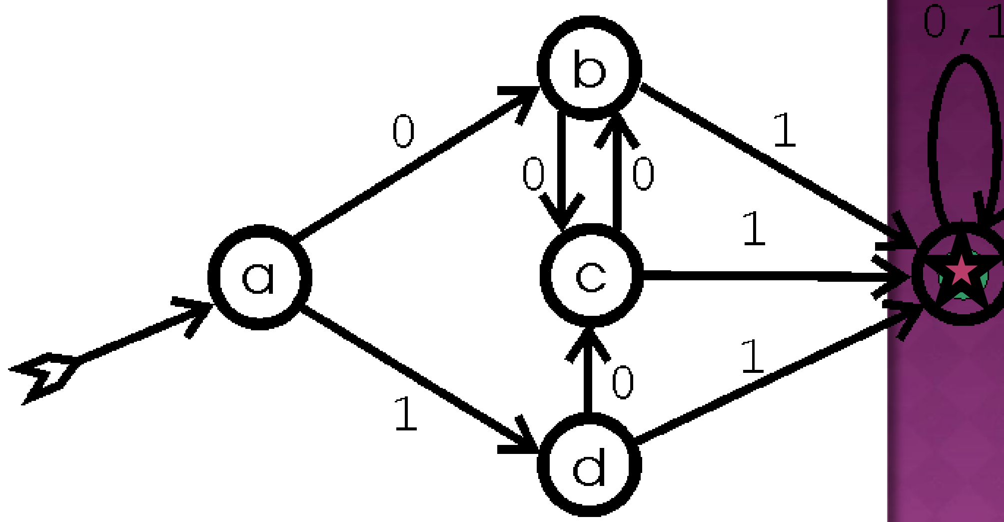


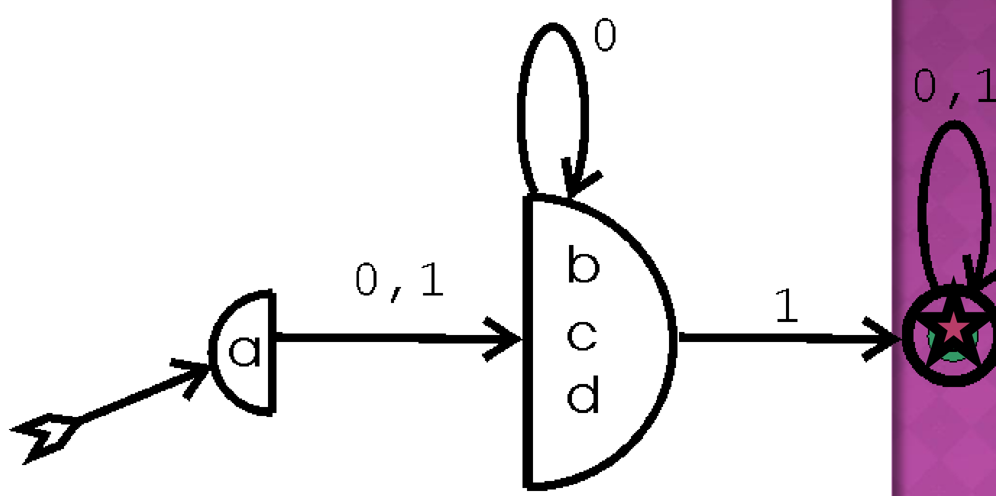


MINIMIZATION EXAMPLE. COMPARE

100100101

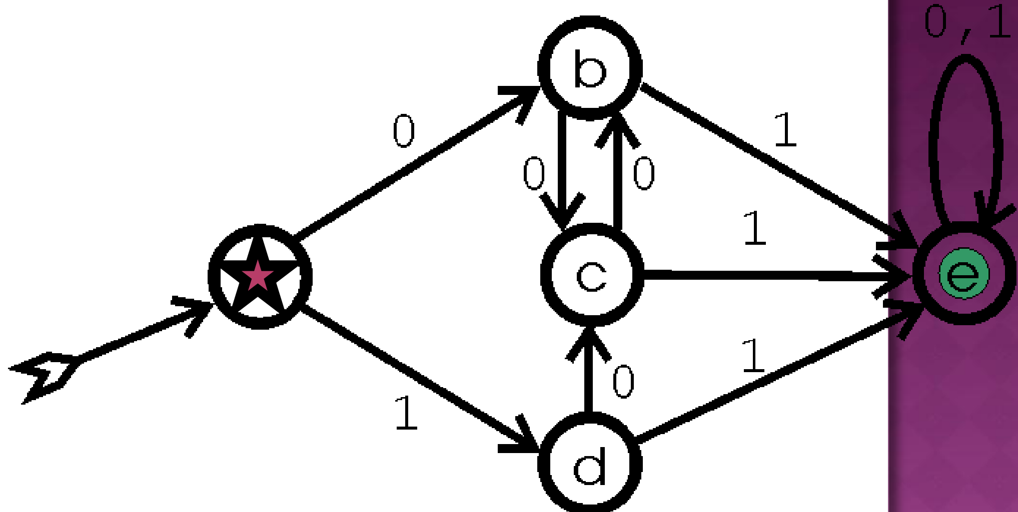
ACCEPTED.

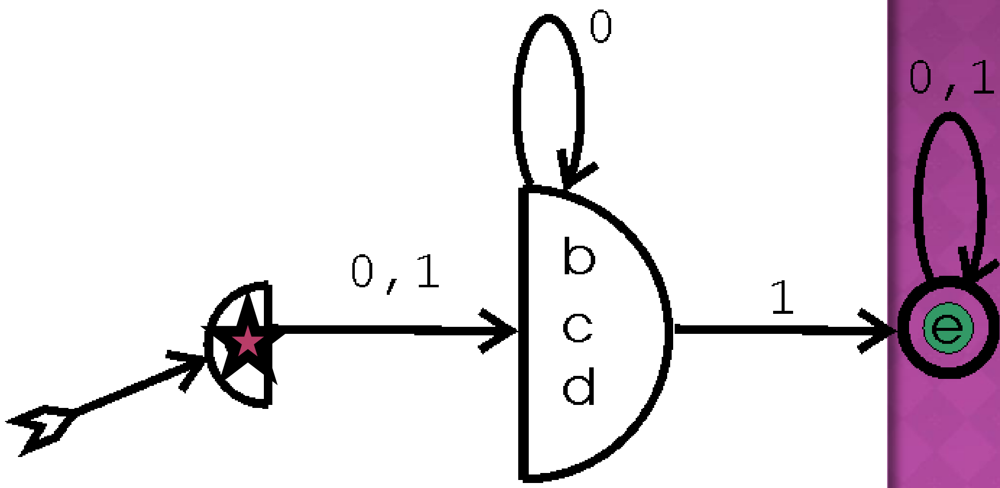




MINIMIZATION EXAMPLE. COMPARE

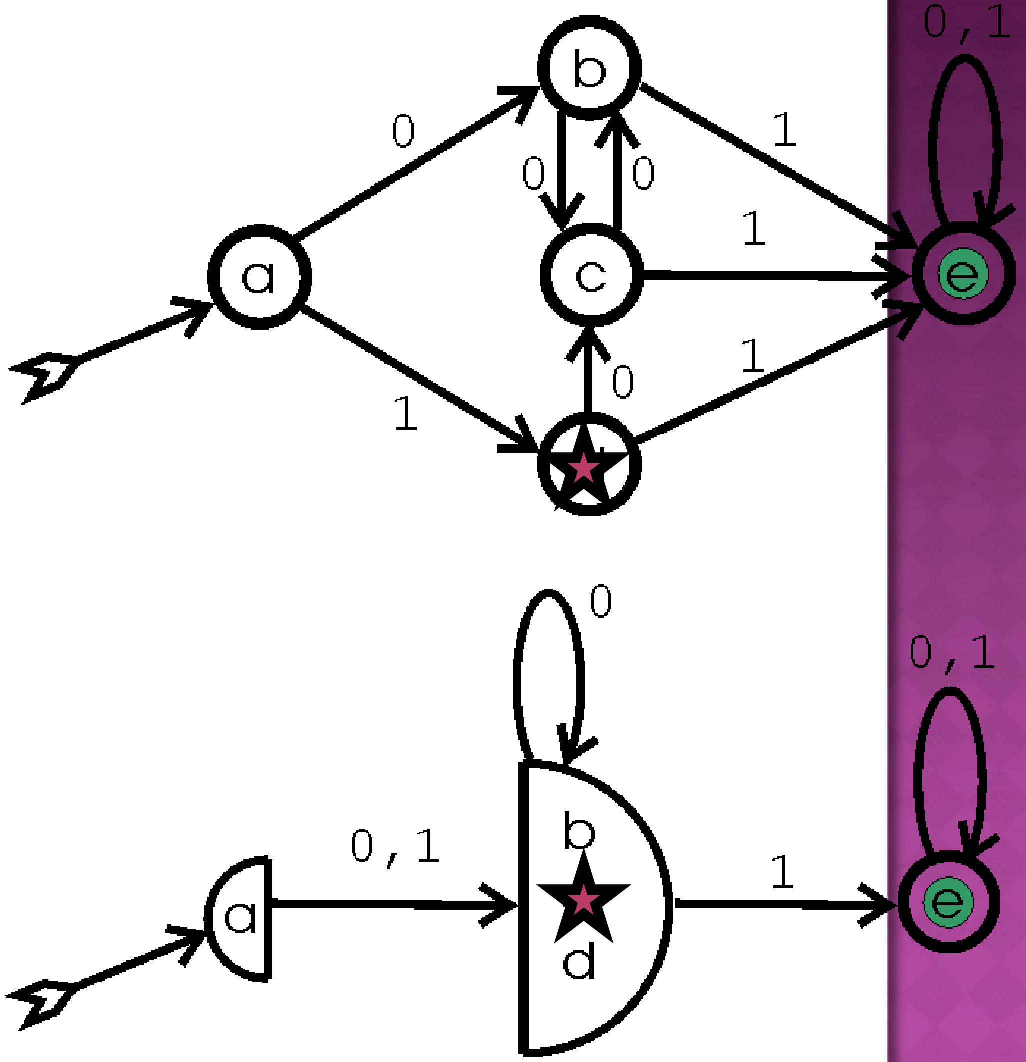
10000





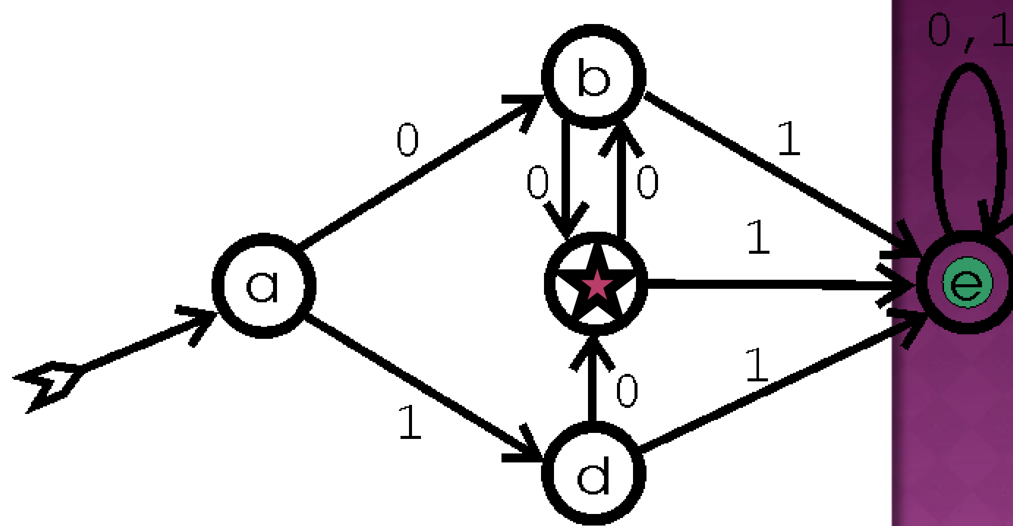
MINIMIZATION EXAMPLE. COMPARE

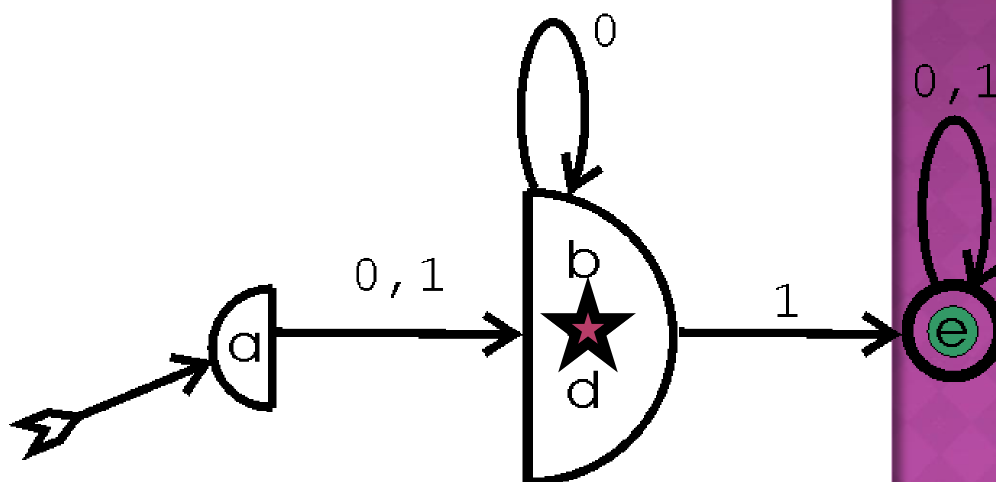
10000



MINIMIZATION EXAMPLE. COMPARE

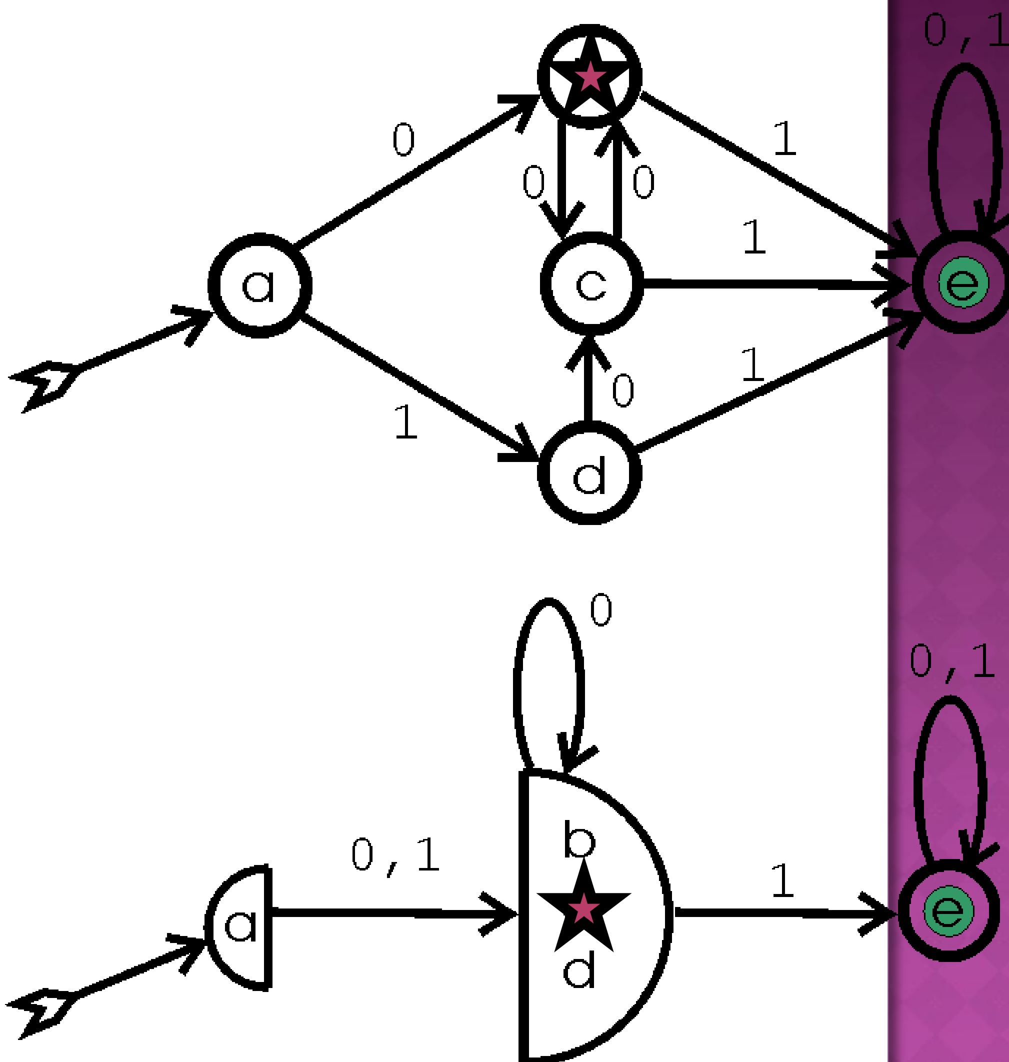
10000





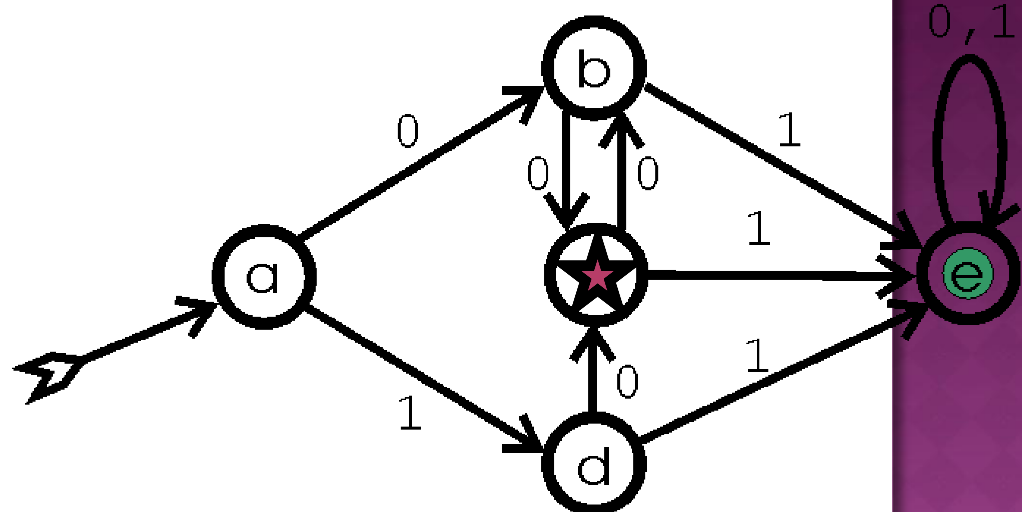
MINIMIZATION EXAMPLE. COMPARE

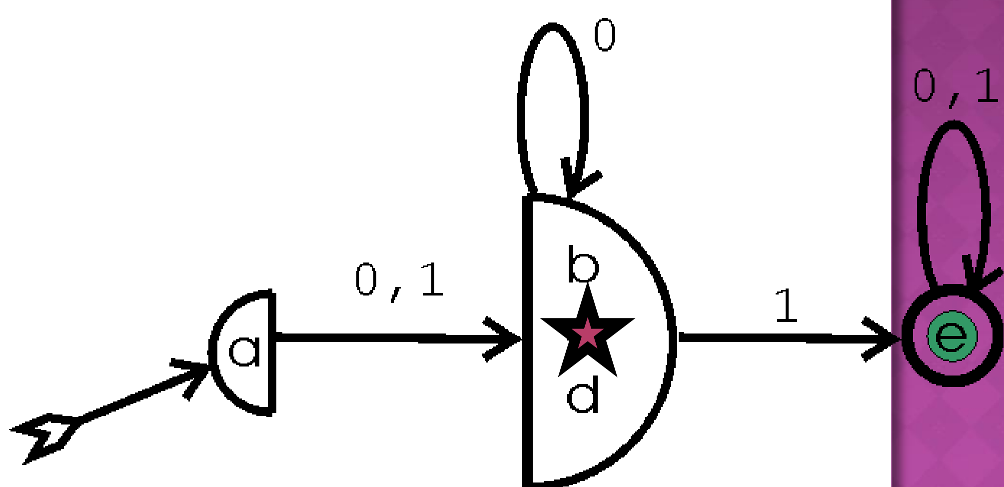
10000



MINIMIZATION EXAMPLE. COMPARE

10000

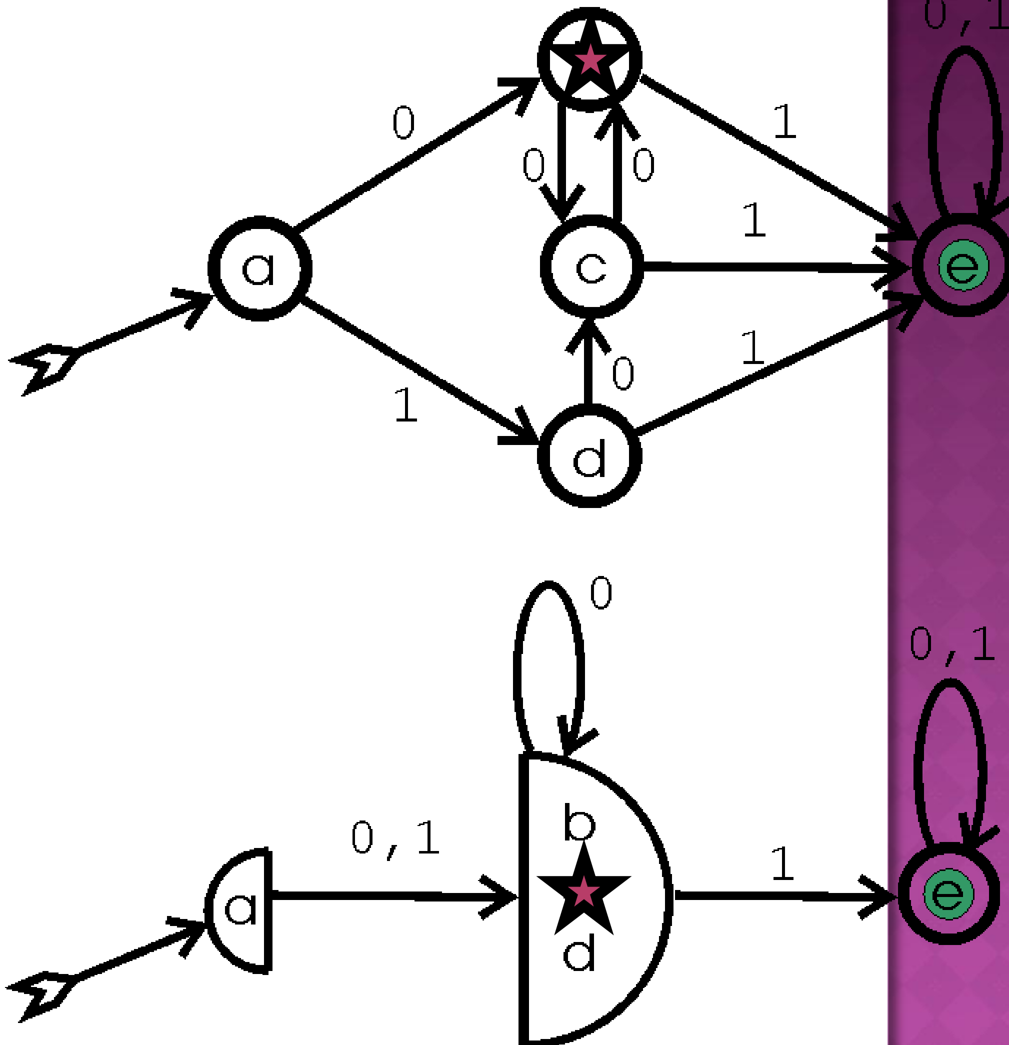




MINIMIZATION EXAMPLE. COMPARE

10000

REJECT.



PROOF OF MINIMAL AUTOMATON

Previous algorithm guaranteed to produce an irreducible FA. Why should that FA be the smallest possible FA for its accepted language?

Analogous question in calculus: Why should a local minimum be a global minimum? *Usually* not the case!

PROOF OF MINIMAL AUTOMATON

THM (Myhill-Nerode): The minimization algorithm produces the smallest possible automaton for its accepted language.

Proof. Show that any irreducible automaton is the smallest for its accepted language L:

We say that two strings $u,v \in \Sigma^*$ are ***indistinguishable*** if for all suffixes x, ux is in L exactly when vx is.

Notice that if u and v *are* distinguishable, the path from their paths from the start state must have different endpoints.

PROOF OF MINIMAL AUTOMATON

Consequently, the number of states in any DFA for *L* must be as great as the number of mutually distinguishable strings for *L*.

But an irreducible DFA has the property that every state gives rise to another mutually distinguishable string!

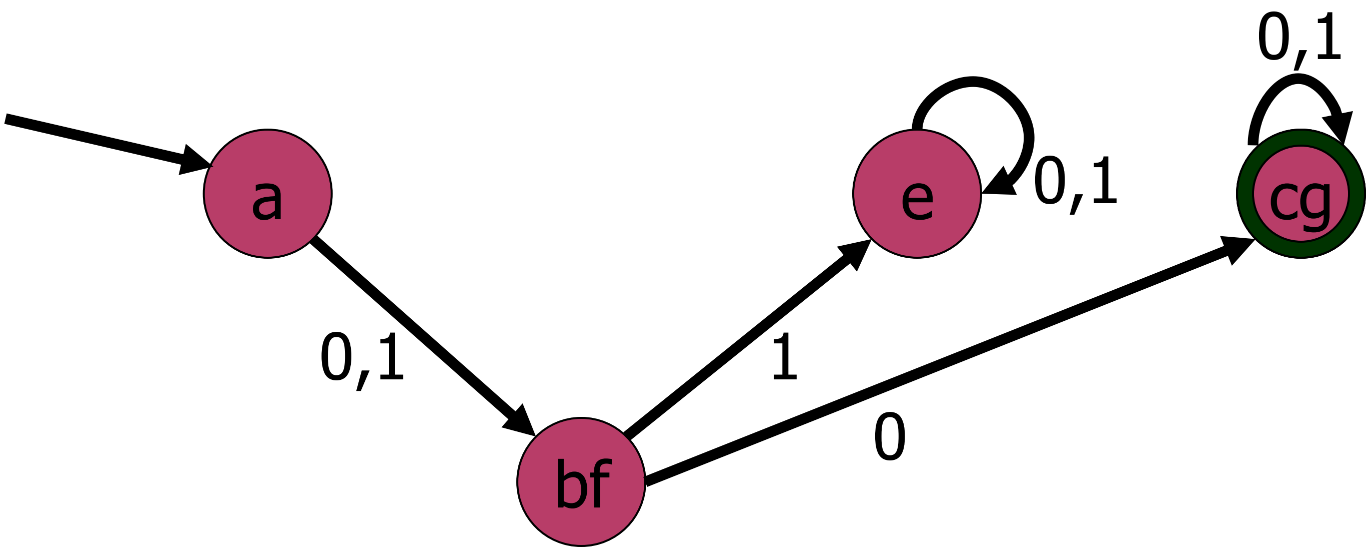
Therefore, any other DFA must have at least as many states as the irreducible DFA

□

Let’s see how the proof works on a previous example:

PROOF OF MINIMAL AUTOMATON. EXAMPLE

The “spanning tree of strings” {ε,0,01,00} is a mutually distinguishable set (otherwise redundancy would occur and hence DFA would be reducible). Any other DFA for *L* has ≥4 states.



THE PUMPING LEMMA MOTIVATION

Consider the language

$$L_1 = 01^* = \{0, 01, 011, 0111, \dots\}$$

The string $0\underline{1}1$ is said to be ***pumpable*** in L_1 because can take the underlined portion, and pump it up (i.e. repeat) as much as desired while *always* getting elements in L_1.

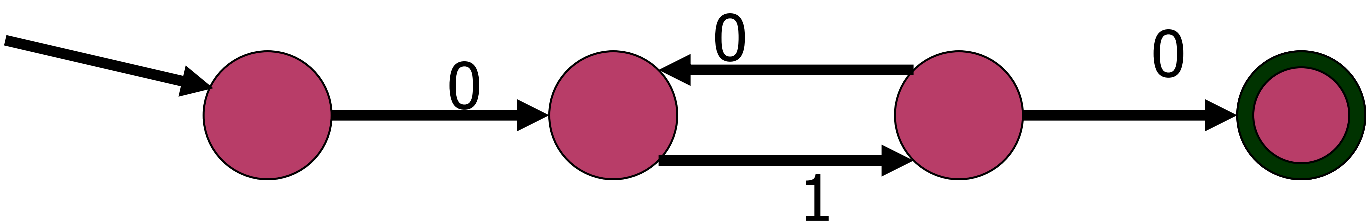
Q: Which of the following are pumpable?

1. 01111
2. 01
3. 0

THE PUMPING LEMMA MOTIVATION

1. Pumpable: $011\underline{1}1$, $0\underline{1}111$, $0\underline{111}1$, $0\underline{1111}$, etc.
2. Pumpable: $0\underline{1}$
3. 0 *not* pumpable because most of 0* not in L_1

Define L_2 by the following automaton:

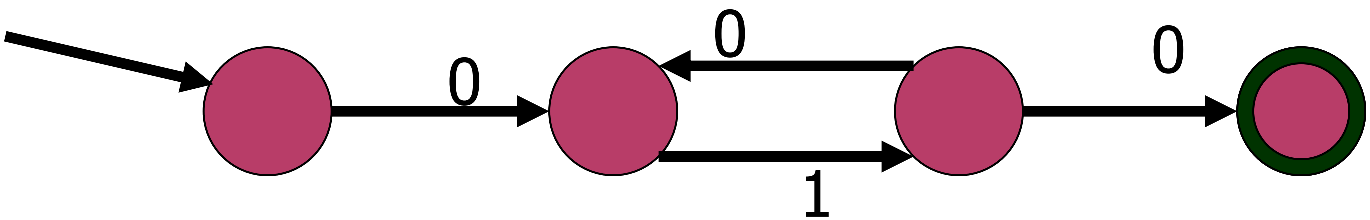


Q: Is 01010 pumpable?

THE PUMPING LEMMA MOTIVATION

A: Pumpable: 0<u>10</u>10, 01<u>01</u>0. Underlined substrings correspond to cycles in the FA!

Cycles in the FA can be repeated arbitrarily often, hence pumpable.



Let $L_3 = \{011, 11010, 000, \varepsilon\}$

Q: Which strings are pumpable?

THE PUMPING LEMMA MOTIVATION

A: None! When pumping any string non-trivially, always result in infinitely many possible strings. So no pumping can go on inside a finite set.

Pumping Lemma give a criterion for when strings can be pumped:

PUMPING LEMMA

THM: Given a regular language L, there is a number p (called the ***pumping number***) such that any string in L of length $\geq p$ is pumpable within its first p letters. In other words, for all $u \in L$ with $|u| \geq p$ we we can write:

- $u = xyz$ (*x* is a prefix, *z* is a suffix)
- $|y| \geq 1$ (mid-portion *y* is non-empty)
- $|xy| \leq p$ (pumping occurs in first *p* letters)
- $xy^iz \in L$ for all $i \geq 0$ (can pump *y*-portion)

PUMPING LEMMA PROOF

EX: Show that **pal**=$\{x \in \Sigma^* | x = x^R\}$ isn't regular.

1. Assume **pal** were regular
2. Therefore it has a pumping no. p
3. But… consider the string 0^p10^p. Can this string be pumped in its first p letters? The answer is NO because any augmenting of the first 0^p-portion results in a non-palindrome
4. (2)$\rightarrow\leftarrow$(3) <contradiction> Therefore our assumption (1) was wrong and conclude that **pal** is *not* a regular language

PUMPING LEMMA TEMPLATE

In general, to prove that *L* isn't regular:

1. Assume *L* were regular
2. Therefore it has a pumping no. *p*
3. *Find a string pattern involving the length p in some clever way, and which cannot be pumped.* ***This is the hard part.***
4. (2)→←(3) <contradiction> Therefore our assumption (1) was wrong and conclude that *L* is *not* a regular language

PUMPING LEMMA EXAMPLES

Since parts 1, 2 and 4 are identical for any pumping lemma proof, following examples will only show part 3 of the proof.

PUMPING LEMMA EXAMPLES

EX: Show that $\{a^n b^n \mid n = 0,1,2, \dots\}$ is not regular.

Part 3) Consider $a^p b^p$. By assumption, we can pump up within the first p letters of this string. Thus we get more a's than b's in the resulting string, which breaks the pattern.

PUMPING LEMMA EXAMPLES PUMPING DOWN

Sometimes it is useful to pump-*down* instead of up. In pumping down we simply erase the y portion of the pattern string. This is allowed by setting $i = 0$ in the pumping lemma:

EX: Show that $\{a^m b^n \mid m > n\}$ is not regular.

Part 3) Consider $a^{p+1} b^p$. By assumption, we can pump *down* within the first p letters of this string. As by assumption y is non-empty, we must decrease the number of a's in the pattern, meaning that the number of a's is less than or equal to the number of b's, which breaks the pattern!

PUMPING LEMMA EXAMPLES NUMERICAL ARGUMENTS

Sometimes we have to look at the resulting pump-ups more carefully:

EX: Show that $\{1^n \mid n \text{ is a prime number}\}$ is not regular.

Part 3) Given p, choose a prime number n bigger than p. Consider 1^n. By assumption, we can pump within the first p letters of this string so we can pump 1^n. Let m be the length of the pumped portion x. Pumping i *times* (i = 0 means we pump-down) results in the string $1^{(n-m)+im} = 1^{n+(i-1)m}$.

Q: Find an i making the exponent non-prime.

PUMPING LEMMA EXAMPLES NUMERICAL ARGUMENTS

A: Set $i = n + 1$. Then the pumped-up string is

$$1^{n+(i-1)m} = 1^{n+(n+1-1)m} = 1^{n+nm} = 1^{n(1+m)}$$

Therefore the resulting exponent is not a prime, which breaks the pattern.

PROOF OF PUMPING LEMMA

Consider a graph with *n* vertices. Suppose you tour around visiting a certain number of nodes.

Q: How many vertices can you visit before you are forced to see some vertex twice?

PROOF OF PUMPING LEMMA

A: If you visit n+1 vertices, you must have seen some vertex twice.

Q: Why?

PROOF OF PUMPING LEMMA. PIGEONHOLE PRINCIPLE

A: The pigeonhole principle.

More precisely. Your visiting n+1 vertices defines the following function:

$$f : \{1, 2, 3, \ldots , n+1\} \rightarrow \{\text{size-}n \text{ set}\}$$

$$f(i) = i \text{ 'th vertex visited}$$

Since domain is bigger than codomain, cannot be one-to-one.

PROOF OF PUMPING LEMMA

Now consider an accepted string *u*. By assumption *L* is regular so let *M* be the FA accepting it. Let $p = |Q|$ = no. of states in *M*. Suppose $|u| \geq p$. The path labeled by *u* visits $p+1$ states in its first *p* letters. Thus *u* must visit some state twice. The sub-path of *u* connecting the first and second visit of the vertex is a loop, and gives the claimed string *y* that can be pumped within the first *p* letters.