Specialized Finite Automata

EFAs, NFAs, and DFAs

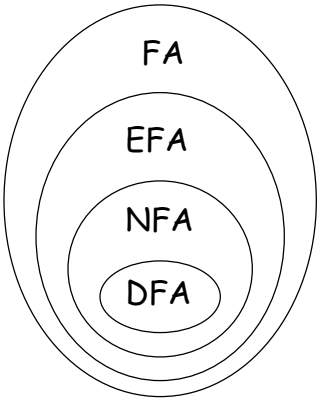
Wednesday, October 10, 2007
Reading: Stoughton 3.8--3.11; Sipser 1.1

CS235 Languages and Automata

Department of Computer Science
Wellesley College

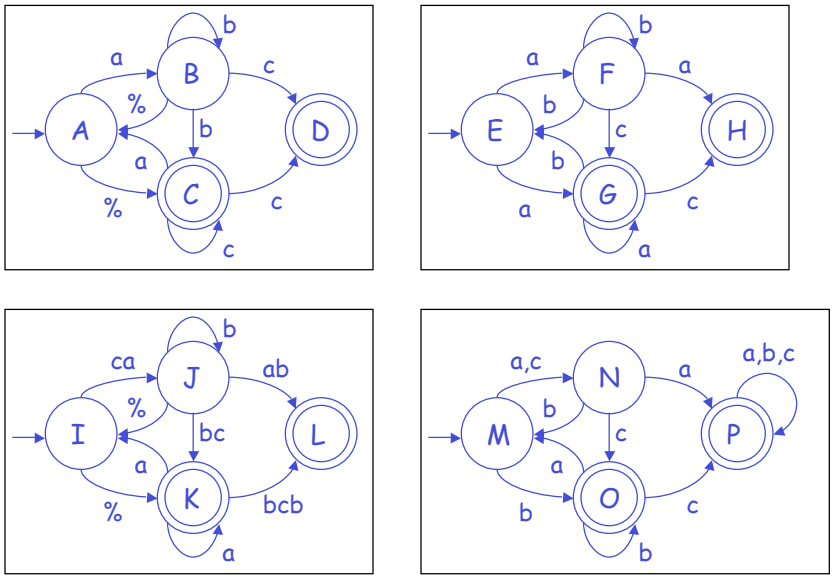
Three Subclasses of FAs*

- **EFA** (Empty-string Finite Automata): FA in which each transition label has length 0 or 1.
- **NFA** (Nondeterministic Finite Automata): FA in which each transition label has length 1.
- **DFA** (Deterministic Finite Automata): FA in which each transition label has length 1, and for every symbol *a* in the alphabet of the automaton, there is exactly one transition labeled *a* from each state. I.e., the transition relation is a *total function* $T : Q \times a \rightarrow Q$.



* This is Stoughton's terminology and classification. Many other authors use NFA for what Stoughton calls EFA, and do not give a distinct name for what Stoughton calls NFA.

Classification Examples



Conversions

We have already studied conversions between Reg and FA, demonstrating their equivalence.

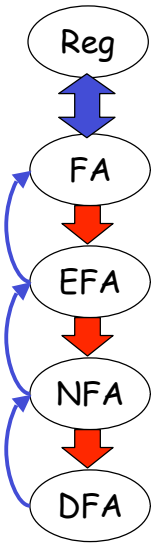
There are obvious injections

EFA → FA

NFA → EFA

DFA → NFA

induced by subset inclusion.

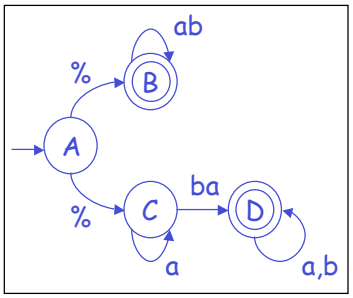


Today, we study the red conversions.

With these, we can convert any of the five representations to any other, demonstrating that they all describe the same set of languages = the **regular languages**.

A Running Example

Here's an example we'll use throughout this lecture:

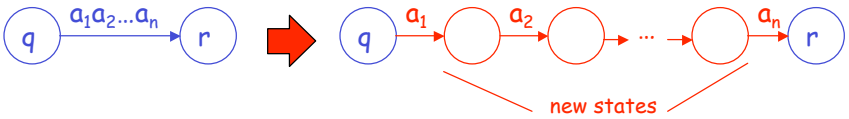


Which of the following strings are accepted, and why?

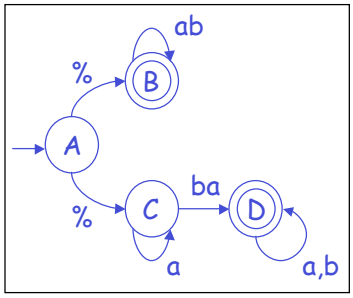
- aaa
- aba
- abaa
- abab
- aaba
- aabb
- aaaba

Converting FA to EFA

Simply break every transition with a multi-character label $a_1a_2...a_n$ into *n* single character transitions involving *n-1* new states:



Converting our running example:



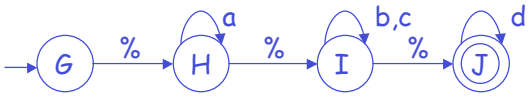
FA to EFA

Converting EFA to NFA: Some Notation

Some notation (mine, not Stoughton's)

- p %⇒ q means p goes to q via a transition labeled %
- p %⇒+ q means p goes to q via one or more transitions labeled % (the transitive closure of %⇒)
- p %⇒* q means p goes to q via zero or more transitions labeled % (the reflexive, transitive closure of %⇒)

Example:



%⇒	G	H	I	J
G				
H				
I				
J				

%⇒+	G	H	I	J
G				
H				
I				
J				

%⇒*	G	H	I	J
G				
H				
I				
J				

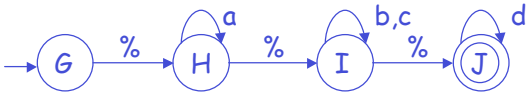
Converting EFA to NFA: Algorithm

If p %⇒+ q and q a⇒ r add a transition p a⇒ r.

If p %⇒+ q, where q is an accepting state, make p an accepting state (since it accepts %).

After performing the above steps everywhere, remove all % transitions.

Example:



Converting EFA to NFA: More Formally

States:*

$Q_{NFA} = Q_{EFA}$

Start State:

$s_{NFA} = s_{EFA}$

Accepting States:

$A_{NFA} = \{p \mid p \%\Rightarrow^* q \text{ in } T_{EFA} \text{ and } q \text{ in } A_{EFA}\}$

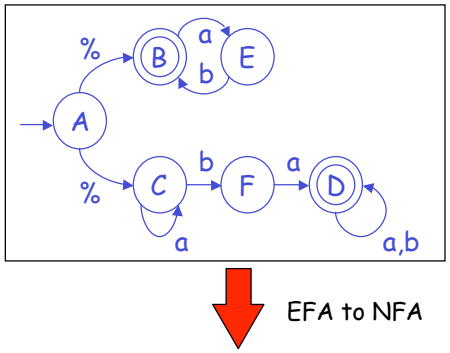
Transitions:

$T_{NFA} = \{(p,a,r) \mid p \%\Rightarrow^* q \text{ in } T_{EFA} \text{ and } (q,a,r) \text{ in } T_{EFA}\}$

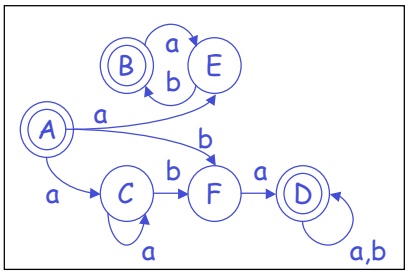
---

* Note: some states may become unreachable, and so can be omitted:

Converting EFA to NFA: Running Example



Current State of our Running Example



Which of the following strings are accepted, and why?

aaa

aba

abaa

abab

aaba

aabb

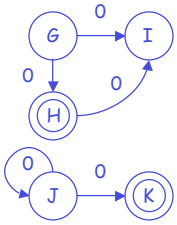
aaaba

Converting NFA to DFA: Idea

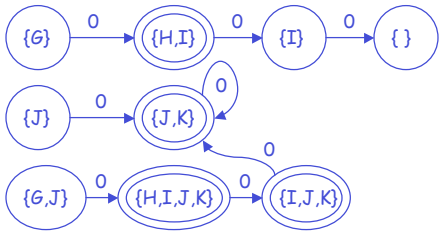
To construct a DFA from an NFA, we explore all possible labeled paths for a string **in parallel** and accept if any such path ends in an accepting state.

- Each state of DFA = set of states in NFA
- Start state of DFA = singleton set of start state of NFA
- An accepting state of DFA = any set of states from NFA containing an accepting state from the NFA
- A transition of DFA = set of all states in NFA that can be reached from a given set of states by following transitions labeled by a particular symbol.

Some NFA transitions



Some corresponding DFA transitions



NFA to DFA: The Subset Construction

The **subset construction** formalizes the sketch from the previous slide:

- Each state of DFA = set of states in NFA

  $Q_{DFA} = P(Q_{NFA})$ (powerset of Q_{NFA})

- Start state of DFA = singleton set of start state of NFA

  $s_{DFA} = \{s_{NFA}\}$

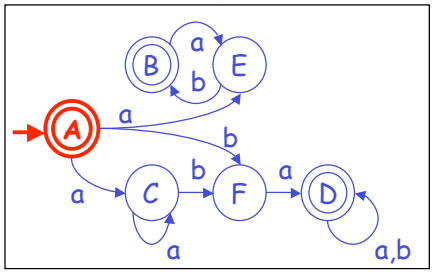
- An accepting state of DFA = any set of states from NFA containing an accepting state from the NFA

  $A_{DFA} = \{S \mid S \subseteq Q_{NFA} \text{ and } S \cap A_{NFA} \neq \emptyset\}$

- A transition of DFA = set of all states in NFA that can be reached from a given set of states by following transitions labeled by a particular symbol.

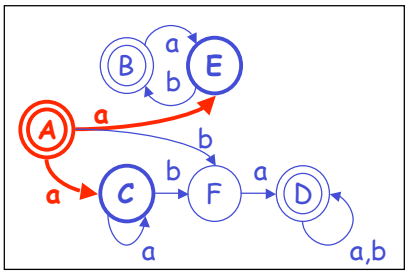
  $T_{DFA} = \{(S, a, S') \mid S \subseteq Q_{NFA} \text{ and } S' = \{q \mid p \in S \text{ and } (p,a,q) \in T_{NFA}\}\}$

Converting NFA to DFA: Running Example

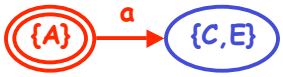


NFA to DFA

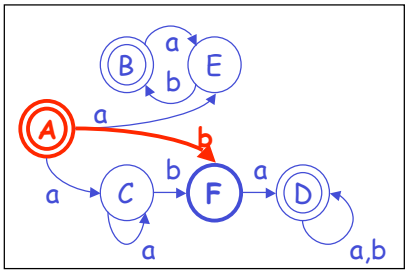
NFA to DFA Example: {A}, a -> {C,E}



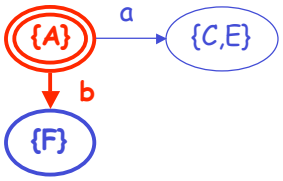
NFA to DFA



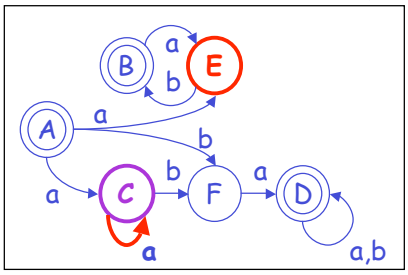
NFA to DFA Example: {A}, b -> {F}



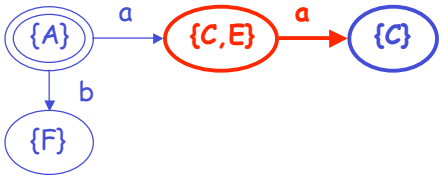
NFA to DFA



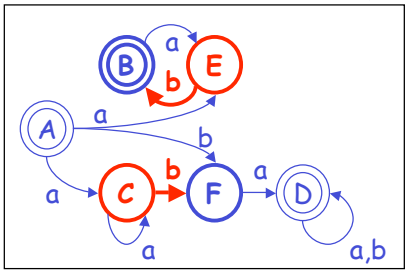
NFA to DFA Example: {C,E}, a -> {C}



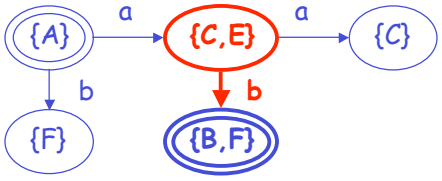
NFA to DFA



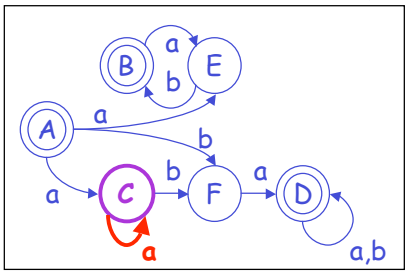
NFA to DFA Example: {C,E}, b -> {B,F}



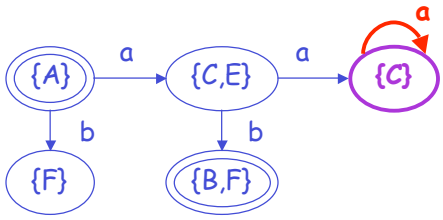
NFA to DFA



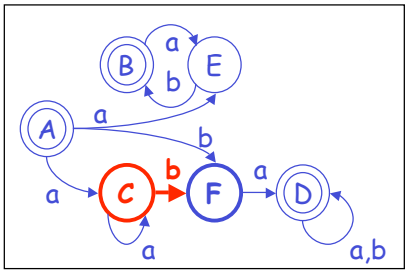
NFA to DFA Example: {C}, a -> {C}



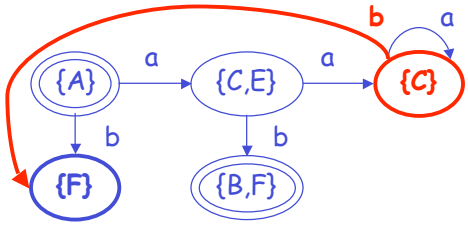
NFA to DFA



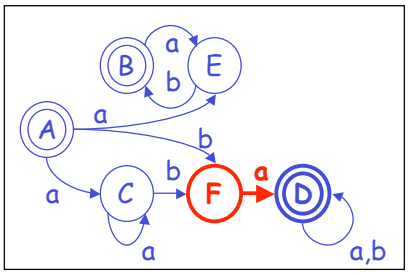
NFA to DFA Example: {C}, b -> {F}



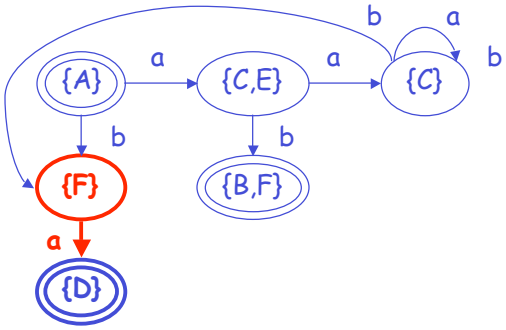
NFA to DFA



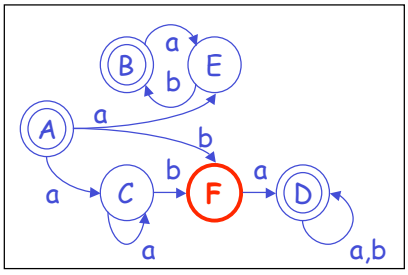
NFA to DFA Example: {F}, a -> {D}



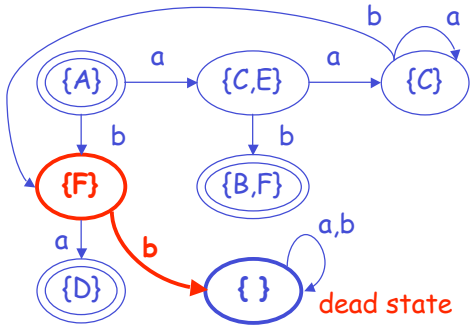
NFA to DFA



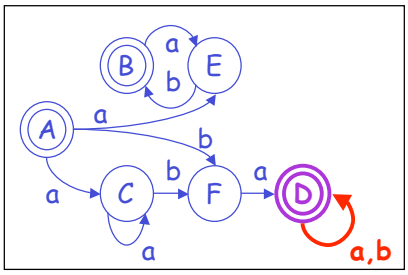
NFA to DFA Example: {F}, b -> { }



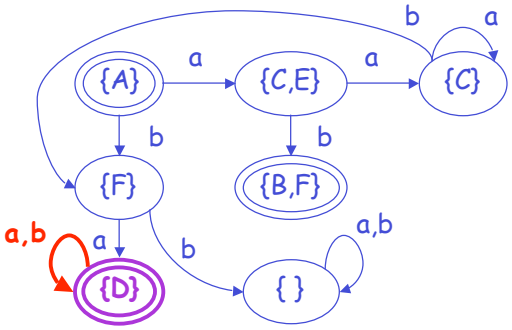
NFA to DFA



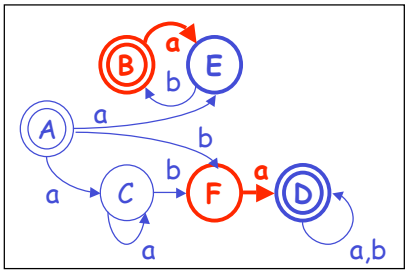
NFA to DFA Example: {D}, (a,b) -> {D}



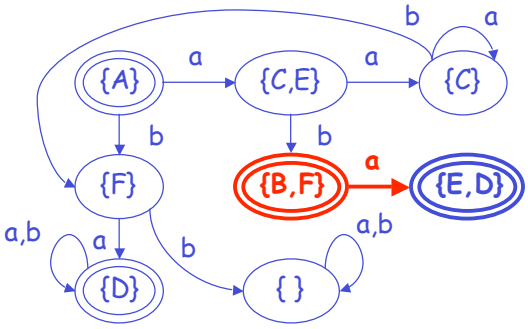
NFA to DFA



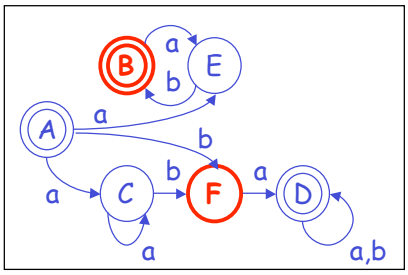
NFA to DFA Example: {B,F}, a -> {E,D}



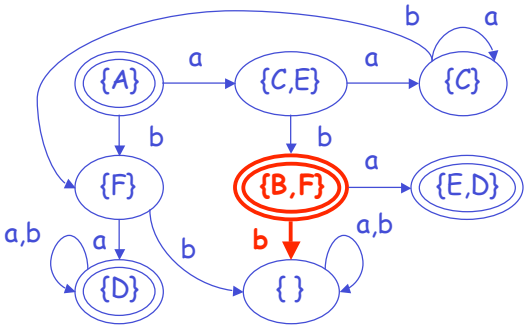
NFA to DFA



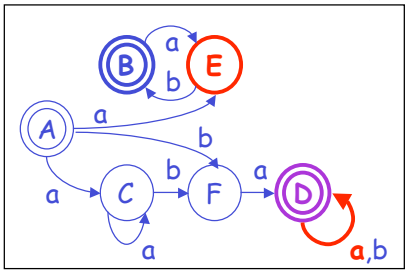
NFA to DFA Example: {B,F}, b -> { }



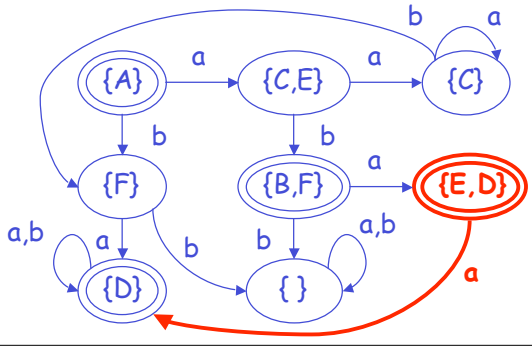
NFA to DFA



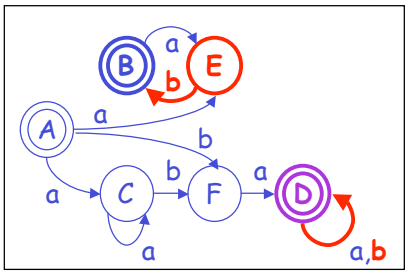
NFA to DFA Example: {E,D}, a -> {D}



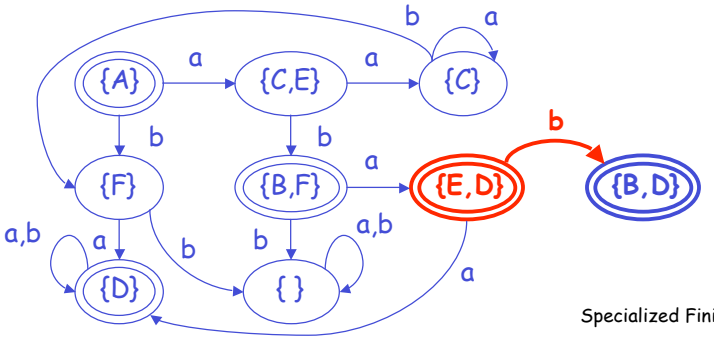
NFA to DFA



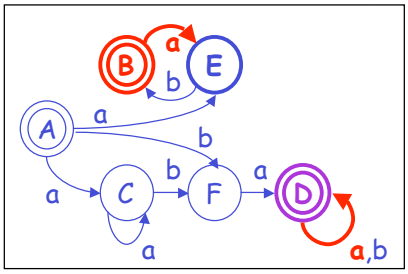
NFA to DFA Example: {E,D}, b -> {B,D}



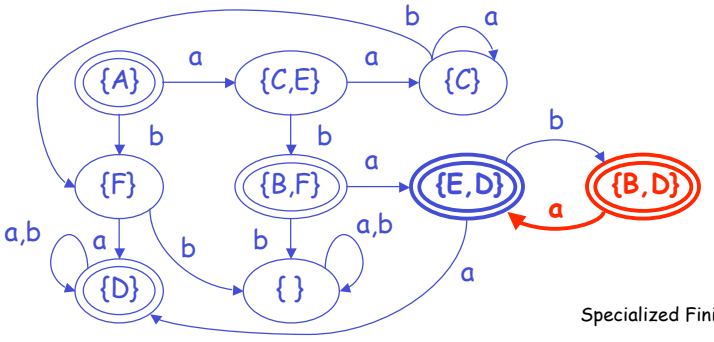
NFA to DFA



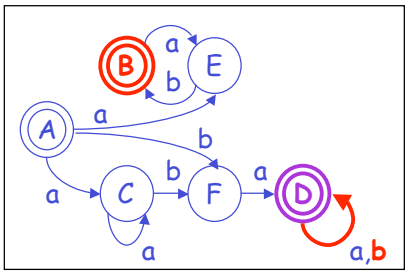
NFA to DFA Example: {B,D}, a -> {E,D}



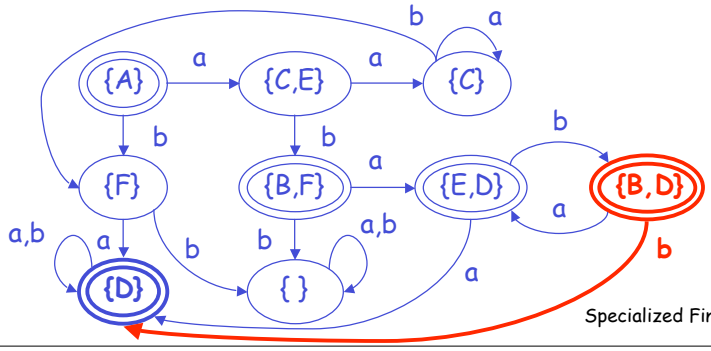
NFA to DFA



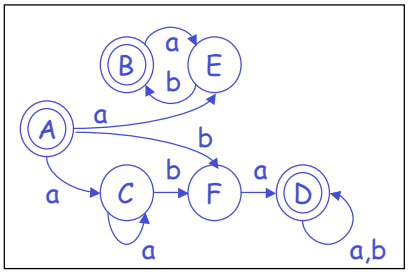
NFA to DFA Example: {B,D}, b -> {D}



NFA to DFA

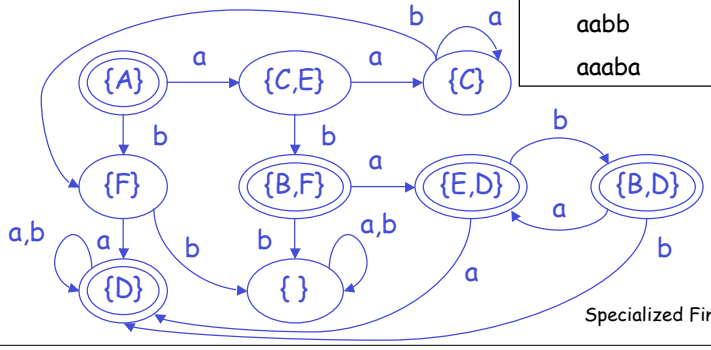


NFA to DFA Example: Final Configuration



Which of the following strings are accepted, and why?

- aaa
- aba
- abaa
- abab
- aaba
- aabb
- aaaba



Discussion

- Given an NFA with n states, how many reachable states could there be in the DFA in the worst case?
- Why does the subset construction algorithm terminate?
- Why is the subset construction algorithm correct?
- All the conversions we studied today are implemented in Forlan:

```
faToEFA : fa -> efa
efaToNFA : efa -> nfa
nfaToDFA : nfa -> dfa
val faToDFA = nfaToDFA o efaToNFA o faToEFA
```