Theory of Computation
Week 7

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure
- ✓ Week 6 – Context-free languages: grammars and parse trees

Week 7 – Pushdown automata

- ❑ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ❑ Week 9 – Decidable languages: Turing Machines
- ❑ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ❑ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

CONTEXT-FREE GRAMMARS

A context-free grammar G is a quadruple, (V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$,
- S (the start symbol) is an element of $V - \Sigma$.

A language L is ***context-free*** if and only if it is generated by some context-free grammar G.

PARSE TREES

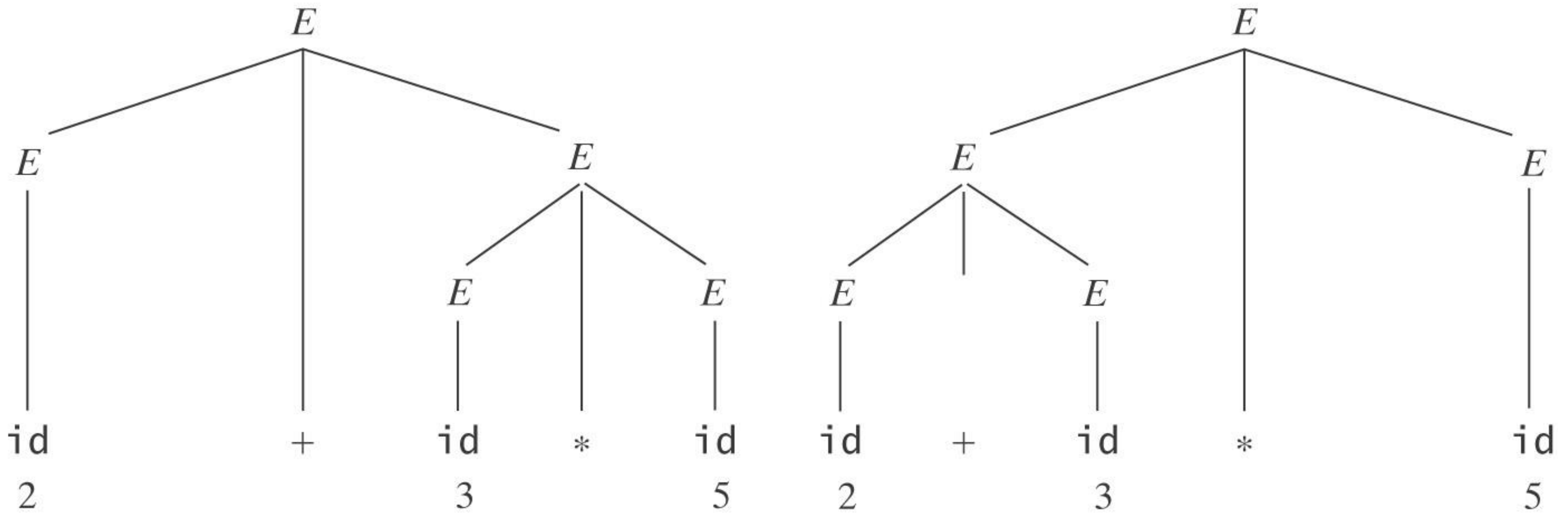
A parse tree, derived by a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

- Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$,
- The root node is labeled S,
- Every other node is labeled with some element of:
  $V - \Sigma$, and
- If m is a nonleaf node labeled X and the children of m are labeled $x_1, x_2, \ldots, x_n$, then R contains the rule
  $X \rightarrow x_1, x_2, \ldots, x_n$.

Ambiguity

$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$E \to \texttt{id}$$

2 + 3 * 5



Week 07 Videos

You already know

- ❑ What is Push Down Automata (PDA)
  - ❑ Formal and informal definition
  - ❑ Difference between PDA and FSM
  - ❑ How PDA operates with its stack
  - ❑ When PDA accepts/rejects
  - ❑ Deterministic/Nondeterministic PDA

Videos to watch before lecture

Additional videos to watch for this week

Week 07 Lecture

Ambiguity, Normal Forms

- ❑ Normal Forms
- ❑ Conversion to Chomsky Normal Form
- ❑ Pushdown Automata (PDA)
    - ❑ Definition
    - ❑ Computation
    - ❑ Accepting/Rejecting
- ❑ Examples of PDA
- ❑ Nondeterminism in PDA

NORMAL FORMS

- A normal form *F* for a set *C* of data objects is a form, i.e., a set of syntactically valid objects, with the following two properties:

- For every element *c* of *C*, except possibly a finite set of special cases, there exists some element *f* of *F* such that *f* is equivalent to *c* with respect to some set of tasks.

- *F* is simpler than the original form in which the elements of *C* are written. By "simpler" we mean that at least some tasks are easier to perform on elements of *F* than they would be on elements of *C*.

NORMAL FORMS

If you want to design algorithms, it is often useful to have a limited number of input forms that you have to deal with.

Normal forms are designed to do just that. Various ones have been developed for various purposes.

Examples:

- Clause form for logical expressions to be used in resolution theorem proving
- Disjunctive normal form for database queries so that they can be entered in a query by example grid.
- Various normal forms for grammars to support specific parsing techniques.

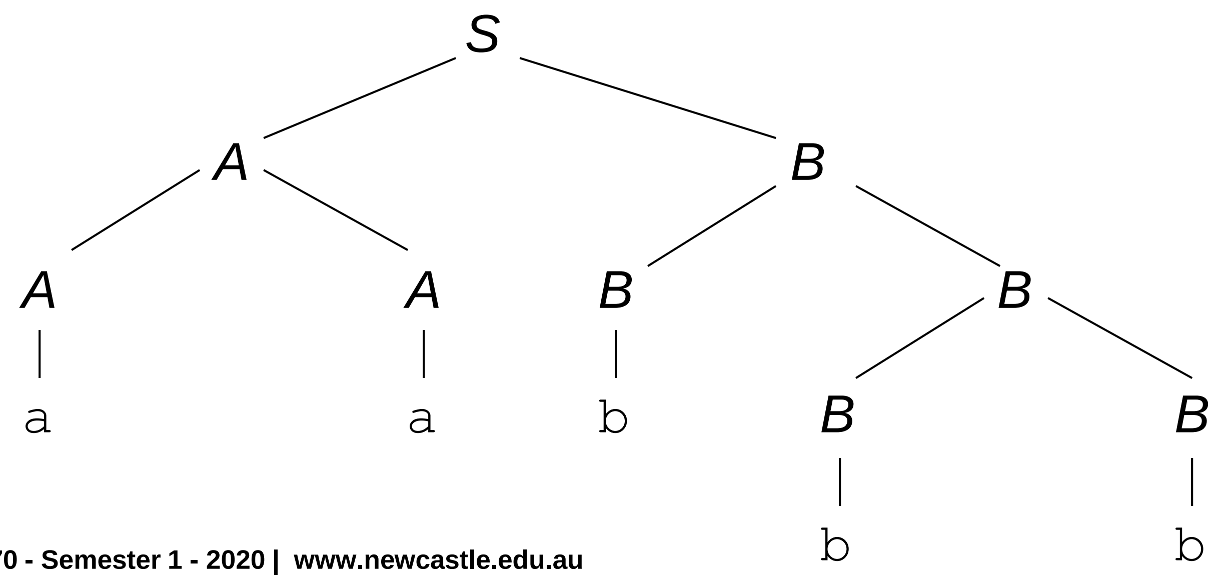
NORMAL FORMS

Chomsky Normal Form, in which all rules are of one of the following two forms:

- $X \rightarrow a$, where $a \in \Sigma$, or
- $X \rightarrow BC$, where B and C are elements of $V - \Sigma$.

Advantages:

- Parsers can use binary trees.
- Exact length of derivations is known:



NORMAL FORMS

Theorem: Given a CFG G, there exists an equivalent Chomsky normal form grammar G_C such that:

$$L(G_C) = L(G) - \{\varepsilon\}.$$

Proof: The proof is by construction.

CONVERSION TO CHOMSKY NORMAL FORM

1. Remove all ε-rules (e.g $X \to \varepsilon$), using the algorithm *removeEps*.

2. Remove all unit productions (rules of the form $A \to B$).

3. Remove all rules whose right hand sides have length greater than 1 and include a terminal:

   (e.g., $A \to \mathtt{a}B$ or $A \to B\mathtt{a}C$ or $A \to \mathtt{ab}$)

4. Remove all rules whose right hand sides have length greater than 2:

   (e.g., $A \to BCDE$)

CONVERTING TO A NORMAL FORM

1. Apply some transformation to *G* to get rid of undesirable property 1. Show that the language generated by *G* is unchanged.

2. Apply another transformation to *G* to get rid of undesirable property 2. Show that the language generated by *G* is unchanged *and* that undesirable property 1 has not been reintroduced.

3. Continue until the grammar is in the desired form.

NORMAL FORMS
Rule Substitution

$$X \rightarrow \mathtt{a}Y\mathtt{c}$$
$$Y \rightarrow \mathtt{b}$$
$$Y \rightarrow ZZ$$

We can replace the X rule with the rules:

$$X \rightarrow \mathtt{abc}$$
$$X \rightarrow \mathtt{a}ZZ\mathtt{c}$$

$$X \Rightarrow \mathtt{a}Y\mathtt{c} \Rightarrow \mathtt{a}ZZ\mathtt{c}$$

NORMAL FORMS
Rule Substitution

Theorem: Let G contain the rules:

$$X \to \alpha Y \beta \qquad \text{and} \qquad Y \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n ,$$

Replace $X \to \alpha Y \beta$ by:

$$X \to \alpha\gamma_1\beta, \quad X \to \alpha\gamma_2\beta, \quad \ldots, \quad X \to \alpha\gamma_n\beta.$$

The new grammar G' will be equivalent to G.

NORMAL FORMS
Rule Substitution

Proof:

Every string in $L(G)$ is also in $L(G')$:

Suppose w is in L(G). We show that w is also in L(G').

If $X \rightarrow \alpha Y\beta$ is not used, then use same derivation.

If it is used, then one derivation is:

In G: $S \Rightarrow \ldots \Rightarrow \delta X\phi \Rightarrow \delta\alpha Y\beta\phi \Rightarrow \delta\alpha\gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

$$\boxed{\begin{array}{l} X \rightarrow \alpha Y\beta \\ Y \rightarrow \gamma_k \end{array}}$$

Use this one instead:

In G′: $S \Rightarrow \ldots \Rightarrow \delta X\phi \Rightarrow \delta\alpha\gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

$$\boxed{X \rightarrow \alpha\gamma_k\beta}$$

Every string in $L(G')$ is also in $L(G)$: Every new rule ($X \rightarrow \alpha\gamma_k\beta$) can be simulated by two old rules $X \rightarrow \alpha Y\beta$ and $Y \rightarrow \gamma_k$.

THE PRICE OF NORMAL FORMS

- CNF version of a grammar may be longer than the original grammar

- Conversion time: Suppose n be the length of the grammar
  - If run in order Step 1, 2, 3, 4 then Total: $O(2^n)$
  - If run in order Step 4, 1, 2, 3 then Total: $O(n^2)$

- Step 1 (*removeEps*) can take $O(2^n)$ time as we need to rewrite a single rule $X \rightarrow A_1A_2\ldots A_k$ into 2^k-1 rules
- But if we apply Step 4 (*removeLong*) first then all rules are of length 2 at most. And none of them will be rewritten with 3 rules. So *removeEps* runs in linear time.

- Step 2 (*removeUnits*) runs in $O(n^2)$ time
- Step 3 (*removeMixed*) runs in linear time
- Step 4 (*removeLong*) runs in linear time

- Conversion doesn't change weak generative capacity but it may change strong generative capacity.

RECOGNIZING CONTEXT-FREE LANGUAGES

We need a device similar to an FSM except that it needs more power.

The insight: Precisely what it needs is a stack, which gives it an unlimited amount of memory with a restricted structure.

Example: Bal (the balanced parentheses language)

(((())))

DEFINITION OF A PUSHDOWN AUTOMATON

A ***pushdown automaton*** is a 6-tuple $M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

- K is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $s \in K$ is the initial state
- $A \subseteq K$ is the set of accepting states, and
- Δ is the transition **relation**.

DEFINITION OF A PUSHDOWN AUTOMATON

Δ, the transition relation, is a finite subset of

$$(\underbrace{K}_{\text{state}} \times (\underbrace{\Sigma \cup \{\varepsilon\}}_{\text{input or } \varepsilon}) \times \underbrace{\Gamma^*}_{\substack{\text{string of}\\ \text{symbols}\\ \text{to pop}\\ \text{from top}\\ \text{of stack}}}) \times (\underbrace{K}_{\text{state}} \times \underbrace{\Gamma^*}_{\substack{\text{string of}\\ \text{symbols}\\ \text{to push}\\ \text{on top}\\ \text{of stack}}})$$

DEFINITION OF A PUSHDOWN AUTOMATON

A configuration of M is an element of $K \times \Sigma^* \times \Gamma^*$.

The initial configuration of M is (s, w, ε).

MANIPULATING THE STACK


```
c
a
b
```

will be written as `cab`

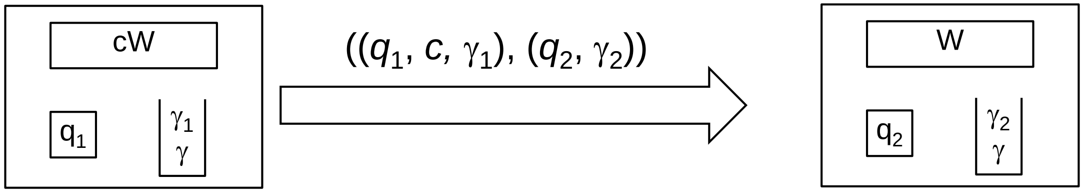
If $c_1c_2\ldots c_n$ (right to left) is pushed onto the stack: $c_1c_2\ldots c_n$`cab`

c_1
c_2
c_n
`c`
`a`
`b`

YIELDS

Let c be any element of $\Sigma \cup \{\varepsilon\}$,
Let γ_1, γ_2 and γ be any elements of Γ^*, and
Let w be any element of Σ^*.

Then:
$(q_1, cw, \gamma_1\gamma) \vdash_M (q_2, w, \gamma_2\gamma)$ iff $((q_1, c, \gamma_1), (q_2, \gamma_2)) \in \Delta$.



Let $\vdash_M^*$ be the reflexive, transitive closure of $\vdash_M$.

C_1 ***yields*** configuration C_2 iff $C_1 \vdash_M^* C_2$

COMPUTATIONS

A ***computation*** by M is a finite sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:

- C_0 is an initial configuration,
- C_n is of the form (q, ε, γ), for some state $q \in K$ and some string γ in Γ^*, and
- $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

NONDETERMINISM

If M is in some configuration (q_1, s, γ) it is possible that:

- Δ contains exactly one transition that matches.
- Δ contains more than one transition that matches.
- Δ contains no transition that matches.
- Δ is a relation not function.

ACCEPTING

A computation C of M is an ***accepting computation*** iff:

- $C = (s, w, \varepsilon) \mid\!-_M^* (q, \varepsilon, \varepsilon)$, and
- $q \in A$.

M ***accepts*** a string w iff at least one of its computations accepts.

ACCEPTING

Other paths may:

- Read all the input and halt in a nonaccepting state,
- Read all the input and halt in an accepting state with the stack not empty,
- Loop forever and never finish reading the input, or
- Reach a dead end where no more input can be read.

The ***language accepted by*** M, denoted $L(M)$, is the set of all strings accepted by M.

REJECTING

A computation C of M is a ***rejecting computation*** iff:

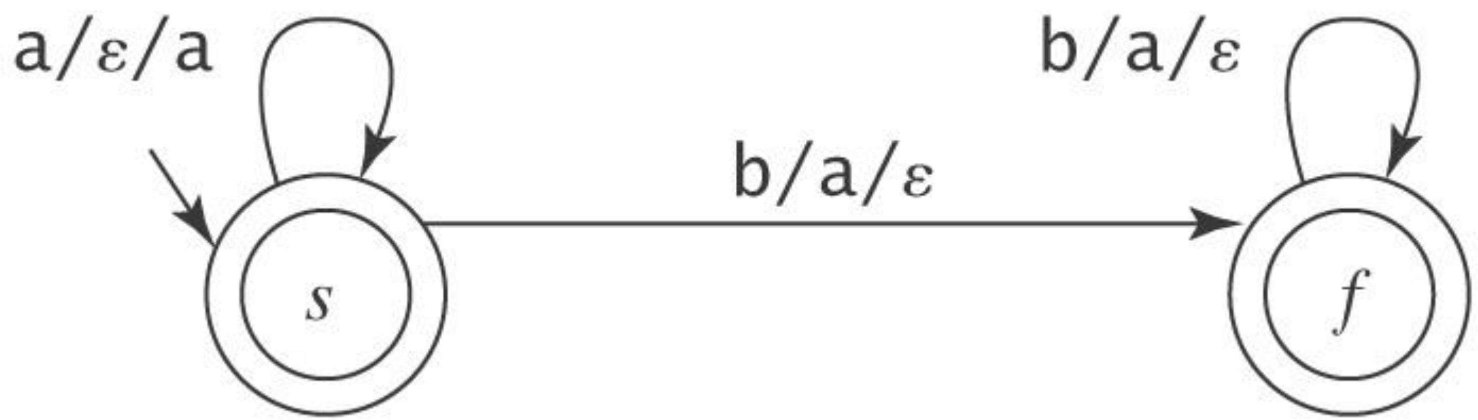
- $C = (s, w, \varepsilon) \mid\!-_M{}^{*} (q, w', \alpha)$,
- C is not an accepting computation, and
- M has no moves that it can make from (q, w', α).

M ***rejects*** a string w iff all of its computations reject.

So note that it is possible that, on input w, M neither accepts nor rejects.

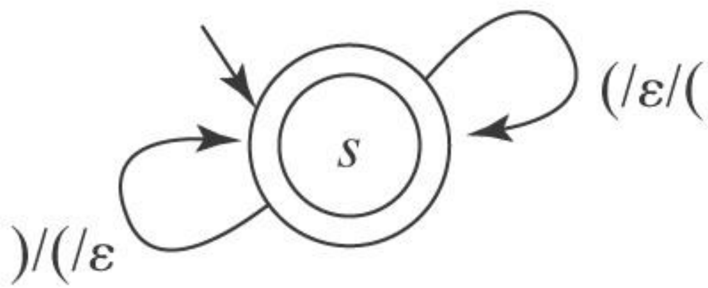
A PDA for $A^nB^n = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$

A PDA for $A^nB^n = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$



A PDA FOR BALANCED PARENTHESES

A PDA FOR BALANCED PARENTHESES



$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

$K = \{s\}$ the states

$\Sigma = \{(,)\}$ the input alphabet

$\Gamma = \{(\}$ the stack alphabet

$A = \{s\}$

Δ contains:

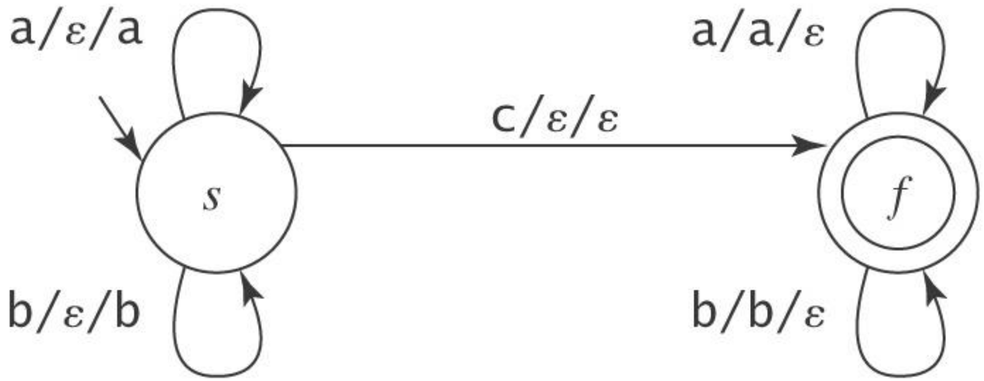
((s, (, ε**), (s, ( ) **)**

((s,), ( ), (s, ε) **)**

**Important: This does not mean that the stack is empty

A PDA for $\{wcw^R: w \in \{a, b\}^*\}$

A PDA for $\{wcw^R: w \in \{a, b\}^*\}$

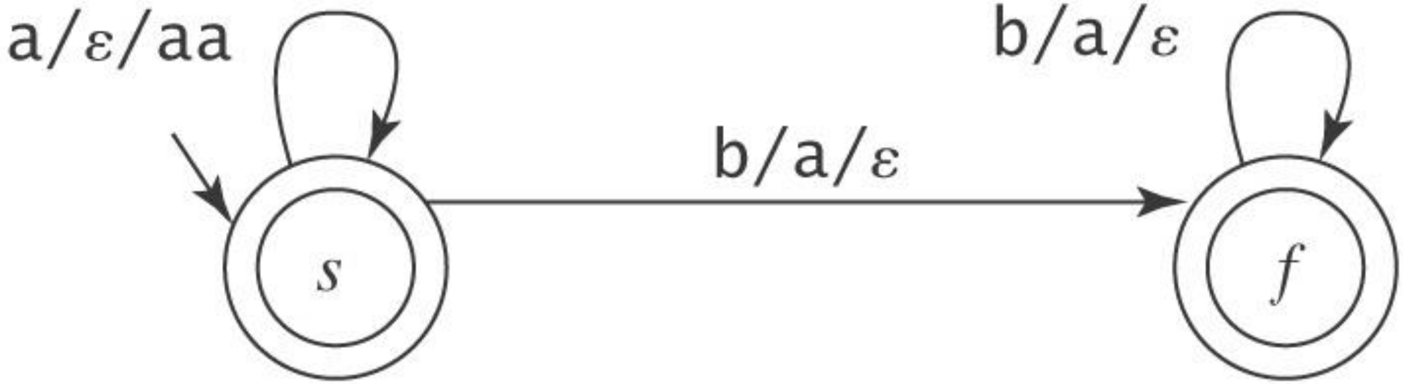


$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

- $K = \{s, f\}$ the states
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ the input alphabet
- $\Gamma = \{\mathtt{a}, \mathtt{b}\}$ the stack alphabet
- $A = \{f\}$ the accepting states
- Δ contains:
  - $((s, \mathtt{a}, \varepsilon), (s, \mathtt{a}))$
  - $((s, \mathtt{b}, \varepsilon), (s, \mathtt{b}))$
  - $((s, \mathtt{c}, \varepsilon), (f, \varepsilon))$
  - $((f, \mathtt{a}, \mathtt{a}), (f, \varepsilon))$
  - $((f, \mathtt{b}, \mathtt{b}), (f, \varepsilon))$

A PDA for {$\mathtt{a}^n\mathtt{b}^{2n}$: $n \geq 0$}

A PDA for $\{a^n b^{2n}: n \geq 0\}$

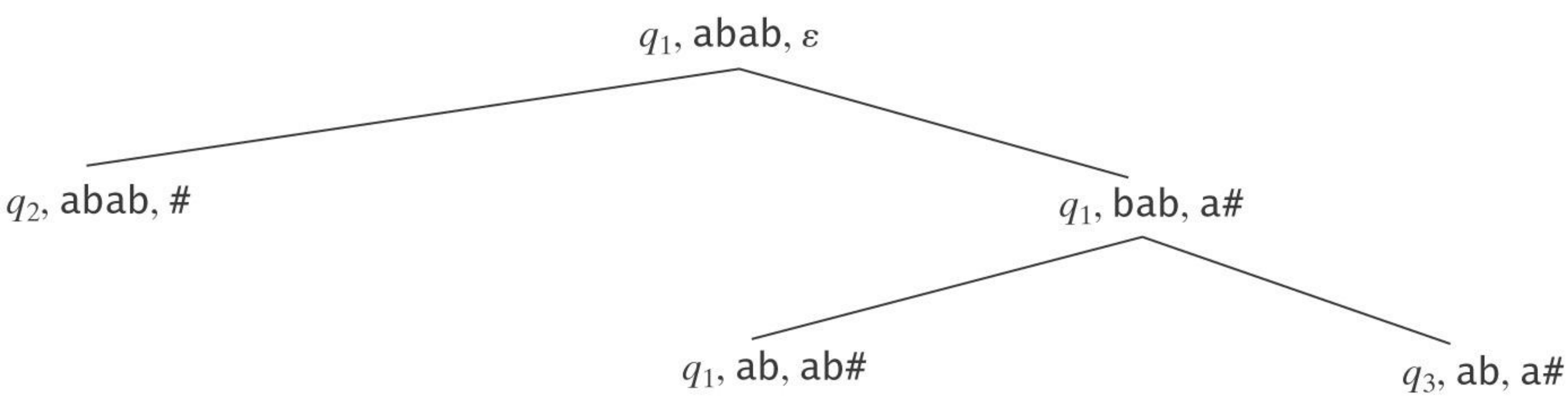


NONDETERMINISM

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 is never forced to choose between accepting and continuing. i.e. no transition $((q,\varepsilon, \varepsilon),(p,a))$ where q is an accepting state.

But many useful PDAs are not deterministic.



A PDA for PalEven $=\{ww^R: w \in \{\mathtt{a}, \mathtt{b}\}^*\}$

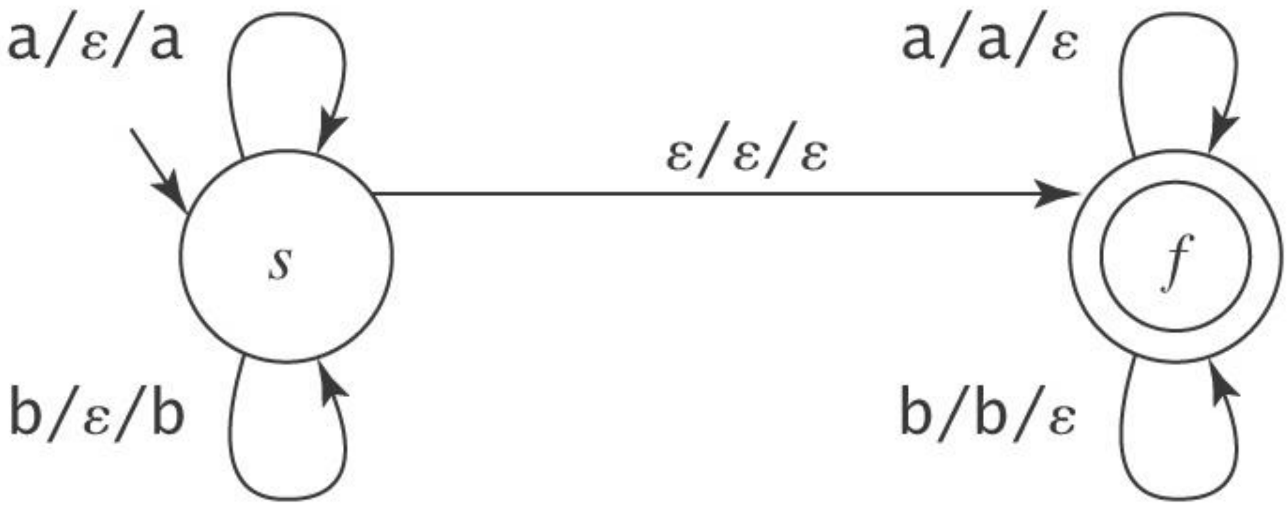
$$S \to \varepsilon$$
$$S \to \mathtt{a}S\mathtt{a}$$
$$S \to \mathtt{b}S\mathtt{b}$$

A PDA:

A PDA for PalEven ={ww^R: $w \in \{\mathtt{a}, \mathtt{b}\}^*$}

$S \rightarrow \varepsilon$
$S \rightarrow \mathtt{a}S\mathtt{a}$
$S \rightarrow \mathtt{b}S\mathtt{b}$

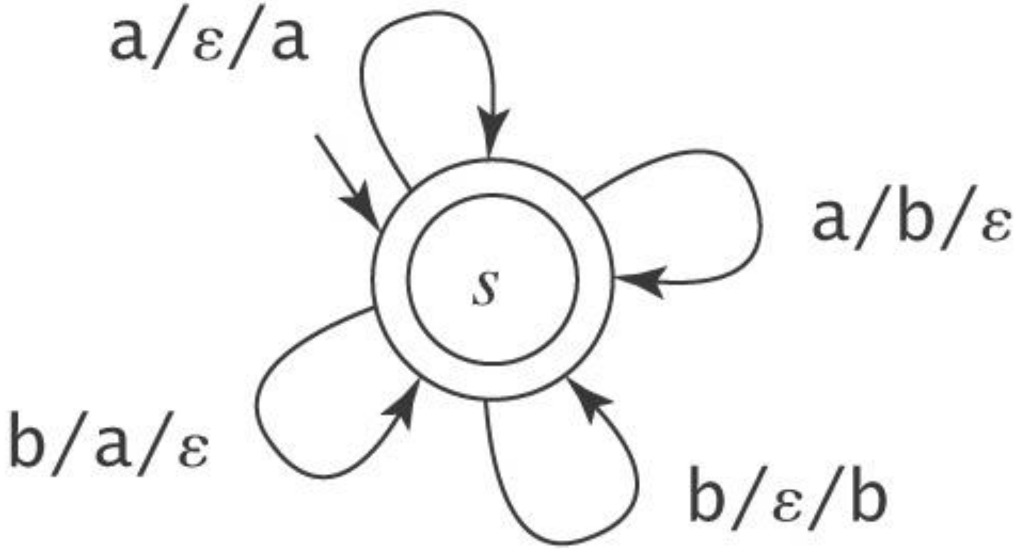
A PDA:



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

A PDA for $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : \#_{\mathtt{a}}(w) = \#_{\mathtt{b}}(w)\}$

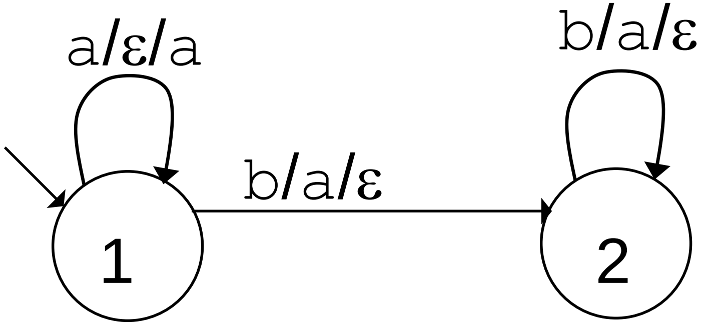
A PDA for $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : \#_{\mathtt{a}}(w) = \#_{\mathtt{b}}(w)\}$



NONDETERMINISM

$L = \{\mathtt{a}^m\mathtt{b}^n : m \neq n; m, n > 0\}$

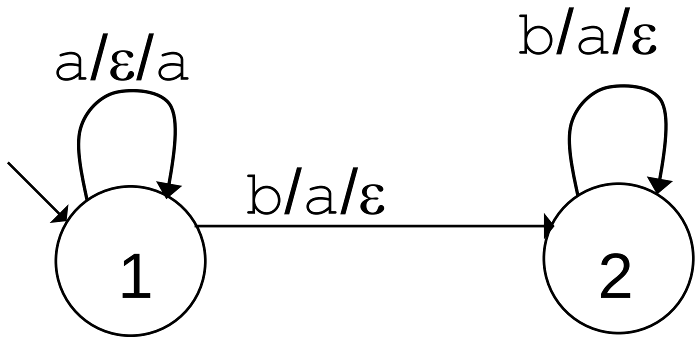
Start with the case where $n = m$:



NONDETERMINISM

$L = \{\mathtt{a}^m\mathtt{b}^n : m \neq n;\ m, n > 0\}$

Start with the case where $n = m$:



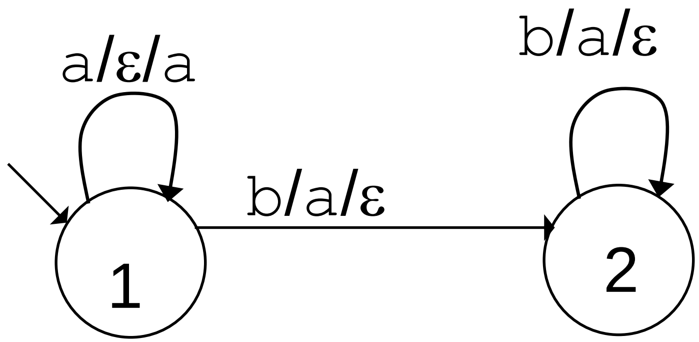
If stack and input are empty, halt and reject.

If input is empty but stack is not ($m > n$) (accept):

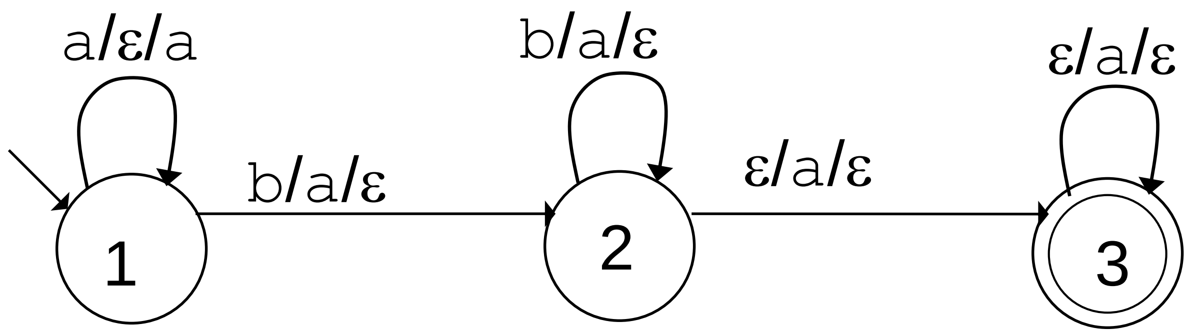
If stack is empty but input is not ($m < n$) (accept):

NONDETERMINISM

$L = \{\mathrm{a}^m\mathrm{b}^n : m \neq n; m, n > 0\}$

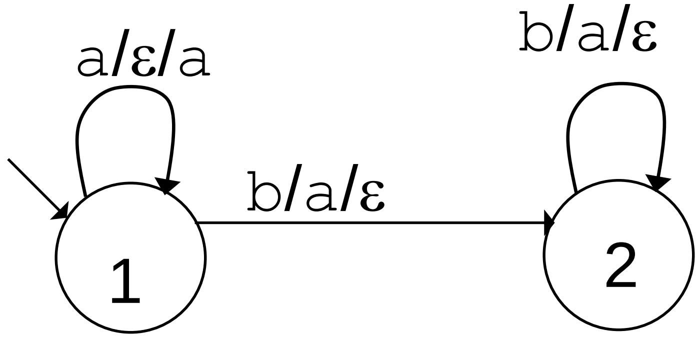


If input is empty but stack is not ($m > n$) (accept):

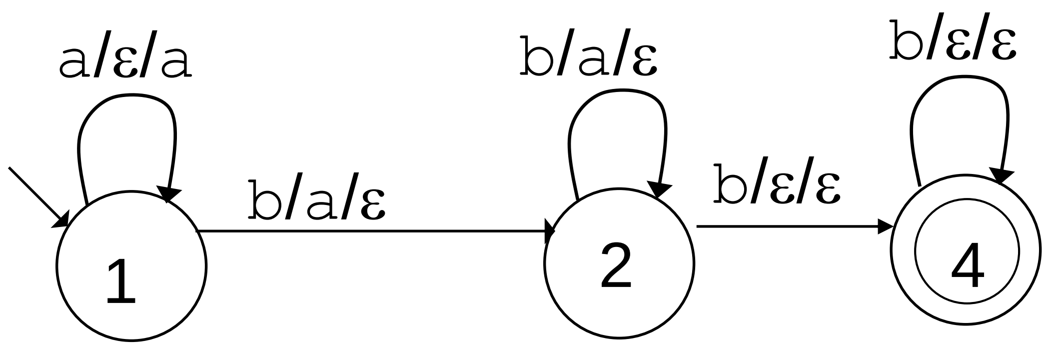


NONDETERMINISM

$L = \{\mathtt{a}^m\mathtt{b}^n : m \neq n; m, n > 0\}$

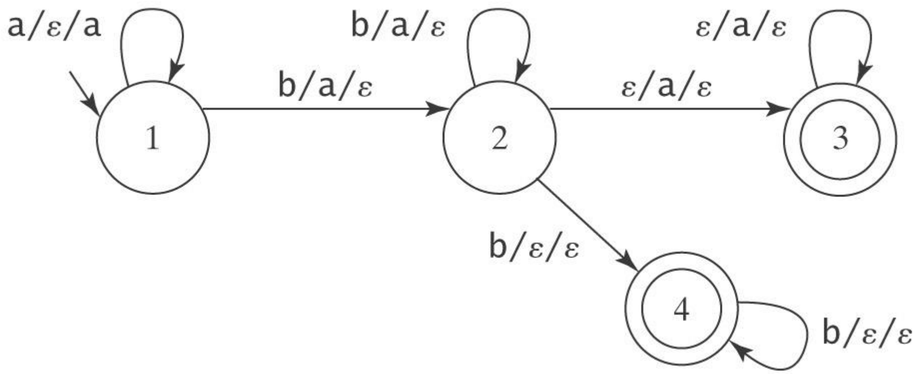


If stack is empty but input is not ($m < n$) (accept):



PUTTING IT TOGETHER

$L = \{\mathtt{a}^m\mathtt{b}^n : m \neq n; m, n > 0\}$



Jumping to the input clearing state 4:
Need to detect bottom of stack.

Jumping to the stack clearing state 3:
Need to detect end of input.

NONDETERMINISM

Consider $A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$.

PDA for it?

NONDETERMINISM

Consider $A^nB^nC^n = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \geq 0\}$.

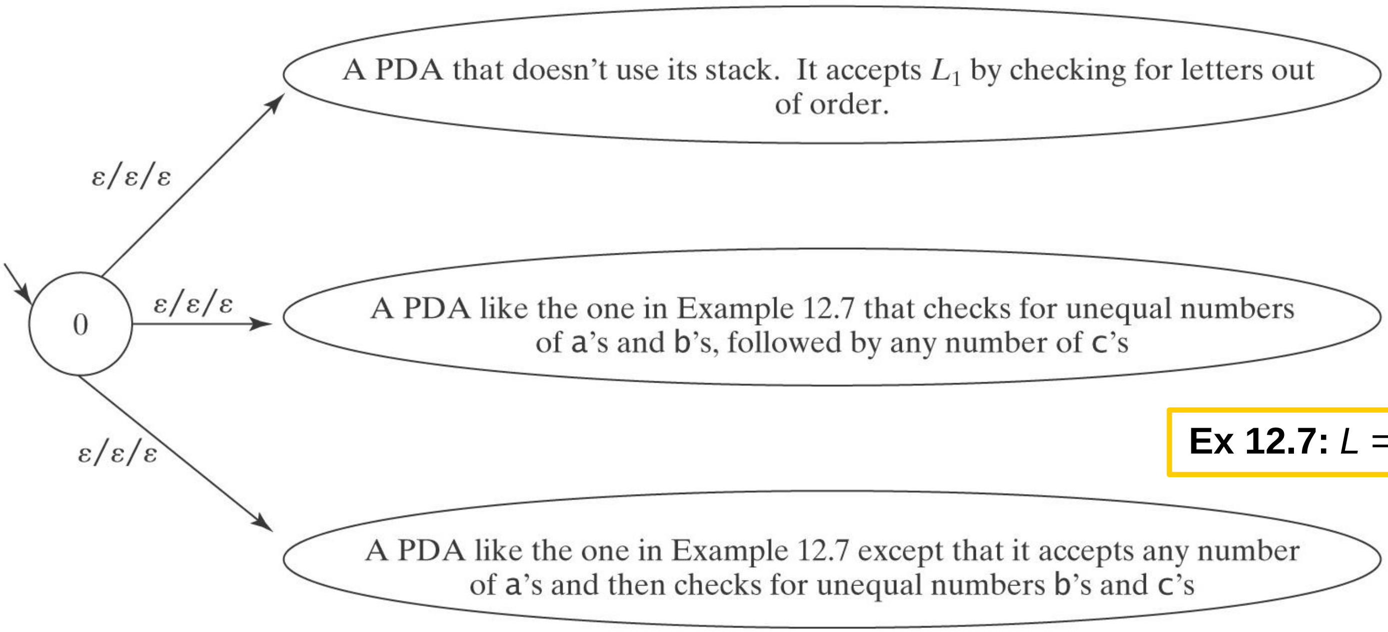
Now consider $L = \neg A^nB^nC^n$. L is the union of two languages:

1. $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$: the letters are out of order$\}$, and

2. $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i, j, k \geq 0$ and $(i \neq j$ or $j \neq k)\}$ (in other words, unequal numbers of $\mathtt{a}$'s, $\mathtt{b}$'s, and $\mathtt{c}$'s).

NONDETERMINISM

A PDA for $L = \neg A^n B^n C^n$



Ex 12.7: $L = \{a^m b^n : m \neq n;\ m, n > 0\}$

ARE THE CONTEXT-FREE LANGUAGES CLOSED UNDER COMPLEMENT?

$\neg A^nB^nC^n$ is context free.

If the CF languages were closed under complement, then

$$\neg\neg A^nB^nC^n \qquad = \; A^nB^nC^n$$

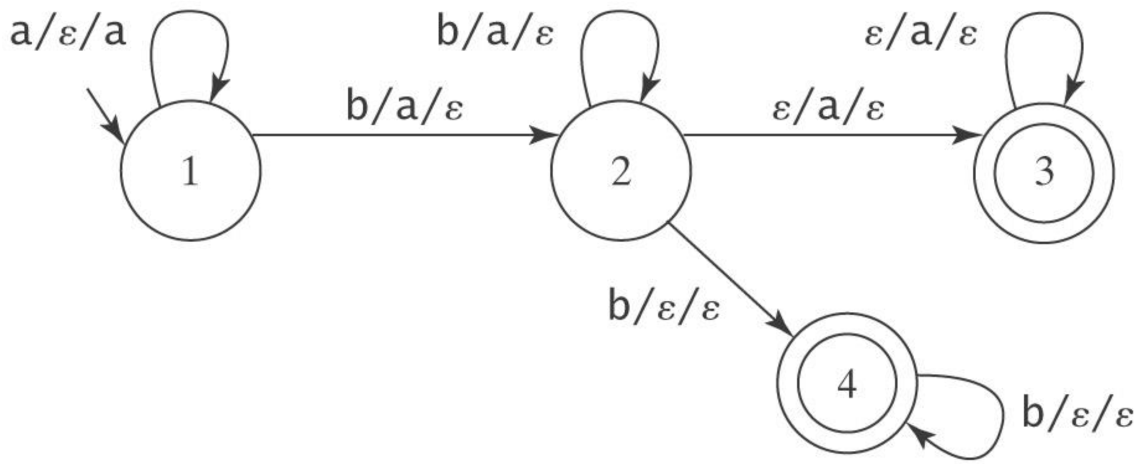
would also be context-free.

But we will prove that it is not.

$L = \{a^n b^m c^p: n, m, p \geq 0 \text{ and } n \neq m \text{ or } m \neq p\}$

$S \rightarrow NC$	/* $n \neq m$, then arbitrary c's
$S \rightarrow QP$	/* arbitrary a's, then $p \neq m$
$N \rightarrow A$	/* more a's than b's
$N \rightarrow B$	/* more b's than a's
$A \rightarrow \mathtt{a}$	
$A \rightarrow \mathtt{a}A$	
$A \rightarrow \mathtt{a}A\mathtt{b}$	
$B \rightarrow \mathtt{b}$	
$B \rightarrow B\mathtt{b}$	
$B \rightarrow \mathtt{a}B\mathtt{b}$	
$C \rightarrow \varepsilon \mid \mathtt{c}C$	/* add any number of c's
$P \rightarrow B'$	/* more b's than c's
$P \rightarrow C'$	/* more c's than b's
$B' \rightarrow \mathtt{b}$	
$B' \rightarrow \mathtt{b}B'$	
$B' \rightarrow \mathtt{b}B'\mathtt{c}$	
$C' \rightarrow \mathtt{c} \mid C'\mathtt{c}$	
$C' \rightarrow C'\mathtt{c}$	
$C' \rightarrow \mathtt{b}C'\mathtt{c}$	
$Q \rightarrow \varepsilon \mid \mathtt{a}Q$	/* prefix with any number of a's

REDUCING NONDETERMINISM



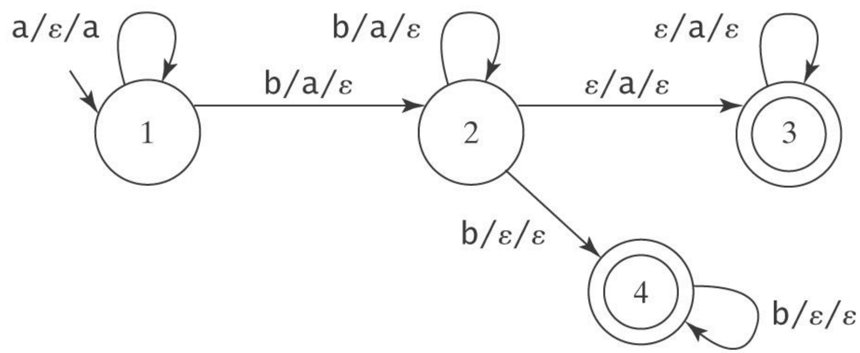
Jumping to the input clearing state 4:

Need to detect bottom of stack, so push # onto the stack before we start.

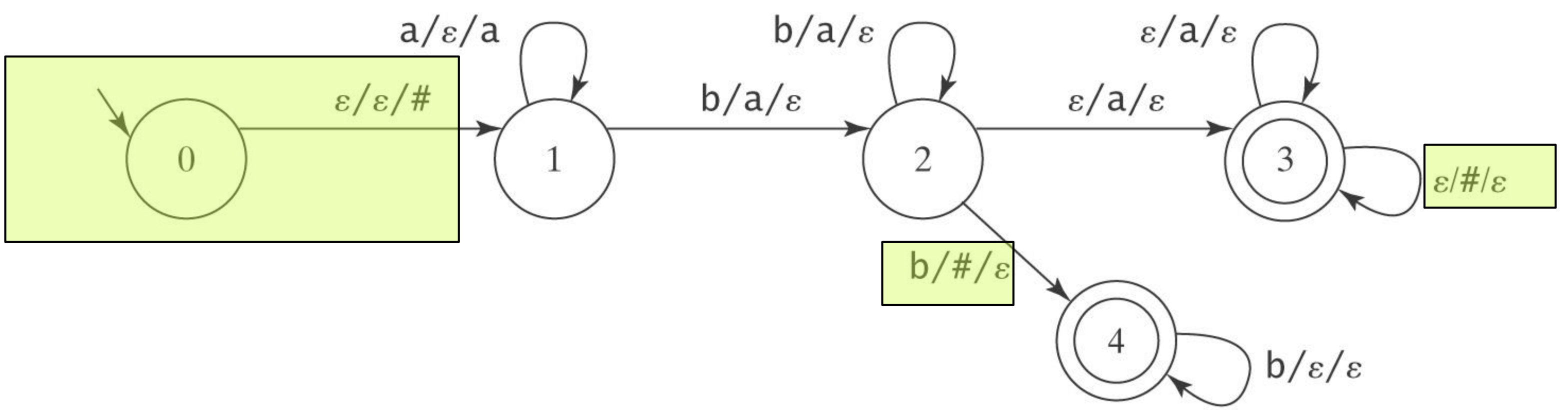
Jumping to the stack clearing state 3:

Need to detect end of input. Add to *L* a termination character (e.g., $)

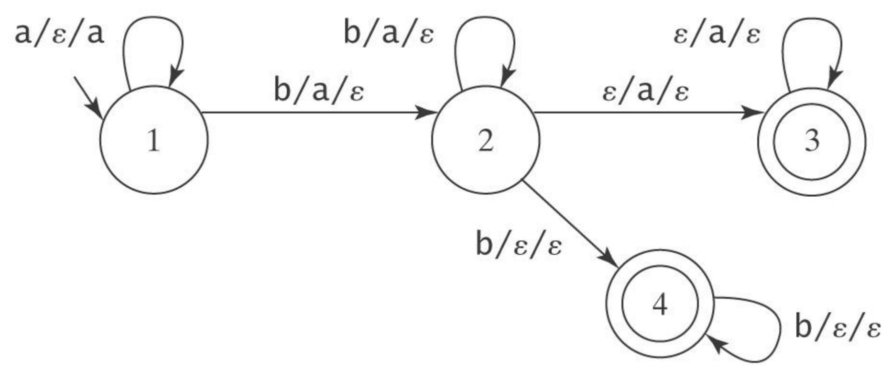
REDUCING NONDETERMINISM



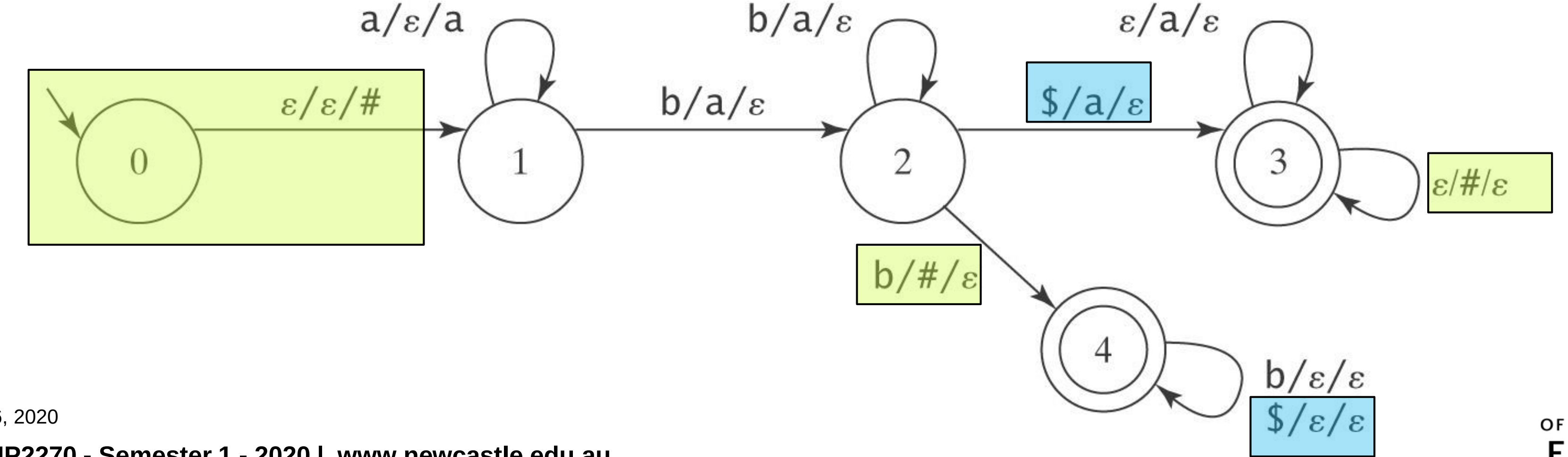
Jumping to the input clearing state 4:



REDUCING NONDETERMINISM

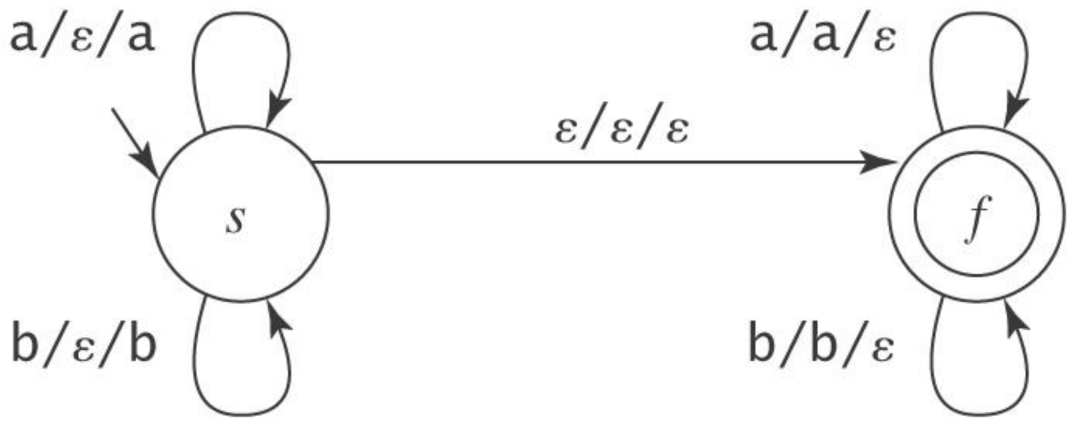


Jumping to the stack clearing state 3:



MORE ON PDAs

A PDA for $\{ww^R : w \in \{a, b\}^*\}$:



What about a PDA to accept $\{ww : w \in \{a, b\}^*\}$?

PDAs AND CONTEXT-FREE GRAMMARS

Theorem: The class of languages accepted by PDAs is exactly the class of context-free languages.

Recall: context-free languages are languages that can be defined with context-free grammars.

Restate theorem:

Can be described with context-free grammar

$=$

Can be accepted by PDA

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA

Lemma*:* Each context-free language is accepted by some PDA.

Proof (by construction):

The idea: Let the stack do the work.

Two approaches:

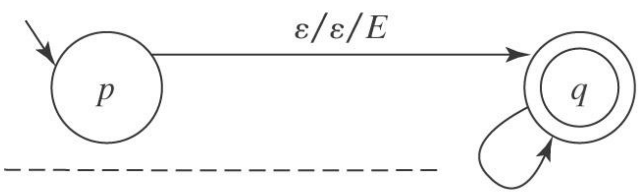
- Top down
- Bottom up

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA - Top Down

The idea: Let the stack keep track of expectations.

Example: Arithmetic expressions :: Σ = { id, +, * , (,) }

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow id$

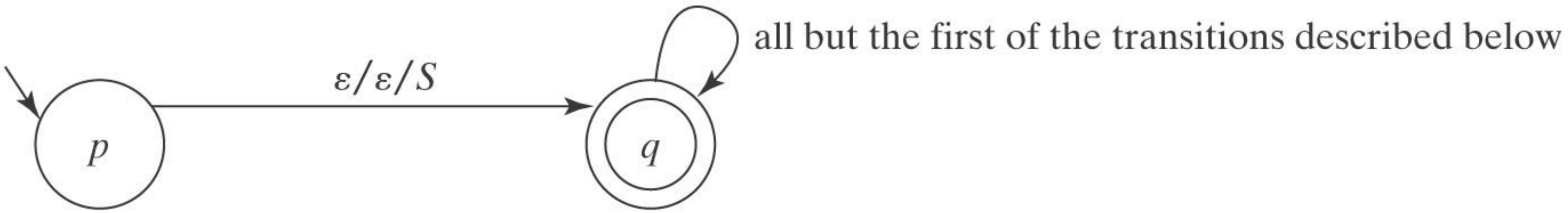


(1) $(q, \varepsilon, E), (q, E{+}T)$
(2) $(q, \varepsilon, E), (q, T)$
(3) $(q, \varepsilon, T), (q, T{*}F)$
(4) $(q, \varepsilon, T), (q, F)$
(5) $(q, \varepsilon, F), (q, (E)\ )$
(6) $(q, \varepsilon, F), (q, id)$

(7) $(q, id, id), (q, \varepsilon)$
(8) $(q, (, (\ \ ), (q, \varepsilon)$
(9) $(q,),)\ \ ), (q, \varepsilon)$
(10) $(q, +, +), (q, \varepsilon)$
(11) $(q, *, *), (q, \varepsilon)$

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA - Top Down

The outline of M is:



$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, 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:
  $((q, c, c), (q, \varepsilon))$.

PDAs AND CONTEXT-FREE GRAMMARS
Example of the construction

$L = \{a^n b^* a^n\}$

(1) $S \rightarrow \varepsilon$
(2) $S \rightarrow B$
(3) $S \rightarrow aSa$
(4) $B \rightarrow \varepsilon$
(5) $B \rightarrow bB$

0. $(p, \varepsilon, \varepsilon), (q, S)$
1. $(q, \varepsilon, S), (q, \varepsilon)$
2. $(q, \varepsilon, S), (q, B)$
3. $(q, \varepsilon, S), (q, aSa)$
4. $(q, \varepsilon, B), (q, \varepsilon)$
5. $(q, \varepsilon, B), (q, bB)$
6. $(q, a, a), (q, \varepsilon)$
7. $(q, b, b), (q, \varepsilon)$

PDAs AND CONTEXT-FREE GRAMMARS
Example of the construction

$L = \{a^n b^* a^n\}$

input = a a b b a a

Trans	*state*	*unread input*	*stack*
	p	a a b b a a	ε
0	q	a a b b a a	S
3	q	a a b b a a	aSa
6	q	a b b a a	Sa
3	q	a b b a a	aSaa
6	q	b b a a	Saa
2	q	b b a a	Baa
5	q	b b a a	bBaa
7	q	b a a	Baa
5	q	b a a	bBaa
7	q	a a	Baa
4	q	a a	aa
6	q	a	a
6	q	ε	ε

0 (p, ε, ε), (q, S)
1 (q, ε, S), (q, ε)
2 (q, ε, S), (q, B)
3 (q, ε, S), (q, aSa)
4 (q, ε, B), (q, ε)
5 (q, ε, B), (q, bB)
6 (q, a, a), (q, ε)
7 (q, b, b), (q, ε)

PDAs AND CONTEXT-FREE GRAMMARS
Another example of the construction

$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^p\mathtt{d}^q : m + n = p + q\}$

PDAs AND CONTEXT-FREE GRAMMARS
Another example of the construction

$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^p\mathtt{d}^q : m + n = p + q\}$

(1) $S \to \mathtt{a}S\mathtt{d}$
(2) $S \to T$
(3) $S \to U$
(4) $T \to \mathtt{a}\mathrm{T}\mathtt{c}$
(5) $T \to V$
(6) $\mathrm{U} \to \mathtt{b}U\mathtt{d}$
(7) $U \to V$
(8) $V \to \mathtt{b}V\mathtt{c}$
(9) $V \to \varepsilon$

input = `a a b c d d`

PDAs AND CONTEXT-FREE GRAMMARS
Another example of the construction

$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^p\mathtt{d}^q : m + n = p + q\}$

(1) $S \to \mathtt{a}S\mathtt{d}$
(2) $S \to T$
(3) $S \to U$
(4) $T \to \mathtt{a}\mathrm{T}\mathtt{c}$
(5) $T \to V$
(6) $\mathrm{U} \to \mathtt{b}U\mathtt{d}$
(7) $U \to V$
(8) $V \to \mathtt{b}V\mathtt{c}$
(9) $V \to \varepsilon$

0. $(p, \varepsilon, \varepsilon), (q, S)$
1. $(q, \varepsilon, S), (q, \mathtt{a}S\mathtt{d})$
2. $(q, \varepsilon, S), (q, T)$
3. $(q, \varepsilon, S), (q, U)$
4. $(q, \varepsilon, T), (q, \mathtt{a}T\mathtt{c})$
5. $(q, \varepsilon, T), (q, V)$
6. $(q, \varepsilon, U), (q, \mathtt{b}U\mathtt{d})$
7. $(q, \varepsilon, U), (q, V)$
8. $(q, \varepsilon, V), (q, \mathtt{b}V\mathtt{c})$
9. $(q, \varepsilon, V), (q, \varepsilon)$
10. $(q, \mathtt{a}, \mathtt{a}), (q, \varepsilon)$
11. $(q, \mathtt{b}, \mathtt{b}), (q, \varepsilon)$
12. $(q, \mathtt{c}, \mathtt{c}), (q, \varepsilon)$
13. $(q, \mathtt{d}, \mathtt{d}), (q, \varepsilon)$

THE OTHER WAY TO BUILD A PDA - DIRECTLY

$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^p\mathtt{d}^q : m + n = p + q\}$

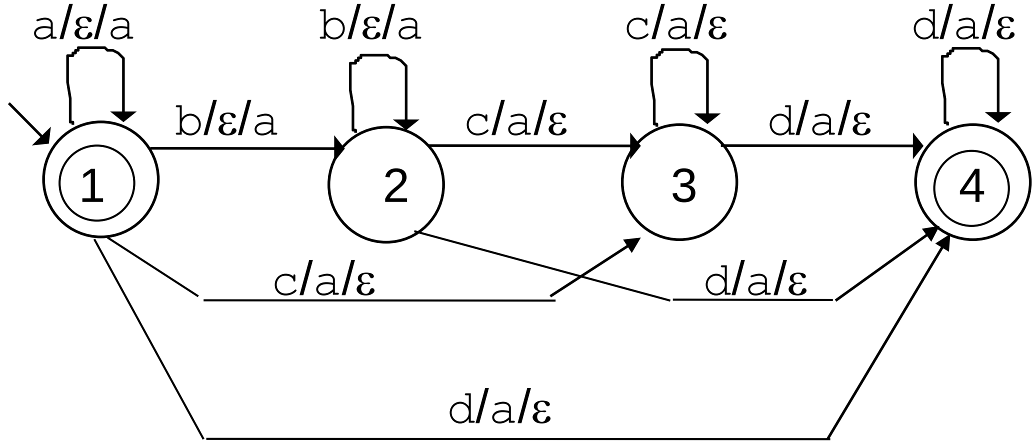
(1) $S \to \mathtt{a}S\mathtt{d}$
(2) $S \to T$
(3) $S \to U$
(4) $T \to \mathtt{a}T\mathtt{c}$
(5) $T \to V$

(6) $U \to \mathtt{b}U\mathtt{d}$
(7) $U \to V$
(8) $V \to \mathtt{b}V\mathtt{c}$
(9) $V \to \varepsilon$

THE OTHER WAY TO BUILD A PDA - DIRECTLY

$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^p\mathtt{d}^q : m + n = p + q\}$

(1) $S \rightarrow \mathtt{a}S\mathtt{d}$
(2) $S \rightarrow T$
(3) $S \rightarrow U$
(4) $T \rightarrow \mathtt{a}T\mathtt{c}$
(5) $T \rightarrow V$
(6) $U \rightarrow \mathtt{b}U\mathtt{d}$
(7) $U \rightarrow V$
(8) $V \rightarrow \mathtt{b}V\mathtt{c}$
(9) $V \rightarrow \varepsilon$



input = a a b c d d

NONDETERMINISM

Machines constructed with the algorithm are often nondeterministic, even when they needn't be. This happens even with trivial languages.

Example: $A^nB^n = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$

A grammar for A^nB^n is:

[1] $S \rightarrow \mathtt{a}S\mathtt{b}$
[2] $S \rightarrow \varepsilon$

A PDA M for A^nB^n is:

(0) $((p, \varepsilon, \varepsilon), (q, S))$
(1) $((q, \varepsilon, S), (q, \mathtt{a}S\mathtt{b}))$
(2) $((q, \varepsilon, S), (q, \varepsilon))$
(3) $((q, \mathtt{a}, \mathtt{a}), (q, \varepsilon))$
(4) $((q, \mathtt{b}, \mathtt{b}), (q, \varepsilon))$

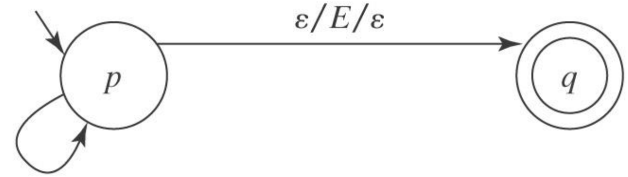
But transitions 1 and 2 make M nondeterministic.

A directly constructed machine for A^nB^n:

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA - Bottom up

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

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



Reduce Transitions:

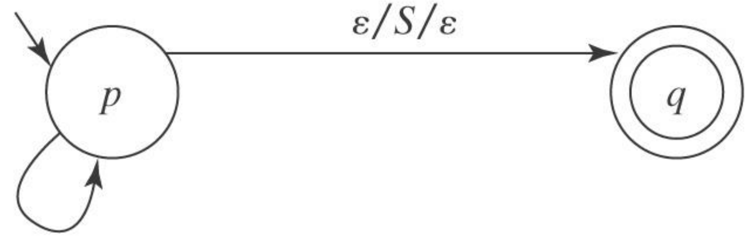
(1) $(p, \varepsilon, T + E), (p, E)$
(2) $(p, \varepsilon, T), (p, E)$
(3) $(p, \varepsilon, F * T), (p, T)$
(4) $(p, \varepsilon, F), (p, T)$
(5) $(p, \varepsilon,)E( ), (p, F)$
(6) $(p, \varepsilon, \text{id}), (p, F)$

Shift Transitions

(7) $(p, \text{id}, \varepsilon), (p, \text{id})$
(8) $(p, (, \varepsilon), (p, ()$
(9) $(p,), \varepsilon), (p,))$
(10) $(p, +, \varepsilon), (p, +)$
(11) $(p, *, \varepsilon), (p, *)$

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA - Bottom up

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 $X \rightarrow s_1 s_2 \ldots s_n.$ in G.
- The finish up transition: $((p, \varepsilon, S), (q, \varepsilon))$.

PDAs AND CONTEXT-FREE GRAMMARS
From PDA to 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:
Step 2: Convert the PDA (in restricted normal form) to a CFG.

Pages: 265~273 (have a look)

NONDETERMINISM AND HALTING

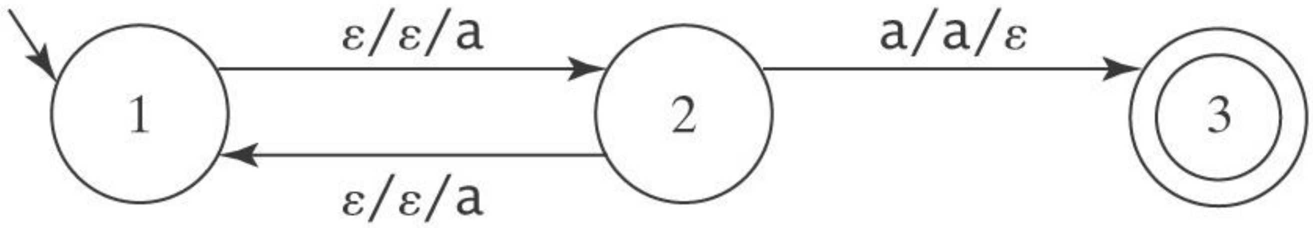
1. There are CFL for which no deterministic PDA exists.
2. There exist no algorithm to minimize a PDA.
   - It is undecidable whether a PDA is already minimal

NONDETERMINISM AND HALTING

It is possible that a PDA may

- not halt,
- not ever finish reading its input.

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



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

On any other input except a:

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

NONDETERMINISM AND HALTING

Solutions to the Problem

For NDFSMs:

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

For NDPDAs:

- Formal solutions that usually involve changing the form of the grammar.
- Practical solutions that:
  - Preserve the structure of the grammar, but
  - Only work on a subset of the CFLs.

COMPARING REGULAR AND CONTEXT-FREE LANGUAGES

Regular Languages

- Regular expressions
  Regular grammars
- **recognize**
- = DFSMs

Context-Free Languages

- Context-free grammars
- **parse**
- = NDPDAs

References

- **Automata, Computability and Complexity. Theory and Applications**
  - By Elaine Rich
- Chapter 11:
  - Page : 224-227, 232-241.
- Chapter 12:
  - Page : 249-275.