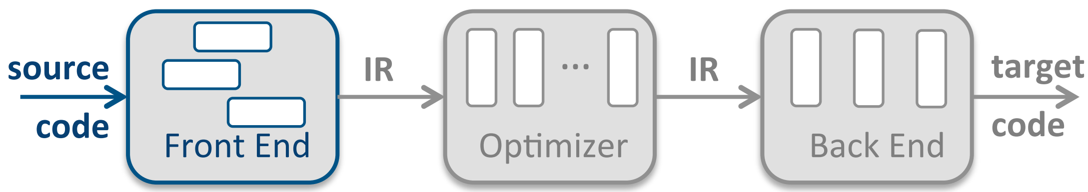
COMP 412
FALL 2017

Lexical Analysis, II

Comp 412

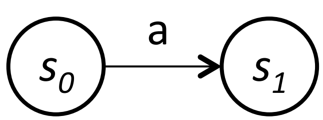


Chapter 2 in EaC2e

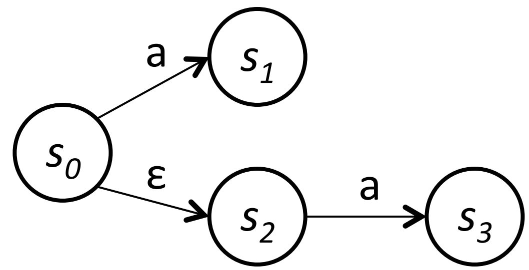
Determinism (or not)

So far, we have only looked at <u>deterministic</u> automata, or DFAs

- **DFA** ≅ <u>D</u>eterministic <u>F</u>inite <u>A</u>utomaton
- Deterministic means that it has only one transition out of a state on a given character



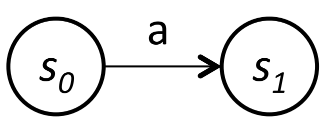
rather than



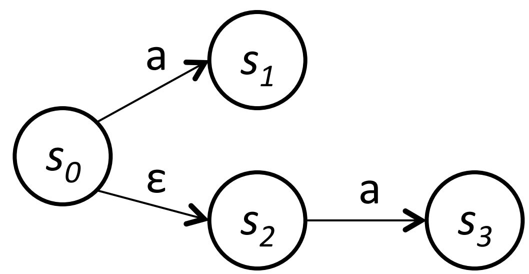
Determinism (or not)

So far, we have only looked at <u>deterministic</u> automata, or DFAs

- **DFA** ≅ <u>D</u>eterministic <u>F</u>inite <u>A</u>utomaton
- Deterministic means that it has only one transition out of a state on a given character



rather than



- Can a finite automaton have multiple transitions out of a single state on the same character?
  - *Yes, we call such an **FA** a <u>N</u>ondeterministic <u>F</u>inite <u>A</u>utomaton*
  - *And, yes, the **NFA** is truly an odd notion … but a useful one*
- **NFA**s and **DFA**s are equivalent
  - Sometimes, it is easier to build an **NFA** than to build a **DFA**

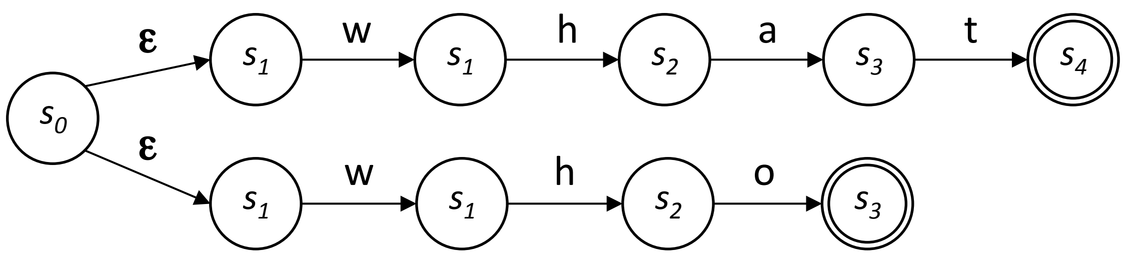
ε–transition does not consume an input character, which should worry us. (O(1) ?)

Whoa. What Does That **NFA** "Mean"?

An NFA accepts a string *x* *iff* ∃ a path though the transition graph from s_0 to a final state such that the edge labels spell *x*, ignoring ε's

Two models for **NFA** execution

1. To "run" the **NFA**, start in s_0 and ***guess*** the right transition at each step †
2. To "run" the **NFA**, start in s_0 and, at each non-deterministic choice, clone the **NFA** to purse all possible paths. If any of the clones succeeds, *accept*



NFA for "what | who"

In some sense, this same operational definition works on a **DFA**

† See page 44 in EaC2e.

Why Do We Care?

We need a construction that takes an RE to a DFA to a scanner.
NFAs will help up get there.

Overview:

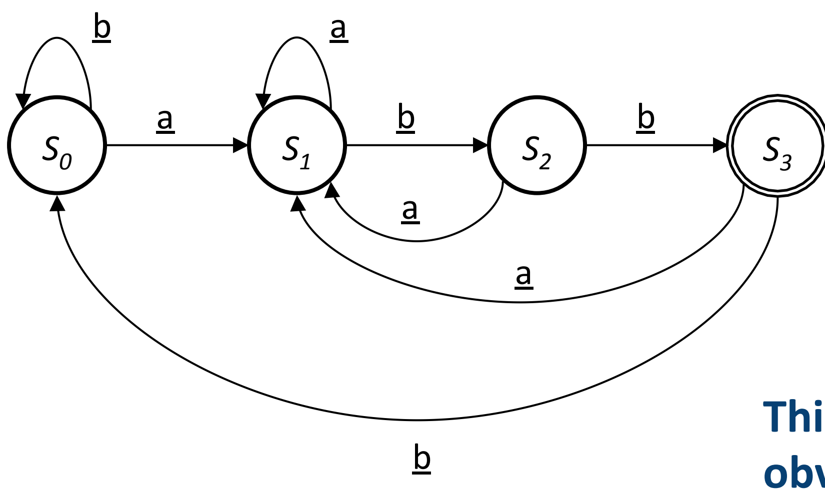
1. Simple and direct construction of a **nondeterministic finite automaton** (**NFA**) to recognize a given **RE**
   – Easy to build in an algorithmic way
   – Key idea is to combine **NFA**s for the terms with ε-transitions
2. Construct a **deterministic finite automaton** (**DFA**) that simulates the **NFA**
   – Use a set-of-states construction
3. Minimize the number of states in the **DFA**
   – We will look at 2 different algorithms: Hopcroft's & Brzozowski's
4. Generate the scanner code
   – Additional specifications needed for the actions

Optional, but worthwhile; reduces **DFA** size

lex and **flex** work along these lines

Example of a **DFA**

Here is a DFA for (a | b)* abb



This DFA is not particularly obvious from the RE.

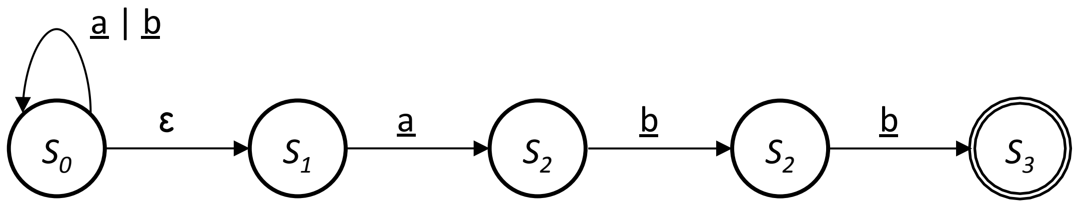
Each **RE** corresponds to one or more ***deterministic finite automatons*** (**DFAs**)

- We know a **DFA** exists for each **RE**
- The **DFA** may be hard to build directly
- Automatic techniques will build it for us …

For algorithm aficionados in the class, this **DFA** is reminiscent of the way that the failure function works in the Knuth, Morris, & Pratt sub-linear time pattern matcher.

Example as an **NFA**

Here is a simpler, more obvious NFA for (a | b)* abb



(a | b)* abb

Here is an **NFA** for the same language

- The relationship between the **RE** and the **NFA** is more obvious
- The ε-transition pastes together two **DFA**s to form a single **NFA**
- We can rewrite this **NFA** to eliminate the ε-transition
  - ε-transitions are an odd and convenient quirk of **NFA**s
  - Eliminating this one makes it obvious that it has 2 transitions on a from s_0

Relationship between **NFAs** and **DFAs**

DFA is a special case of an **NFA**

- **DFA** has no ε transitions
- **DFA**'s transition function is single-valued
- Same rules will work

DFA can be simulated with an **NFA**

  – *Obviously*

NFA can be simulated with a **DFA** *(less obvious, but still true)*

- Simulate sets of possible states
- Possible exponential blowup in the state space
- Still, one state per character in the input stream

⇒ **NFA** & **DFA** are equivalent in ability to recognize languages

Rabin & Scott, 1959

The Plan for Scanner Construction

RE → NFA *(Thompson's construction)*

- Build an **NFA** for each term in the **RE**
- Combine them in patterns that model the operators

NFA → DFA *(Subset construction)*

- Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**

- Hopcroft's algorithm
- Brzozowski's algorithm

Minimal **DFA** → Scanner

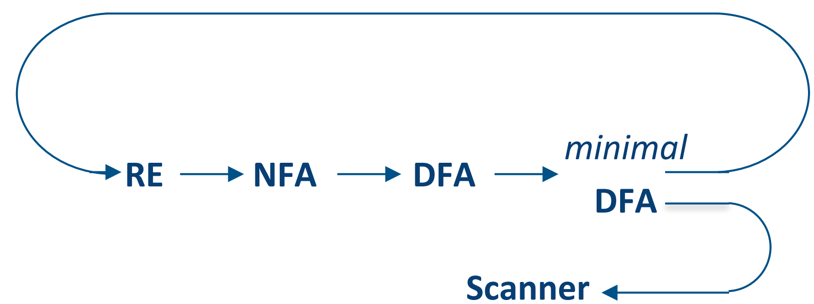
- See § 2.5 in EaC2e

DFA → RE

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

Automata Theory Moment
Taken together, the constructions on the cycle show that **RE**s, **NFA**s, and **DFA**s are all equivalent in their expressive power.

The Cycle of Constructions

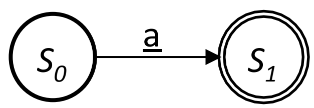


Taken together, these constructions prove that DFAs and REs are equivalent.

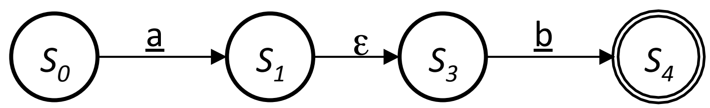
RE → NFA using Thompson's Construction

Key idea

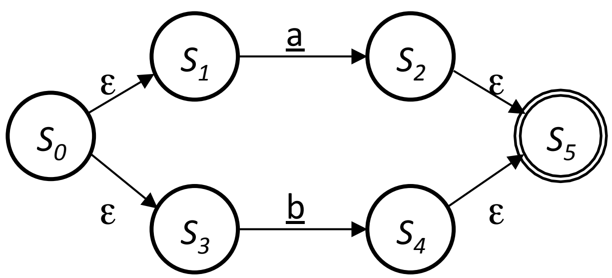
- **NFA** pattern for each symbol & each operator
- Join them with ε moves in precedence order



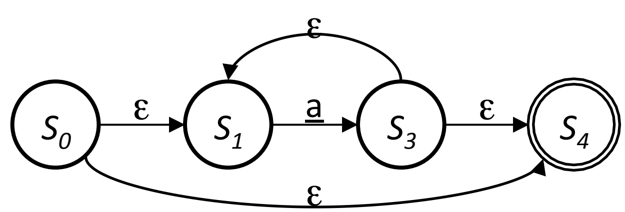
NFA for a



NFA for ab



NFA for a | b



NFA for a^*

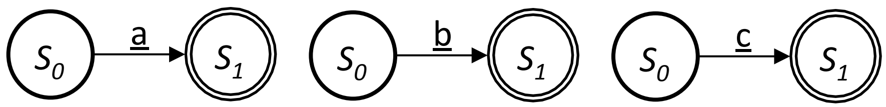
Precedence in REs:
Closure
Concatenation
Alternation

Ken Thompson, CACM, 1968

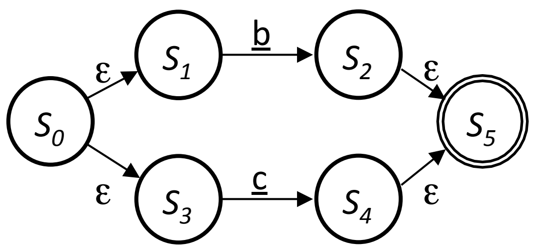
Example of Thompson's Construction

Let's build an NFA for a (b | c)*

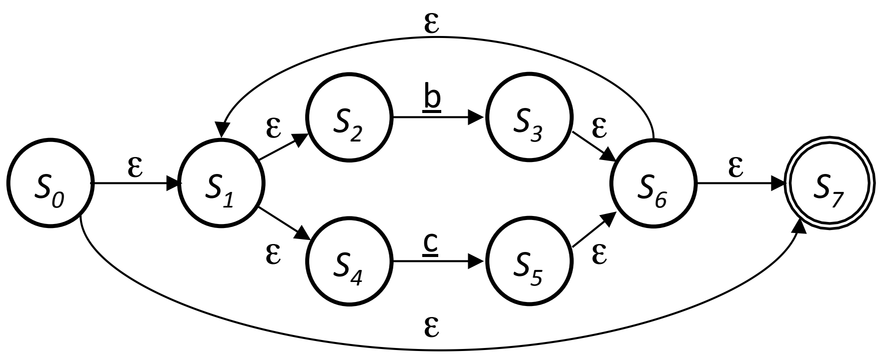
1. a, b, & c



2. b | c



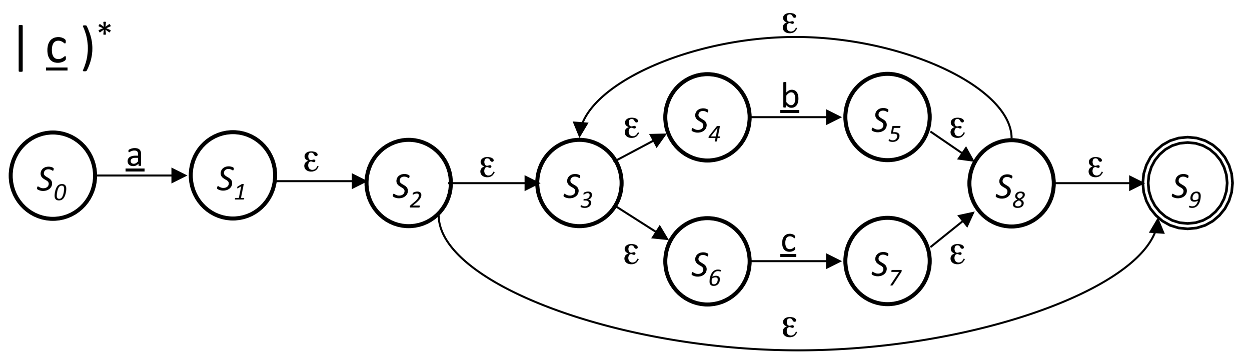
3. (b | c)*



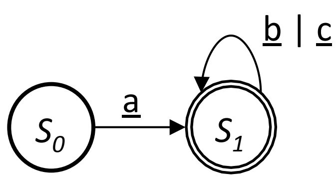
Note that states are being renamed at each step.

Example of Thompson's Construction

4. $\underline{a}\ (\ \underline{b}\ |\ \underline{c}\ )^*$



Of course, a human would design something simpler ...

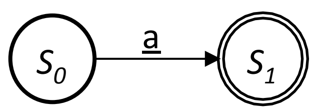


But, we can automate production of the more complex NFA version ...

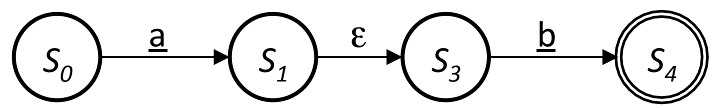
Thompson's Construction

Warning

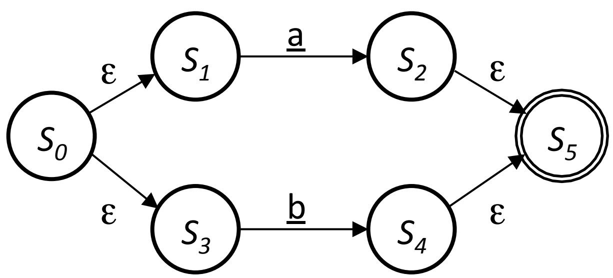
- You will be tempted to take shortcuts, such as leaving out some of the ε transitions
- Do not do it. Memorize these four patterns. They will keep you out of trouble.



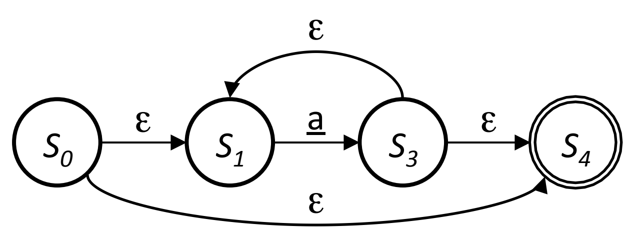
NFA for a



NFA for ab



NFA for a | b



NFA for a^*

Ken Thompson, CACM, 1968

The Plan for Scanner Construction

RE → NFA *(Thompson's construction)* ✔
- Build an **NFA** for each term in the **RE**
- Combine them in patterns that model the operators

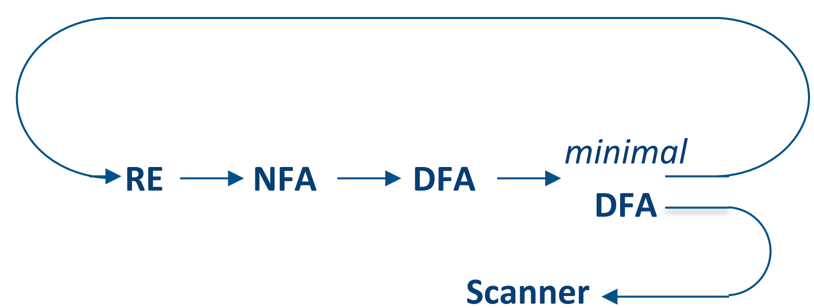
NFA → DFA *(Subset construction)*
- Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**
- Hopcroft's algorithm
- Brzozowski's algorithm

Minimal **DFA** → Scanner
- See § 2.5 in EaC2e

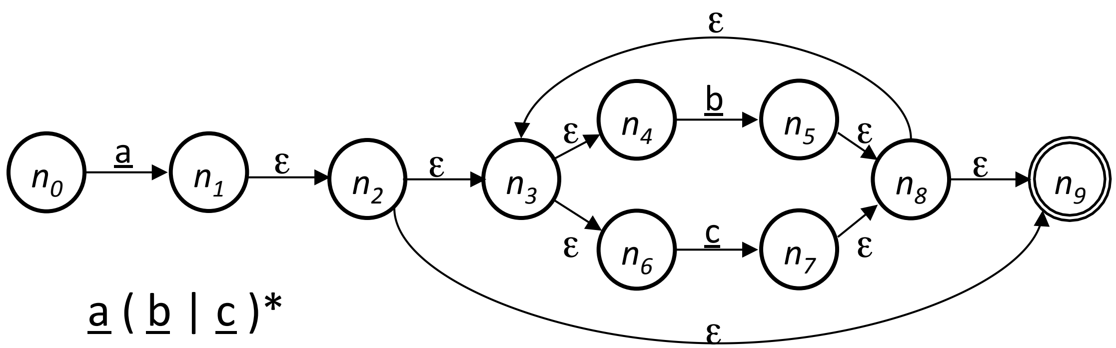
DFA → RE
- All pairs, all paths problem
- Union together paths from s_0 to a final state

The Cycle of Constructions



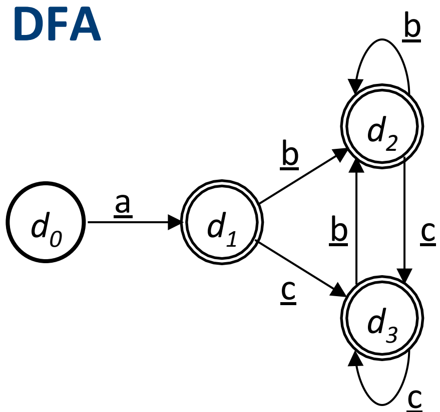
Simulating an **NFA** with a **DFA**

NFA



a (b | c)*

DFA



Where the mapping between **NFA** states and **DFA** states is:

DFA	NFA
d_0	n_0
d_1	n_1 n_2 n_3 n_4 n_6 n_9
d_2	n_5 n_8 n_9 n_3 n_4 n_6
d_3	n_7 n_8 n_9 n_3 n_4 n_6

NFA →DFA with Subset Construction

The subset construction builds a DFA that simulates the NFA

Two key functions

- *Move*(s_i, $\underline{a}$) is the set of states reachable from s_i by $\underline{a}$
- *FollowEpsilon*(s_i) is the set of states reachable from s_i by ε

The algorithm

- Derive the **DFA**'s start state from n_0 of the **NFA**
- Add all states reachable from n_0 by following ε
  - $d_0 = \mathit{FollowEpsilon}(\{n_0\})$
  - Let $\boldsymbol{D} = \{ d_0 \}$
- For $\alpha \in \Sigma$, compute $\mathit{FollowEpsilon}(\mathit{Move}(d_0, \alpha))$
  - If this creates a new state, add it to ***D***
- Iterate until no more states are added

It sounds more complex than it is…

Rabin & Scott, 1959

Any **DFA** state that contains a final state of the **NFA** becomes a final state of the **DFA**.

NFA $\rightarrow$**DFA** with Subset Construction

The algorithm:

$d_0 \leftarrow$ *FollowEpsilon(* $\{ n_0 \}$ *)*
$D \leftarrow \{ d_0 \}$
$W \leftarrow \{ d_0 \}$
while ($W \neq \emptyset$ *) {*
    select and remove s from W
    for each $\alpha \in \Sigma$ *{*
        $t \leftarrow$ *FollowEpsilon(Move(s,*α*))*
        $T[s,\alpha] \leftarrow t$
        if ($t \notin D$ *) then {*
            add t to D
            add t to W
        }
    }
}

d_0 is a set of states
D & W are sets of sets of states

The algorithm halts:

1. *D* contains no duplicates (test before addition)
2. $2^{\{\text{NFA states}\}}$ is finite
3. while loop adds to *D*, but does not remove from *D* *(monotone)*

$\Rightarrow$ the loop halts

D contains all the reachable **NFA** states

It tries each character in each d_i.

It builds every possible ***NFA*** *configuration.*

$\Rightarrow$ *D and T form the* ***DFA***

This test is a little tricky

NFA →DFA with Subset Construction

Example of a *fixed-point* computation

- Monotone construction of some finite set
- Halts when it stops adding to the set
- Proofs of halting & correctness are similar
- These computations arise in many contexts

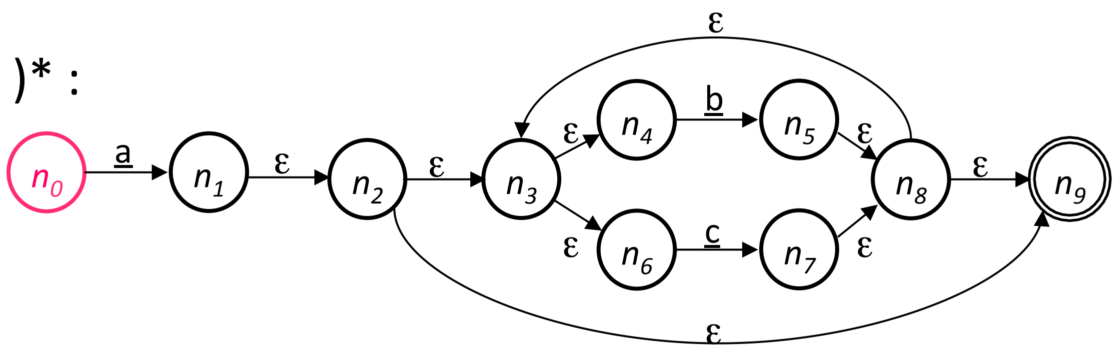
Other fixed-point computations

- Canonical construction of sets of LR(1) items
  - Quite similar to the subset construction
- Classic data-flow analysis & Gaussian Elimination
  - Solving sets of simultaneous set equations

We will see many more fixed-point computations

NFA → DFA with Subset Construction

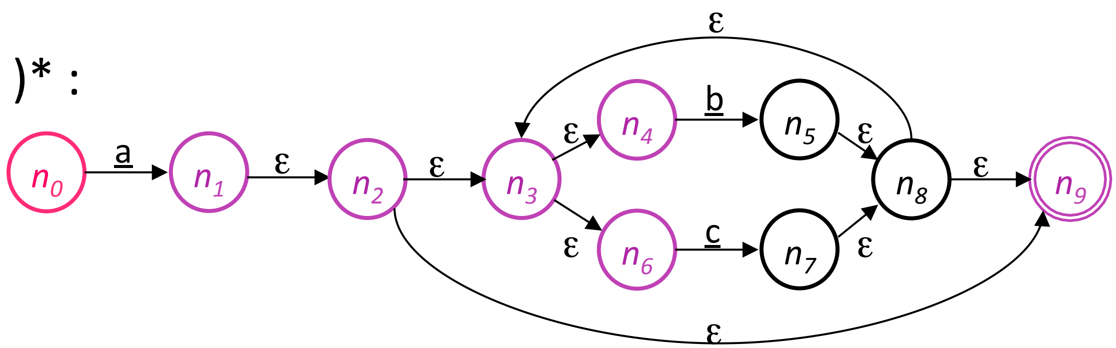
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0			

NFA → DFA with Subset Construction

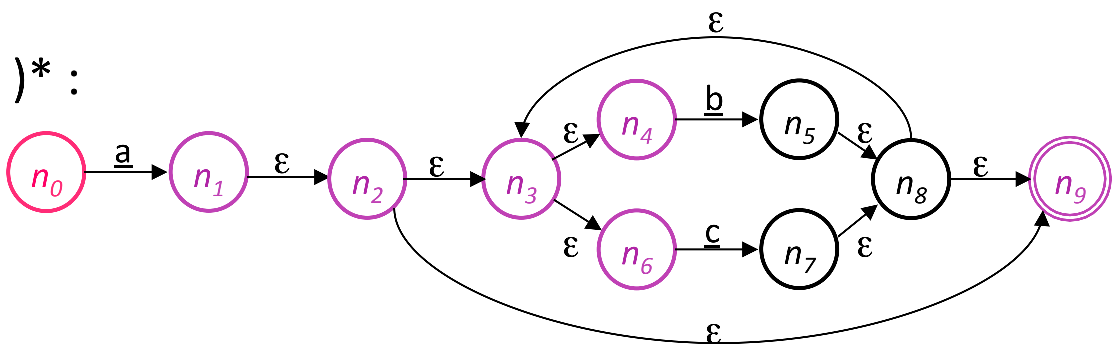
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9		

NFA → DFA with Subset Construction

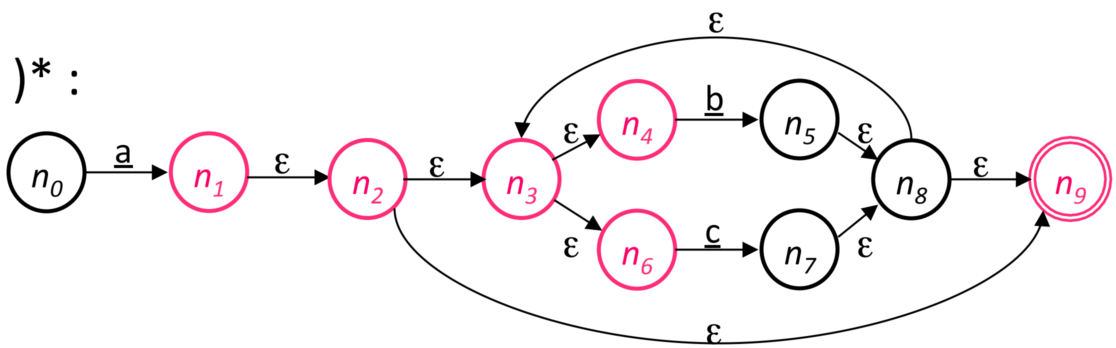
a (b | c)* :



States		FollowEpsilon (Move(s,*))		
DFA	NFA	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	none	none

NFA → DFA with Subset Construction

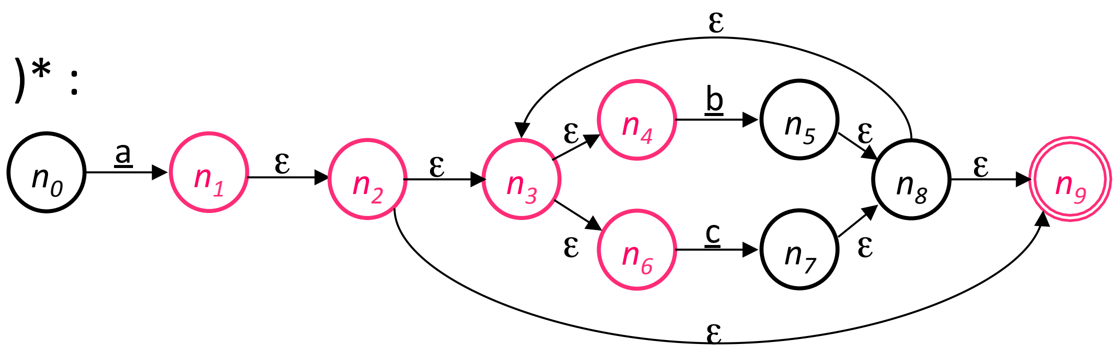
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	NFA	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9			

NFA → **DFA** with Subset Construction

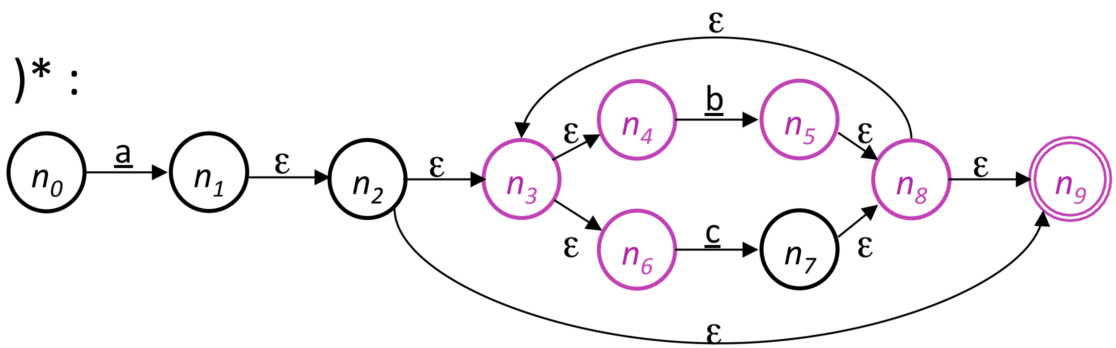
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9	*none*		

NFA → DFA with Subset Construction

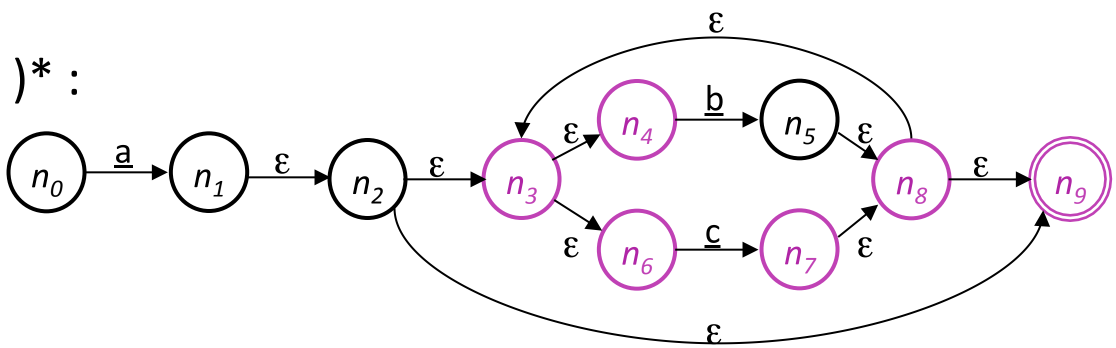
<u>a</u> (<u>b</u> | <u>c</u>)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	<u>a</u>	<u>b</u>	<u>c</u>
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9	*none*	n_5 n_8 n_9 n_3 n_4 n_6	

NFA → DFA with Subset Construction

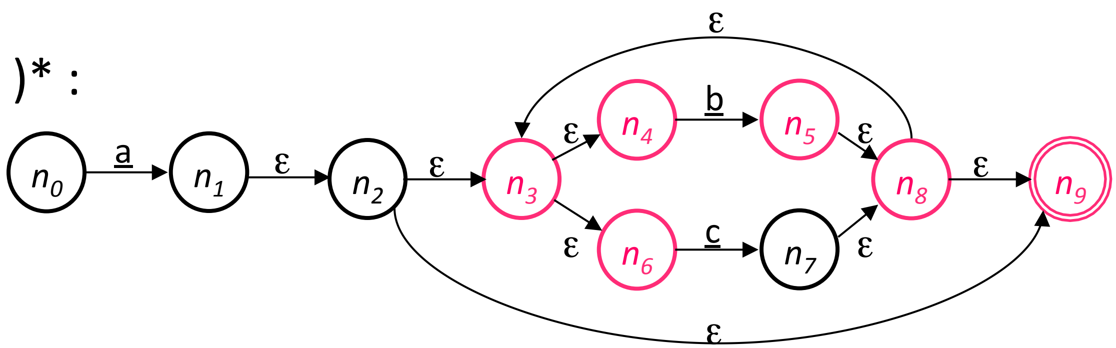
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9	*none*	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6

NFA → DFA with Subset Construction

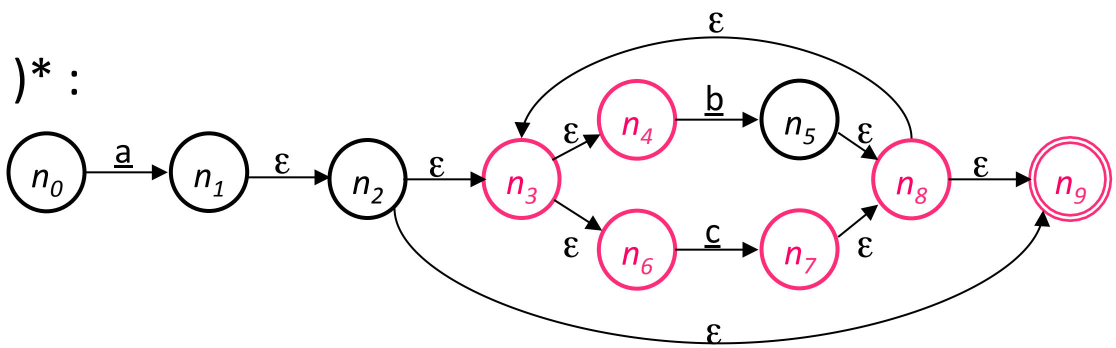
a (b | c)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9	*none*	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6
d_2	n_5 n_8 n_9 n_3 n_4 n_6			

NFA → DFA with Subset Construction

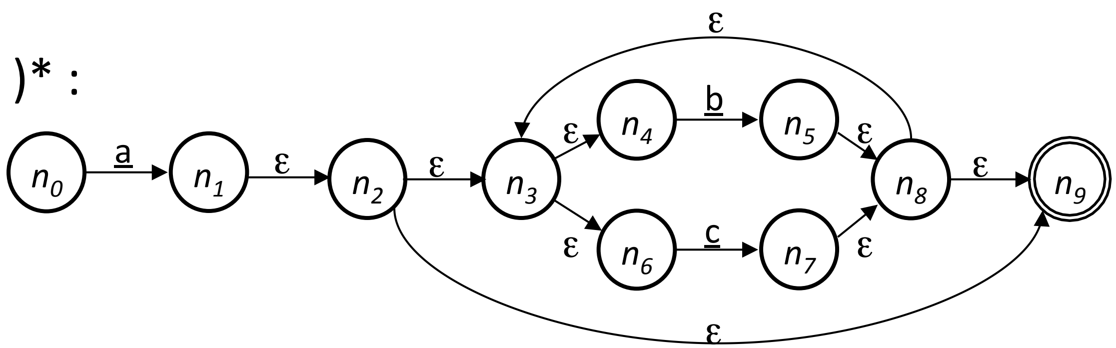
a (b | c)* :



States		FollowEpsilon (Move(s,*))		
DFA	NFA	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	none	none
d_1	n_1 n_2 n_3 n_4 n_6 n_9	none	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6
d_2	n_5 n_8 n_9 n_3 n_4 n_6			
d_3	n_7 n_8 n_9 n_3 n_4 n_6			

NFA → DFA with Subset Construction

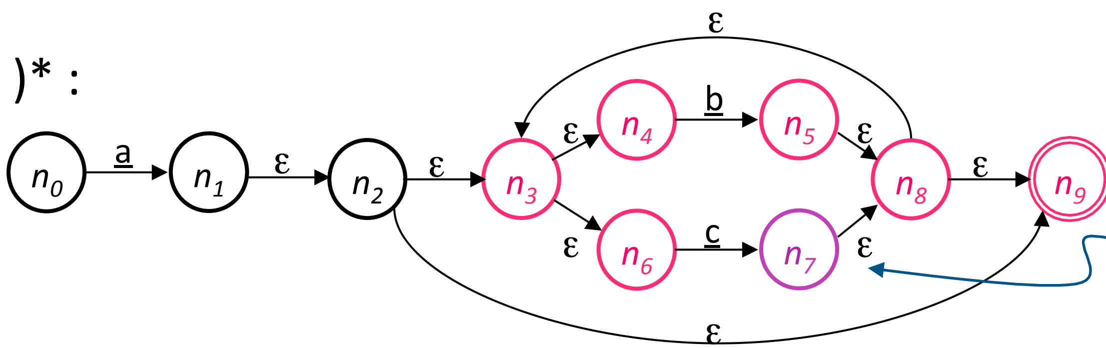
<u>a</u> (<u>b</u> | <u>c</u>)* :



States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	<u>a</u>	<u>b</u>	<u>c</u>
d_0	n_0	$n_1\ n_2\ n_3$ $n_4\ n_6\ n_9$	*none*	*none*
d_1	$n_1\ n_2\ n_3$ $n_4\ n_6\ n_9$	*none*	$n_5\ n_8\ n_9$ $n_3\ n_4\ n_6$	$n_7\ n_8\ n_9$ $n_3\ n_4\ n_6$
d_2	$n_5\ n_8\ n_9$ $n_3\ n_4\ n_6$	*none*		
d_3	$n_7\ n_8\ n_9$ $n_3\ n_4\ n_6$	*none*		

NFA → DFA with Subset Construction

a (b | c)* :

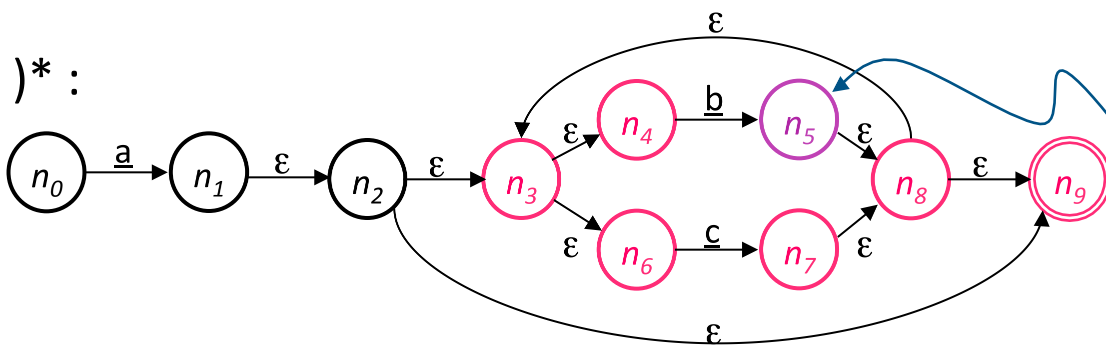


n_7 is the core state of d_3

States		FollowEpsilon (Move(s,*))		
DFA	NFA	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	none	none
d_1	n_1 n_2 n_3 n_4 n_6 n_9	none	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6
d_2	n_5 n_8 n_9 n_3 n_4 n_6	none	d_2	d_3
s_3	n_7 n_8 n_9 n_3 n_4 n_6	none		

NFA → DFA with Subset Construction

a (b | c)* :

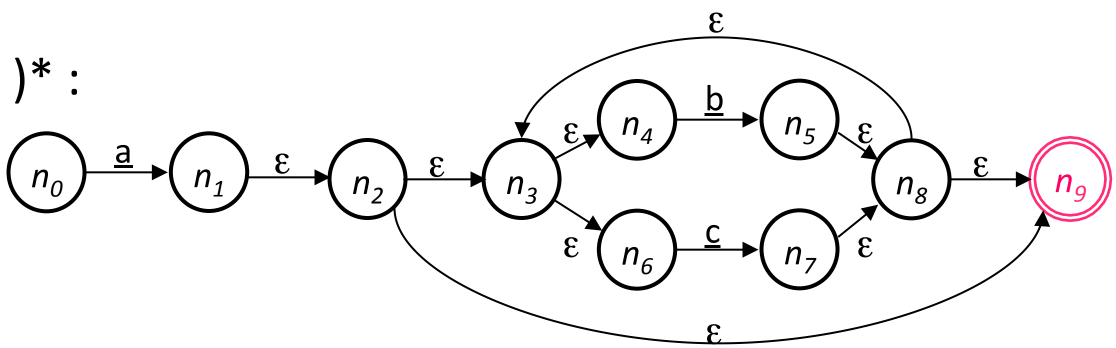


n_5 is the core state of d_2

States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	*none*	*none*
d_1	n_1 n_2 n_3 n_4 n_6 n_9	*none*	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6
d_2	n_5 n_8 n_9 n_3 n_4 n_6	*none*	d_2	d_3
d_3	n_7 n_8 n_9 n_3 n_4 n_6	*none*	d_2	d_3

NFA → DFA with Subset Construction

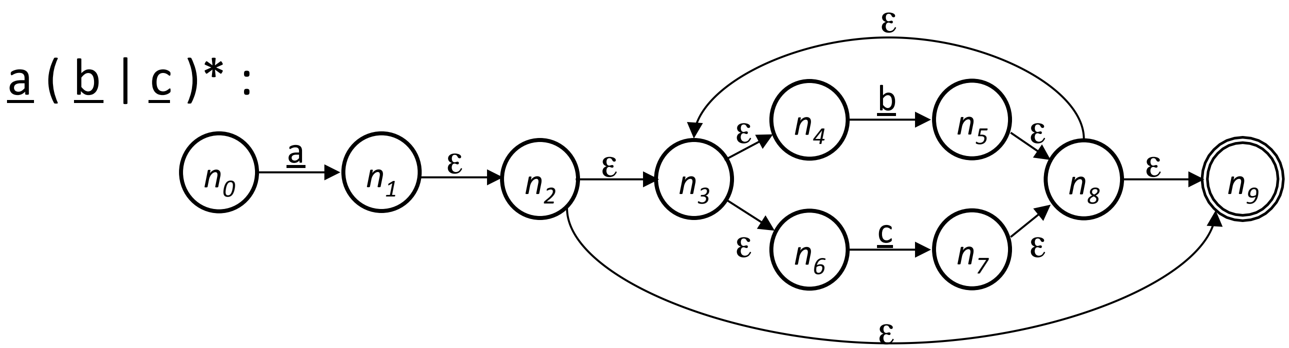
a (b | c)* :



States		FollowEpsilon (Move(s,*))		
DFA	NFA	a	b	c
d_0	n_0	n_1 n_2 n_3 n_4 n_6 n_9	none	none
d_1	n_1 n_2 n_3 n_4 n_6 n_9	none	n_5 n_8 n_9 n_3 n_4 n_6	n_7 n_8 n_9 n_3 n_4 n_6
d_2	n_5 n_8 n_9 n_3 n_4 n_6	none	d_2	d_3
d_3	n_7 n_8 n_9 n_3 n_4 n_6	none	d_2	d_3

Final states because of n_9

NFA → DFA with Subset Construction

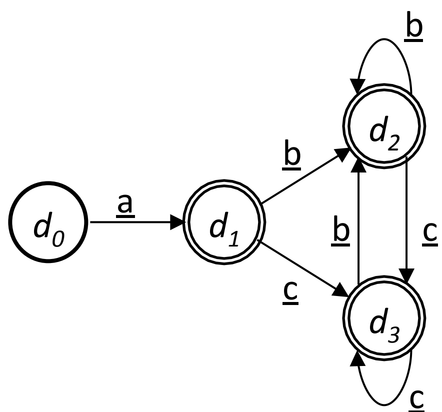


States		*FollowEpsilon (Move(s,*))*		
DFA	**NFA**	a	b	c
d_0	n_0	d_1	*none*	*none*
d_1	$n_1\ n_2\ n_3\ n_4\ n_6\ n_9$	*none*	d_2	d_3
d_2	$n_5\ n_8\ n_9\ n_3\ n_4\ n_6$	*none*	d_2	d_3
d_3	$n_7\ n_8\ n_9\ n_3\ n_4\ n_6$	*none*	d_2	d_3

Transition table for the DFA

NFA → **DFA** with Subset Construction

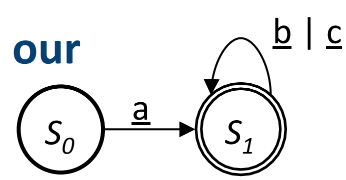
The **DFA** for a (b | c)*



	a	b	c
d_0	d_1	*none*	*none*
d_1	*none*	d_2	d_3
d_2	*none*	d_2	d_3
d_3	*none*	d_2	d_3

- Much smaller than the **NFA** (no ε-transitions)
- All transitions are deterministic
- Use same code skeleton as before

But, remember, our goal was:



Rabin and Scott, 1959 (page 8)

chines are more general than the ordinary ones, but this is not the case. We shall give a direct construction of an ordinary automaton, defining exactly the same set of tapes as a given nondeterministic machine.

Definition 11. *Let* $\mathfrak{A}=(S,M,S_0,F)$ *be a nondeterministic automaton.* $\mathfrak{D}(\mathfrak{A})$ *is the system* (T,N,t_0,G) *where* T *is the set of all subsets of* S, N *is a function on* $T\times\Sigma$ *such that* $N(t,\sigma)$ *is the union of the sets* $M(s,\sigma)$ *for* s *in* t, $t_0=S_0$, *and* G *is the set of all subsets of* S *containing at least one member of* F.

Clearly $\mathfrak{D}(\mathfrak{A})$ is an ordinary automaton, but it is actually equivalent to $\mathfrak{A}$.

Theorem 11. *If* $\mathfrak{A}$ *is a nondeterministic automaton, then* $T(\mathfrak{A})=T(\mathfrak{D}(\mathfrak{A}))$.

You must appreciate the "*clearly.*"

Proof: Assume first tha ... $T(\mathfrak{A})$ and let $s_0,s_1,\ldots$ ... states satisfying the conditions of Definition 10. We show by induction that for $k\leq n$, s_k is in $N(t_0,{}_0x_k)$. For $k=0$, $N(t_0,{}_0x_k)=N(t_0,\Lambda)=t_0=S_0$ and we were given that s_0 is in S_0. Assume the result for $k-1$. By definition, $N(t_0,{}_0x_k)=N(N(t_0,{}_0x_{k-1}),\sigma_{k-1})$. But we have assumed s_{k-1} is in $N(t_0,{}_0x_{k-1})$ so that from the definition of N we have $M(s_{k-1},\sigma_{k-1})\subset N(t_0,{}_0x_k)$. However, s_k is in $M(s_{k-1},\sigma_{k-1})$, and so the result is established. In particular s_n is in $N(t_0,{}_0x_n)=N(t_0,x)$, and since s_n is in F, we have $N(t_0,x)$ in G, which proves that x is in $T(\mathfrak{D}(\mathfrak{A}))$. Hence, we have shown that

$$T(\mathfrak{A})\subset T(\mathfrak{D}(\mathfrak{A})).$$

Assume next that a tape $x=\sigma_0\sigma_1\ldots\sigma_{n-1}$ is in $T(\mathfrak{D}(\mathfrak{A}))$. Let for each $k\leq n$, $t_k=N(t_0,{}_0x_k)$. We shall work backwards. First, we know that t_n is in G. Let then s_n be any internal state of $\mathfrak{A}$ such that s_n is in t_n and s_n is in F. Since s_n is in

$$t_n=N(t_0,{}_0x_n)=N(t_{n-1},\sigma_{n-1}),$$

we have from the definition of N that s_n is in $M(s_{n-1},\sigma_{n-1})$ for some s_{n-1} in t_{n-1}. But

Definition 12. *Let* $\mathfrak{A}=(S,M,S_0,F)$ *be a nondeterministic automaton. The dual of* $\mathfrak{A}$ *is the machine* $\mathfrak{A}^*=(S,M^*,F,S_0)$ *where the function* M^* *is defined by the condition*

s' *is in* $M^*(s,\sigma)$ *if and only if* s *is in* $M(s',\sigma)$.

Notice that we have at once the equation $\mathfrak{A}^{**}=\mathfrak{A}$. The relation between the sets defined by an automaton and its dual is as follows.

Theorem 12. *If* $\mathfrak{A}$ *is a nondeterministic automaton, then* $T(\mathfrak{A}^*)=T(\mathfrak{A})^*$.

Proof: In view of the equality $\mathfrak{A}^{**}=\mathfrak{A}$, we need only show $T(\mathfrak{A}^*)\subset T(\mathfrak{A})^*$. Let $x=\sigma_0\sigma_1\ldots\sigma_{n-1}$ be a tape ... ow that x^* is in $T(\mathfrak{A})$. Let $s_0,s_1,$ ... e of internal states of $\mathfrak{A}^*$ such ... S_0 and s_k is in $M^*(s_{k-1},\sigma_{k-1})$ for $k=1,2,\ldots,n$. Define a new sequence $s'_0,s'_1,\ldots,s'_n$ by the equation $s'_k=s_{n-k}$ for $k\leq n$. Obviously, s'_0 is in S_0 and s'_n is in F. Further, for $k>0$ and $k\leq n$, $s'_{k-1}=s_{n-k+1}$ is in $M^*(s_{n-k},\sigma_{n-k})$, or in other words, $s_{n-k}=s'_k$ is in $M(s'_{k-1},\sigma_{n-k})$. Now defining a new sequence of symbols $\sigma'_0\sigma'_1\ldots\sigma'_{n-1}$ by the formula $\sigma'_k=\sigma_{n-k-1}$, we see that $\sigma'_{k-1}=\sigma_{n-k}$ and $\sigma'_0\sigma'_1\ldots\sigma'_{n-1}=x^*$. Thus, x^* is in $T(\mathfrak{A})$ as was to be proved.

It should be noted that Theorem 12 together with Theorem 11 yields a direct construction and proof for Theorem 4 of Section 3 which was first proved by the indirect method of Theorem 1. In the next section we make heavy use of the direct constructions supplied by the nondeterministic machines to obtain results not easily apparent from the mathematical characterizations of Theorems 1 and 2.

6. *Further closure properties*

Simplifying a result due originally to Kleene, Myhill in unpublished work has shown that the class $\mathcal{T}$ can be characterized as the least class of sets of tapes containing the finite sets and closed under some simple operations on sets of tapes. We indicate here a different proof using

The Plan for Scanner Construction

RE → **NFA** *(Thompson's construction)* ✔
- Build an **NFA** for each term in the **RE**
- Combine them in patterns that model the operators

NFA → **DFA** *(Subset construction)* ✔
- Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**
- Hopcroft's algorithm
- Brzozowski's algorithm

Minimal **DFA** → Scanner
- See § 2.5 in EaC2e

DFA → **RE**
- All pairs, all paths problem
- Union together paths from s_0 to a final state

The Cycle of Constructions