COMP 412
FALL 2010

Lexical Analysis: DFA Minimization

Comp 412

Automating Scanner Construction

RE→NFA *(Thompson's construction)* ✓

- Build an NFA for each term
- Combine them with ε-moves

NFA →DFA (*subset construction*) ✓

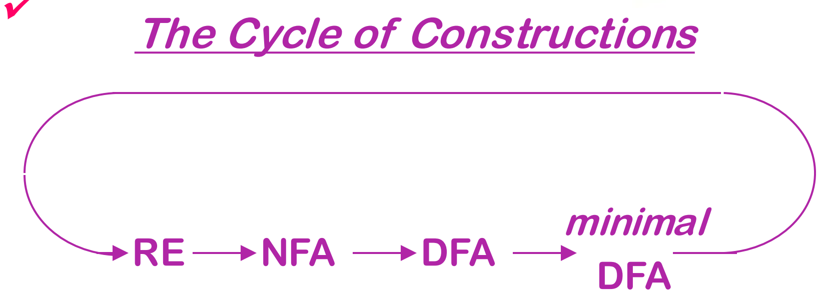
- Build the simulation

DFA →Minimal DFA

- Brzozowski's Algorithm ✓
- Hopcroft's algorithm *(today)*

DFA →RE (*not really part of scanner construction*)

- All pairs, all paths problem
- Union together paths from s_0 to a final state



DFA Minimization

The Big Picture

- Discover sets of equivalent states in the DFA
- Represent each such set with a single state

DFA Minimization

The Big Picture

- Discover sets of equivalent states in the DFA
- Represent each such set with a single state

Two states are equivalent if and only if:

- The set of paths leading to them are equivalent, *and*
- $\forall\ \alpha \in \Sigma$, transitions on α lead to equivalent states (DFA)

$\Rightarrow$ Must split a state that has α-transitions to distinct sets

DFA Minimization

The Big Picture

- Discover sets of equivalent states in the DFA
- Represent each such set with a single state

Two states are equivalent if and only if:

- The set of paths leading to them are equivalent, *and*
- $\forall\ \alpha \in \Sigma$, transitions on α lead to equivalent states (DFA)

$\Rightarrow$ Must split a state that has α-transitions to distinct sets

A partition *P* of *S*

- A collection of sets *P* s.t. each $s \in S$ is in exactly one $p_i \in P$
- The algorithm iteratively constructs partitions of the DFA's states

DFA Minimization

Maximally sized sets $\Rightarrow$ minimal number of sets

Details of the algorithm

- Group states into maximally sized initial sets, *optimistically*
- Iteratively subdivide those sets, based on transition graph
- States that remain grouped together are equivalent

Initial partition, P_0, has two sets: $\{F\}$ & $\{S-F\}$ $\quad D = (S, \Sigma, \delta, s_0, F)$

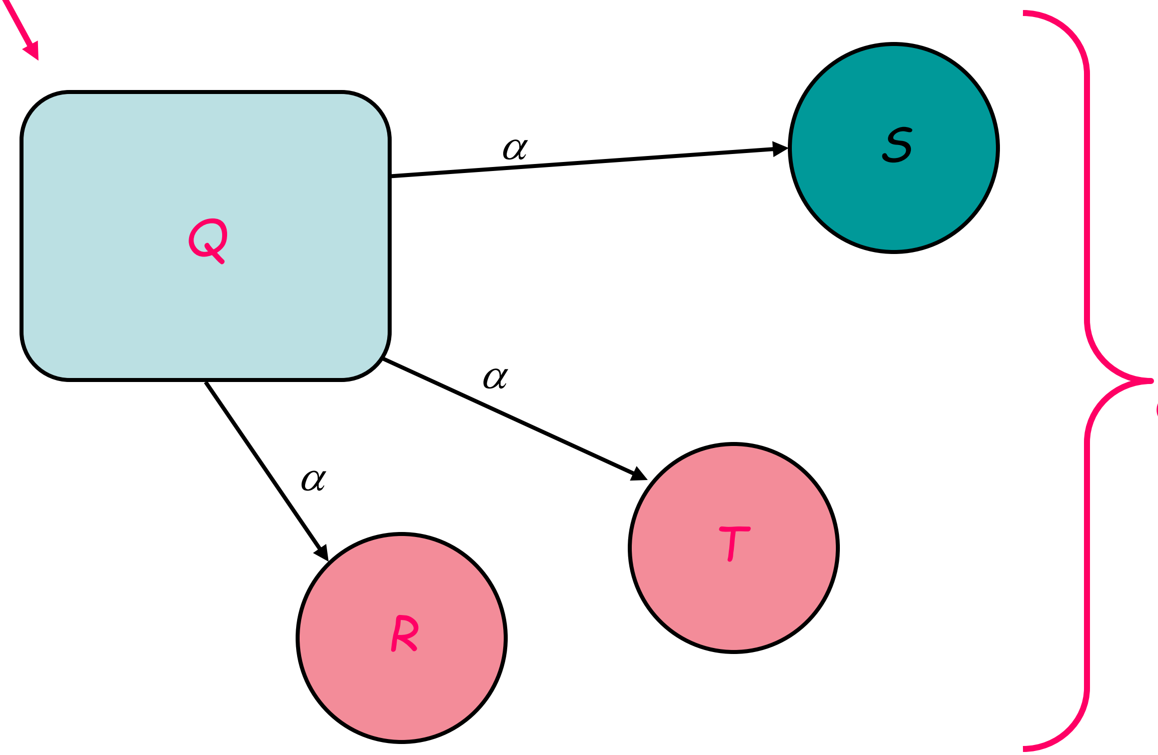
final states others

Splitting a set ("partitioning a set by a")

- Assume s_a & $s_b \in p_i$, and $\delta(s_a, \underline{a}) = s_x$, & $\delta(s_b, \underline{a}) = s_y$
- If s_x & s_y are not in the same set p_j, then p_i must be split
  - s_a has transition on a, s_b does not $\Rightarrow$ a splits p_i
- One state in the final DFA cannot have two transitions on a
- The algorithm works backward, from a pair $(p, \underline{a})$ to the subset of the states in some other set q that reach p on a

Key Idea: Splitting S around α

Original set Q

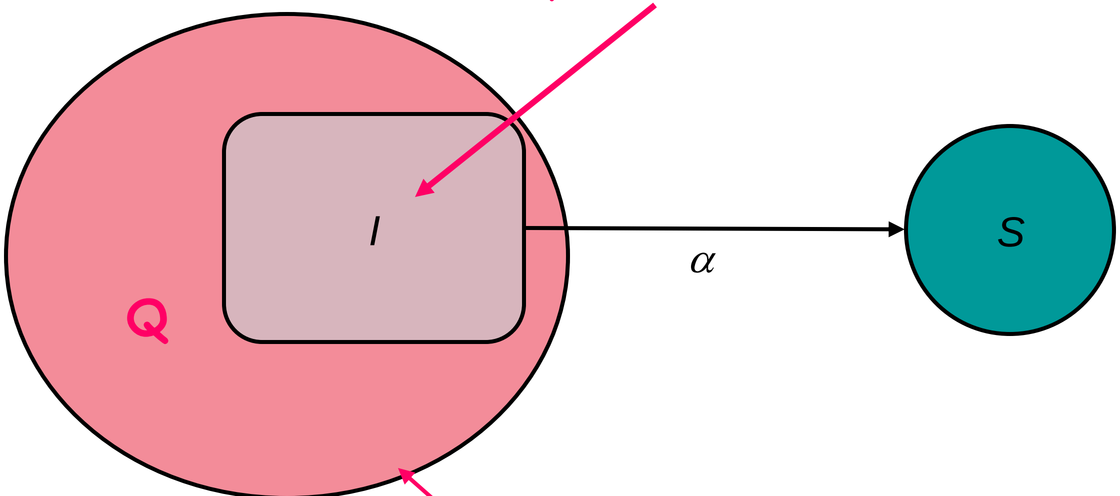


Q has transitions on α to R, S, & T

The algorithm partitions Q around α

Key Idea: Splitting Q around (S,α)

Find maximal partition I that has an α-transition into S



This part must have an α-transition to one or more other states in one or more other partitions.

Otherwise, it does not split!

Think of I as the image of S under the inverse of the transition function

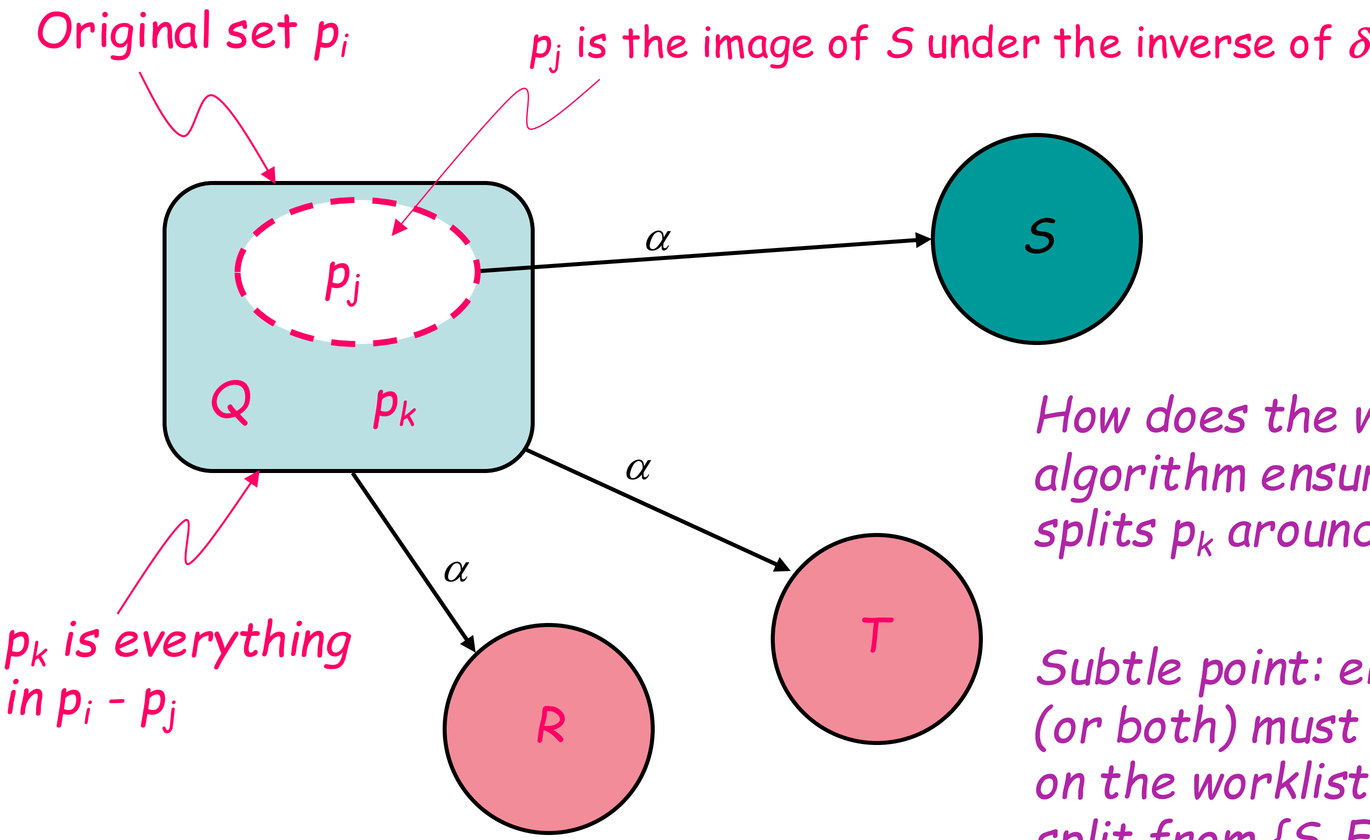
$$I \leftarrow \delta^{-1}(s, \alpha)$$

Hopcroft's Algorithm

$W \leftarrow \{F, S\text{-}F\}$; $P \leftarrow \{F, S\text{-}F\}$; *// W is the worklist, P the current partition*

```
while ( W is not empty ) do begin
      select and remove s from W;  // s is a set of states
      for each α in Σ do begin
            let I_α ← δ_α^{-1}( s );  // I_α is set of all states that can reach s on α
            for each p ∈ P such that p ∩ I_α is not empty
               and p is not contained in I_α do begin
                  partition p into p1 and p2 such that p1 ← p ∩ I_α ; p2 ← p – p1;
                  P ← (P – p) ∪ p1 ∪ p2 ;
                  if p ∈ W
                        then W ← (W – p) ∪ p1 ∪ p2 ;
                        else if |p1| ≤ |p2 |
                              then W ← W ∪ p1;
                              else W ← W ∪ p2;
            end
      end
end
```

Key Idea: Splitting p_i around α



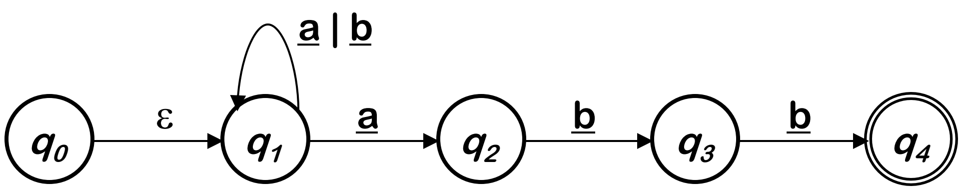
How does the worklist algorithm ensure that it splits p_k around R & T ?

Subtle point: either R or T (or both) must already be on the worklist. (R & T have split from {S-F}.)

Thus, it can split p_i around one state (S) & add either p_j or p_k to the worklist.

A Detailed Example

Remember (a | b)* abb ?   *(from last lecture)*



Our first NFA

Applying the subset construction:

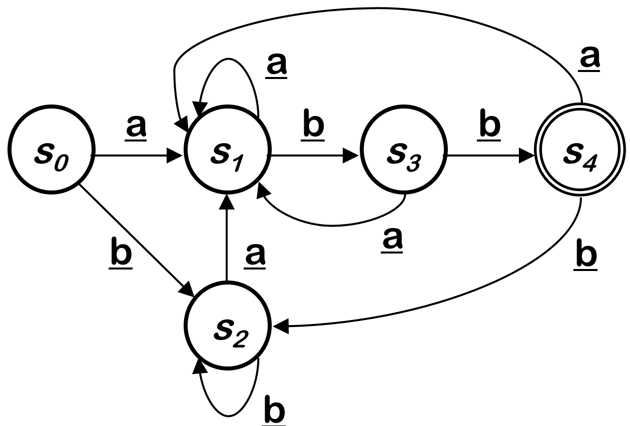
	State		ε-closure(move(s_i,*))	
Iter.	DFA	NFA	a	b
0	s_0	q_0, q_1	q_1, q_2	q_1
1	s_1	q_1, q_2	q_1, q_2	q_1, q_3
	s_2	q_1	q_1, q_2	q_1
2	s_3	q_1, q_3	q_1, q_2	q_1, q_4
3	s_4	q_1, q_4	q_1, q_2	q_1

Iteration 3 adds nothing to *S*, so the algorithm halts

contains q_4 (final state)

A Detailed Example

The DFA for (a | b)* abb

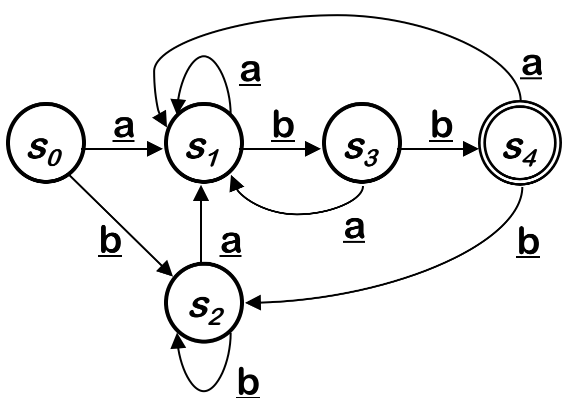


	Character	
State	a	b
s_0	s_1	s_2
s_1	s_1	s_3
s_2	s_1	s_2
s_3	s_1	s_4
s_4	s_1	s_2

- Not much expansion from NFA *(we feared exponential blowup)*
- Deterministic transitions
- Use same code skeleton as before

A Detailed Example (DFA Minimization)

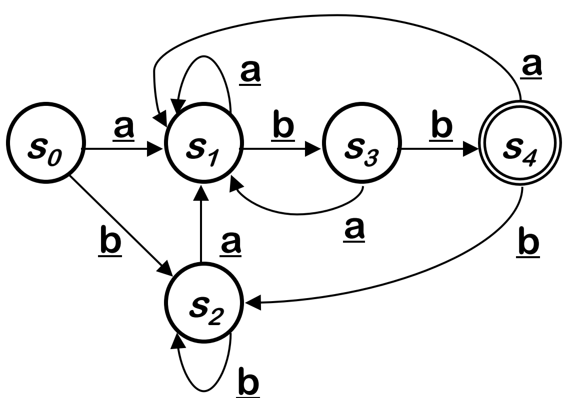
	Current Partition	Worklist	*s*	Split on a	Split on b
P_0	$\{s_4\}\,\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\,\{s_0,s_1,s_2,s_3\}$			



For the record, example was right in 1999, broken in 2000

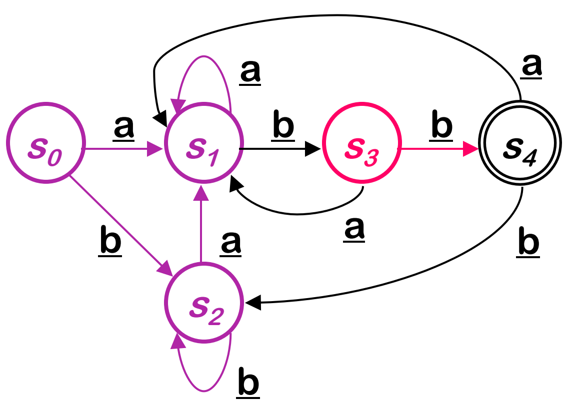
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on* $\underline{a}$	*Split on* $\underline{b}$
P_0	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	



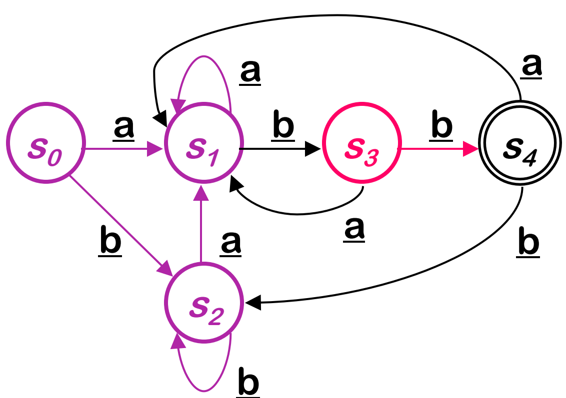
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on a*	*Split on b*
P_0	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\ \{s_0,s_1,s_2\}$



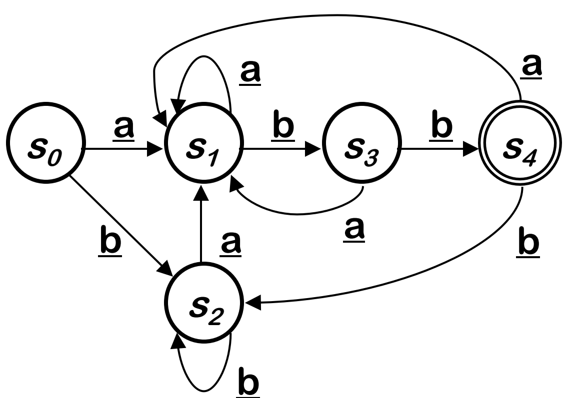
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on* a	*Split on* b
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$			



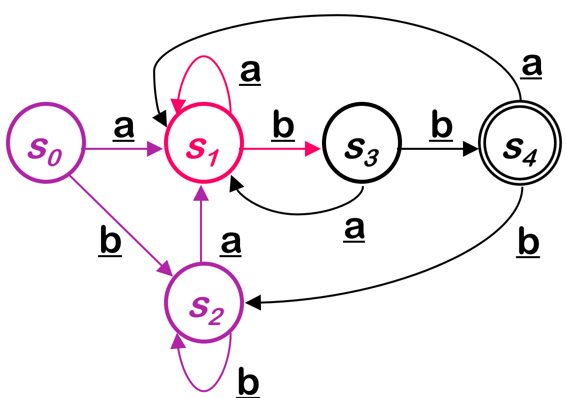
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on a*	*Split on b*
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	



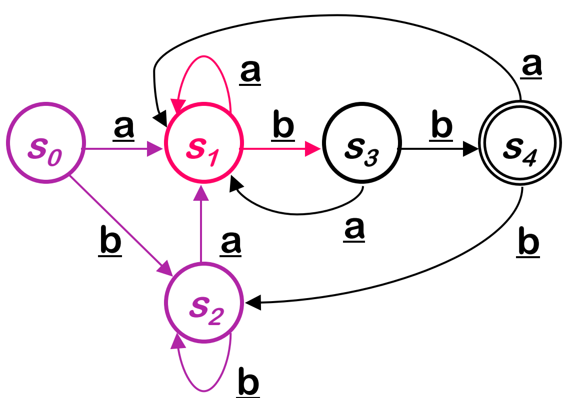
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on* a	*Split on* b
P_0	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}\ \{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\ \{s_0,s_1,s_2\}$
P_1	$\{s_4\}\ \{s_3\}\ \{s_0,s_1,s_2\}$	$\{s_3\}\ \{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	$\{s_1\}\ \{s_0,s_2\}$



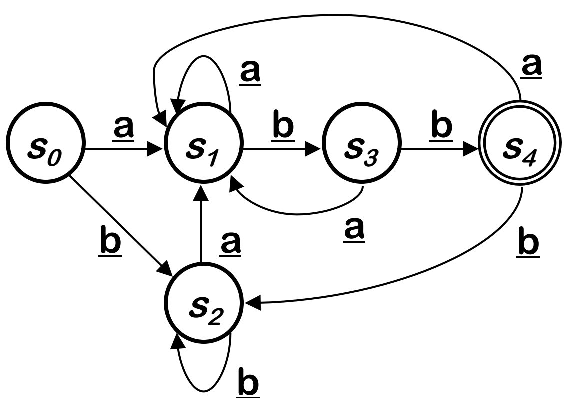
A Detailed Example (DFA Minimization)

	Current Partition	*Worklist*	*s*	*Split on* a	*Split on* b
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	$\{s_1\}\{s_0,s_2\}$
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$			



A Detailed Example (DFA Minimization)

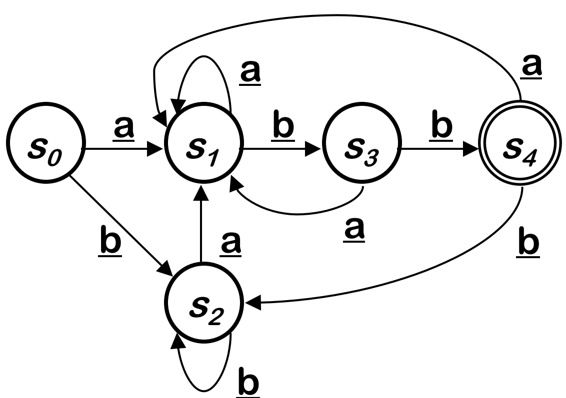
	Current Partition	*Worklist*	*s*	*Split on* a	*Split on* b
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	$\{s_1\}\{s_0,s_2\}$
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$	$\{s_1\}$	*none*	*none*



A Detailed Example (DFA Minimization)

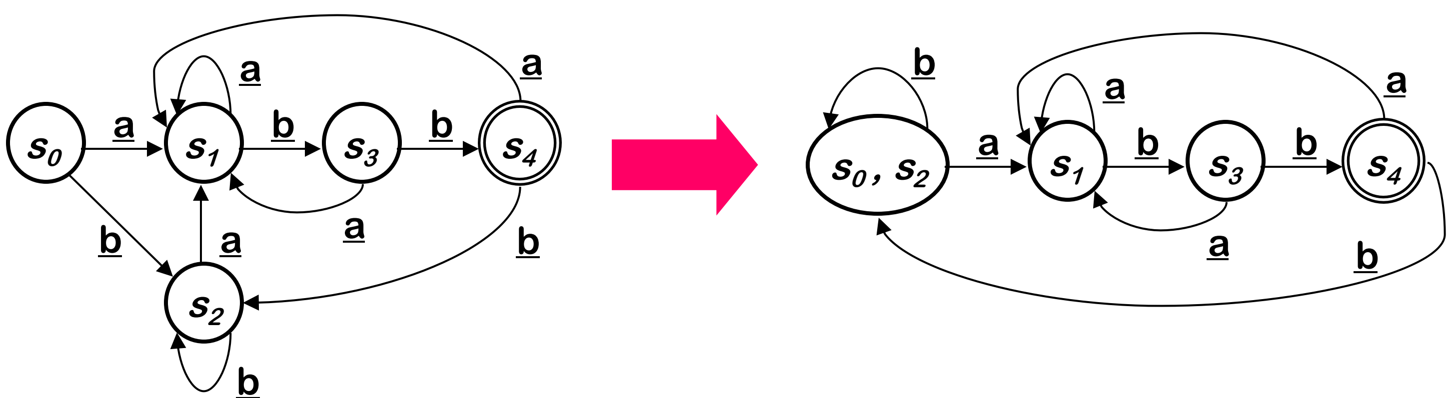
	Current Partition	*Worklist*	*s*	*Split on a*	*Split on b*
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	$\{s_1\}\{s_0,s_2\}$
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$	$\{s_1\}$	*none*	*none*
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$	$\{s_0,s_2\}$	*none*	*none*

Empty worklist $\Rightarrow$ done!



A Detailed Example (DFA Minimization)

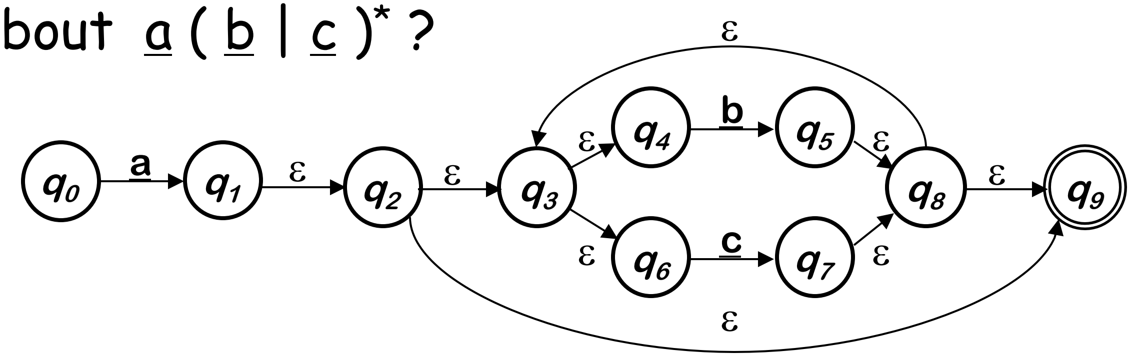
	Current Partition	*Worklist*	*s*	*Split on a*	*Split on b*
P_0	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}\{s_0,s_1,s_2,s_3\}$	$\{s_4\}$	*none*	$\{s_3\}\{s_0,s_1,s_2\}$
P_1	$\{s_4\}\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}\{s_0,s_1,s_2\}$	$\{s_3\}$	*none*	$\{s_1\}\{s_0,s_2\}$
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$	$\{s_1\}$	*none*	*none*
P_2	$\{s_4\}\{s_3\}\{s_1\}\{s_0,s_2\}$	$\{s_1\}\{s_0,s_2\}$	$\{s_0,s_2\}$	*none*	*none*



20% reduction in number of states

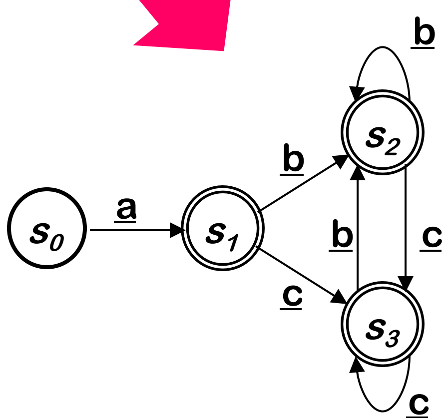
DFA Minimization

What about $\underline{a}\ (\ \underline{b}\ |\ \underline{c}\ )^*$?



First, the subset construction:

States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	s_1	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	s_2	s_3
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3

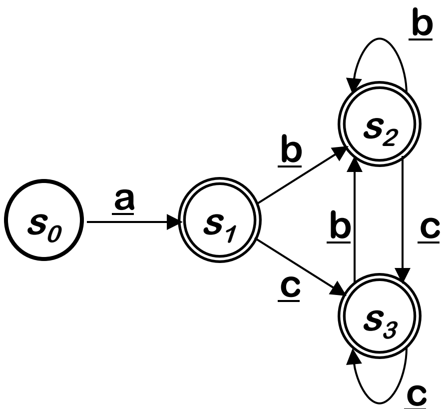


From last lecture …

DFA Minimization

Then, apply the minimization algorithm

		Split on		
	Current Partition	*a*	*b*	*c*
P_0	$\{s_1,s_2,s_3\}\ \{s_0\}$	*none*	*none*	*none*



It splits no states after the initial partition

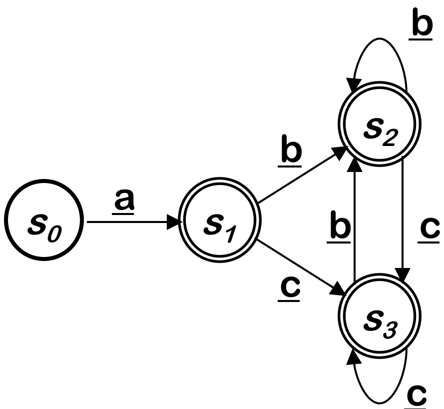
⇒ The minimal DFA has two states

- One for $\{s_0\}$
- One for $\{s_1,s_2,s_3\}$

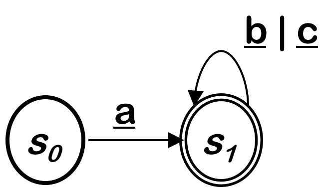
DFA Minimization

Then, apply the minimization algorithm

		Split on		
	Current Partition	*a*	*b*	*c*
P_0	$\{s_1,s_2,s_3\}$ $\{s_0\}$	*none*	*none*	*none*



It produces this DFA



In lecture 5, we observed that a human would design a simpler automaton than Thompson's construction & the subset construction did.

Minimizing that DFA produces the one that a human would design!

Abbreviated Register Specification

Start with a regular expression

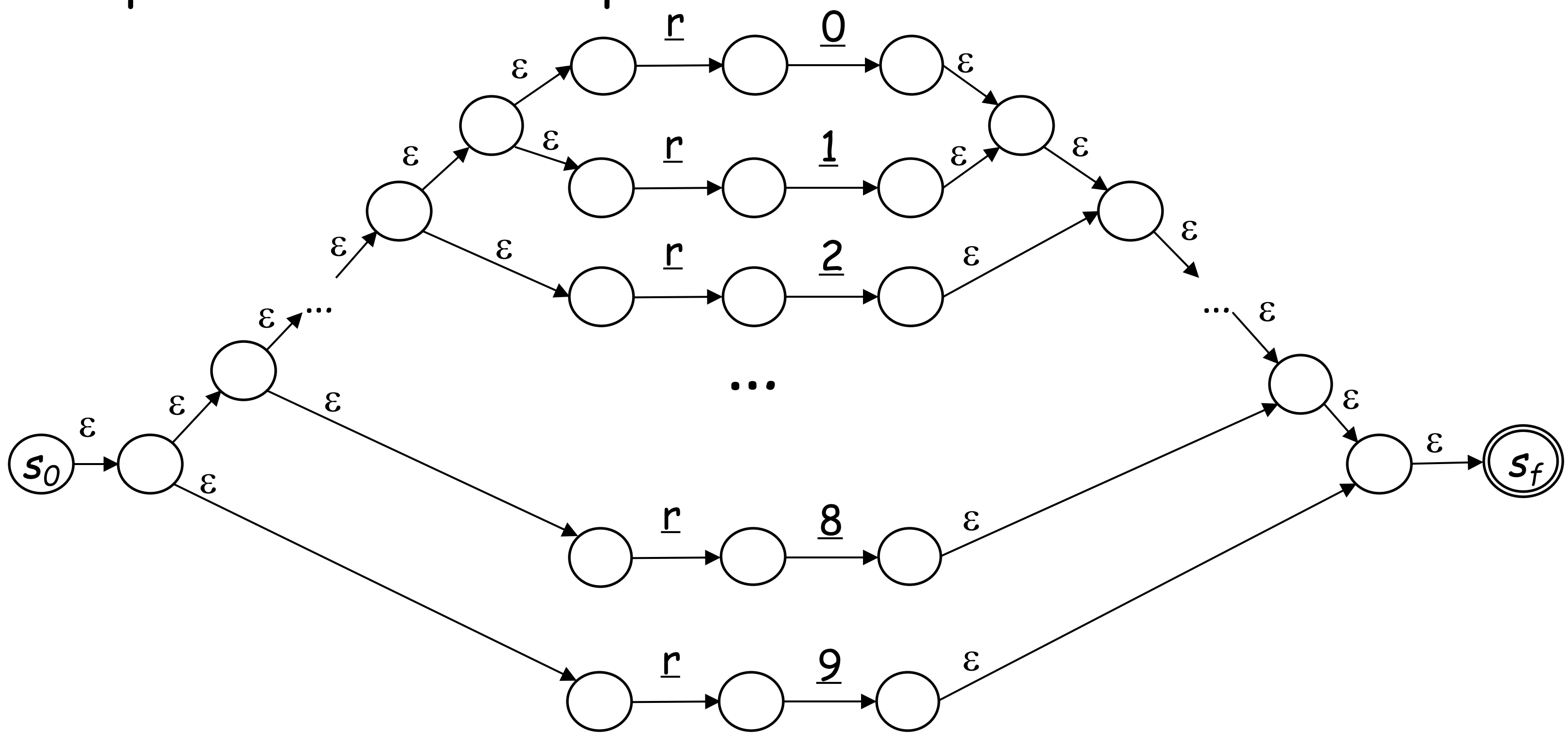
r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9

Register names from zero to nine

The Cycle of Constructions

Abbreviated Register Specification

Thompson's construction produces

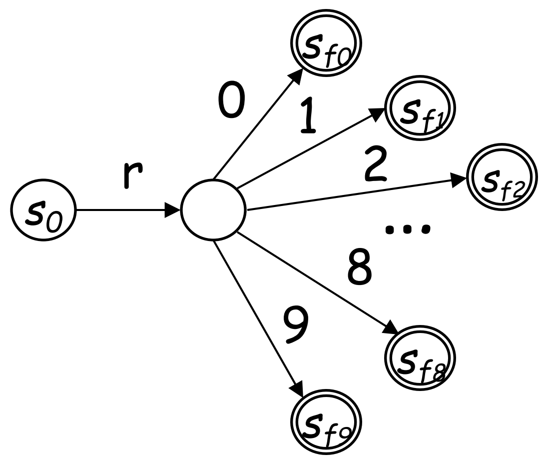


The Cycle of Constructions

To make the example fit, we have eliminated some of the ε-transitions, e.g., between r and 0

Abbreviated Register Specification

The subset construction builds

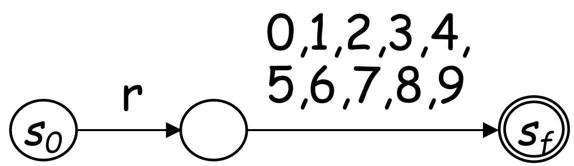


This is a DFA, but it has a lot of states …

The Cycle of Constructions

Abbreviated Register Specification

The DFA minimization algorithm builds



This looks like what a skilled compiler writer would do!

The Cycle of Constructions

Automating Scanner Construction

RE→NFA *(Thompson's construction)* ✓

- Build an NFA for each term
- Combine them with ε-moves

NFA →DFA (*subset construction*) ✓

- Build the simulation

DFA →Minimal DFA

- Brzozowski's Algorithm ✓
- Hopcroft's algorithm ✓

DFA →RE *(not really part of scanner construction)*

- All pairs, all paths problem
- Union together paths from s_0 to a final state

The Cycle of Constructions

RE Back to DFA

Kleene's Construction

for $i \leftarrow 0$ to $|D| - 1$; // label each immediate path
  for $j \leftarrow 0$ to $|D| - 1$;
    $R^0_{ij} \leftarrow \{ a \mid \delta(d_i, a) = d_j \}$;
    if $(i = j)$ then
      $R^0_{ii} = R^0_{ii} \mid \{\varepsilon\}$;

for $k \leftarrow 0$ to $|D| - 1$; // label nontrivial paths
  for $i \leftarrow 0$ to $|D| - 1$;
    for $j \leftarrow 0$ to $|D| - 1$;
      $R^k_{ij} \leftarrow R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} \mid R^{k-1}_{ij}$

$L \leftarrow \{\}$ // union labels of paths from
For each final state s_i // s_0 to a final state s_i
  $L \leftarrow L \mid R^{|D|-1}_{0i}$

R^k_{ij} is the set of paths from i to j that include no state higher than k

The Cycle of Constructions

Limits of Regular Languages

Not all languages are regular

$RL's \subset CFL's \subset CSL's$

You cannot construct DFA's to recognize these languages

- $L = \{ p^k q^k \}$ *(parenthesis languages)*
- $L = \{ wcw^r \mid w \in \Sigma^* \}$

Neither of these is a regular language *(nor an RE)*

But, this is a little subtle. You can construct DFA's for

- Strings with alternating 0's and 1's

  $(\varepsilon \mid 1) (01)^* (\varepsilon \mid 0)$
- Strings with and even number of 0's and 1's

RE's can count bounded sets and bounded differences

Limits of Regular Languages

Advantages of Regular Expressions

- Simple & powerful notation for specifying patterns
- Automatic construction of fast recognizers
- Many kinds of syntax can be specified with REs

Example — an expression grammar

$$\begin{aligned} &\mathit{Term} &&\rightarrow [\text{a-zA-Z}]\,([\text{a-zA-Z}] \mid [\underline{0}\text{-}\underline{9}])^* \\ &\mathit{Op} &&\rightarrow \underline{+} \mid \underline{-} \mid \underline{*} \mid \underline{/} \\ &\mathit{Expr} &&\rightarrow (\ \mathit{Term}\ \mathit{Op}\ )^*\ \mathit{Term} \end{aligned}$$

Of course, this would generate a DFA …

If REs are so useful …

Why not use them for everything?

EXTRA SLIDES START HERE

<u>*This is a fixed-point algorithm!*</u>

DFA Minimization

The algorithm

```
T ← { F, {S-F}}
 P ← { }
while ( P ≠ T)
   P ← T
   T ← { }
    for each set p_i ∈ P
        T ← T ∪ Split(p_i)

Split(S)
    for each c ∈ Σ
        if c splits S into s_1 & s_2
            then return {s_1, s_2}
    return S
```

mild abuse of notation

Why does this work?

- Partition $P \in 2^S$
- Start off with 2 subsets of S: $\{F\}$ and $\{S-F\}$
- The *while* loop takes $P^i \rightarrow P^{i+1}$ by splitting 1 or more sets
- P^{i+1} is at least one step closer to the partition with $|S|$ sets
- Maximum of $|S|$ splits

Note that

- Partitions are <u>never</u> combined
- Initial partition ensures that final states remain final states

DFA Minimization

Refining the algorithm

- As written, it examines every $p_i \in P$ on each iteration
  - This strategy entails a lot of unnecessary work
  - Only need to examine p_i if some T, reachable from p_i, has split
- Reformulate the algorithm using a "worklist"
  - Start worklist with initial partition, F and $\{S-F\}$
  - When it splits p_i into p_1 and p_2, place p_2 on worklist

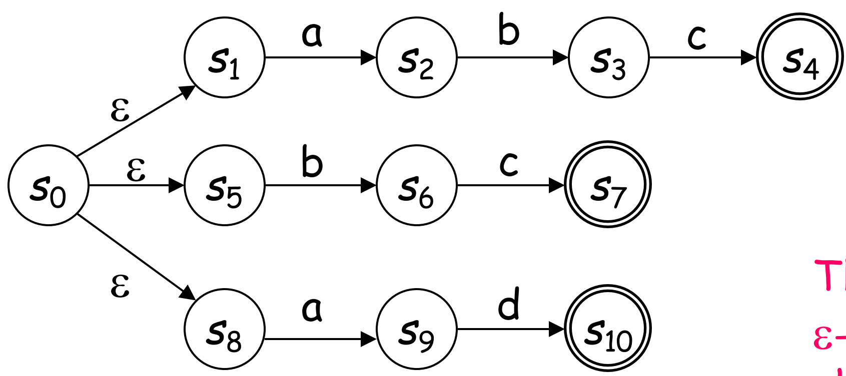
This version looks at each $p_i \in P$ many fewer times

- Well-known, widely used algorithm due to John Hopcroft

Alternative Approach to DFA Minimization

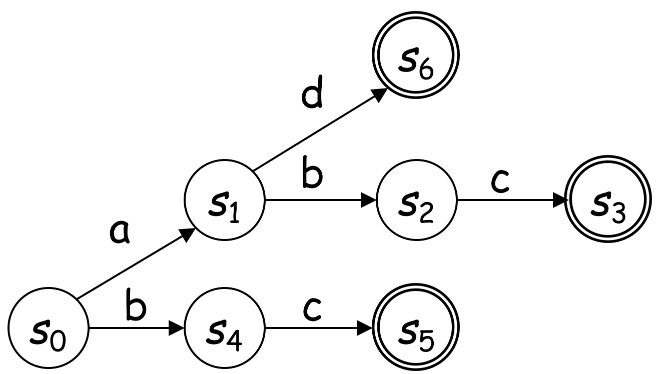
The Intuition

- The subset construction merges prefixes in the NFA



abc | bc | ad

Thompson's construction would leave ε-transitions between each single-character automaton



Subset construction eliminates ε-transitions and merges the paths for a. It leaves duplicate tails, such as bc.

Alternative Approach to DFA Minimization

Idea: use the subset construction twice

- For an NFA *N*
  - Let *reverse(N)* be the NFA constructed by making initial states final (& vice-versa) and reversing the edges
  - Let *subset(N)* be the DFA that results from applying the subset construction to *N*
  - Let *reachable(N)* be *N* after removing all states that are not reachable from the initial state
- Then,

reachable(subset(reverse[reachable(subset(reverse(N))]))

is the minimal DFA that implements *N* [Brzozowski, 1962]

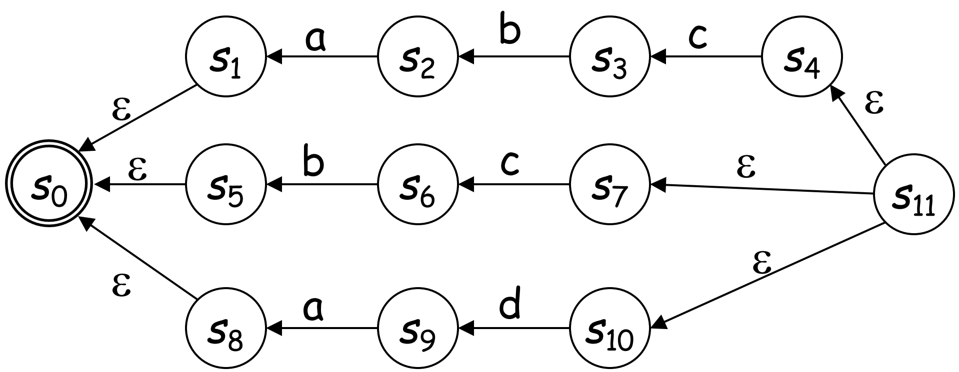
This result is not intuitive, but it is true.

Neither algorithm dominates the other.

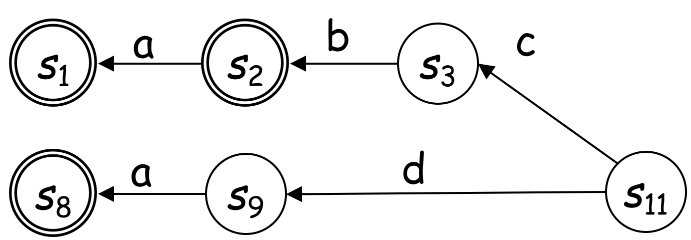
Alternative Approach to DFA Minimization

Step 1

- The subset construction on *reverse(NFA)* merges suffixes in original NFA



Reversed NFA

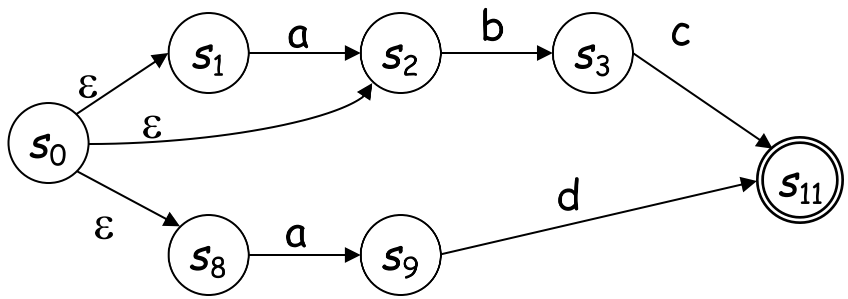


subset(reverse(NFA))

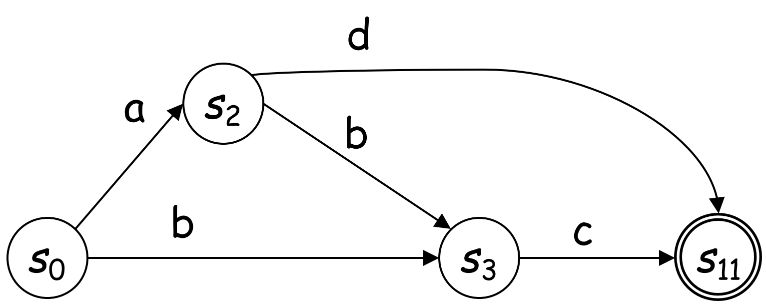
Alternative Approach to DFA Minimization

Step 2

- Reverse it again & use subset to merge prefixes …



Reverse it, again



Minimal DFA

And subset it, again

The Cycle of Constructions