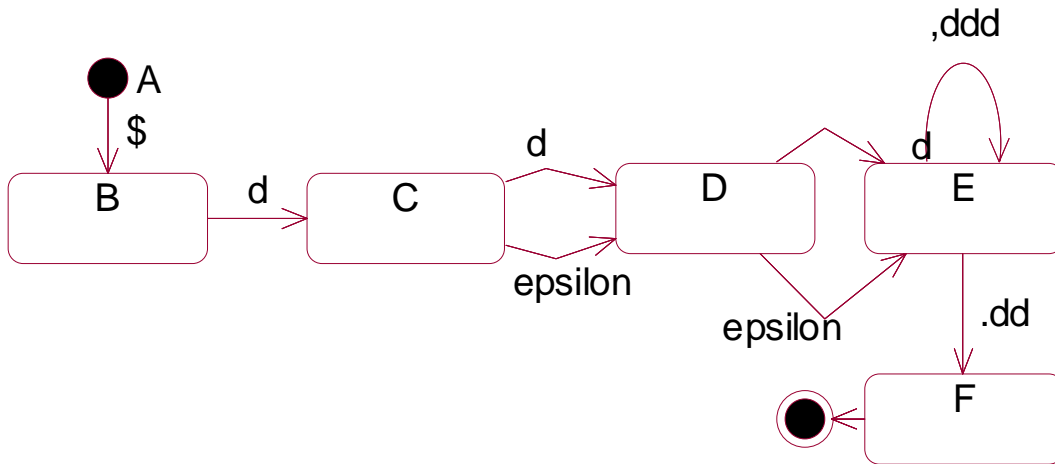


Finite State Machines

Nondeterministic Machines

We can describe a regular language using a finite state machine in which each edge (transition) is labelled with a sequence of terminals. For example:



Defines the language defined by

$$A \longrightarrow \$B$$

$$B \longrightarrow dC$$

$$C \longrightarrow D \mid dD$$

$$D \longrightarrow E \mid dE$$

$$E \longrightarrow ,dddE|.dd$$

The



indicates the start state (A in this case)

Any state with a transition to



is an end state (only F in this case).

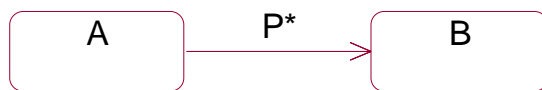
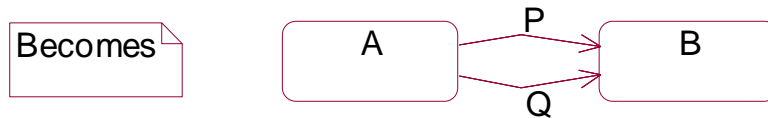
A string w is in the language iff there is a path from the start state to an end state in which the catenation of the labels is w .

Every regular expression corresponds to a finite state machine

Proof: Start with a transition labeled by a regular expression:



Now apply the following transformations



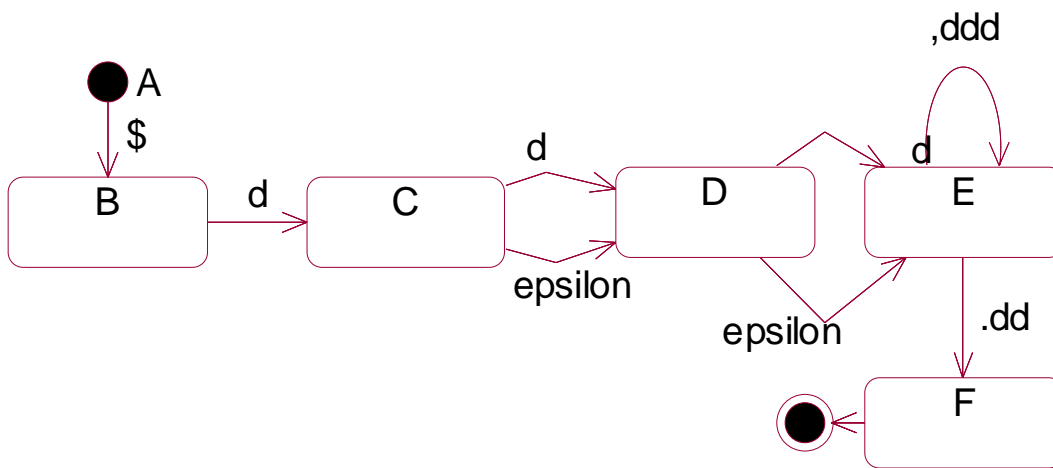
Furthermore, every finite state machine can be turned into an equivalent regular expression.

Thus nondeterministic finite state machines and regular expressions describe the same set of languages.

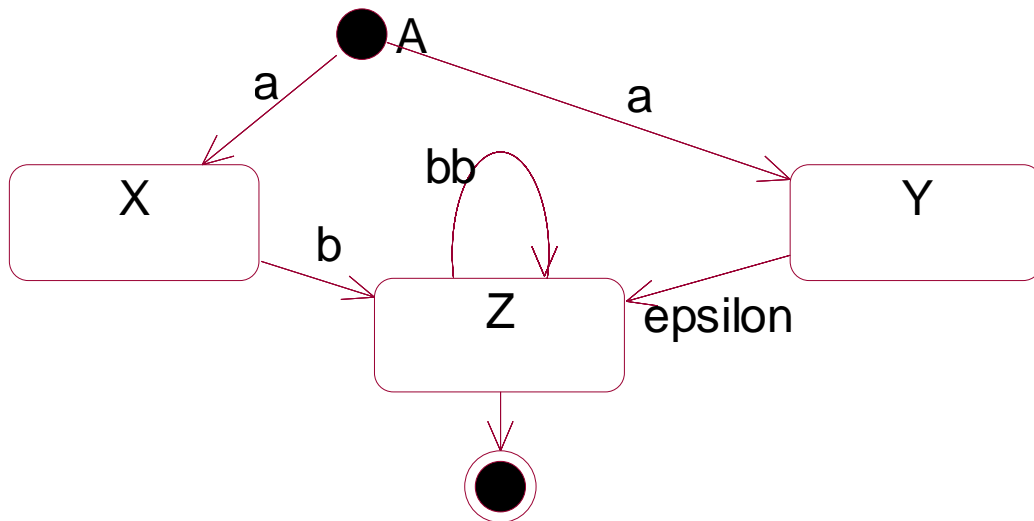
Deterministic finite state machines

The nondeterministic machines above describe languages better than they describe parsers.

Consider this nondeterministic machine



Consider a parser “in” state (C). If the next input terminal is a **d** should it go to state E by the **d** transition or the ϵ transition? In the above example, it is always safe to take the **d** transition, but consider this example:



From state A , with next input terminal being \mathbf{a} , there is a choice of whether to go to state X or Y .

The correct choice depends on whether the number of \mathbf{b} 's is odd or even.

The equivalent right-linear grammar is

$$A \longrightarrow aX \mid aY$$

$$X \longrightarrow bZ$$

$$Y \longrightarrow Z$$

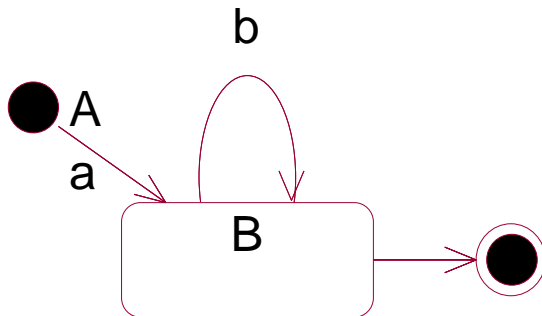
$$Z \longrightarrow bbZ \mid \epsilon$$

Deterministic finite state machines:

- Have no transitions labelled by ϵ
- No state has two exiting transitions whose labels start with

the same terminal.

A deterministic finite state machine for the above language:



The equivalent right-linear grammar is

$$\begin{aligned}
 A &\longrightarrow aB \\
 B &\longrightarrow bB \mid \epsilon
 \end{aligned}$$

Does determinism cost descriptive power

Recall that not every context-free language has an LL(1) grammar.

But, every regular language has a deterministic finite state machine.

Proof idea: *You can find an algorithm to convert a nondeterministic FSM into a deterministic FSM.*

Once you have a deterministic finite state machine recognizing in $O(1)$ space and $O(N)$ time is easy. The only memory your parser needs is the current state and the finite state machine itself.

Conversely if you can recognize a language in linear time and constant space, then it is a regular language.

The equivalence of four formalisms

Thus:

- regular expressions,
- right-linear grammars,
- nondeterministic finite state machines, and
- deterministic finite state machines

all describe the same set of languages: the regular languages.