COMP3630 / COMP6363

week 3: **Pushdown Automata**

This Lecture Covers Chapter 6 of HMU: Pushdown Automata

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Pushdown Automata (PDA)
- Language accepted by a PDA
- Equivalence of CFGs and the languages accepted by PDAs
- Deterministic PDAs

Additional Reading: Chapter 6 of HMU.

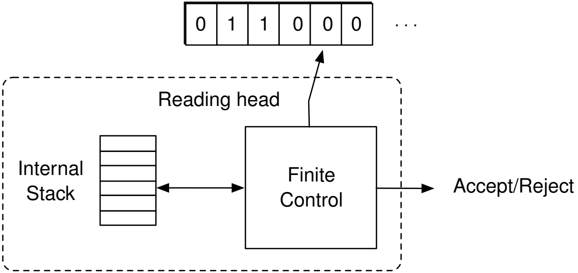
Introduction to PDAs

- PDA '=' ϵ-NFA + Stack (LIFO)
- At each instant, the PDA uses:

  (a) the input symbol, if read; (b) present state; and (c) symbol atop the stack

  to transition to a new state and alter the top of the stack.
- Once the string is read, the PDA decides to accept/reject the input string.
- Note: The PDA can only read a symbol once (i.e., the reading head is unidirectional).

PDA '=' ϵ-NFA + Stack



PDA: Formal Definition

Definition 6.1.1

A PDA is a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

- Just like in DFAs: Q is the (finite) set of internal states; Σ is the finite alphabet of input tape symbols; $q_0 \in Q$ is the (unique) start state; F is the set of final or accepting states of the PDA.
- Γ is the finite alphabet of stack symbols;
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ is a partial function such that $\delta(q, a, \gamma)$ (if defined) is a finite set of pairs $(q', \gamma') \in Q \times \Gamma^*$. **// This is non-deterministic! Why?**
- We have a partial function since we don't need to define transitions for all possible values in $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$.
- However, note that using a "normal" (i.e., non-partial) function is still correct when defining $\delta(q, a, A) = \emptyset$ for $q \in Q$, $a \in \Sigma^* \cup \{\epsilon\}$, and $A \in \Gamma$ whenever we want no transition for $\delta(q, a, A)$. (Since we don't actually have a transition in these cases when mapping to an empty set.)

PDA: Formal Definition

Definition 6.1.1

A PDA is a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

- Just like in DFAs: Q is the (finite) set of internal states; Σ is the finite alphabet of input tape symbols; $q_0 \in Q$ is the (unique) start state; F is the set of final or accepting states of the PDA.
- Γ is the finite alphabet of stack symbols;
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q\times\Gamma^*}$ is a partial function such that $\delta(q, a, \gamma)$ (if defined) is a finite set of pairs $(q', \gamma') \in Q \times \Gamma^*$. **// This is non-deterministic! Why?**
- $Z_0 \in \Gamma$ is the sole symbol atop the stack at the start; and

Input symbol (or ϵ)

Next state

The number of possible transitions

Present state

$$\delta(q, a, A) = \{(q'_i, \gamma_i) : i = 1, \ldots, \ell\}$$

Stack symbol on top

The string replacing A on top of the stack

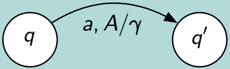
Convention: lower case symbols s, a, and b will denote input symbols; lower case symbols u, v, w, x, ... will exclusively denote strings (sequences!) of input symbols; stack symbols are indicated by upper case letters (e.g., A, B, etc); strings of stack symbols are indicated by greek letters (e.g., α, β, etc);

A PDA Example

Transition Diagram Notation

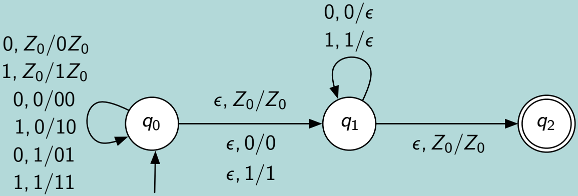
Notation: The label $a, A/\gamma$ on the edge from a state q to q' indicates a possible transition from state q to state q' by reading the symbol a when the top of the stack contains the symbol A. This stack symbol is then replaced by the string γ.

$(q', \gamma) \in \delta(q, a, A) \quad \Leftrightarrow$



(Note: q' can be q itself)

PDA that accepts $L = \{ww^R : w \in \{0.1\}^*\}$



Language Accepted by a PDA

Definitions

- The **Configuration** or **Instantaneous Description (ID)** of a PDA P is a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$ where:
  (i) q is the state of the PDA;
  (ii) w is the unread part of input string; and
  (iii) γ is the stack content from top to bottom.
- An ID tracks the trajectory/operation of the PDA as it reads the input string.
- **One-step computation** of a PDA P, denoted by $\underset{P}{\vdash}$, indicates configuration change due to one transition. Suppose $(q', \gamma) \in \delta(q, a, A)$. For $w \in \Sigma^*$, $\alpha \in \Gamma^*$,
  $(q, aw, A\alpha) \underset{P}{\vdash} (q', w, \gamma\alpha)$, [one-step computation] (What if we "read" ϵ?)
- **(multi-step) computation**, denoted by $\overset{*}{\underset{P}{\vdash}}$, indicates configuration change due to zero or any finite number of consecutive PDA transitions.
  - $ID \overset{*}{\underset{P}{\vdash}} ID'$ if there are k IDs $ID_1, \ldots, ID_k$ (for some $k \geq 1$) such that:
    (i) $ID_1 = ID$ and $ID_k = ID'$, and
    (ii) for each $i = 1, \ldots, k-1$, $ID_i \underset{P}{\vdash} ID_{i+1}$.

Beware of IDs!

Lemma 6.2.1

Let PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be given. Let $q, q' \in Q$, $x, y, w \in \Sigma^*$, and $\alpha, \beta, \gamma \in \Sigma^*$. Then the following hold.

$$(q, x, \alpha) \overset{*}{\underset{P}{\vdash}} (q', y, \beta) \quad \Leftrightarrow \quad (q, xw, \alpha) \overset{*}{\underset{P}{\vdash}} (q', yw, \beta) \tag{1}$$

$$(q, x, \alpha) \overset{*}{\underset{P}{\vdash}} (q', y, \beta) \quad \Rightarrow \quad (q, x, \alpha\gamma) \overset{*}{\underset{P}{\vdash}} (q', y, \beta\gamma) \tag{2}$$

Proof Idea

- (1) What hasn't been read cannot affect configuration changes
- (2) PDA transitions cannot occur on empty stack. So the $(q, x, \alpha) \overset{*}{\underset{P}{\vdash}} (q', y, \beta)$ must not access any location beneath the last symbol of x.

Remark: Think about why is the reverse implication of (2) not true.

Language Accepted by PDAs

Definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, the language accepted by P by final states is

$$L(P) = \left\{ w \in \Sigma^* : (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \alpha) \text{ for some } q \in F, \alpha \in \Gamma^* \right\}.$$

The language accepted by P by empty(ing its) stack is

$$N(P) = \left\{ w \in \Sigma^* : (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \epsilon) \text{ for some } q \in Q \right\}.$$

Can $L(P)$ and $N(P)$ be different?

- Pick a DFA A such that $L(A) \neq \emptyset$. Convert it to a PDA P by pushing each symbol that is read onto the stack, increasing the stack size each time a symbol is read. For the derived PDA, $L(P) = L(A)$. However, $N(P) = \emptyset$.

- Which of the two definitions accepts 'more' languages?

Equivalence of the two Notions of Language Acceptance

Motivation: If true, why would that result be useful?

Because then we are free which criterion we use!

- Sometimes it's easier to construct a PDA P and consider $L(P)$,
- sometimes it's easier to construct a PDA P' and consider $N(P')$.

Assuming accepting by final state would be exactly as powerful as accepting by empty stack, what would have to hold?

Let P be a PDA.

- Then, there is a PDA P', such that $L(P) = N(P')$.

$\rightarrow$ Shows that accepting by empty stack is at least as powerful as accepting by final state.

- Then, there is a PDA P'', such that $N(P) = L(P'')$.

$\rightarrow$ Shows that accepting by final state is at least as powerful as accepting by empty stack.

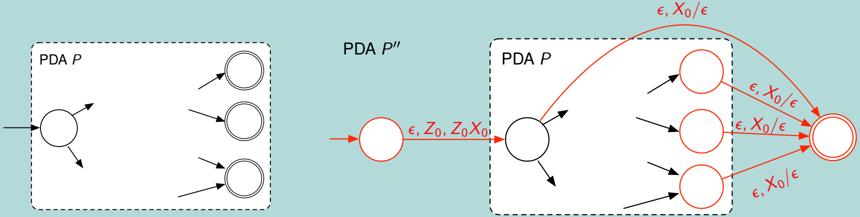
Taken both together we have, if true, that both criteria are equally expressive.

Equivalence of the Two Notions of Language Acceptance

Theorem 6.2.2

Given PDA P, there exist PDAs P' and P'' such that $L(P) = N(P')$ and $N(P) = L(P'')$.

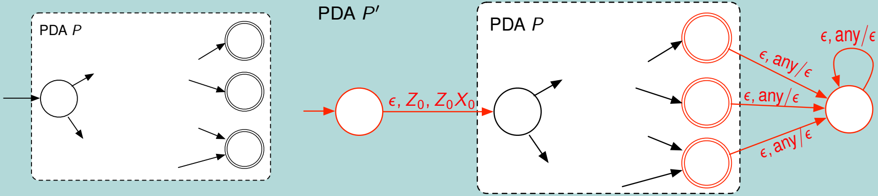
Proof of Existence of P'' such that $N(P) = L(P'')$ (i.e., P accepts by empty stack)



- Introduce a new start state and a new final state with the transitions as indicated.
- The start state first replaces the stack symbol Z_0 by Z_0X_0.
- If and only if $w \in N(P)$ will the computation by P end with the stack containing precisely X_0.
- The PDA P'' then transitions to the final state popping X_0. Hence, $N(P) = L(P'')$.

Equivalence of the two Notions of Language Acceptance

Proof of Existence of P' such that $L(P) = N(P')$ (i.e., P accepts by final states)



- Introduce a new start state and a special state with the transitions as indicated.
- The start state first replaces the stack symbol Z_0 by Z_0X_0.
- If and only if $w \in L(P)$ will the computation by P end in a final state with the stack containing (at least) X_0. **Question:** Why is this required?
- The PDA P' then transitions to the special state and starts to pop stack symbols one at time until the stack is empty. Hence, $L(P) = N(P')$.

CFGs and PDAs

Is every CFL accepted by some PDA and vice versa?

Theorem 6.3.1

For every CFG G, there exists a PDA P such that $N(P) = L(G)$.

Proof

- Let $G = (V, T, \mathcal{P}, S)$ be given.
- Construct PDA $P = (\{q_0\}, T, V \cup T, \delta, S, \{q_0\})$ with δ defined by

$$[\text{Type 1}] \;\; \delta(q_0, a, a) = \{(q_0, \epsilon)\}, \quad \text{whenever } a \in \Sigma,$$

$$[\text{Type 2}] \;\; \delta(q_0, \epsilon, A) = \{(q_0, \alpha) : A \longrightarrow \alpha \text{ is a production rule in } \mathcal{P}\}.$$

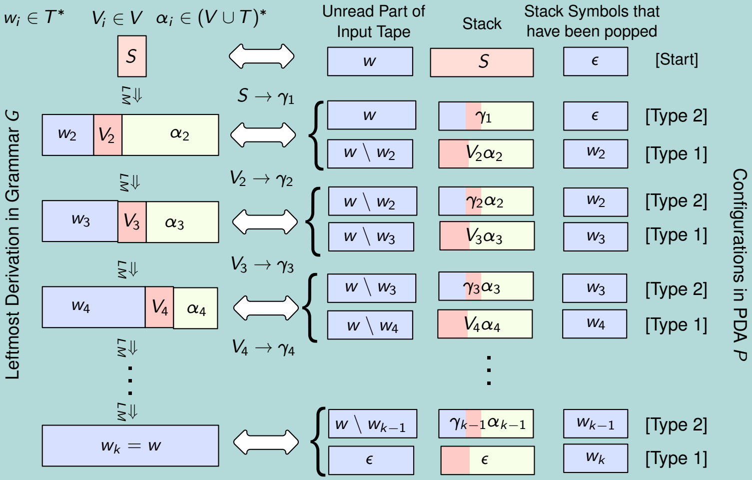
- This PDA mimics all possible leftmost derivations.
- We use induction to show that $L(G) = N(P)$

Remark: That's a great (fun!) exercise to practice by yourself! Just take any grammar.

CFGs and PDAs (see appendix slide!)

Proof of 1-1 Correspondence between PDA Moves and Leftmost Derivations

Suppose $w \in T^*$ and $S \overset{*}{\underset{LM}{\Rightarrow}} w$. $x \setminus y$:=suffix of y in x.



CFGs and PDAs

Theorem 6.3.2

For every PDA P, there exists a CFG G such that $L(G) = N(P)$.

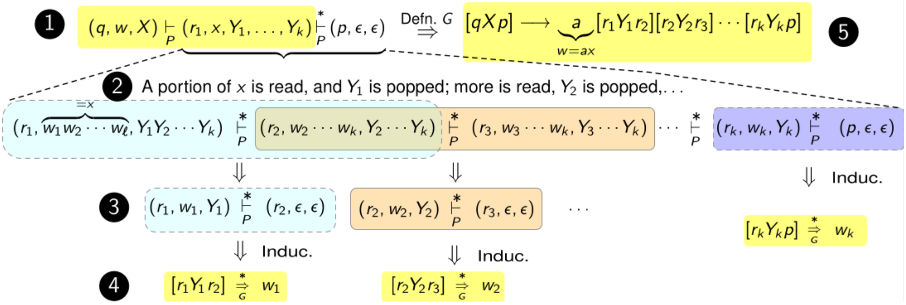
Proof

- Given $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, we define $G = (V, T, \mathcal{P}, S)$ as follows.
  - $T = \Sigma$;
  - $V = \{S\} \cup \{[pXq] : p, q \in Q, X \in \Gamma\}$;
    Interpretation: Each variable $[pXq]$ will generate a terminal string w iff upon reading w (in finite steps) P moves from state p to q popping X from the stack.
  - $\mathcal{P}$ contains only the following rules:
    - $S \longrightarrow [q_0 Z_0 p]$ for all $p \in Q$.
    - Suppose that $(r, X_1 \cdots X_\ell) \in \delta(q, a, X)$. Then, for any states $p_1, \ldots, p_\ell \in Q$,
      $[qXp_\ell] \longrightarrow a[rX_1p_1][p_1X_2p_2] \cdots [p_{\ell-1}X_\ell p_\ell]$. (So these are $\mathcal{O}(|Q|^\ell)$ rules!)
      Note that if $(r, \epsilon) \in \delta(q, a, X)$, then $[qXr] \longrightarrow a$.
- We will show $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w \Leftrightarrow (q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$. We will choose $q = q_0$, $X = Z_0$.
- **Remark:** Not 100% clear? Translate a small PDA! (One with few states.)

CFGs and PDAs (see appendix slide!)

Proof of $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon) \Rightarrow [qXp] \overset{*}{\underset{G}{\Rightarrow}} w$. (Induction on # of steps of computation)

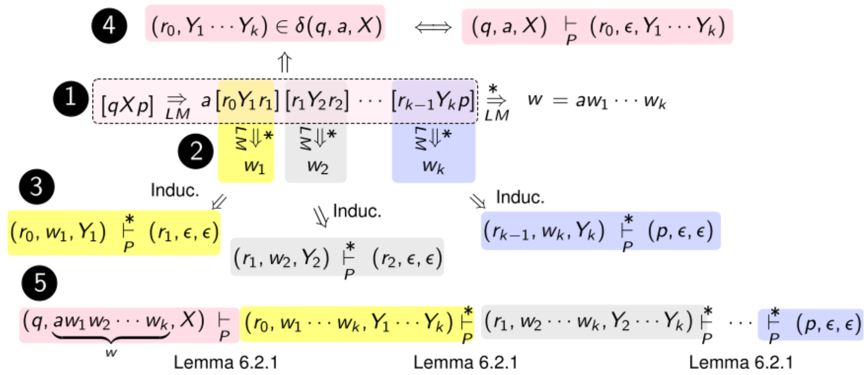
- Basis: Let $w \in N(P)$. Suppose there is a one-step computation $(q, w, X) \underset{P}{\vdash} (p, \epsilon, \epsilon)$. Then, $w \in \Sigma \cup \{\epsilon\}$. Since $(p, \epsilon) \in \delta(q, w, X)$, $[qXp] \longrightarrow w$ is a production rule.
- Induction: Let $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$. Let a be read in the first step of the computation, and let $w = ax$. Then the following argument completes the proof.



CFGs and PDAs (see appendix slide!)

Proof of $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w \Rightarrow (q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$. (Induction on # of steps of derivation)

- Basis: Let $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w$ in one step. Then, $[qXp] \longrightarrow w$ must be a production rule. Consequently, $(p, \epsilon) \in (q, w, X)$ and $(q, w, X) \underset{P}{\vdash} (p, \epsilon, \epsilon)$.
- Induction: Let $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w$.



Deterministic PDAs (DPDAs)

- PDAs are (by definition) non-deterministic.
- Deterministic PDAs are defined to have **no choice** in their transitions.

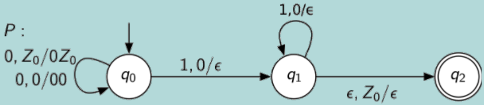
Definition

A DPDA P is a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ such that for each $q \in Q$ and $X \in \Gamma$,

- $|\delta(q, a, X)| \leq 1$ *for any* $a \in \Sigma \cup \{\epsilon\}$,
  i.e., a configuration cannot transition to more than one configuration.
- $|\delta(q, a, X)| = 1$ *for some* $a \in \Sigma \Rightarrow \delta(q, \epsilon, X) = \emptyset$,
  i.e., both reading or not reading (a tape symbol) cannot be options.

- DPDAs have a computation power that is strictly better than DFAs

Example: $L(P) = N(P) = \{0^n 1^n : n \geq 1\}$



- DPDAs have a computation power that is strictly worse that PDAs.
  (We will discuss this later)

Languages Accepted by DPDAs

- The two notions of acceptance (empty stack and final state) are **not equivalent** in the case of DPDAs.
- There are languages L such that $L = L(P)$ for some DPDA P, but there exists no DPDA P' such that $L = N(P')$.

Theorem 6.4.1

Every regular language L is the language accepted by some DPDA accepting by final states.

Proof

Simply view the DFA accepting L as a DPDA (with the stack always containing Z_0).

Languages Accepted by DPDAs

- The two notions of acceptance (empty stack and final state) are **not equivalent** in the case of DPDAs.
- There are languages L such that $L = L(P)$ for some DPDA P, but there exists no DPDA P' such that $L = N(P')$.

Theorem 6.4.2

Not every regular language L is the language accepted by some DPDA accepting by empty stack.

Proof

- Let $L = \{0\}^*$ (which is regular). It cannot equal $N(P)$ for any DPDA P.
- Suppose DPDA P accepts L by emptying its stack. Since 0 is accepted, P eventually reaches a configuration (p, ϵ, ϵ) for some state p.
- Now, suppose that P is fed with the input 00. Since P is **deterministic**, P reads a 0 and eventually has to get to $(p, 0, \epsilon)$. However, it hangs at this configuration and cannot read any further input symbols. Hence, P cannot accept 00.

Languages Accepted by DPDAs

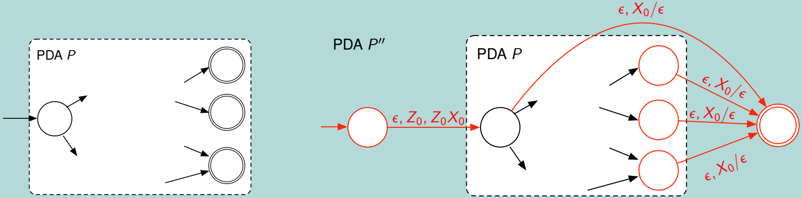
- A language L is said to have the **prefix property** if no two distinct strings in the L are prefixes of one another. (So no prefix of any $w \in L$ is in the language!)

Theorem 6.4.3

A language L is the language for some DPDA P accepting by empty stack, $L = N(P)$ iff L has the prefix property and $L = L(P'')$ for some DPDA P''.

Proof $\Rightarrow$

$\Rightarrow$ Let $L = N(P)$ for some DPDA P. Let w, ww' be in L with $w' \neq \epsilon$. Then $(q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$ for some $p \in Q$. The DPDA hangs at this state since the stack is empty. Hence, it cannot accept ww'. The fact that $L = L(P'')$ for some DPDA P'' follows from Theorem 6.2.2 since the construction yields a **deterministic** PDA.

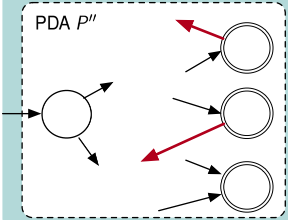


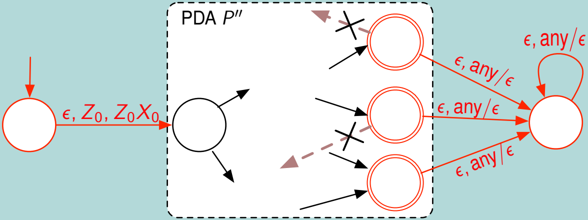
Languages Accepted by DPDAs

Proof $\Leftarrow$

$\Leftarrow$ Let DPDA P'' be given. Let $w \in L(P'')$, $(q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \gamma)$ for some $p \in F$, and $\gamma \in \Gamma$. Since $L(P'')$ satisfies the prefix property, the PDA cannot enter any final state before reading all of w.

- Then we can delete all transitions from final states; this does not alter $L(P'')$.
- Then, the construction of Theorem 6.2.2 yields a **deterministic** PDA P' such that $N(P') = L(P'') = L$.





DPDAs and Unambiguous Grammars

Theorem 6.4.4

If $L = N(P)$ for some DPDA P, then L has an unambiguous CFG.

Proof

- Let G be the CFG constructed in Theorem 6.3.2.
- Suppose G is ambiguous. Then, some $w \in L$ has 2 leftmost derivations.
- However, each derivation corresponds to a unique trajectory of configurations in P that also accepts w by emptying the stack.
- Since P is deterministic, the trajectories, and hence, the derivations have to be identical. Hence, G is unambiguous.

DPDAs and unambiguous Grammars

Theorem 6.4.5

If $L = L(P)$ for some DPDA P, then L has an unambiguous CFG.

Proof

- Let $\$$ be a symbol not in the alphabet of L.
- Consider $L' = \{w\$: w \in L\}$. Then, L' has the prefix property.
- By Theorem 6.4.3, there must exist a DPDA P' such that $L' = N(P')$.
- By Theorem 6.4.4, L' has an unambiguous CFG $G' = (V, T, \mathcal{P}, S)$.
- Define CFG $G = (V \cup \{\$\}, T \setminus \{\$\}, \mathcal{P} \cup \{\$ \longrightarrow \epsilon\}, S)$. G generates L.
- Proof by contradiction: Suppose G is ambiguous.
- Then, some $w \in L$ has 2 leftmost derivations.
- The last steps in the two leftmost derivations of w **must** use the production $\$ \longrightarrow \epsilon$. (So this can't cause ambiguity.)
- Thus, the portions of the two leftmost derivations without the last production step (which corresponds to G') must allow two different LM derivations.
- Hence, G' must be ambiguous, contradiction! Hence, G must unambiguous.

Explanation for Slide 13

- $\Rightarrow$ Suppose we want to show that if there is a derivation in G generating w, then there is a trajectory in P accepting w. To do that let $S \overset{*}{\underset{LM}{\Rightarrow}} w$.
- Then there must be a LM derivation as in the left column. In each step of the leftmost derivation, a part of the string w is uncovered, and the uncovered part is succeeded by a non-terminal.
- Let after $i = 1, \ldots, k-2$ production uses: (1) the prefix w_{i+1} of w be uncovered (shown in purple); (2) the leftmost non-terminal be V_{i+1} (shown in orange); and (3) is the string to the right of the leftmost non-terminal α_{i+1} that contains both terminal and non-terminal symbols (shown in beige).
- After the k^{th} production rule, we have derived $w_k = w$.
- Now suppose $S \to \gamma_1 = w_2 V_2 \alpha_2$, $V_2 \to \gamma_2$, ..., $V_{k-1} \to \gamma_{k-1}$ be the $k-1$ production rules used in the leftmost derivation.
- Now let us show that a trajectory exists for P using the above information we have laid out.
- Since there is only one state for the PDA, the right part of the slide presents only the portion of tape yet to be read, and the stack contents; additionally, it also gives the **string of terminals** that has been popped up until any point in time.
- Initially, the tape contains w, the stack contains S, and ϵ has been popped thus far.

Explanation for Slide 13 (Continued)

- Now since $S \to \gamma_1$ is a valid production rule, by the definition of P, there is a Type-22 transition that reads nothing from the input tape, reads S from the stack and pushes $\gamma_1 := w_2 V_2 \alpha_2$ onto the stack. Thus, the following one-step computation is valid

$$(q_0, w, S) \underset{P}{\vdash} (q_0, w, w_2 V_2 \alpha_2).$$

- Note that w_1 is the prefix of w uncovered after the first step of the derivation, and hence matches the first few symbols of w. Then, it is clear that one can perform $|w|$ Type-1 transitions that pop each of these symbols from the stack. Thus, after popping $|w_1|$ symbols, we see that:

$$(q_0, w, S) \underset{P}{\vdash} (q_0, w, w_2 V_2 \alpha_2) \overset{*}{\underset{P}{\vdash}} (q_0, w \setminus w_2, V_2 \alpha_2),$$

  where we let $w \setminus w_2$ to denote the suffix of w_2 in w.

- Now, note that $V_2 \to \gamma_2$ is a valid production rule; hence, there is a valid one-step computation from $(q_0, w \setminus w_2, V_2 \alpha_2)$ that uses the corresponding Type-2 transition. The resultant configuration change will then be

$$(q_0, w, S) \underset{P}{\vdash} (q_0, w, w_2 V_2 \alpha_2) \overset{*}{\underset{P}{\vdash}} (q_0, w \setminus w_2, V_2 \alpha_2) \underset{P}{\vdash} (q_0, w \setminus w_2, (w_3 \setminus w_2) V_3 \alpha_3),$$

  where $(w_3 \setminus w_2) V_3 \alpha_3 := \gamma_2 \alpha_2$.

Explanation for Slide 13 (Continued)

- Again, we see that a portion of the top of the stack contains $w \setminus w_2$, which matches the initial segment of the input tape. Then there is a valid multi-step computation involving $|w_3 \setminus w_2|$ Type-1 transitions that pops $w_3 \setminus w_2$. The resultant configuration will then be $q_0, w \setminus w_3, V_3\alpha_3)$.
- Now, this proceeds until all of w is exhausted (read) from the input tape, and the configuration at the end will be $(q_0, \epsilon, \epsilon)$. Since the stack is empty, the original string w will be accepted.
- $\Leftarrow$ The direction that a trajectory accepting w in P implies a derivation of w in G is simply arguing the above in the reverse direction using the facts that:
  - a trajectory for accepting w in P must consist only of Type-1 and Type-2 transitions, and each Type-2 transition corresponds to a unique production in G.
  - The argument is literally the same as above except that we now uncover the production rule from the corresponding Type-2 transition.

Explanation for Slide 15

Inductive proof for $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon) \Rightarrow [qXp] \overset{*}{\underset{G}{\Rightarrow}} w$ based on length of computation.

- Basis: Let $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$ be a one-step computation. Thus, w has to be an input symbol or ϵ. Then, by definition of one-step computation it **must** be true that $(p, \epsilon) \in \delta(q, w, x)$, where $X = xX'$. Then, by the construction of G, we have $[qXp] \rightarrow w$ (see Slide 12 for the construction), and hence $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w$.
- Induction: $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon)$ in say $k > 1$ steps. Let us assume that the in the first step of the computation, the symbol a is read from the input tape (or $a = \epsilon$). Let $w = ax$. Let's break the k-step computation to a single step followed by a $k-1$-step computation as detained in 1 (encircled in black). Let r_1 be the state of the PDA after the first step and let X be popped and $Y_1 \cdots Y_k$ be pushed onto the stack after the first step/transition/move.
- Now, the claim is that the $k-1$ step portion of the computation can be expanded into the sequence of computations as given in 2 (encircled in black). The reasoning is as follows. The ID $(r_1, x, Y_1 \cdots Y_k)$ eventually changes to (p, ϵ, ϵ). There must be a finite number of moves after which the effective stack change is the popping of Y_1, i.e., after a finite number of steps Y_2 is at the top **for the very first time**. The steps until then could have popped Y_1, pushed a string, and then popped it eventually to reveal Y_2 at the top.

Explanation for Slide 15 (Continued)

- Let w_1 be the portion of the input tape read and r_2 be the state pf the PDA when this intermediate ID where Y_2 is at the top of the stack (i.e., the stack contains $Y_2 \cdots, Y_k$) is attained. Thus,

$$(r, x, Y_1 \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_2, x \setminus w_1, Y_2, \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon),$$

  where again we let $w \setminus w_1$ to be the suffix of w_1 in w.
- By a similar argument, after reading another segment, say w_2, of the input tape and reaching (some) state r_3, the top of the stack of the PDA contains Y_3 **for the very first time**. Thus,

$$(r, x, Y_1 \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_2, x \setminus w_1, Y_2, \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_3, x \setminus (w_1 w_2), Y_3, \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon).$$

- Proceeding inductively, we see that 2 (encircled in black) holds. Note that x is then equal to the concatenation of the w_i's, i.e., $x = w_1 \cdots w_k$.
- Now focus on the computation within the blue block in 2. In no intermediate ID of the computation is Y_2 at the top of the stack (since $(r_2, x \setminus w_1, Y_2, \cdots Y_k)$ is the very first time Y_2 is at the top of the stack). Thus, the stack contents $Y_2 \cdots Y_k$ are never visited in this first set of moves, and hence, we see that

$$(r_1, x, Y_1 \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_2, x \setminus w_1, Y_2, \cdots Y_k) \Rightarrow (r_1, w_1, Y_1) \overset{*}{\underset{P}{\vdash}} (r_2, \epsilon, \epsilon). \tag{3}$$

Explanation for Slide 15 (Continued)

- Similarly, we see that the in portion of the computation in orange, no intermediate ID of the computation has Y_3 at the top of the stack (since $(r_3, x \setminus (w_1 w_2), Y_3, \cdots Y_k)$ is the very first time Y_3 is at the top of the stack). Hence,

$$(r_2, x \setminus w_2 \cdots w_k, Y_2, \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_3, w_2 \cdots w_k, Y_3 \cdots Y_k) \Rightarrow (r_2, w_2, Y_2) \overset{*}{\underset{P}{\vdash}} (r_3, \epsilon, \epsilon). \quad (4)$$

- We can proceed inductively to argue that $(r_i, w_i, Y_i) \overset{*}{\underset{P}{\vdash}} (r_{i+1}, \epsilon, \epsilon)$ for $i = 1, \ldots, k-1$.
- Now each of these derivations $(r_i, w_i, Y_i) \overset{*}{\underset{P}{\vdash}} (r_{i+1}, \epsilon, \epsilon)$ for $i = 1, \ldots, k-1$ contain $k-1$ or less steps, because the number of steps they contain is at least one-less than the number of steps in the computation in 1 (encircled in black).
- Consequently, by the induction hypothesis, we have $[r_i Y_i r_{i+1}] \overset{*}{\underset{G}{\Rightarrow}} w_i$, $i = 1, \ldots, k-1$.

  By the very same argument $[r_k Y_k p] \overset{*}{\underset{G}{\Rightarrow}} w_k$.
- Now focus on the yellow box at the top, the first one-step computation guarantees that there exists a production rule

$$[qXp] \rightarrow a[r_1 Y_1 r_2][r_2 Y_2 r_3] \cdots [r_{k-1} Y_{k-1} r_k][r_k Y_k p]. \quad (5)$$

  Now combining the above production with the known derivations in 4 (encircled in black), we see that $[qXp] \overset{*}{\underset{G}{\Rightarrow}} a w_1 \cdots w_k = ax = w$.

Explanation for Slide 16

Inductive proof for $(q, w, X) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \epsilon) \Leftarrow [qXp] \overset{*}{\underset{G}{\Rightarrow}} w$ based on length of leftmost derivation.

- Basis: $[qXp] \overset{*}{\underset{LM}{\Rightarrow}} w$ be a one-step derivation. This can be possible only if $(p, \epsilon) \in (q, w, X)$, which then means $(q, w, X) \underset{P}{\vdash} (p, \epsilon, \epsilon)$.
- Induction: Let $[qXp] \overset{*}{\underset{G}{\Rightarrow}} w$ in $k > 1$ steps. As in the previous direction, let us split the leftmost derivation into the first step and then rest.
- The first step must involve the application of some production rule, say, $[qXp] \to a[r_0Y_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kp]$.
- By 1 (encircled in 1) each non-terminal $[r_{i-1}Y_ir_i]$ $i = 1, \ldots, k$ must derive (via a leftmost derivation) a segment of w, say w_i in $k-1$ steps or less. [w_i is the yield of the parse subtree in the parse tree of $[qXp]$ with yield w, and the depth of the subtree is at most 1 less than the depth of the parse tree of $[qXp]$.).
- Hence, $[r_{i-1}Y_ir_i] \overset{*}{\underset{LM}{\Rightarrow}} w_i$ for $i = 1, \ldots, k$ in $k-1$ steps or less (I've set $r_k = p$ here).

  By induction hypothesis, then $(r_{i-1}, w_i, Y_i) \overset{*}{\underset{P}{\vdash}} (r_i, \epsilon, \epsilon)$.
- Then by Lemma 6.2.1, $(r_{i-1}, w_i \cdots w_k, Y_i \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_i, w_{i+1} \cdots w_k, Y_{i+1} \cdots Y_k)$. Thus,

$$(q, w, X) \underset{P}{\vdash} (r_0, w_1 \cdots w_k, Y_1 \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_1, w_2 \cdots w_k, Y_2 \cdots Y_k) \overset{*}{\underset{P}{\vdash}} (r_k, \epsilon, \epsilon) = (p, \epsilon, \epsilon).$$