MA/CSSE 474
Theory of Computation

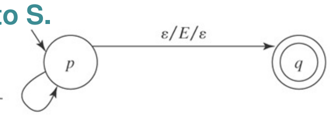
Bottom-up Parsing
CFL Closure properties
Decision Problems
Turing Machine Introduction

Bottom-Up PDA

The idea: Let the stack keep track of what has been found.

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow id$

Discover a rightmost derivation in reverse order. Start with the sentence and try to "pull it back" to S.



Reduce Transitions:
(1) (p, ε, $T + E$), (p, E)
(2) (p, ε, T), (p, E)
(3) (p, ε, $F * T$), (p, T)
(4) (p, ε, F), (p, T)
(5) (p, ε,)E( ), (p, F)
(6) (p, ε, id), (p, F)

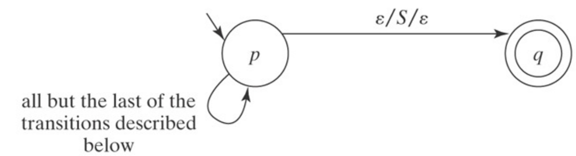
Shift Transitions:
(7) (p, id, ε), (p, id)
(8) (p, (, ε), (p, ()
(9) (p,), ε), (p,))
(10) (p, +, ε), (p, +)
(11) (p, ∗, ε), (p, ∗)

When the right side of a production is on the top of the stack, we can replace it by the left side of that production…

…or not! That's where the nondeterminism comes in: choice between shift and reduce; choice between two reductions.

A Bottom-Up Parser

The outline of *M* is:



$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

- The shift transitions: $((p, c, \varepsilon), (p, c))$, for each $c \in \Sigma$.
- The reduce transitions: $((p, \varepsilon, (s_1 s_2 \ldots s_n.)^R), (p, X))$, for each rule $X \to s_1 s_2 \ldots s_n.$ in G.
- The finish up transition: $((p, \varepsilon, S), (q, \varepsilon))$.

Sketch of PDA→CFG

Lemma: If a language is accepted by a pushdown automaton *M*, it is context-free (i.e., it can be described by a context-free grammar).

Proof (by construction):

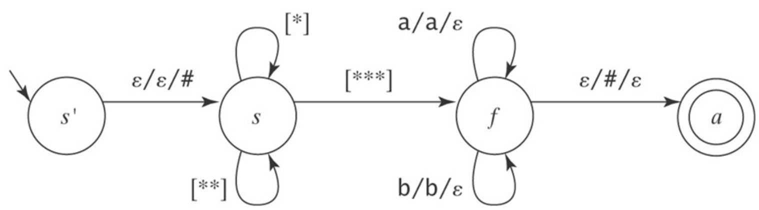
Step 1: Convert *M* to restricted normal form:

- *M* has a start state s' that does nothing except push a special symbol # onto the stack and then transfer to a state *s* from which the rest of the computation begins. There must be no transitions back to s'.
- *M* has a single accepting state *a*. All transitions into *a* pop # and read no input.
- Every transition in *M*, except the one from s', pops exactly one symbol from the stack.

Second Step - Creating the Productions

Example: W c W^R

M =



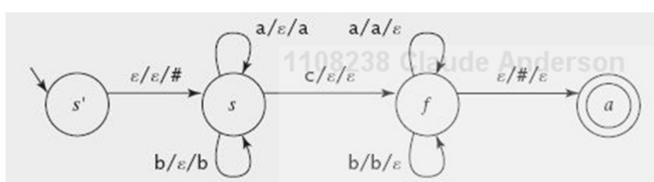
The basic idea –

simulate a leftmost derivation of *M* on any input string.

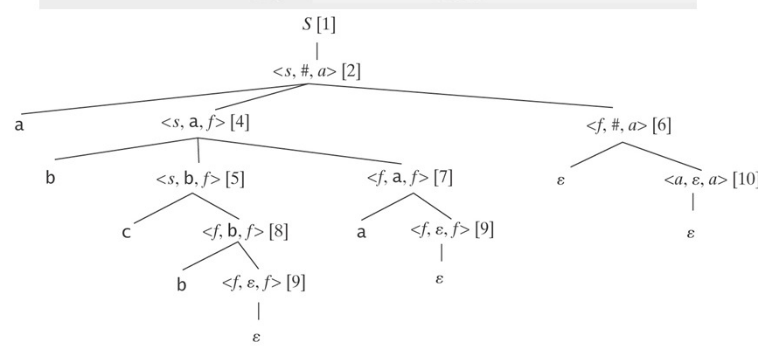
Step 2 - Creating the Productions

The basic idea: A leftmost derivation simulates the actions of M on an input string.

Example:



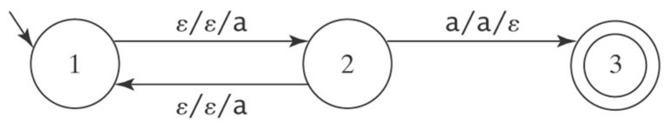
abcba



Halting

It is possible that a PDA may
- not halt,
- not ever finish reading its input.

Let Σ = {a} and consider *M* =



L(*M*) = {a}: (1, a, ε) |- (2, a, a) |- (3, ε, ε)

On any other input except a:
- *M* will never halt.
- *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 that usually involve changing the grammar.
    - Such as Chomsky or Greibach Normal form.
  - Practical solutions that:
    - Preserve the structure of the grammar, but
    - Only work on a subset of the CFLs.

What About These Variations?

- In HW, we see that Acceptance by "accepting tate" only is equivalent to acceptance by empty stack and accepting state.
- FSM plus FIFO queue (instead of stack)?
- FSM plus two stacks?

Comparing Regular and Context-Free Languages

Regular Languages	Context-Free Languages
● regular exprs. or regular grammars	● context-free grammars
● recognize	● parse
● = DFSMs	● = NDPDAs

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

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.

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 $(([q_1, q_2], \varepsilon, \beta), ([p_1, q_2], \gamma))$.

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

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.

Why are the Context-Free Languages Not Closed under Complement, Intersection and Subtraction But the Regular Languages Are?

Given an NDFSM M_1, build an FSM M_2 such that $L(M_2) = \neg L(M_1)$:

1. From M_1, construct an equivalent deterministic FSM M', using *ndfsmtodfsm*.
2. If M' is described with an implied dead state, add the dead state and all required transitions to it.
3. Begin building M_2 by setting it equal to M'. Then swap the accepting and the nonaccepting states. So:

$$M_2 = (K_{M'}, \Sigma, \delta_{M'}, s_{M'}, K_{M'} - A_{M'}).$$

We could do the same thing for CF languages if we could do step 1, but we can't.

The need for nondeterminism is the key.

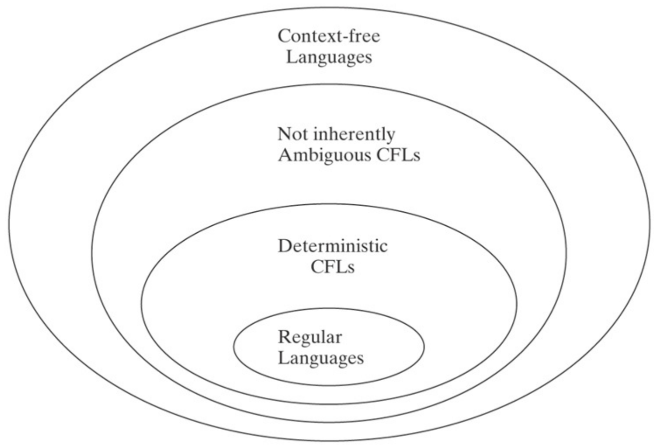
DCFL Properties (skip the details)

.

The Deterministic CF Languages are closed under complement.

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

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):

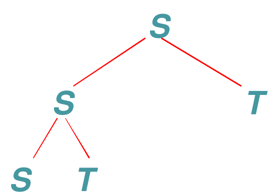
Decision Procedures for CFLs

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*.

$S \rightarrow S\ T \mid$ `a` Try to derive `aaa`



Decision Procedures for CFLs

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

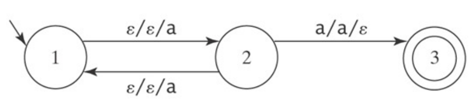
Two approaches:

- 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:

- 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 $L(G) = L(M)$.

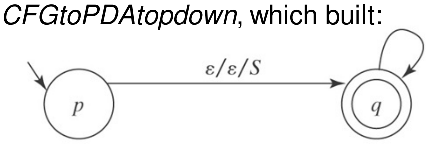
2. If $w = \varepsilon$ then if S_G is nullable then accept, else reject.

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 $2 \cdot |w| - 1$ steps. Try all derivations in G of $2 \cdot |w| - 1$ steps. If one of them derives w, accept. Otherwise reject.

Using a PDA

Recall *CFGtoPDAtopdown*, which built:



$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

- The start-up transition $((p, \varepsilon, \varepsilon), (q, S))$.
- For each rule $X \rightarrow s_1 s_2 \ldots s_n$. in R, the transition $((q, \varepsilon, X), (q, s_1 s_2 \ldots s_n))$.
- For each character $c \in \Sigma$, the transition $((q, c, c), (q, \varepsilon))$.

Can we make this work so there are no ε-transitions? If every transition consumes an input character then M would have to halt after $|w|$ steps.

Put the grammar into Greibach Normal form:
All rules are of the following form:

- **$X \rightarrow a\ A$, where $a \in \Sigma$ and $A \in (V - \Sigma)^*$.**

Greibach Normal Form

All rules are of the following form:

- $X \rightarrow a\ A$, where $a \in \Sigma$ and $A \in (V - \Sigma)^*$.

No need to push the *a* and then immediately pop it.

So $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

1. The start-up transitions:
   For each rule $S \rightarrow cs_2 \ldots s_n$, the transition:
   $((p, c, \varepsilon), (q, s_2 \ldots s_n))$.

2. For each rule $X \rightarrow cs_2 \ldots s_n$ (where $c \in \Sigma$ and s_2 through s_n are elements of $V - \Sigma$), the transition:

   $((q, c, X), (q, s_2 \ldots s_n))$

An Algorithm to Decide Whether *M* Accepts *w*

decideCFLusingPDA(*L*: CFL, *w*: string) =

1. If *L* is specified as a PDA, use *PDAtoCFG* to construct a grammar *G* such that $L(G) = L(M)$.
2. If *L* is specified as a grammar *G*, simply use *G*.
3. If $w = \varepsilon$ then if S_G is nullable then accept, otherwise reject.
4. If $w \neq \varepsilon$ then:
   4.1 From *G*, construct *G′* such that $L(G') = L(G) - \{\varepsilon\}$ and *G′* is in Greibach normal form.
   4.2 From *G′* construct a PDA *M* such that $L(M) = L(G')$ and *M′* has no ε-transitions.
   4.3 All paths of *M* are guaranteed to halt within a finite number of steps. So run *M* on *w*. Accept if it accepts and reject otherwise.
   Each individual path of *M* must halt within $|w|$ steps.
   - The total number of paths pursued by *M* must be less than or equal to $P = B^{|w|}$, where *B* is the maximum number of competing transitions from any state in *M*.
   - The total number of steps that will be executed by all paths of *M* is bounded by $P * |w|$

Emptiness

Given a context-free language L, is $L = \varnothing$?

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?

Equivalence of DCFLs

Theorem: Given two *deterministic* context-free languages L_1 and L_2, there exists a decision procedure to determine whether $L_1 = L_2$.

Proof: Given in [Sénizergues 2001].

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

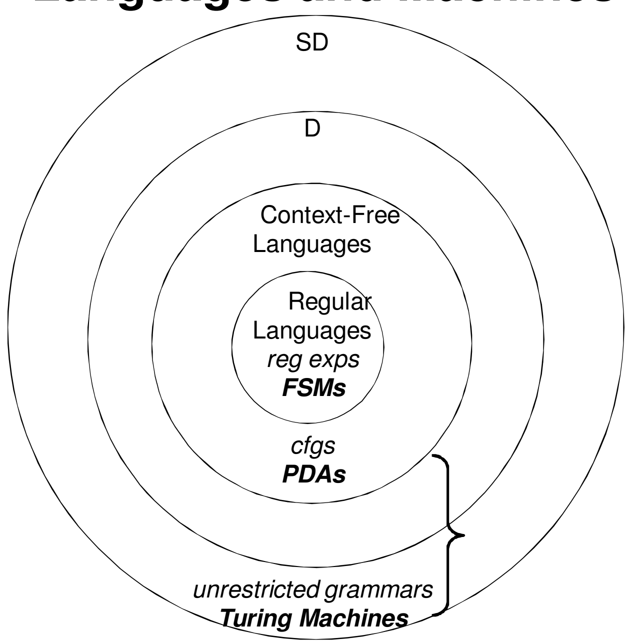
Regular Languages

- regular exprs.
  - or
- regular grammars
- = DFSMs
- recognize
- minimize FSMs
- closed under:
  - ♦ concatenation
  - ♦ union
  - ♦ Kleene star
  - ♦ complement
  - ♦ intersection
- pumping theorem
- D = ND

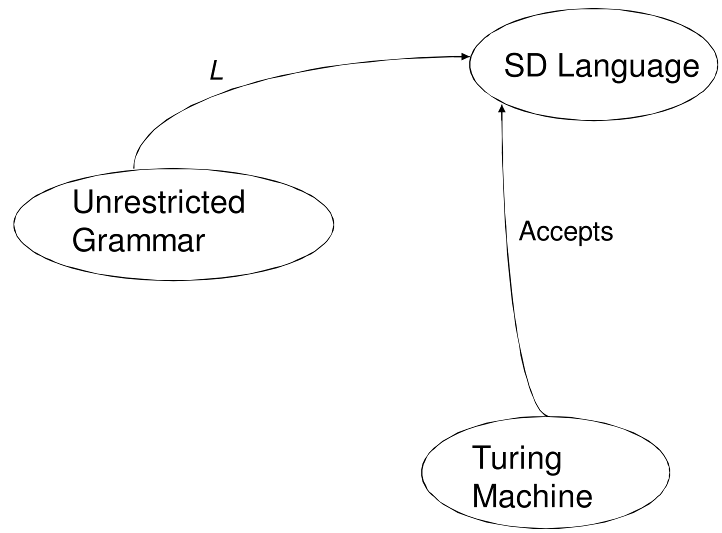
Context-Free Languages

- context-free grammars
- = NDPDAs
- parse
- find unambiguous grammars
- reduce nondeterminism in PDAs
- find efficient parsers
- closed under:
  - ♦ concatenation
  - ♦ union
  - ♦ Kleene star
  - ♦ intersection w/ reg. langs
- pumping theorem
- D $\neq$ ND

Languages and Machines



Grammars, SD Languages, and Turing Machines



Turing Machines

We want a new kind of automaton:

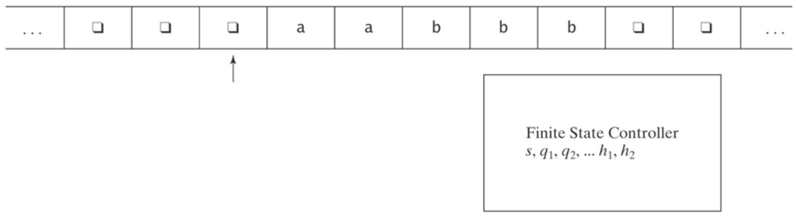
- powerful enough to describe all computable things

  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 $(K, \Sigma, \Gamma, \delta, s, H)$:

- K is a finite set of states;
- Σ is the input alphabet, which does not contain ❑;
- Γ is the tape alphabet, which must contain ❑ 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:

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

non-halting state $\times$ tape char $\rightarrow$ state $\times$ tape char $\times$ 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, input) 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.

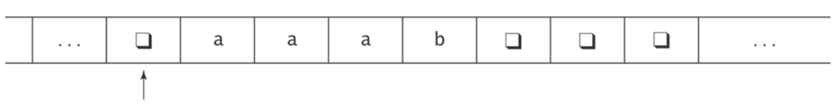
An Example

M takes as input a string in the language:

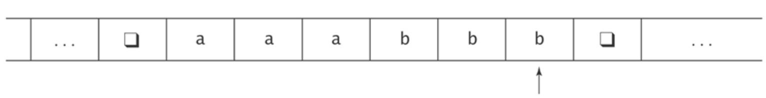
$\{a^i b^j, 0 \leq j \leq i\}$,

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:

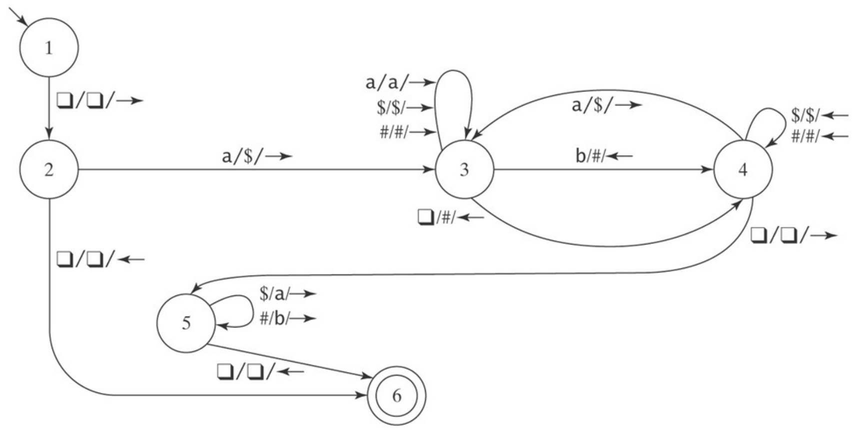


The output should be:



The Details

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



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

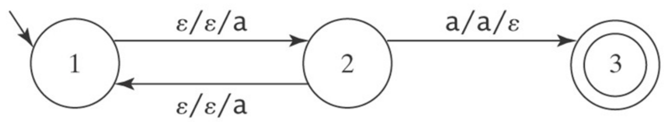
There are two common ways to scan back and forth marking things off.

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

Even a very simple machine is a nuisance to write.

Halting

- A DFSM *M*, on input *w*, is guaranteed to halt in |*w*| steps.
- A PDA *M*, on input *w*, is not guaranteed to halt. To see why, consider again *M* =



But there exists an algorithm to construct an equivalent PDA *M´* that is guaranteed to halt.

A TM *M*, on input *w*, is not guaranteed to halt. And there is no algorithm to construct an equivalent TM that is guaranteed to halt.