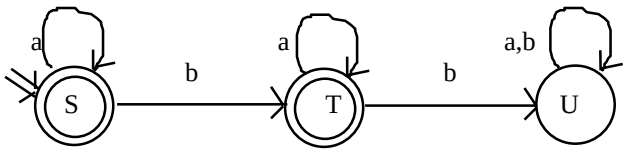
Interpreters for Finite State Machines

Deterministic FSAs as Algorithms

Example: No more than one b




```
S:  s := get-next-symbol;
    if s = end-of-file then accept;
    else if s = a then go to S;
    else if s = b then go to T;
T:  s:= get-next-symbol;
    if s = end-of-file then accept;
    else if s = a then go to T;
    else if s = b then go to U;
etc.
```

Length of Program: $|K| \times (|\Sigma| + 2)$
Time required to analyze string w: $O(|w| \times |\Sigma|)$

We have to write new code for every new FSM.

Until accept or reject do:

A Deterministic FSA Interpreter

To simulate M = $(K, \Sigma, \delta, s, F)$:

```
ST := s;
Repeat
        i := get-next-symbol;
        if i ≠ end-of-string then
                ST := δ(ST, i)
Until i = end-of-string;
If ST ∈ F then accept else reject
```

Simulate the no more than one b machine on input: aabaa

Nondeterministic FSAs as Algorithms

Real computers are deterministic, so we have three choices if we want to execute a nondeterministic FSA:

1. Convert the NDFSA to a deterministic one:
   - Conversion can take time and space 2^K.
   - Time to analyze string w: $O(|w|)$

2. Simulate the behavior of the nondeterministic one by constructing sets of states "on the fly" during execution
   - No conversion cost
   - Time to analyze string w: $O(|w| \times K^2)$

3. Do a depth-first search of all paths through the nondeterministic machine.

A Nondeterministic FSA Interpreter

To simulate $M = (K, \Sigma, \Delta, s, F)$:

```
SET ST;
ST := E(s);
Repeat
        i := get-next-symbol;
        if  i ≠ end-of-string then
                ST1 := ∅
                For all q ∈ ST do
                        For all r ∈ Δ(q, i) do
                                ST1 := ST1 ∪ E(r);
                ST := ST1;

Until i = end-of-string;
If  ST ∩  F ≠ ∅ then accept else reject
```

A Deterministic Finite State Transducer Interpreter

To simulate $M = (K, \Sigma, O, \delta, s, F)$, given that:

δ_1(state, symbol) returns a single new state (i.e., M is deterministic), and

δ_2(state, symbol) returns an element of O^*, the string to be output.

```
ST := s;
Repeat:
        i := get-next-symbol;
        if  i ≠ end-of-string then
                write(δ2(ST, i));
                ST := δ1(ST, i)
Until i = end-of-string;
If  ST ∈ F then accept else reject
```