Example 2: \[ L = \{ w \# w^r \mid w \in \{a, b\}^* \} \]

palindromes with center marker

Using no heuristics mentioned in the previous class notes:

- **PDA**
  - use the stack to remember the "first half" (prior to the center marker), and, later, check against the "second half" after the center marker
  - use two loops (states) to process the remembering-loop and checking-off-loop
  - when the bottom-stack is exposed (using "Zo") and the input has been processed (guessing), ready to accept.
(q4, ε, ε) includes (q4, z₀)
(s (remember, q, ε) includes (remember, A))
(s (remember, b, ε) includes (remember, B))

loop at q to remember the "first half" prior to ""

(s (remember, $, ε) includes (q check off, exit q remember to enter q check off

(s (check off, a, A) includes (check off, ε)
(s (check off, b, B) includes (check off, ε)

loop at check off to check off the "first half"
(remembered in stack)
versus the second half (being processed)
in reverse manner

(s (check off, ε, z₀) includes (check off, ε)

"z₀" is exposed: first-half of second-half (so far in consumed input) are in memory
and guess the completion of input.
Check:

If \( x \in L \) then \( M \) accepts \( x \):

- General: \( w \# w^r \) : \( M \) works correctly
- Extreme: \( E \# E^r = \$ \) also works correctly

If \( x \notin L \) then \( M \) should not accept \( x \):

- \( w \# u \) when \( u \neq w^r \):
  - Case 1: \( w \# w^r v \) and \( v \neq E \)
    - \( E \# E^r \) and \( c \neq c' \)
    - \( c \# w \# w^r \)
    - \( c' \# w \# w^r \)
  - Case 2: \( \ldots \)
In the previous example, \( L = \{ w \# w \mid w \in \{a,b\}^* \} \), no PDA \( M \) behaves "deterministically".

**Example 3:** \( L = \{ w w r \mid w \in \{a,b\}^* \} \)

*Remember/Warning:* a typical string in \( L \) is of the form \( w w r \) for some \( w \in \{a,b\}^* \), but, for PDAs or CFGs, the gadgets do not really know the "mid-point" (in Example 2, if such mid-point exists, it would be "\#")

So, \( L = \{ w w r \mid w \in \{a,b\}^* \} \)

\[
= \{ x \mid x \text{ is of the form } w w r \text{ for some } w \in \{a,b\}^* \}
\]

= the set of all even-length palindromes

Without any hint for the mid-point, how can \( M \) exit the remembering-loop and enter the checking-off loop?

— Use nondeterminism of its transition function
Most comments on $S$-function are similar to those in Example 2.

\[ S(q_1, 3, 3) \] includes \((q_{\text{remember}}, z_0)\).

\[ S(q_{\text{remember}}, 0, 3) \] includes \((q_{\text{remember}}, A)\).

\[ S(q_{\text{remember}}, b, 3) \] includes \((q_{\text{remember}}, B)\).

\[ S(q_{\text{remember}}, 3, 3) \] includes \((q_{\text{checkoff}}, \varepsilon)\).

Non-deterministically guess the mid-point and enter the check-off.

\[ S(q_{\text{checkoff}}, a, A) \] includes \((q_{\text{checkoff}}, \varepsilon)\).

\[ S(q_{\text{checkoff}}, b, B) \] includes \((q_{\text{checkoff}}, \varepsilon)\).

\[ S(q_{\text{checkoff}}, \varepsilon, z_0) \] includes \((q_{\text{accept}}, \varepsilon)\).
Check:

\[
\text{if } x \in L \text{ then } M \text{ accepts } x :
\]

general \( x \): \( M \) works correctly of the form \( \text{ww}^R \) (appreciate the "nondeterministic power" of \( \Sigma \))

extreme \( x \): \( y \) \( \text{ww}^R \) also works correctly

\[
E_1 \xrightarrow{\epsilon} E_2 \xrightarrow{\epsilon} E_3 \xrightarrow{\epsilon} \text{ reject}
\]

\[
\text{if } x \notin L \text{ then } M \text{ does not accept } x :
\]

How can an input \( x \notin L \)?

Case when \( x \) has odd length:

\( M \) can never make a correct guess to check off the stack contents versus the residual input.

Case when \( x \) has even length, but is not of the form \( \text{ww}^R \):

Out of many possible nondeterministic guesses, \( M \) makes one at the "mid-point". But, \( M \) cannot complete the check off due to the fact that \( x \) is not of the form \( \text{ww}\text{ww}^R \) for any \( w \in \Sigma^* \).
For the language \( \{wwr \mid w \in \{a, b\}^* \} \) in Example 3, the PDA is "truly" non-deterministic versus the "deterministic PDA" for the language \( \{w \# w \mid w \in \{a, b\}^* \} \) in Example 2.

E-transitions are allowed, but a deterministic PDA must satisfy the following:

1. a deterministic PDA does not have the option to choose between an E-transition and a non-E-transition:

\[
\forall q \in Q, \forall A \in T' \cup \{\varepsilon\},
\delta(q, \varepsilon, A) = \emptyset \quad \text{or} \quad \delta(q, a, A) = \emptyset
\]

for every \( a \in \Sigma \)

Can not have both E-transition and regular transition from the same configuration \((q, A)\)

2. as expected, as most one transition out of a configuration

\[
\forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall A \in T' \cup \{\varepsilon\},
|\delta(q, a, A)| \leq 1
\]

Check that:
- PDA constructed for Example 2 is deterministic.
- PDA constructed for Example 3 is not deterministic.

Consider the configuration \((q, \varepsilon, \{\text{top-stack}\})\)
[Sip12] Section 2.4
Deterministic CFLs — skipped
Example 4: \[ L = \{ \text{wcwr} \mid w \in \{a, b\}^*, c \in \{a, b\} \} \]

odd-length palindromes

\[ = \{ \epsilon \mid \epsilon = \text{wcwr} \text{ for some } w \in \{a, b\}^* \text{ and } c \in \{a, b\} \} \]

Idea: Similar to the construction in Example 3

- (instead of guessing the "mid-point",
  guess the "center-symbol" (and consume))

\[ s(q_4, \epsilon, \epsilon) \text{ includes } (q_{\text{remember}}, \epsilon_0) \]

\[ \begin{cases} 
  s(q_{\text{remember}}, a, \epsilon) \text{ includes } (q_{\text{remember}}, A) & \quad (1) \\
  s(q_{\text{remember}}, b, \epsilon) \text{ includes } (q_{\text{remember}}, B) & \quad (2) 
\end{cases} \]

\[ \begin{cases} 
  s(q_{\text{remember}}, a, \epsilon) \text{ includes } (q_{\text{checkoff}}, \epsilon) & \quad (1') \\
  s(q_{\text{remember}}, b, \epsilon) \text{ includes } (q_{\text{checkoff}}, \epsilon) & \quad (2') 
\end{cases} \]

Nondeterministically guess the center-symbol (and consume),
then exit to \(q_{\text{checkoff}}\).

Note: Can group together the two "inclusions"

\[ \begin{cases} 
  s(q_{\text{remember}}, a, \epsilon) = \{ (q_{\text{remember}}, A), (q_{\text{checkoff}}, \epsilon) \} \\
  s(q_{\text{remember}}, b, \epsilon) = \{ (q_{\text{remember}}, B), (q_{\text{checkoff}}, \epsilon) \} 
\end{cases} \]
Check the correctness of the PDA (exercise)

Example 5: \( L = \{ x \mid x \text{ is a palindrome in } \{a,b\}^* \} \)

\[ L = \left\{ \begin{array}{l}
\left( x \mid x \text{ is a palindrome in } \{a,b\}^* \right) \\
\text{odd-length or even-length } \end{array} \right\} 
\]

Example 3

Example 4

a desired PDA:

\( S(q_{start}, \varepsilon, \varepsilon) \text{ includes } (q_{even}, \varepsilon), (q_{odd}, \varepsilon) \)

PDA for Example 3

PDA for Example 4
Example 6: \[ L = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \} \]

Idea - use nondeterministic \( \varepsilon \)-transitions to implement the "or"

Read state transition diagram in [Sip12] Example 2.17.

**Figure 2.17**
State diagram for PDA \( M_2 \) that recognizes
\[ \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \} \]
Computational Equivalence

\[ \text{CFG} \equiv \text{PDA} \iff \text{nondeterministic} \]

\( \text{CFG} \leq \text{PDA} : \)

\( \forall \text{CFG} \ G \ \exists \text{PDA} \ M \ \text{such that} \ L(M) = L(G) \)

\( \text{PDA} \leq \text{CFG} : \)

\( \forall \text{PDA} \ M \ \exists \text{CFG} \ G \ \text{such that} \ L(G) = L(M) \)

Both theorems and proofs are detailed in [Sip12].

We cover "CFG \leq \text{PDA}" and leave "PDA \leq \text{CFG}" to interested readers.

(to appreciate the nondeterministic power of CFGs)

With these two theorems, \( \text{CFG} \equiv \text{PDA} \),

we can say that a language is CF

if and only if the language is generated by a CFG,

or equivalently, the language is accepted by a PDA.
A CFG $G \in$ PDA $M$ such that $L(M) = L(G)$.

Idea: Say, CFG $G = (V, \Sigma, P, S)$.

We aim to construct a PDA $M$ such that

$\forall x \in \Sigma^* \quad x \in L(G) \iff x \in L(M)$

$S \Rightarrow x$

Expand the multi-step derivation

in a "leftmost" manner

(always replace/substitute a leftmost variable by a production)

$S \Rightarrow \cdots \quad \text{"leftmost variable"}$

$\Rightarrow \begin{array}{c}
1 0 1 1 \\
\text{processed}
\end{array}$

not-yet-processed

derivation used $A \rightarrow \alpha$ production

$\Rightarrow \begin{array}{c}
1 0 1 1 \\
\alpha B 1 D \\
\text{processed}
\end{array}$

Simulate the 1-step "leftmost" derivation in $M$:

$\begin{array}{c}
1 0 1 1 \\
\text{processed}
\end{array}$

not-yet-processed

\begin{align*}
\text{control} & \quad \rightarrow \\
\alpha & \quad \rightarrow
\end{align*}
Note: It is not difficult to see that we may assume/modify the transition function \( \delta \) such that 
\[
\delta(q, a, A) \text{ includes } (p, \alpha)
\]
with \( \alpha \in \Gamma^* \) 
(\text{instead of } \alpha \in \Gamma \cup \{ \epsilon \}).

See [Sip12] for details: adding more states to simulate pushing \( \alpha \) onto the stack.

Now, a desired PDA simulates

- \( \delta \)-step leftmost derivation as shown in the previous page, or

\[
S \Rightarrow \ldots \\
\Rightarrow \langle 1011 \rangle 4 \ldots \\
\Rightarrow \langle 10114 \rangle \ldots
\]

\[
\text{processed}\quad \begin{array}{c}
4 \ldots \\
\text{not-yet-processed}
\end{array}
\]

when top-stack (\( q \) not-yet-processed) is not a variable (in CFG) check off the input symbol versus top-stack (if they match).
PDA M:

$s(q_{\text{start}}, \varepsilon, S) \quad \text{includes} \quad (q_{\text{loop}}, S \varepsilon_0)$

a single loop to simulate/process the leftmost derivation of the input

7. $A \rightarrow \alpha \in F$

$s(q_{\text{loop}}, \varepsilon, A) \quad \text{includes} \quad (q_{\text{loop}}, \alpha)$

nondeterministically guess a leftmost derivation (in $G$) when top-stack is a variable

4. $a \in \Sigma$

$s(q_{\text{loop}}, a, a) \quad \text{includes} \quad (q_{\text{loop}}, \varepsilon)$

$s(q_{\text{loop}}, \varepsilon, Z_0) \quad \text{includes} \quad (q_{\text{accept}}, \varepsilon)$

PDA

$S \Rightarrow \cdots$

$
\Rightarrow \quad \text{accept}
$
Section 2.3

(Closure Properties and) Non-Context-Freedom

Similar to the closure properties/operators for regularity, an operator \( \Theta \) preserves context-freedom (a closure property for context-freedom) if

\[
\text{for all CFLs } L_1 \text{ and } L_2, \quad L_1 \Theta L_2 \text{ is CF}
\]

(assume that \( \Theta \) is a binary operator)

**Theorem:** The set-theoretic "union" operator preserves context-freedom, but

the set-theoretic "intersection" and "difference"

(hence, the "complementation") operators

do not preserve context-freedom, in general.

For the union "\( \cup \)" operator:

Consider arbitrary CFLs \( L_1 \) and \( L_2 \),

with \( L_1 = L(G_i) \) for some CFGs \( G_i \) for \( i = 1, 2 \).

(we apply the notion of CFGs - easier,

rather than "PDAs"
Say, \( G_1 = (V_1, \Sigma, P_1, S_1) \)

\( G_2 = (V_2, \Sigma, P_2, S_2) \),

and without loss of generality, \( V_1 \cap V_2 = \emptyset \).

A desired CFG \( G = (V, \Sigma, P, S) \):

\[ V = V_1 \cup V_2 \cup \{S\} \quad \text{S - new variable} \]

\[ P = \{ S \rightarrow S_1 | S_2 \} \cup P_1 \cup P_2 \]

For the intersection "\( \cap \)" operator:

We show that there exist CFLs \( L_1 \) and \( L_2 \),

with \( L_1 \cap L_2 \) is not CF.

Consider \( L_1 = \{ a^i b^i c^i \mid i, j \geq 0 \} \)

\( L_2 = \{ a^i b^j c^i \mid i, j \geq 0 \} \).

[Sip12] Exercise 2.2

Both \( L_1 \) and \( L_2 \) are CF (why?)

But, \( L_1 \cap L_2 \) is not CF.

Using Pumping Lemma for Context-Freedom (studied later),

\[ \{ a^x b^i c^3 \mid x \geq 3 \} \]

This language is not CF.
For the difference "−" (and complementation) operator:

Use the above non-closure result (that "\(\cap\)" does not necessarily preserve context-freedom) and De Morgan's laws. — Exercise.

**Theorem:** The language-operators: concatenation, Kleene-star closure, and reversal, preserve context-freedom.

For the concatenation "•" operator:

Use the same denotations for the closure property of "\(\cap\):"

add: \(S \rightarrow S_1 S_2\)

For the star-closure "∗" operator:

add: \(S \rightarrow SS_1 \mid \varepsilon\)

For the reversal "r" operator:

exercise
We have used the closure property of "∪" to show the context-freedom of a "target" language (see Examples 5 and 6 in these notes).

Same framework:
Say, educated guess that \( L \) is CF.

Prepare
CFLs: \( L_1, L_2, \ldots \)  \( \vdash \) known
operations: \( \Theta_1, \Theta_2 \ldots \)  \( \vdash \) preserving CF

Use them to result in \( L \) for example:
\[
\left[ \left( L_1 \cup L_2 \right) \left. L_3 \right. \right] \cup L_2^r = L
\]
\( \vdash \) CFL

Another example:
Recall that we claimed (without proof) that \( L = \\{ a^i b^j c^i \mid i, j \geq 0 \} \) is CFL.

Using closure properties:
\( \{ a^i b^j \mid x \geq 0 \} \) is CF (known)
\( \{ c^i \mid \beta \geq 0 \} = c^* \) is regular (so, it is CF, why?)

Then, observe that \( \{ a^i b^j c^i \mid i, j \geq 0 \} \) is CF.