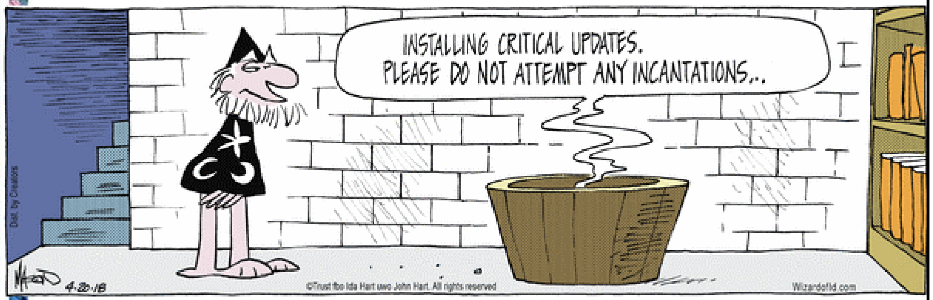
MA/CSSE 474
Theory of Computation

CFL Hierarchy
CFL Decision Problems

Your Questions?

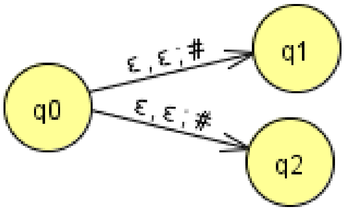
- Previous class days' material
- Reading Assignments
- HW 12 or 13 problems
- Anything else

I have included some slides online that we will not have time to do in class, but may be helpful to you anyway.


INSTALLING CRITICAL UPDATES.
PLEASE DO NOT ATTEMPT ANY INCANTATIONS...
Dist. by Creators
4-20-18
©Trust fbo Ida Hart uwo John Hart. All rights reserved
Wizardofid.com

{*xcy* : *x*, *y* ∈ {0, 1}* and *x* ≠ *y*}

- SURPRISINGLY, it is Context-free! HW 13. Here is the beginning of a proof:
- We can build a PDA *M* to accept *L*. All *M* has to do is to find one way in which *x* and *y* differ.
- *M* starts by pushing a bottom of stack marker # onto the stack.
- Then it nondeterministically chooses to go to state 1 or 2.



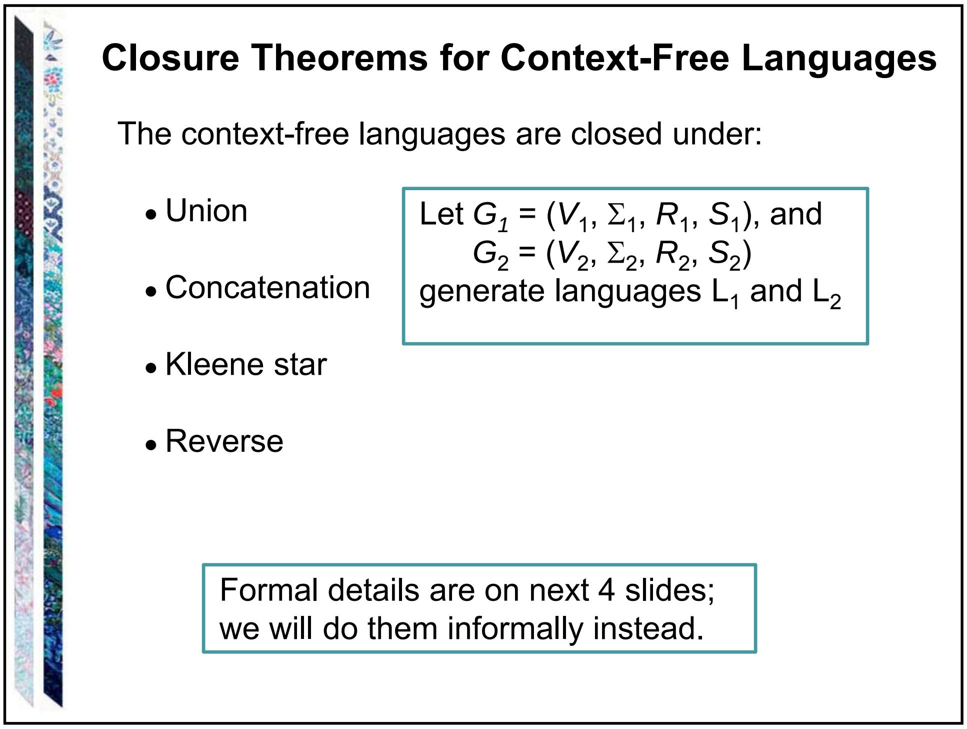
PDA Variations?

- In HW12, we see that acceptance by "accepting state only" is equivalent to acceptance by empty stack and accepting state.

  Equivalent In this sense: Given a language L, there is a PDA that accepts L by accepting state and empty stack iff there is a PDA that accepts L by accepting state only.

- FSM plus two stacks?

- FSM plus FIFO queue (instead of stack)?



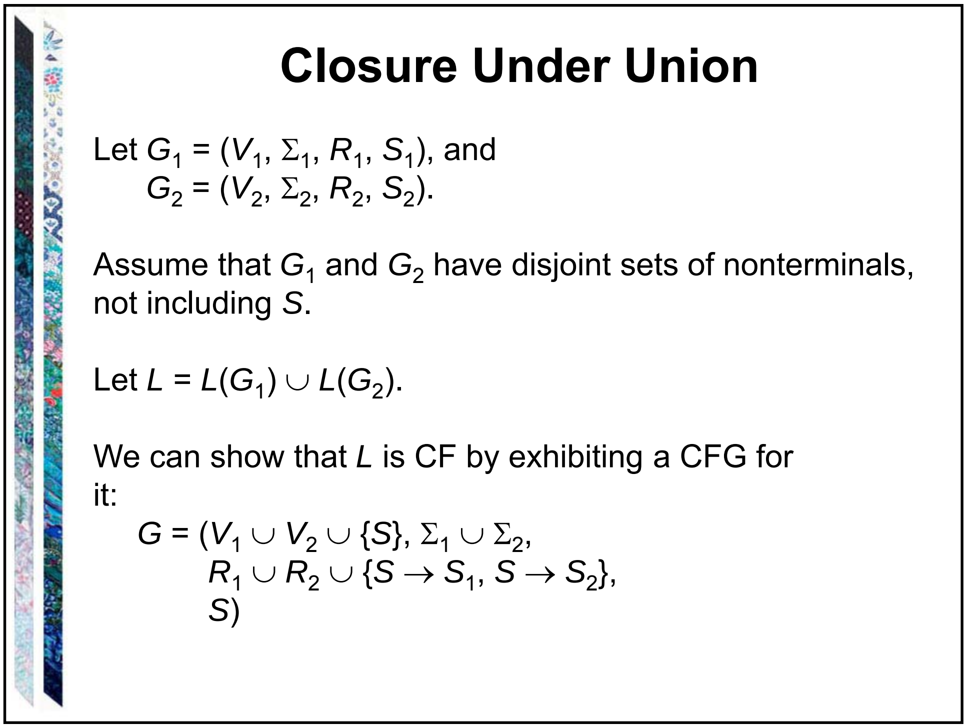
Closure Theorems for Context-Free Languages

The context-free languages are closed under:

- Union
- Concatenation
- Kleene star
- Reverse

Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$, and
$G_2 = (V_2, \Sigma_2, R_2, S_2)$
generate languages L_1 and L_2

Formal details are on next 4 slides;
we will do them informally instead.



Closure Under Union

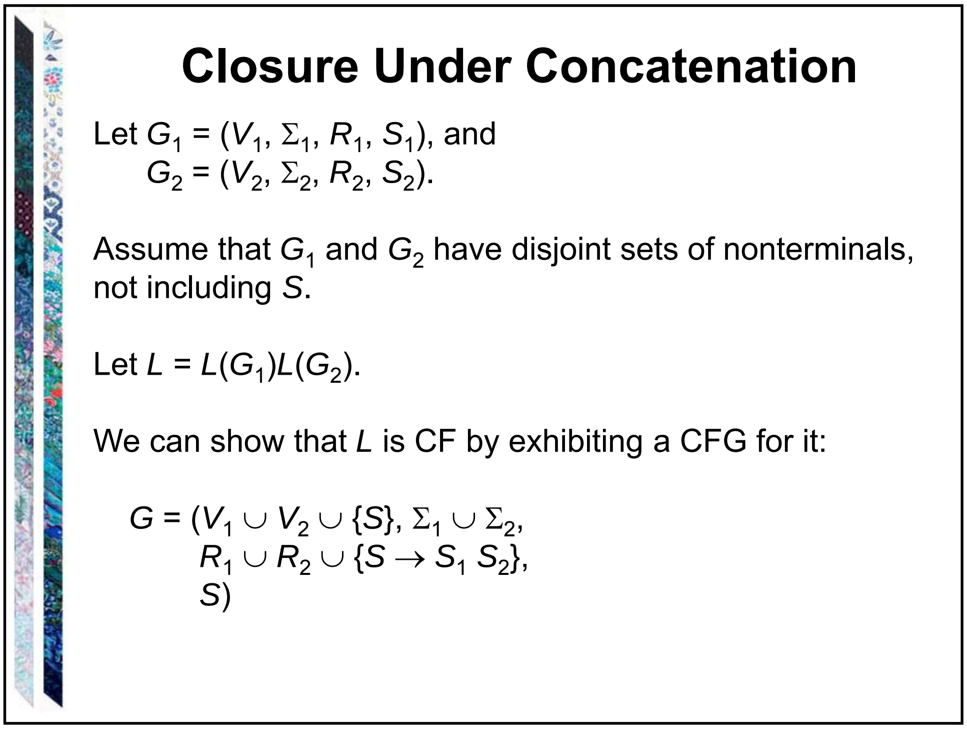
Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$, and
$G_2 = (V_2, \Sigma_2, R_2, S_2)$.

Assume that G_1 and G_2 have disjoint sets of nonterminals, not including S.

Let $L = L(G_1) \cup L(G_2)$.

We can show that L is CF by exhibiting a CFG for it:

$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2,$
$R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\},$
$S)$



Closure Under Concatenation

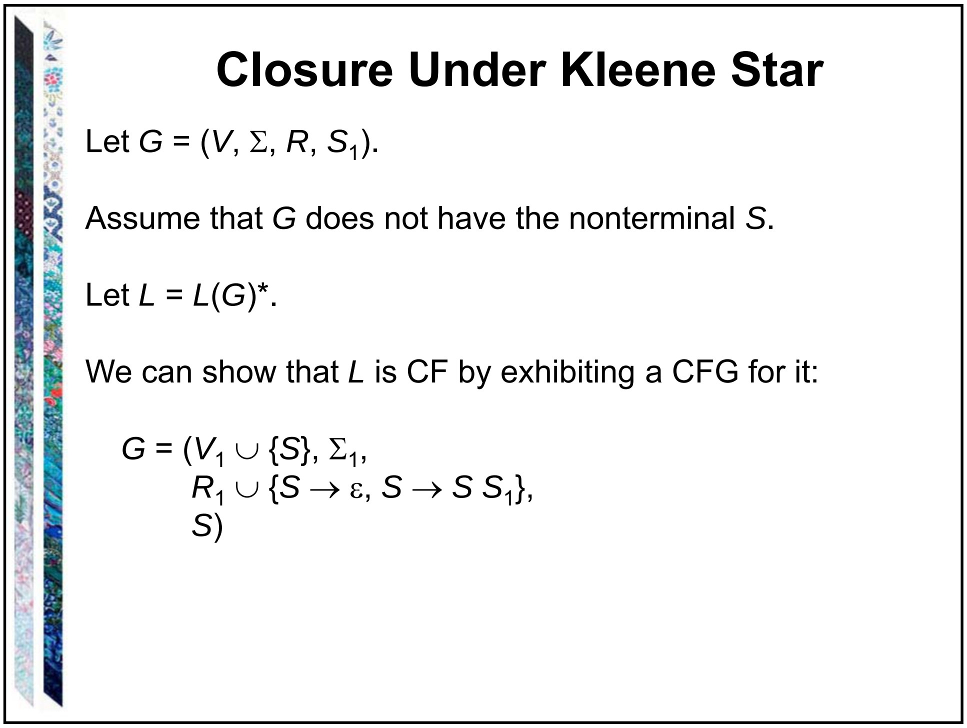
Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$, and
$G_2 = (V_2, \Sigma_2, R_2, S_2)$.

Assume that G_1 and G_2 have disjoint sets of nonterminals, not including S.

Let $L = L(G_1)L(G_2)$.

We can show that L is CF by exhibiting a CFG for it:

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2,\\ R_1 \cup R_2 \cup \{S \rightarrow S_1\ S_2\},\\ S)$$



Closure Under Kleene Star

Let $G = (V, \Sigma, R, S_1)$.

Assume that G does not have the nonterminal S.

Let $L = L(G)^*$.

We can show that L is CF by exhibiting a CFG for it:

$$G = (V_1 \cup \{S\}, \Sigma_1,\\ R_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S\ S_1\},\\ S)$$

Closure Under Reverse

$L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$.

Let $G = (V, \Sigma, R, S)$ be in Chomsky normal form.

Every rule in G is of the form $X \rightarrow BC$ or $X \rightarrow a$, where X, B, and C are elements of $V - \Sigma$ and $a \in \Sigma$.

- $X \rightarrow a$: $L(X) = \{a\}$. $\{a\}^R = \{a\}$.
- $X \rightarrow BC$: $L(X) = L(B)L(C)$. $(L(B)L(C))^R = L(C)^R L(B)^R$.

Construct, from G, a new grammar G', such that $L(G') = L^R$:
$G' = (V_G, \Sigma_G, R', S_G)$, where R' is constructed as follows:

- For every rule in G of the form $X \rightarrow BC$, add to R' the rule $X \rightarrow CB$.
- For every rule in G of the form $X \rightarrow a$, add to R' the rule $X \rightarrow a$.

Closure Under Intersection

The context-free languages are not closed under intersection:

The proof is by counterexample. Let:

$L_1 = \{a^n b^n c^m : n, m \geq 0\}$ /* equal a's and b's.
$L_2 = \{a^m b^n c^n : n, m \geq 0\}$ /* equal b's and c's.

Both L_1 and L_2 are context-free, since there exist straightforward context-free grammars for them.

But now consider:
$L = L_1 \cap L_2$
$= \{a^n b^n c^n : n \geq 0\}$

Recall: Closed under union but not closed under intersection implies not closed under complement.
And we saw a specific example of a CFL whose complement was not CF.

Closure Under Complement

$L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$

The context-free languages are closed under union, so if they were closed under complement, they would be closed under intersection (which they are not).

Alternative approach:
In a previous class, we demonstrated that the complement of L = $A^nB^nC^n$ is context-free, while L itself is not context-free,

The Intersection of a Context-Free Language and a Regular Language is Context-Free

$L = L(M_1)$, a PDA = $(K_1, \Sigma, \Gamma_1, \Delta_1, s_1, A_1)$.
$R = L(M_2)$, a deterministic FSM = $(K_2, \Sigma, \delta, s_2, A_2)$.

We construct a new PDA, M_3, that accepts $L \cap R$ by simulating the parallel execution of M_1 and M_2.

$M = (K_1 \times K_2, \Sigma, \Gamma_1, \Delta, [s_1, s_2], A_1 \times A_2)$.

Insert into Δ:

For each rule $((q_1, a, \beta), (p_1, \gamma))$ in Δ_1,
and each rule (q_2, a, p_2) in δ,
Δ contains $(([q_1, q_2]\ a, \beta), ([p_1, p_2], \gamma))$.

For each rule $((q_1, \varepsilon, \beta), (p_1, \gamma)$ in Δ_1,
and each state q_2 in K_2,
Δ contains $(([q_1, q_2], \varepsilon, \beta), ([p_1, q_2], \gamma))$.

I use square brackets for ordered pairs of states from $K_1 \times K_2$, to distinguish them from the tuples that are part of the notations for transitions in M_1, M_2, and M.

This works because: we can get away with only one stack.

The Difference between a Context-Free Language and a Regular Language is Context-Free

Theorem: The difference ($L_1 - L_2$) between a context-free language L_1 and a regular language L_2 is context-free.

Proof: $L_1 - L_2 = L_1 \cap \neg L_2$.

If L_2 is regular then so is $\neg L_2$.

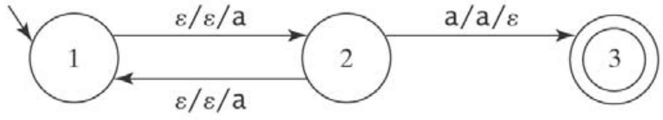
If L_1 is context-free, so is $L_1 \cap \neg L_2$.

Halting

It is possible that a PDA may

- not halt,
- never finish reading its input.

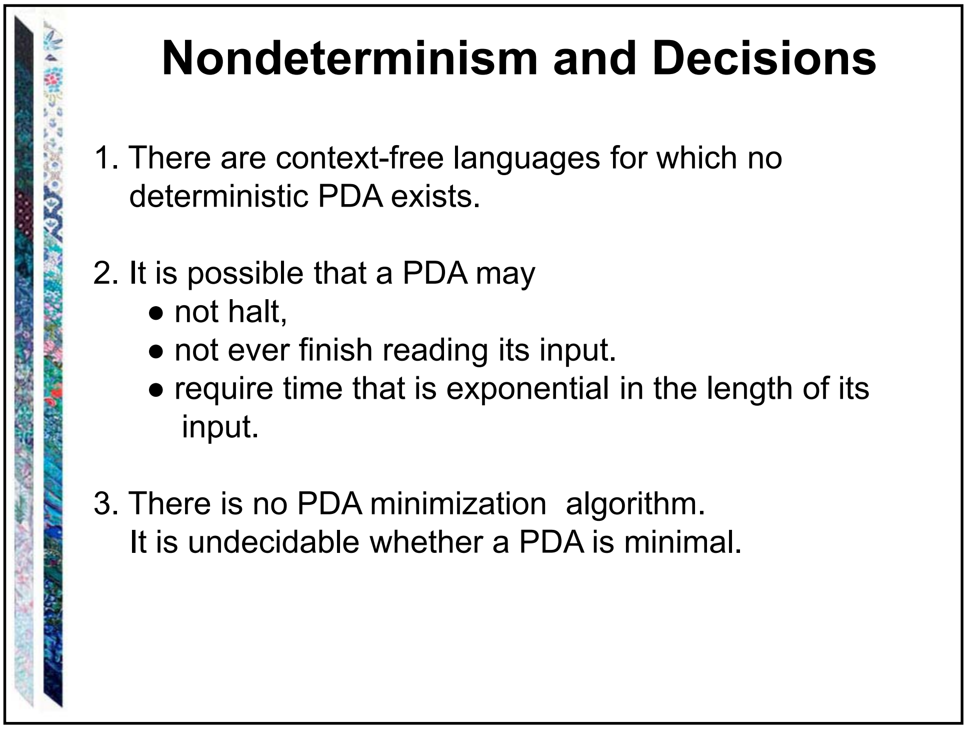
Let $\Sigma = \{a\}$ and consider M =



$L(M) = \{a\}$: $(1, a, \varepsilon) \vdash (2, a, a) \vdash (3, \varepsilon, \varepsilon)$

On any other input except a:

- *M* will never halt, or
- *M* will never finish reading its input unless its input is ε.

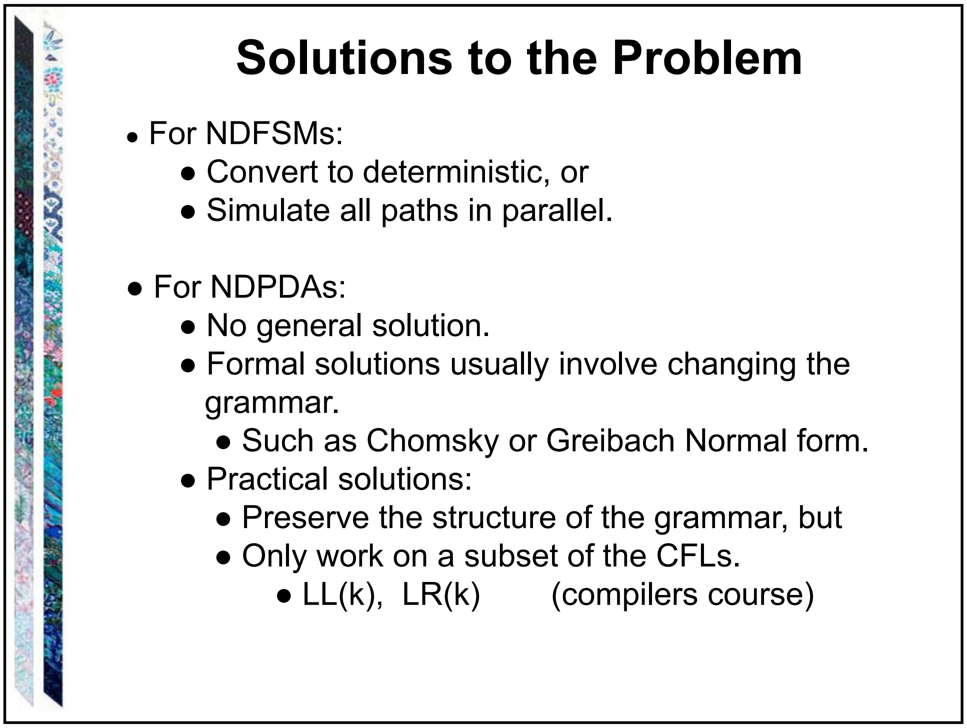


Nondeterminism and Decisions

1. There are context-free languages for which no deterministic PDA exists.

2. It is possible that a PDA may
   - not halt,
   - not ever finish reading its input.
   - require time that is exponential in the length of its input.

3. There is no PDA minimization algorithm.
   It is undecidable whether a PDA is minimal.



Solutions to the Problem

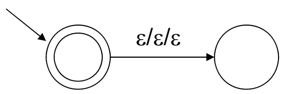
- For NDFSMs:
  - Convert to deterministic, or
  - Simulate all paths in parallel.

- For NDPDAs:
  - No general solution.
  - Formal solutions usually involve changing the grammar.
    - Such as Chomsky or Greibach Normal form.
  - Practical solutions:
    - Preserve the structure of the grammar, but
    - Only work on a subset of the CFLs.
      - LL(k), LR(k) (compilers course)

Deterministic PDAs

A PDA *M* is ***deterministic*** iff:

- Δ_M contains no pairs of transitions that compete with each other, and
- Whenever *M* is in an accepting configuration it has no available moves.



M can choose between accepting and taking the ε-transition, so it is not deterministic.

Deterministic CFLs (very quick overview without many details)

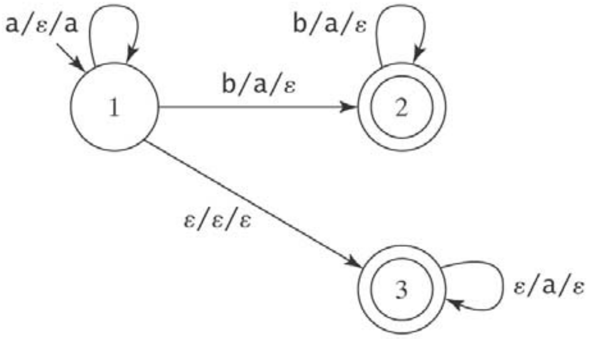
A language *L* is ***deterministic context-free*** iff *L*\$ can be accepted by some deterministic PDA.

Why \$?

Let $L = \text{a}^* \cup \{\text{a}^n\text{b}^n : n > 0\}$.

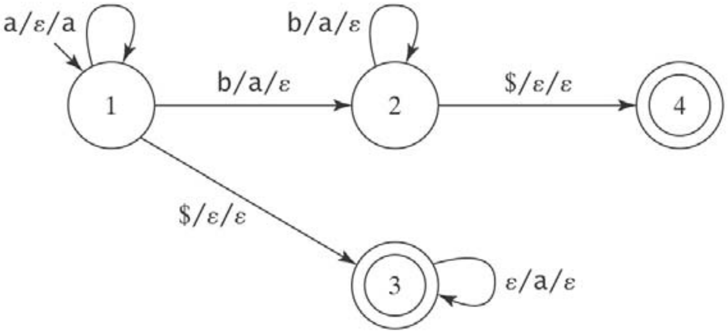
An NDPDA for *L*

$L = \text{a}^* \cup \{\text{a}^n\text{b}^n : n > 0\}$.



A DPDA for *L*$

$L = \text{a}^* \cup \{\text{a}^n\text{b}^n : n > 0\}$.



DCFL Properties (skip the details)

.

The Deterministic CF Languages are closed under complement.

The Deterministic CF Languages are not closed under intersection or union.

Nondeterministic CFLs

Theorem: There exist CLFs that are not deterministic.

Proof: By example. Let $L = \{a^ib^jc^k, i \neq j \text{ or } j \neq k\}$. L is CF. If L is DCF then so is:

$$\begin{aligned} L' &= \neg L. \\ &= \{a^ib^jc^k, i, j, k \geq 0 \text{ and } i = j = k\} \cup \\ &\quad \{w \in \{a, b, c\}^* : \text{the letters are out of order}\}. \end{aligned}$$

But then so is:

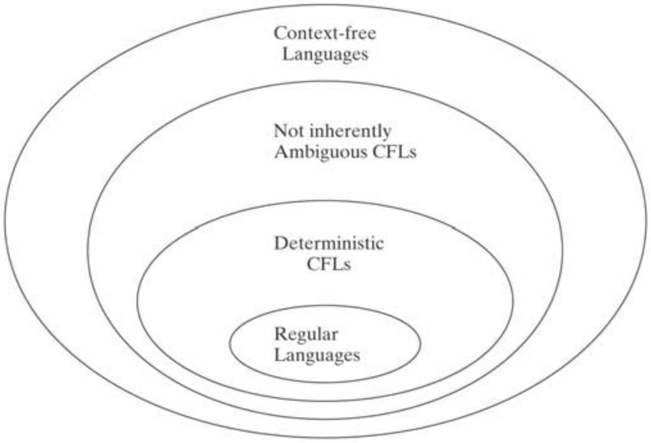
$$\begin{aligned} L'' &= L' \cap a^*b^*c^*. \\ &= \{a^nb^nc^n, n \geq 0\}. \end{aligned}$$

But it isn't. So L is CF but not DCF.

This simple fact poses a real problem for the designers of efficient context-free parsers.

Solution: design a language that is deterministic. LL(k) or LR(k).

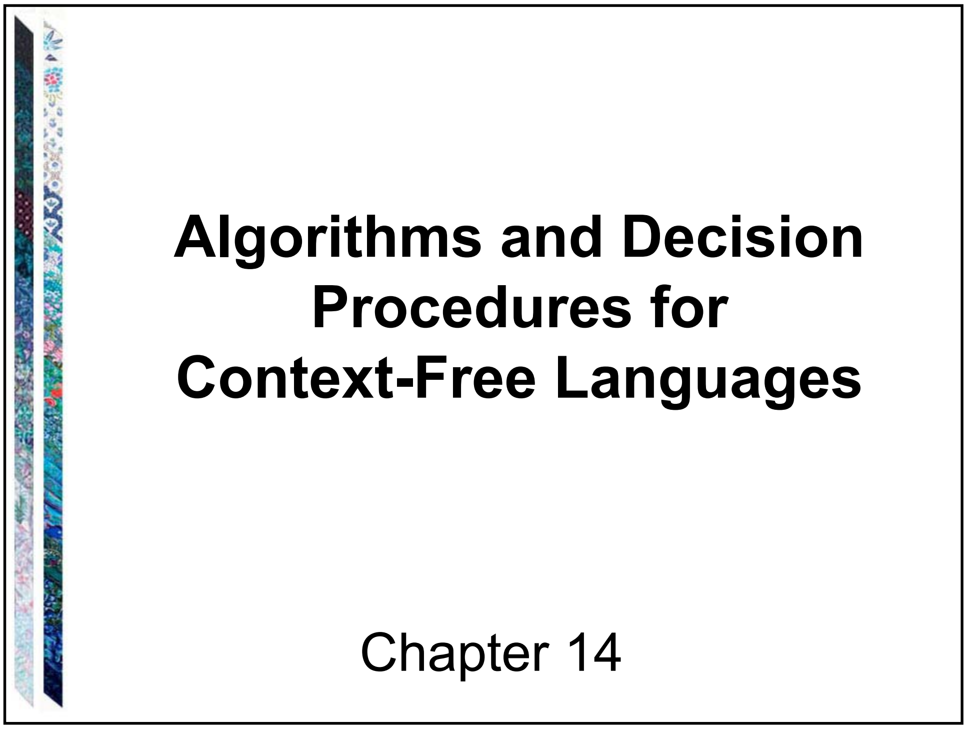
The CFL Hierarchy



Context-Free Languages Over a Single-Letter Alphabet

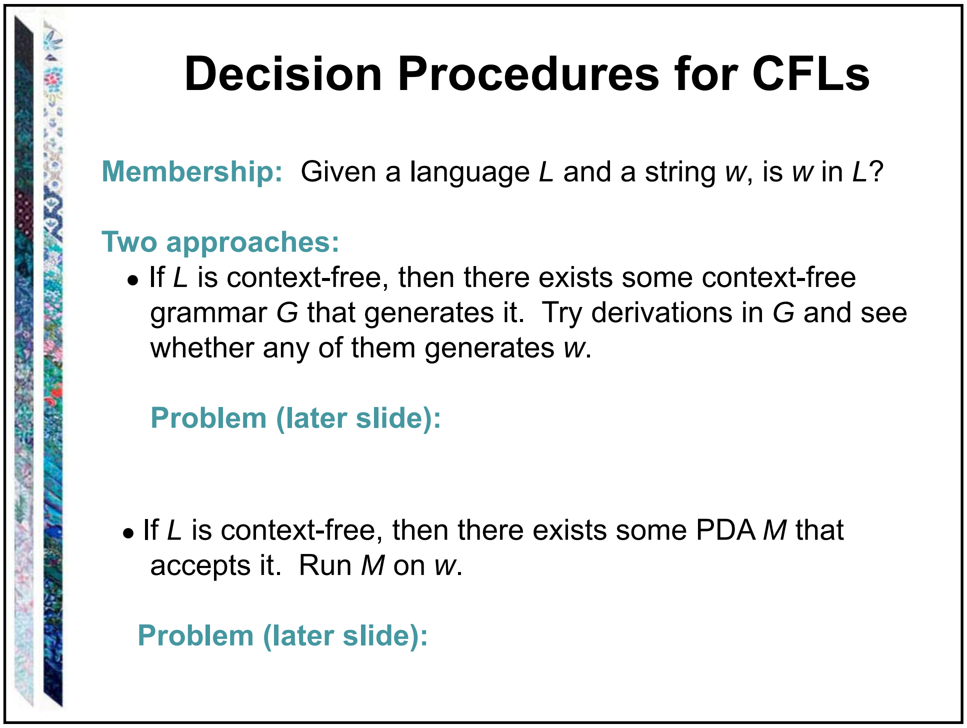
***Theorem**:* Any context-free language over a single-letter alphabet is regular.

***Proof**:* Requires Parikh's Theorem, which we are skipping



Algorithms and Decision Procedures for Context-Free Languages

Chapter 14



Decision Procedures for CFLs

Membership: Given a language *L* and a string *w*, is *w* in *L*?

Two approaches:

- If *L* is context-free, then there exists some context-free grammar *G* that generates it. Try derivations in *G* and see whether any of them generates *w*.

  Problem (later slide):

- If *L* is context-free, then there exists some PDA *M* that accepts it. Run *M* on *w*.

  Problem (later slide):

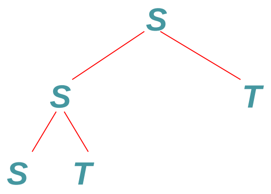
Decision Procedures for CFLs

Membership: Given a language *L* and a string *w*, is *w* in *L*?

Two approaches:

- If *L* is context-free, then there exists some context-free grammar *G* that generates it. Try derivations in *G* and see whether any of them generates *w*.

S $\rightarrow$ ***S T* | a** **Try to derive aaa**



S

S *T*

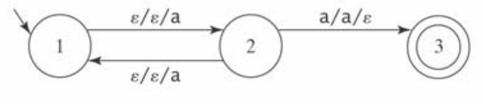
S *T*

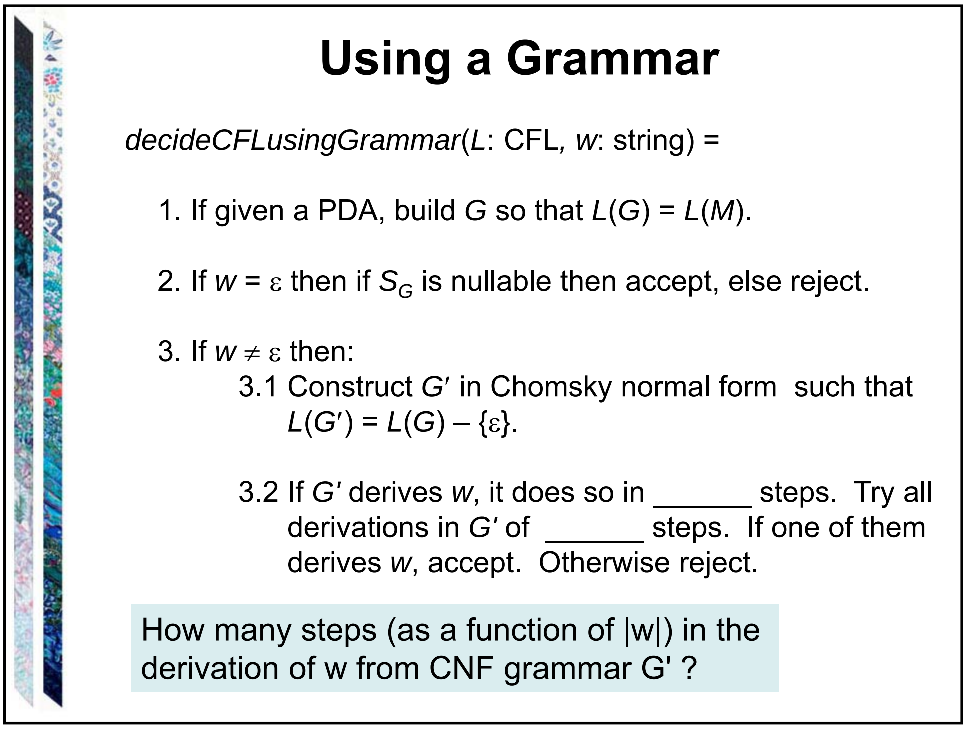
Decision Procedures for CFLs

Membership: Given a language *L* and a string *w*, is *w* in *L*?

- If *L* is context-free, then there exists some PDA *M* that accepts it. Run *M* on *w*.

Problem:

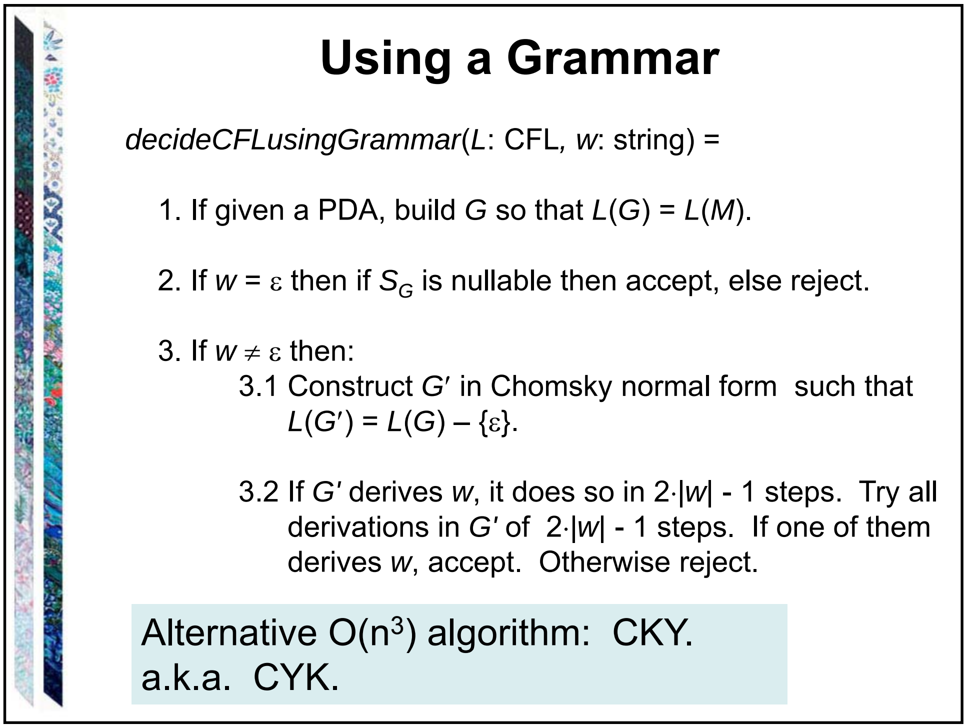




Using a Grammar

```
decideCFLusingGrammar(L: CFL, w: string) =
    1. If given a PDA, build G so that $L(G) = L(M)$.
    2. If $w = \varepsilon$ then if $S_G$ is nullable then accept, else reject.
    3. If $w \neq \varepsilon$ then:
        3.1 Construct $G'$ in Chomsky normal form such that
            $L(G') = L(G) - \{\varepsilon\}$.
        3.2 If $G'$ derives w, it does so in ______ steps. Try all
            derivations in $G'$ of ______ steps. If one of them
            derives w, accept. Otherwise reject.
```

How many steps (as a function of |w|) in the derivation of w from CNF grammar G' ?



Using a Grammar

```
decideCFLusingGrammar(L: CFL, w: string) =
    1. If given a PDA, build G so that $L(G) = L(M)$.
    2. If $w = \varepsilon$ then if $S_G$ is nullable then accept, else reject.
    3. If $w \neq \varepsilon$ then:
        3.1 Construct $G'$ in Chomsky normal form such that
            $L(G') = L(G) - \{\varepsilon\}$.
        3.2 If $G'$ derives w, it does so in $2 \cdot |w| - 1$ steps. Try all
            derivations in $G'$ of $2 \cdot |w| - 1$ steps. If one of them
            derives w, accept. Otherwise reject.
```

Alternative $O(n^3)$ algorithm: CKY.
a.k.a. CYK.

Emptiness

Given a context-free language *L*, is *L* = ∅?

decideCFLempty(*G*: context-free grammar) =

1. Let *G′* = *removeunproductive*(*G*).
2. If *S* is not present in *G′* then return *True*
   else return *False*.

Finiteness

Given a context-free language *L*, is *L* infinite?

decideCFLinfinite(*G*: context-free grammar) =

1. Lexicographically enumerate all strings in Σ^* of length greater than b^n and less than or equal to $b^{n+1} + b^n$.
2. If, for any such string *w*, *decideCFL*(*L, w*) returns *True* then return *True*. *L* is infinite.
3. If, for all such strings *w*, *decideCFL*(*L, w*) returns *False* then return *False*. *L* is not infinite.

Why these bounds?

Some Undecidable Questions about CFLs

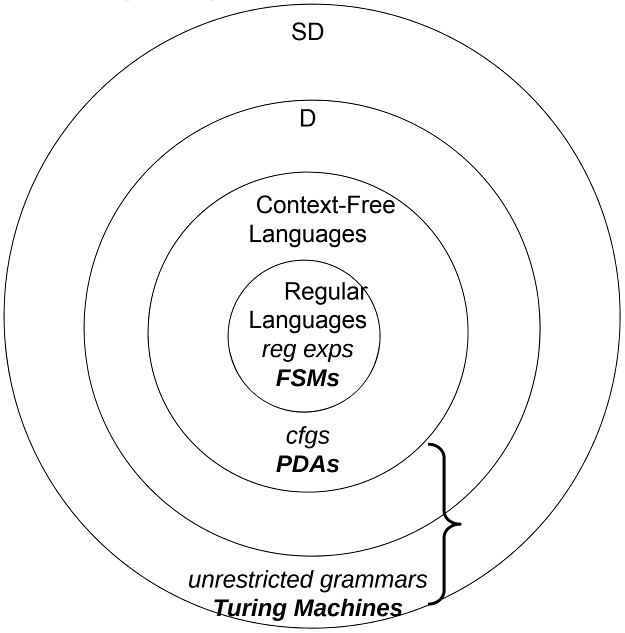
- Is $L = \Sigma^*$?
- Is the complement of L context-free?
- Is L regular?
- Is $L_1 = L_2$?
- Is $L_1 \subseteq L_2$?
- Is $L_1 \cap L_2 = \varnothing$?
- Is L inherently ambiguous?
- Is G ambiguous?

Regular and CF Languages

Regular Languages	**Context-Free Languages**
● regular exprs.	● context-free grammars
● or	
● regular grammars	
● = DFSMs	● = NDPDAs
● recognize	● parse
● minimize FSMs	● try to find unambiguous grammars
	● try to reduce nondeterminism in PDAs
	● find efficient parsers
● closed under:	● closed under:
♦ concatenation	♦ concatenation
♦ union	♦ union
♦ Kleene star	♦ Kleene star
♦ complement	
♦ intersection	♦ intersection w/ reg. langs
● pumping theorem	● pumping theorem
● D = ND	● D ≠ ND

TURING MACHINE INTRO

Languages and Machines



SD

D

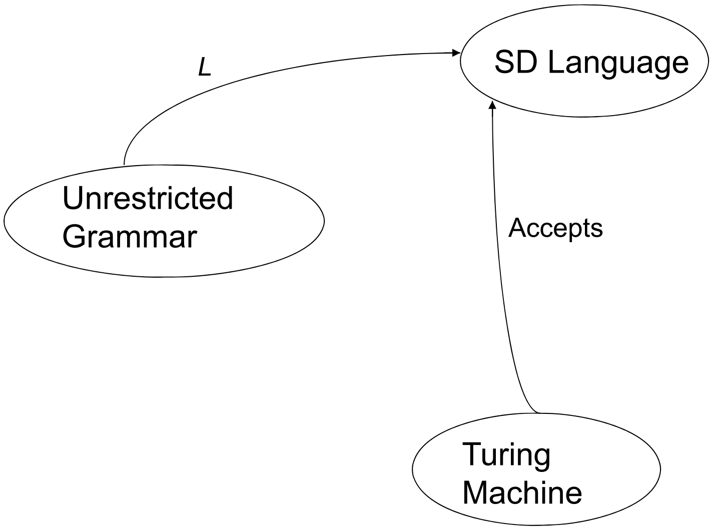
Context-Free Languages

Regular Languages *reg exps* ***FSMs***

cfgs ***PDAs***

unrestricted grammars ***Turing Machines***

Grammars, SD Languages, and Turing Machines



Turing Machines (TMs)

We want a new kind of automaton:

- powerful enough to describe all computable things,

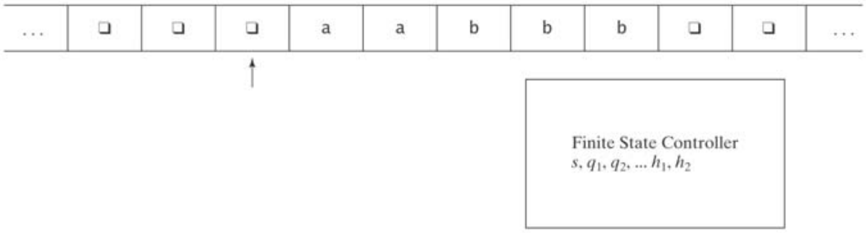
  unlike FSMs and PDAs.

- simple enough that we can reason formally about it

  like FSMs and PDAs,
  unlike real computers.

Goal: Be able to prove things about what can and cannot be computed.

Turing Machines



At each step, the machine must:

- choose its next state,
- write on the current square, and
- move left or right.

A Formal Definition

A (deterministic) Turing machine *M* is (K, Σ, Γ, δ, s, H):

- *K* is a finite set of states;
- Σ is the input alphabet, which does not contain Æ ;
- Γ is the tape alphabet,
  which must contain Æ and have Σ as a subset.
- $s \in K$ is the initial state;
- $H \subseteq K$ is the set of halting states;
- δ is the transition function:

$(K - H) \quad \times \quad \Gamma$ to $K \quad \times \quad \Gamma \quad \times \quad \{\rightarrow, \leftarrow\}$

non-halting state × tape char → state × tape char × direction to move (R or L)

Notes on the Definition

1. The input tape is infinite in both directions.

2. δ is a function, not a relation. So this is a definition for deterministic Turing machines.

3. δ must be defined for all (state, tape symbol) pairs unless the state is a halting state.

4. Turing machines do not necessarily halt (unlike FSM's and most PDAs). Why? To halt, they must enter a halting state. Otherwise they loop.

5. Turing machines generate output, so they can compute functions.

An Example

M takes as input a string in the language:

$\{a^i b^j, 0 \leq j \leq i\}$,

and adds b's as required to make the number of b's equal the number of a's.

The input to *M* will look like this:

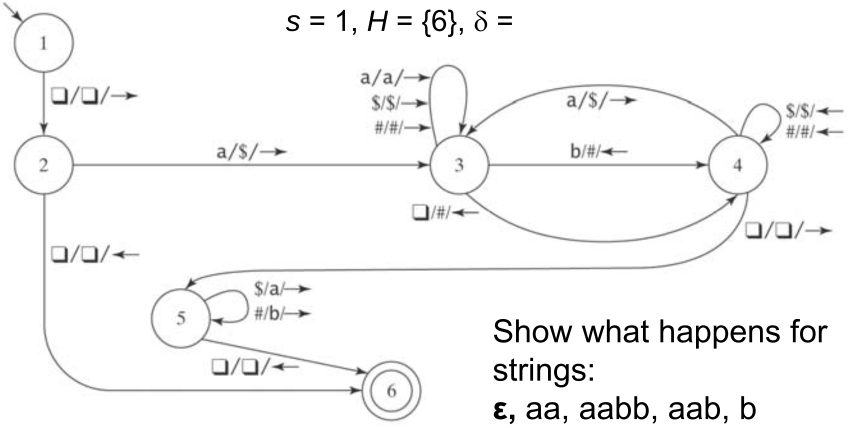
...	❑	a	a	a	b	❑	❑	❑	...
	↑								

The output should be:

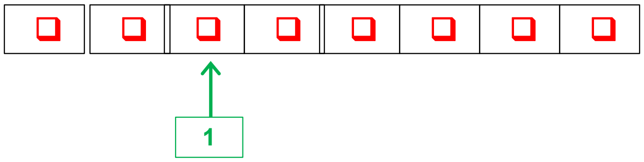
...	❑	a	a	a	b	b	b	❑	...
							↑		

The Details (ε)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =

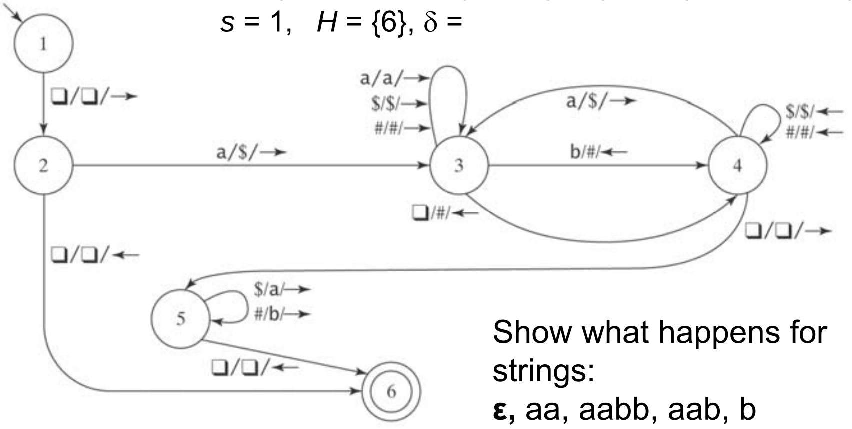


Show what happens for strings:
ε, aa, aabb, aab, b

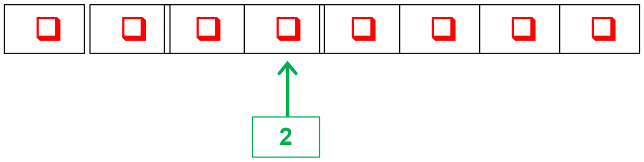


The Details (ε)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =

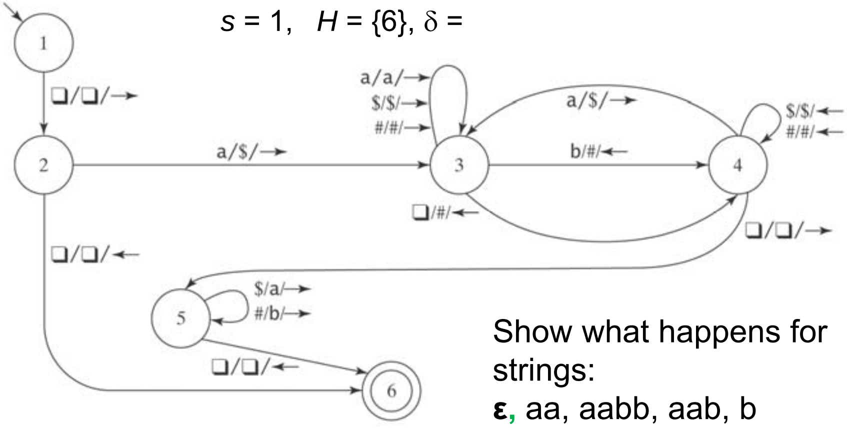


Show what happens for strings:
ε, aa, aabb, aab, b



The Details (ε)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

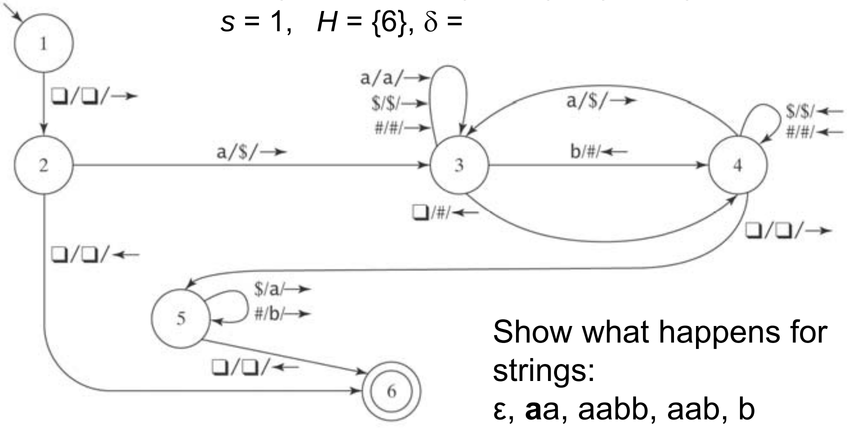
❑	❑	❑	❑	❑	❑	❑	❑

↑

6

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



Show what happens for strings:
ε, **a**a, aabb, aab, b

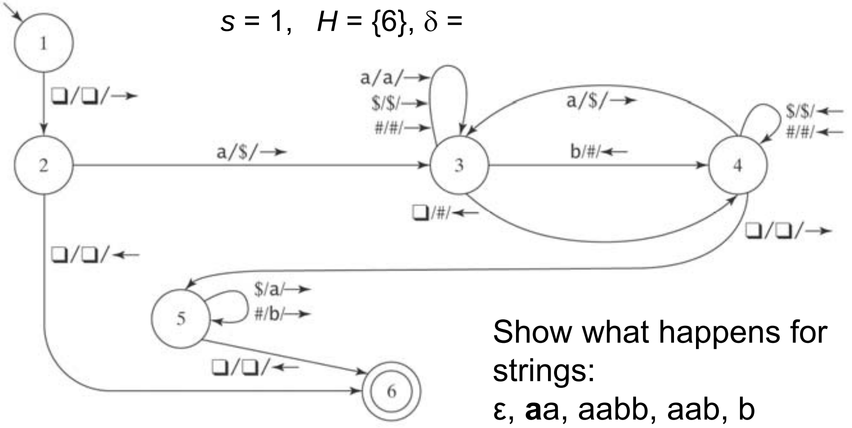
❑	❑	❑	a	a	❑	❑	❑

↑

1

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



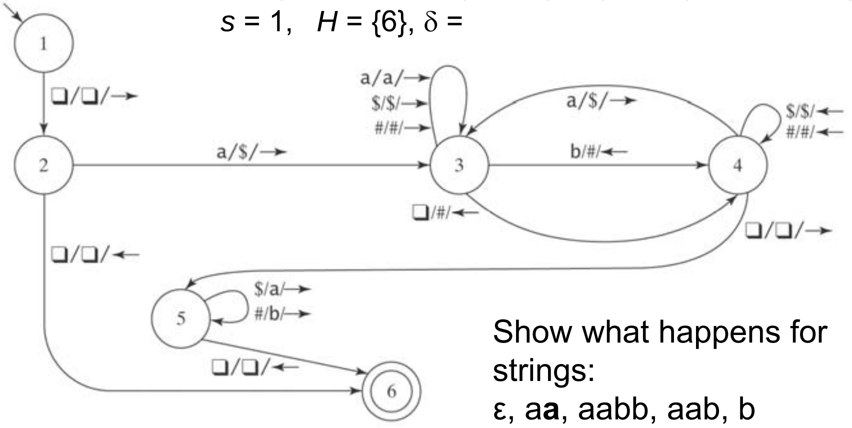
Show what happens for strings:
ε, **a**a, aabb, aab, b

❑	❑	❑	a	a	❑	❑	❑

↑
2

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



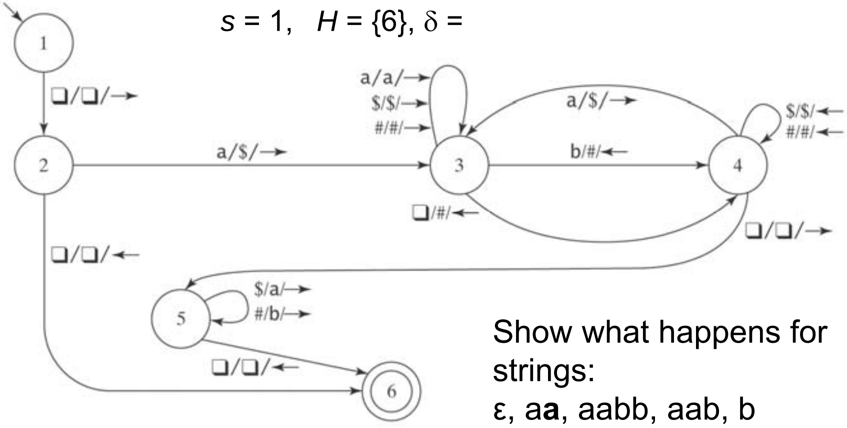
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	a	❑	❑	❑

↑
3

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



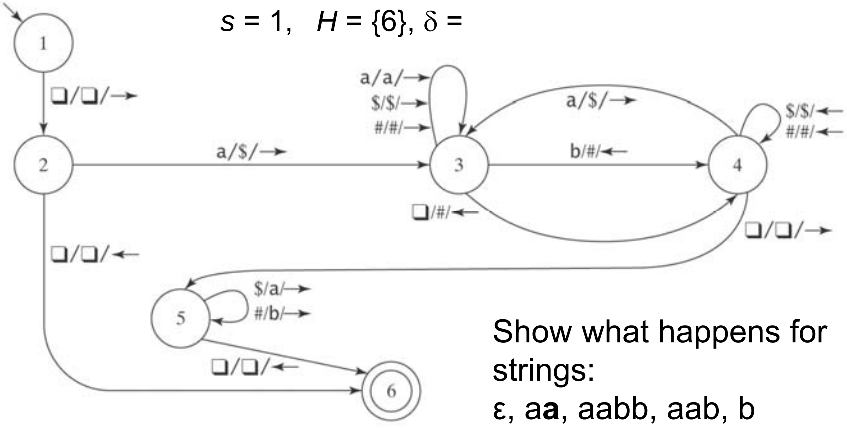
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	a	❑	❑	❑

↑
3

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



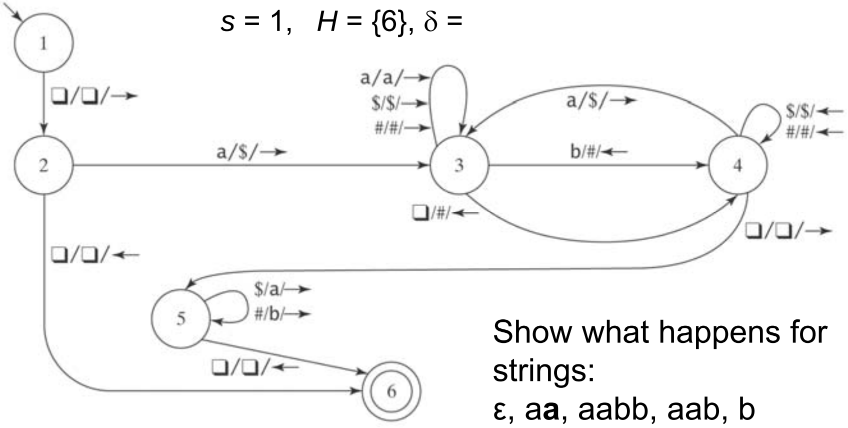
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	a	#	❑	❑

↑
4

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



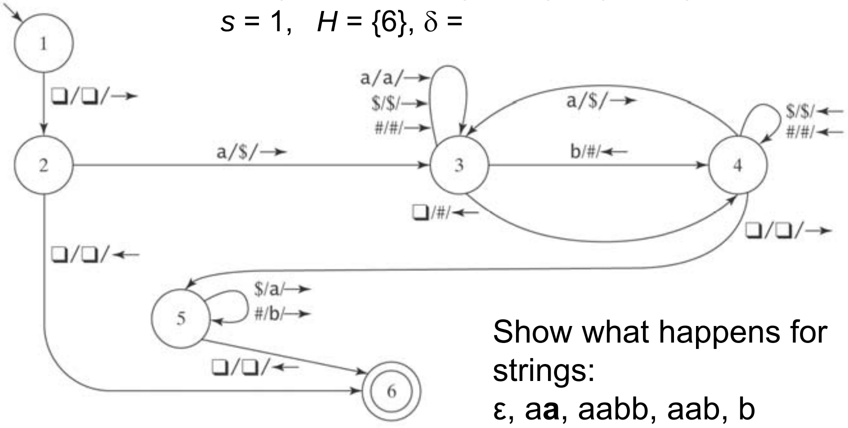
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	❑	❑

↑
3

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



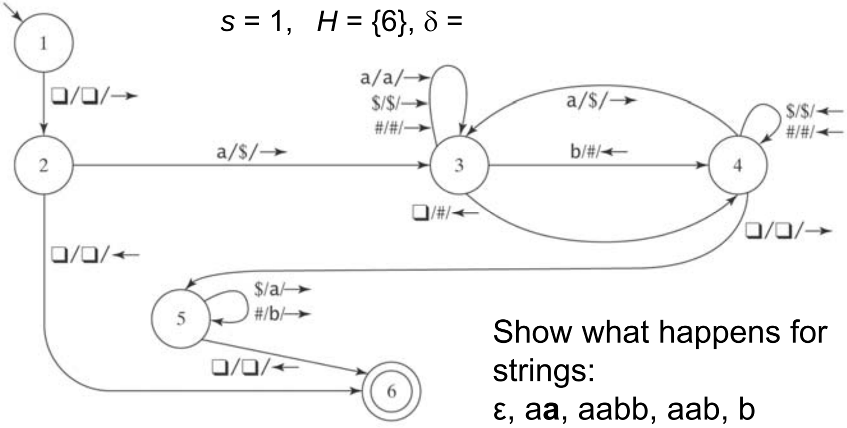
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	❑	❑

↑
3

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



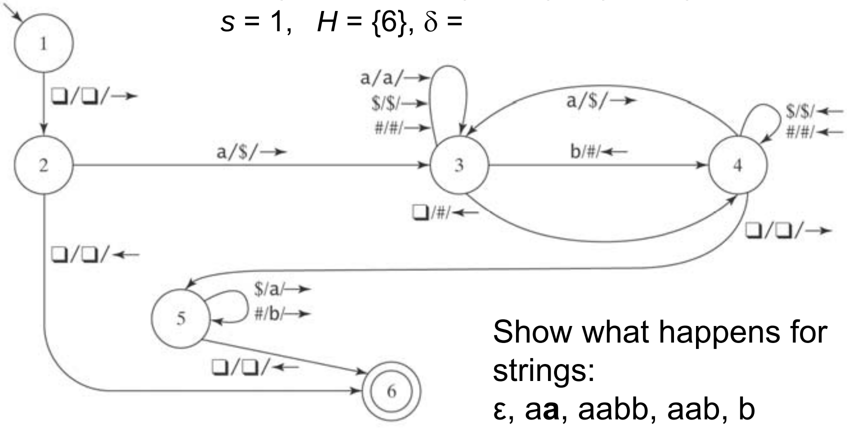
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	❑	❑

3

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



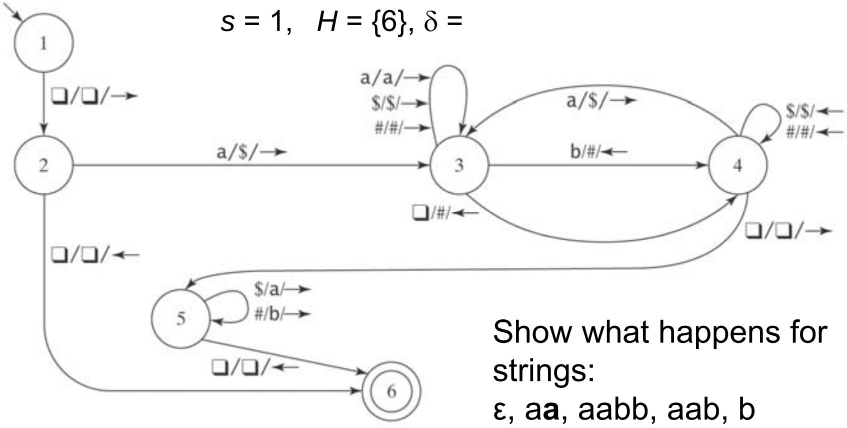
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

4

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



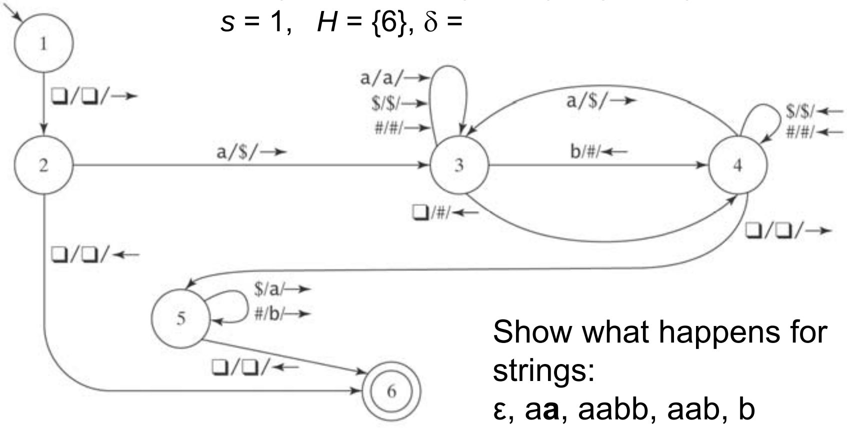
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

↑
4

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



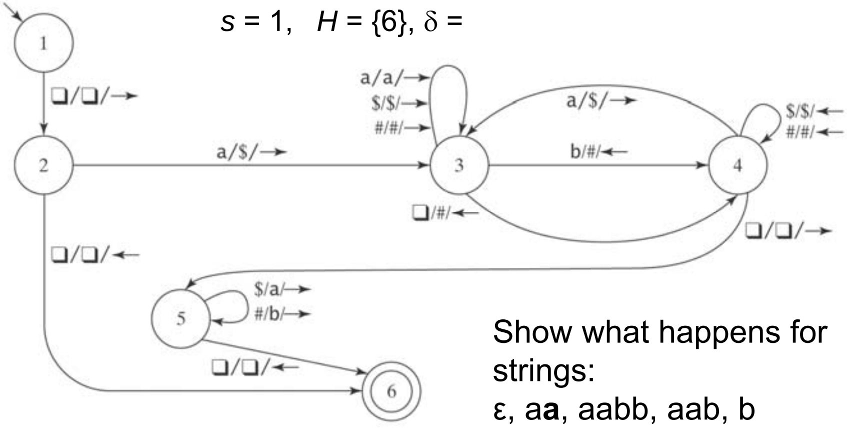
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

↑
4

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



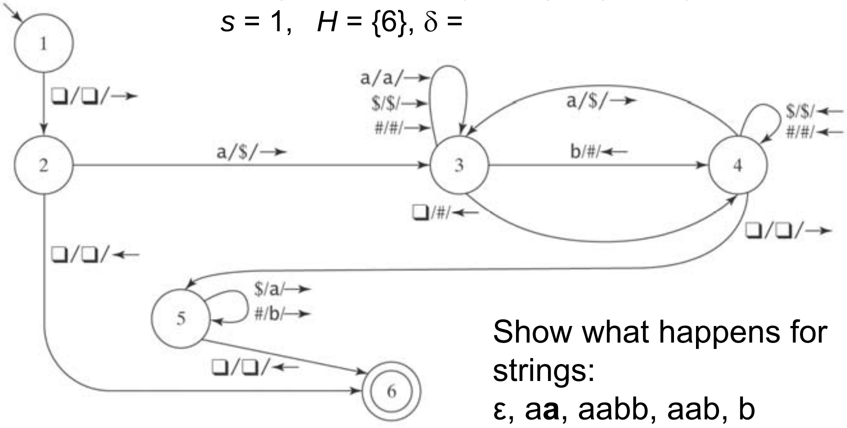
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

↑
4

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1,   *H* = {6}, δ =



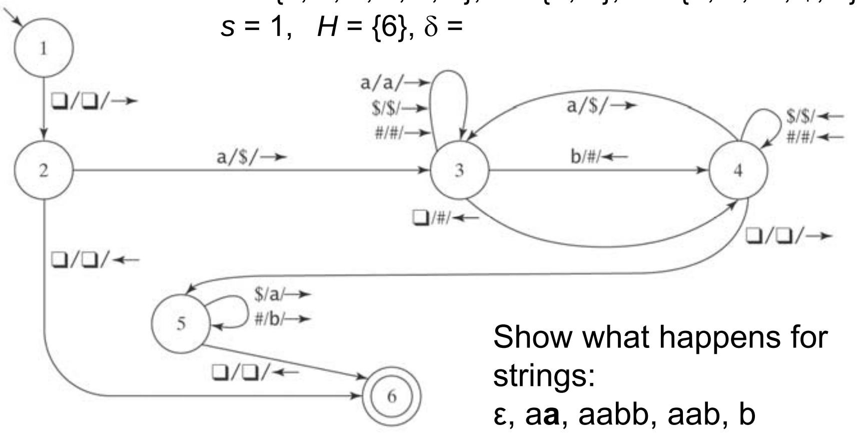
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

↑
5

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



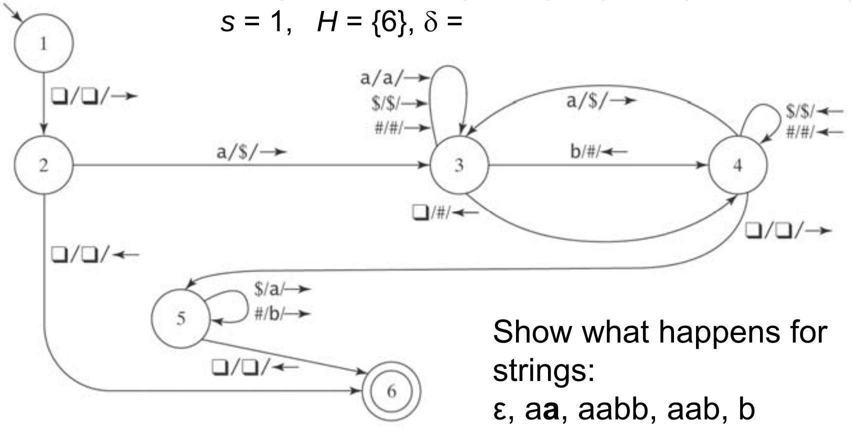
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	a	$	#	#	❑

5

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



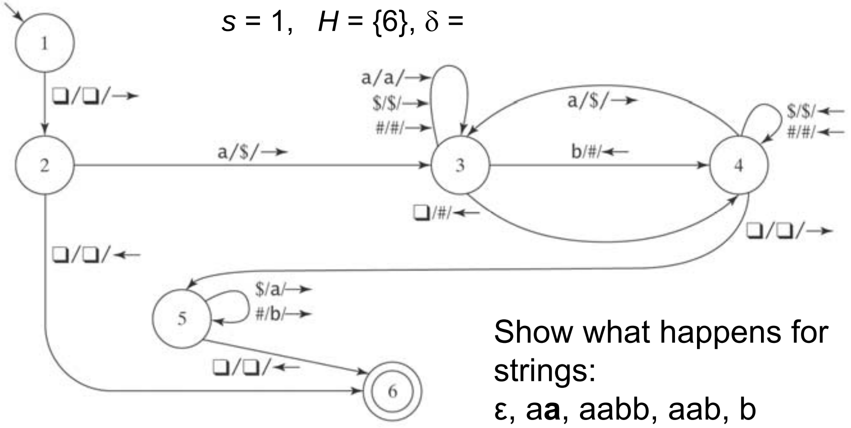
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	a	a	#	#	❑

5

The Details(aa)

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \text{Æ}, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$



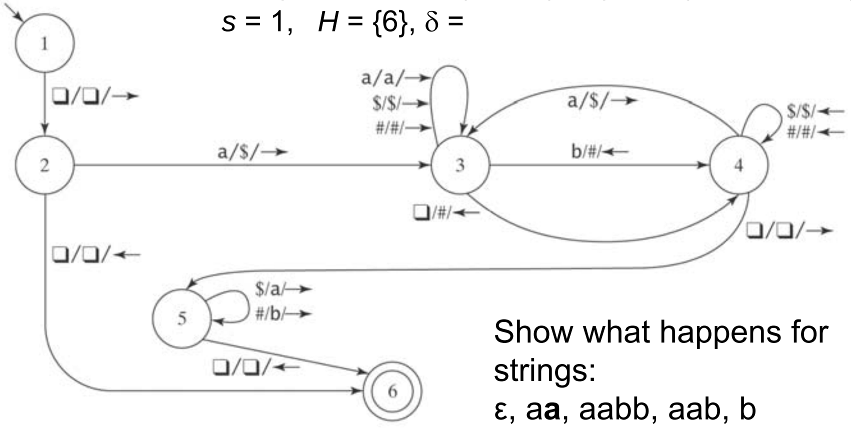
Show what happens for strings:
ε, a**a**, aabb, aab, b

□	□	□	a	a	b	#	□

5

The Details(aa)

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \text{Æ}, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$



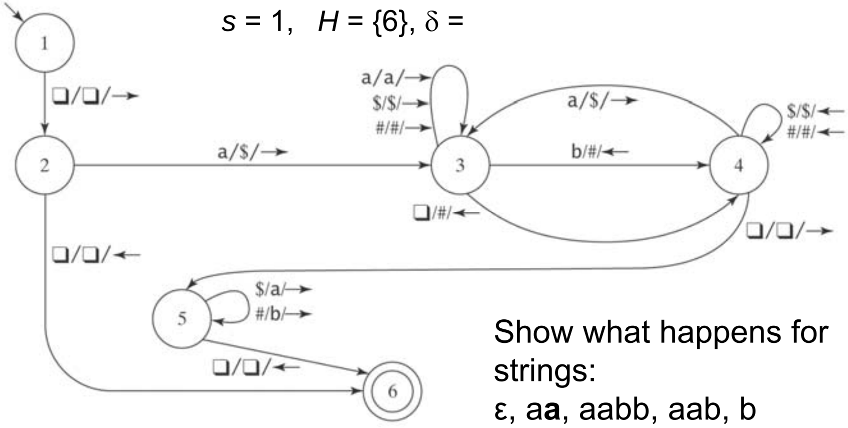
Show what happens for strings:
ε, a**a**, aabb, aab, b

□	□	□	a	a	b	b	□

5

The Details(aa)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



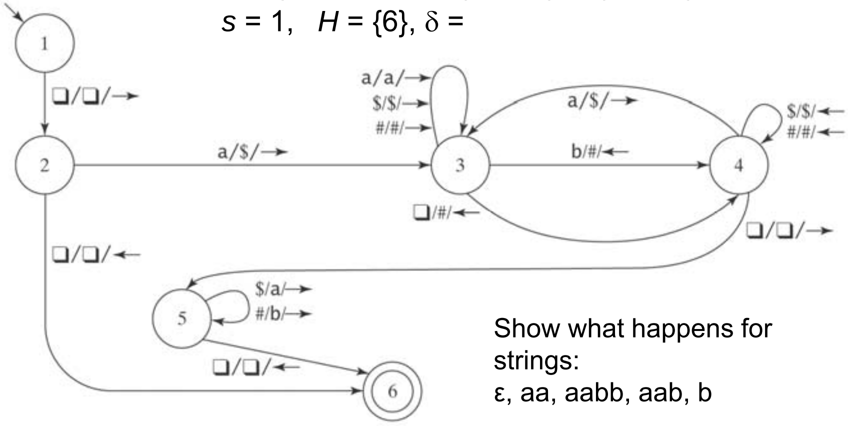
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	a	a	b	b	❑

6

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

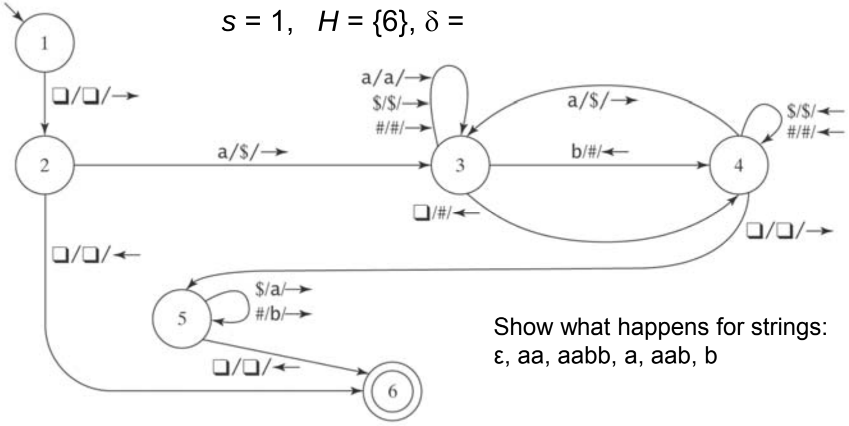
❑	❑	❑	a	a	b	b	❑

1

The steps are the same as previous example until we read the b; skip to there

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #}, *s* = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, a, aab, b

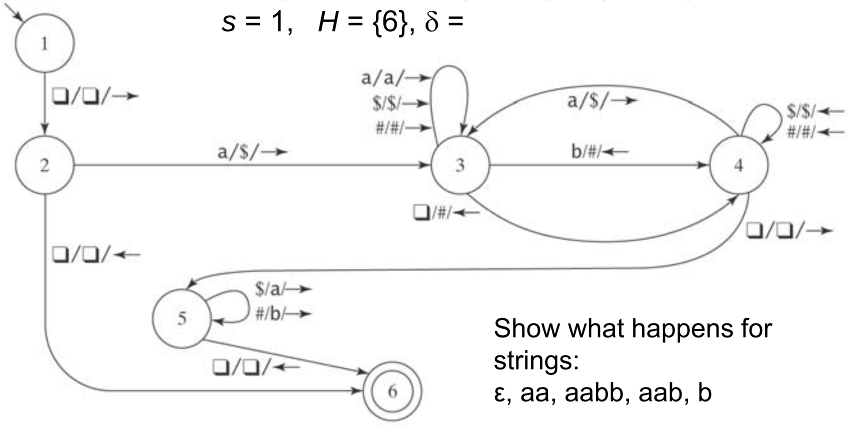
❑	❑	❑	$	a	b	b	❑

↑

3

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #}, *s* = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

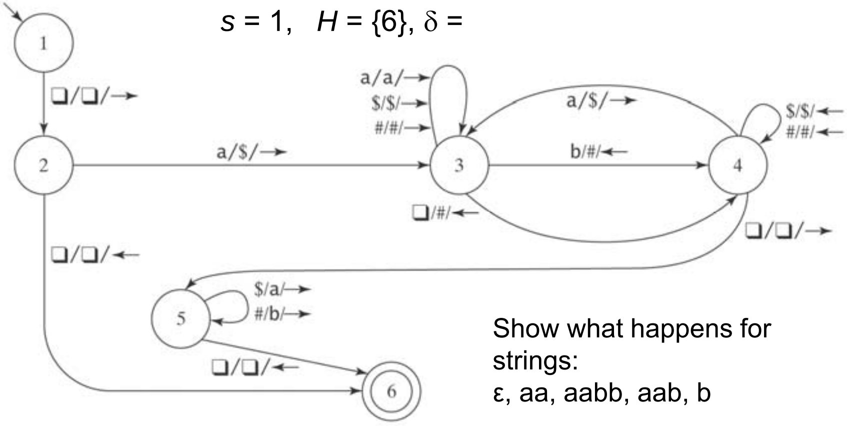
❑	❑	❑	$	a	#	b	❑

↑

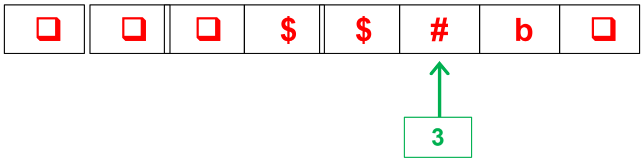
4

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



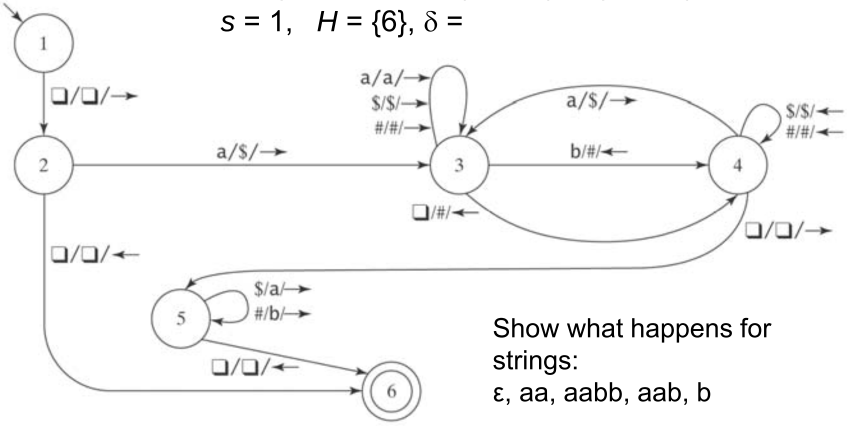
Show what happens for strings:
ε, aa, aabb, aab, b


❑ ❑ ❑ $ $ # b ❑

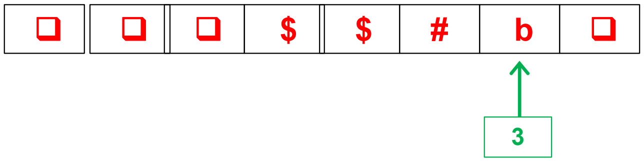
3

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



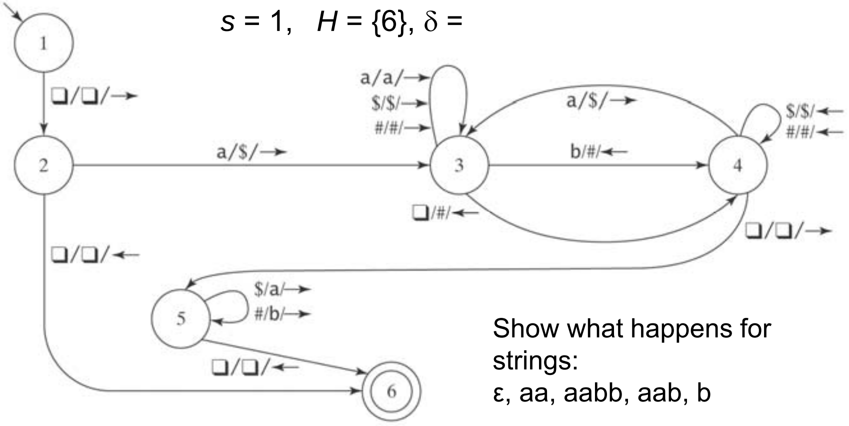
Show what happens for strings:
ε, aa, aabb, aab, b


❑ ❑ ❑ $ $ # b ❑

3

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

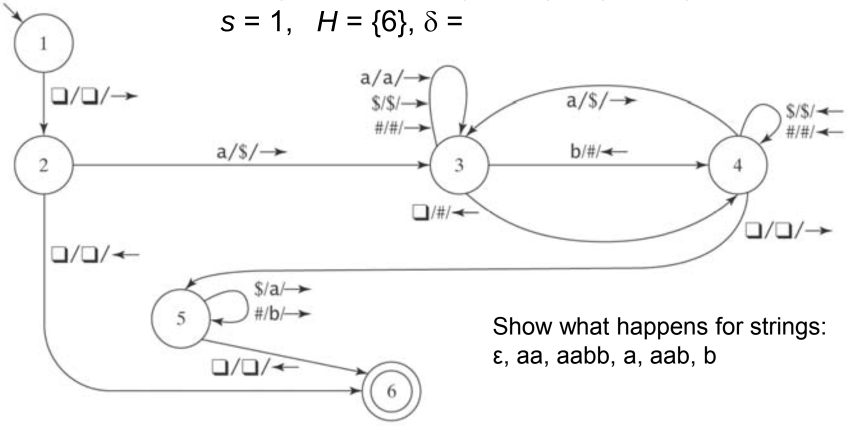
❑	❑	❑	$	$	#	#	❑

In state 4, go left until we hit a blank

4

The Details (aabb)

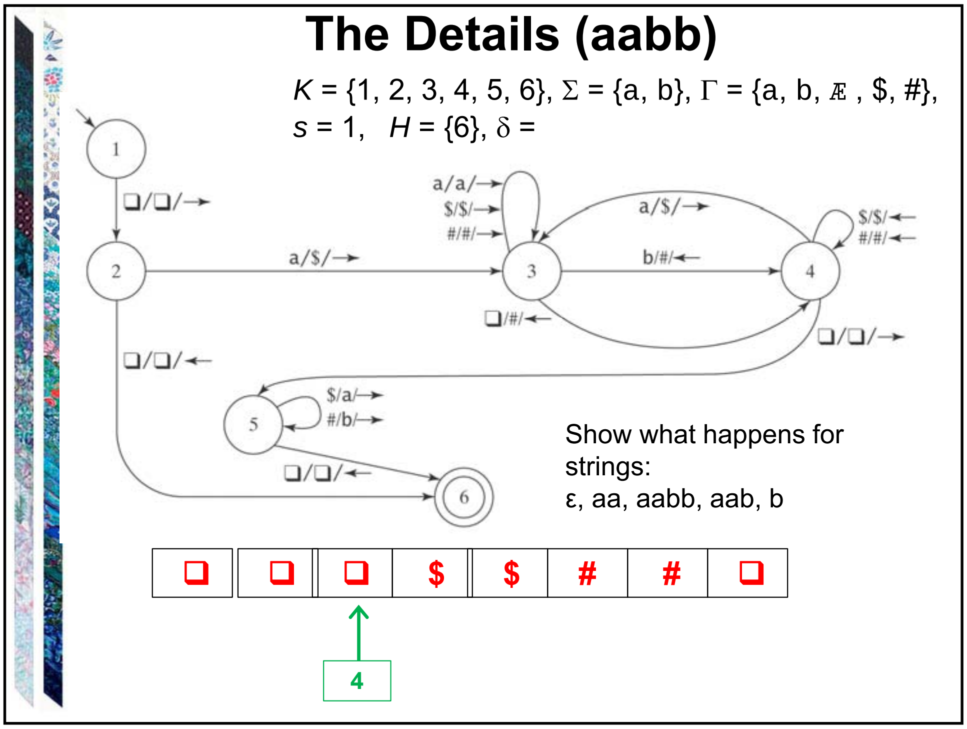
K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, a, aab, b

❑	❑	❑	$	$	#	#	❑

4



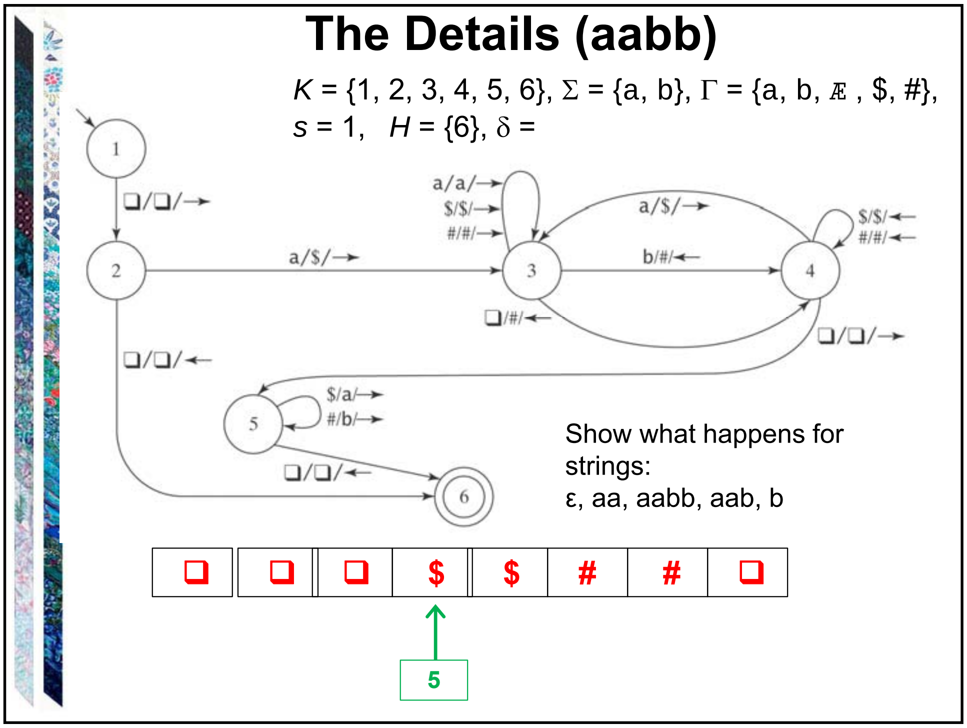
The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #}, *s* = 1, *H* = {6}, δ =

Show what happens for strings:
ε, aa, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

4



The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #}, *s* = 1, *H* = {6}, δ =

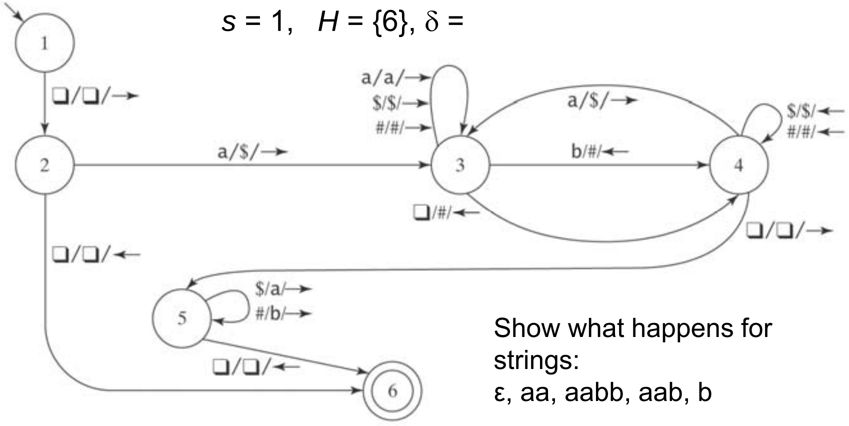
Show what happens for strings:
ε, aa, aabb, aab, b

❑	❑	❑	$	$	#	#	❑

5

The Details (aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

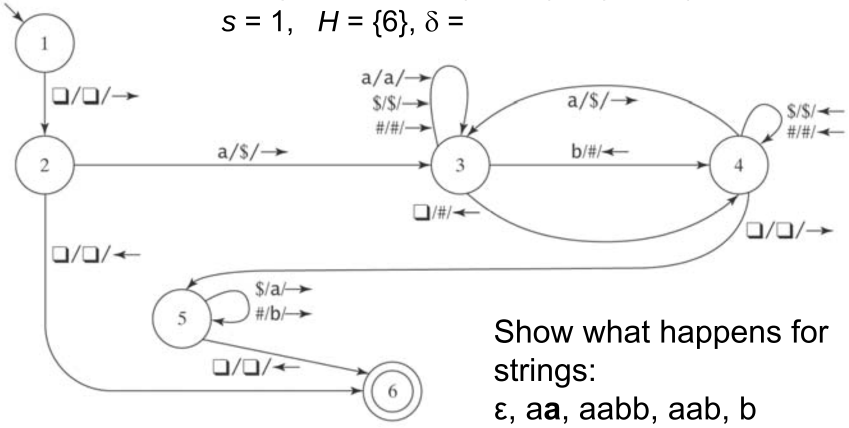
❑	❑	❑	$	$	#	#	❑

5

Go right, replacing $ with a and # with b, then move left, as in the last part of the previous example.

The Details(aabb)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



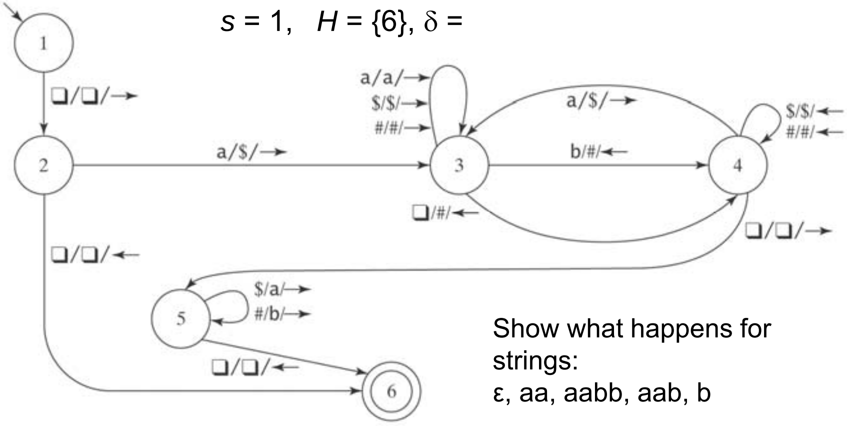
Show what happens for strings:
ε, a**a**, aabb, aab, b

❑	❑	❑	a	a	b	b	❑

6

The Details (aab)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =



Show what happens for strings:
ε, aa, aabb, aab, b

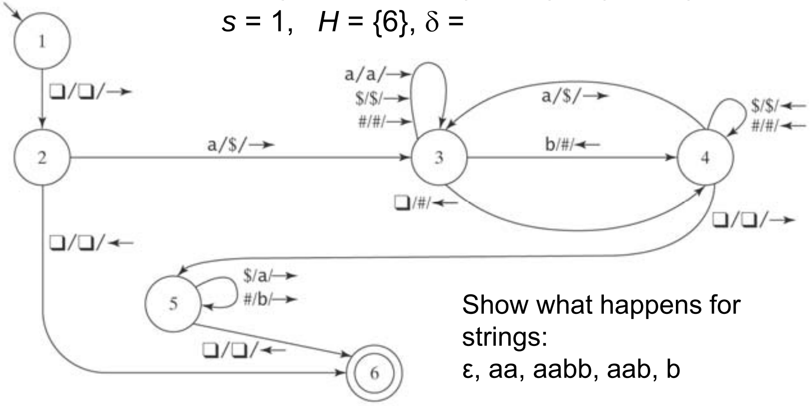
□	□	□	a	a	b	□	□

1

You should try this one.

The Details (b)

K = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, Γ = {a, b, Æ , $, #},
s = 1, *H* = {6}, δ =

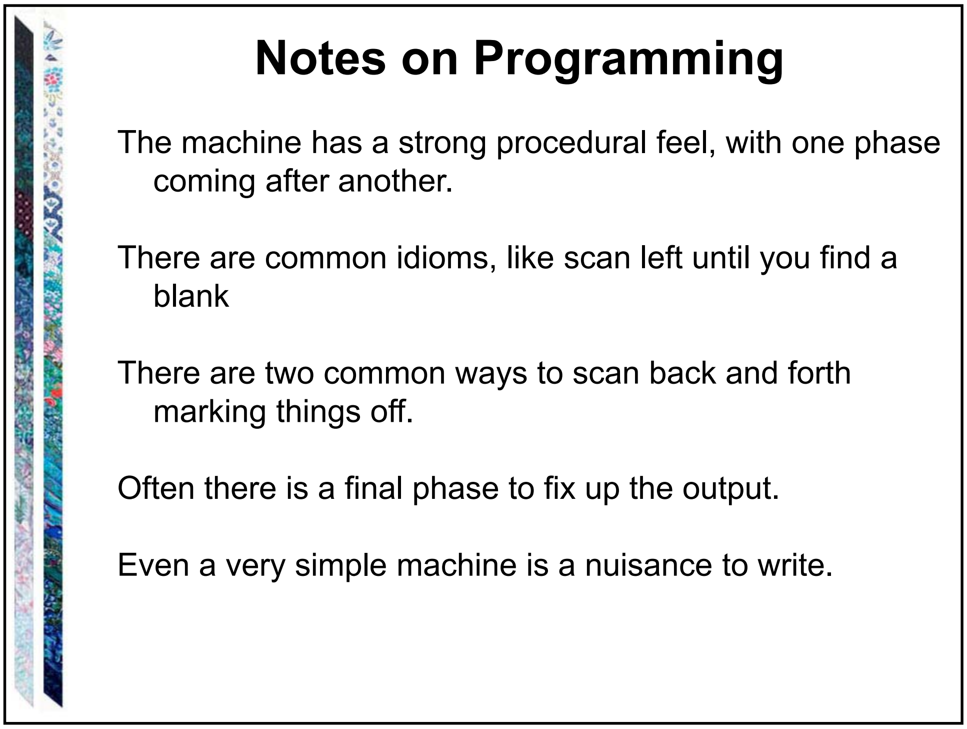


Show what happens for strings:
ε, aa, aabb, aab, b

□	□	□	b	□	□	□	□

1

In the first step, move right. Then there is no transition in the diagram. But there is an implied transition to a dead state, i.e. a new halting state that does not accept.



Notes on Programming

The machine has a strong procedural feel, with one phase coming after another.

There are common idioms, like scan left until you find a blank

There are two common ways to scan back and forth marking things off.

Often there is a final phase to fix up the output.

Even a very simple machine is a nuisance to write.