

Pushdown Automata

Definition

Informal: PDA is a ϵ -NFA with a stack

Formal: A PDA is a 7-tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

- Q : a finite set of states
- Σ : a finite set of input symbols
- Γ : a finite stack alphabet
- δ : the transition function
- q_0 : the start state
- Z_0 : the start symbol in stack
- F : the set of accepting states

考虑 PDA 的迁移函数 δ , 其接受三个参数 (q, a, X) , 其中

1. q 是 Q 中的状态
2. a 是 Σ 中的符号或是 ϵ
3. X 是 Γ 中的符号

其输出为有序对 (p, γ) 的集合, 其中 p 是一个新状态, 而 γ 是一个 stack symbol 的 string, 用于**取代** X , 当 $\gamma = \epsilon$, 表示弹出 X , 而当 $\gamma = X$ 则不变, 若 $\gamma = YZ$, 则将 X 替换为 Z , 再将 Y 压入。

需注意 δ 的输出是一个集合, 即 PDA 可以从中选出一个对, 但是不能迁移到某个对的状态但是将栈中内容替换为另一个对的内容

PDA 也可以像 FA 一样使用图表示, 其中

- 节点代表 PDA 的 states
- 两个圈的节点表示接收状态
- 一条标号为 $a, X/\alpha$ 的从 q 到 p 的边表示 $(p, \alpha) \in \delta(q, a, X)$

Instantaneous Descriptions

不同于 FA, 描述状态机的运行的只有状态, PDA 的运行包括了状态与栈的内容, 故定义 Instantaneous Description 为一个 3-tuple (q, w, γ)

- q 是一个状态
- w 是余下的输入
- γ 是栈的内容

PDA 的 ID 可以完全表示其运行时某一个时刻的格局。

为了描述 ID 随着 PDA 的运行而发生的转换，定义 \vdash 。若 $\delta(q, a, X)$ 包含 (p, α) ，则对于任意 $w \in \Sigma^*, \beta \in \Gamma^*$ 有

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

显然，余下的输入和栈中剩余的部分不影响 ID 的转换

\vdash 表示 PDA 的一步动作，可以定义 \vdash^* 表示 0 步或多步动作

Basis. $I \vdash^* I$ for any ID I

Induction. If $I \vdash^* K, K \vdash J$, then $I \vdash^* J$

对于 ID 的 transition，有

If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, and $(q, x, \alpha) \vdash^* (p, y, \beta)$, then for any strings $w \in \Sigma^*$ and $\gamma \in \Gamma^*$, it is also true that

$$(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma)$$

Proof. 对 ID 的转换步数归纳。由于 PDA 未读入的输入不会影响其行为，故在输入后添加后缀 w 不会改变其行为。同样的，在原本的转换中，没有用到 α 以外的内容，故在栈底加入 γ 也不会改变其行为

同样的，有

If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, and

$$(q, xw, \alpha) \vdash^* (p, yw, \beta)$$

then $(q, x, \alpha) \vdash^* (p, y, \beta)$

需要注意的是未读入的输入不会对 PDA 产生影响，故可以将其共同的后缀去除，但是不能将栈共同的后缀去除，考虑转换前栈中为 $\alpha\gamma$ ，转换后为 $\beta\gamma$ ，在转换的过程中可能弹出了 γ 中的符号，然后之后又将其压入，若将 γ 去除则转换不能成立

添加或删除输入串的后缀不影响转换，但只能添加栈底内容而不能删除

The Language of a PDA

有两种方式可以让一个 PDA 接受输入

- acceptance by final state: 当输入结束时，PDA 状态停止在接收状态

- acceptance by empty stack: 当输入结束时, PDA 栈空

可以得出这两种方式是等价的, 即给定一个语言 L , 存在一个 PDA 以 acceptance by final state 的方式定义 L , 也存在一个 PDA 以 acceptance by empty stack 的方式定义 L 。但对于同一个 PDA 而言, 以两种方式定义的语言一般是不同的

Acceptance by Final State

设 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, 则 language accepted by P by final state 定义为

$$L(P) = \{w : (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha), q \in F\}$$

即在输入结束后, PDA 到达接收状态, 而此时栈中可以是任意内容

Acceptance by Empty Stack

设 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, 则 language accepted by P by empty stack 定义为

$$N(P) = \{w : (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$$

即输入结束后栈清空则视为接受, PDA 此时可以处于任意状态

From Empty Stack to Final State

事实上, $L(P), N(P)$ 定义的语言集合是相同的, 即 CFL。

If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, F)$, then there is a PDA P_F such that $L = L(P_F)$

Proof.

引入一个新的符号 $X_0 \notin \Gamma$, X_0 是 P_F 的开始符号, 其思想在于当 P_F 栈中仅有 X_0 时, 可以得知对于相同的输入, P_N 会清空其栈, 即接受该输入。

同样需要引入一个新的开始符号 p_0 , 用于将 Z_0 压入栈中, 在此之后 P_F 将模拟 P_N 的运行, 直至栈顶为 X_0 , 此时代表 P_N 对于同样的输入清空了栈, 即接受该输入。引入一个新的状态 p_f 用于在栈顶为 X_0 且输入结束时转移到该状态。 p_f 是 P_F 的唯一接受状态

P_F 的精确定义为

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

其中 δ_F 的定义为

- $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$, 即开始时 P_F 将 Z_0 压入栈中, 同时转移到 P_N 的开始状态
- 对于所有 $q \in Q, a \in \Sigma, a = \epsilon, Y \in \Gamma$, 有 $\delta_N(q, a, Y) \subseteq \delta_F(q, a, Y)$, 即 P_F 模拟 P_N 的运行
- 对于所有 $q \in Q$, 有 $(p_f, \epsilon) \in \delta_F(q, \epsilon, X_0)$

只需证明

$$w \in L(P_F) \iff w \in N(P_N)$$

(\Leftarrow): 已知 $(q_0, w, Z_0) \vdash_{P_N}^* (q, \epsilon, \epsilon)$, 则可以在栈底插入符号 X_0 , 即

$$(q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \epsilon, X_0)$$

而由于 P_F 模拟了所有 P_N 的运行, 故有

$$(q_0, w, Z_0 X_0) \vdash_{P_F}^* (q, \epsilon, X_0)$$

根据 δ_F 有

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0 X_0) \vdash_{P_F}^* (q, \epsilon, X_0) \vdash_{P_F} (p_f, \epsilon, \epsilon)$$

即 $(p_0, w, X_0) \vdash_{P_F}^* (p_f, \epsilon, \epsilon)$, 可得 $w \in L(P_F)$

(\Rightarrow): 显然, 对于 P_F , 其第一步转换必然是 $(p_0, \epsilon, X_0) \vdash (q_0, Z_0 X_0)$, 因为这是开始状态唯一的转换函数, 且若其接受 string, 其最后一步转换必然是 $(q, \epsilon, X_0) \vdash (p_f, \epsilon)$, 因为 p_f 是唯一的接受状态, 且所有输出包含 p_f 的转换函数其输入都要求栈中仅有 X_0 , 而显然 X_0 仅会出现在栈底

这样任意接受 w 的转换都有如下形式

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0 X_0) \vdash_{P_F}^* (q, \epsilon, X_0) \vdash_{P_F} (p_f, \epsilon, \epsilon)$$

且除去开始的一步和结尾的一步, 中间的转换都是 P_N 的转换, 且转换过程中 X_0 都在栈的底部 (X_0 若出现在栈顶则下一步运行就会结束)。故可将 X_0 去掉, 得到

$$(q_0, w, Z_0) \vdash_{P_N}^* (q, \epsilon, \epsilon)$$

即 $w \in N(P_N)$

From Final State to Empty State

If $L = L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$, then there is a PDA P_N such that $L = N(P_N)$

Proof. 基本思想同样是用 P_N 去模拟 P_F 的运行, 每当 P_F 接受输入, P_N 便将栈清空, 接受同样的输入。

为了防止在 P_F 运行过程中还未接受就将栈清空使得 P_N 提前接受, 引入新的符号 $X_0 \notin \Gamma$, 在开始时压入栈, 这样无论如何模拟 P_F 的运行都不会清空栈, 直至其到达接受状态, 再将栈中的剩余符号连同 X_0 一起弹出

P_N 的精确定义为

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, F)$$

其中 δ_N 的定义为

- $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$, 开始时 P_N 将 Z_0 压入栈中, 转到 P_F 的开始状态
- 对于所有 $q \in Q, a \in \Sigma, a = \epsilon, Y \in \Gamma$, 有 $\delta_F(q, a, Y) \subseteq \delta_N(q, a, Y)$, 即 P_N 模拟 P_F 的运行
- 对于所有 $q \in F, Y \in \Gamma, Y = X_0$ 有 $(p, \epsilon) \in \delta_N(q, \epsilon, Y)$, 即当 P_F 接受输入时, P_N 开始将自己的栈清空, 不再消耗输入
- 对于所有的 $Y \in \Gamma, Y = X_0$ 有 $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$, 即当进入状态 p 后, P_N 将栈中所有的符号弹出直至栈为空, 然后接受该输入

只需证明

$$w \in N(P_N) \iff w \in L(P_F)$$

(\Leftarrow): 已知 $(q_0, w, Z_0) \vdash_{P_F}^* (q, \epsilon, \alpha), q \in F$, 显然每一步 P_F 的转换都是 P_N 的转换, 且可以将 X_0 插入栈底, 故有

$$(q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \epsilon, \alpha X_0)$$

则根据 δ_N 的定义有

$$(p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \epsilon, \alpha X_0) \vdash_{P_N}^* (p, \epsilon, \epsilon)$$

于是有 $w \in N(P_N)$

(\Rightarrow): 考虑 P_N 清空其栈的唯一条件是进入状态 p , 因为 X_0 在栈底, 且 P_F 的栈符号集 Γ 中没有 X_0 。而 P_N 进入状态 p 的条件是 P_F 达到某个接受状态 q , 而根据 δ_N 的定义, P_N 的第一个动作一定是 $(p_0, w, X_0) \vdash (q_0, w, Z_0 X_0)$

故任意接受 w 的转换都有如下的形式

$$(p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0 X_0) \vdash_{P_N}^* (q, \epsilon, \alpha X_0) \vdash_{P_N}^* (p, \epsilon, \epsilon)$$

而所有 $(q_0, w, Z_0 X_0) \xrightarrow{P_N^*} (q, \epsilon, \alpha X_0)$ 之间的转换都是 P_F 的转换，故 P_F 也可以有同样的转换，且栈中没有符号 X_0 （因为 X_0 在栈底且不会被 P_F 的转换影响），即

$$(q_0, w, Z_0) \xrightarrow{P_F^*} (q, \epsilon, \alpha), q \in F$$

即 $w \in L(P_F)$

Equivalence of PDA and CFG

正如 CFG 一样，PDA 定义的语言正好是 CFL（无论是 acceptance by empty stack 还是 acceptance by final state），这说明 PDA 和 CFG 表达能力是等价的

From Grammars to Pushdown Automata

基本思路为，给出一个 CFG，可以构造一个 PDA 模拟其最左推导过程。而最左推导的每一步是由最左句型来完全表示的。

任何非 terminal string 的最左句型都可以写为 $x A \alpha$ ，其中 A 是最左的 variable。称 $A \alpha$ 为句型的 **tail**，对于一个仅包含 terminal 的最左句型，其 tail 为 ϵ

使用 PDA 模拟最左推导的思路在于令最左句型 $x A \alpha$ 的 tail $A \alpha$ 出现在栈中，而 x 是已经消耗的输入，考虑输入字符串 $w = xy$ ， y 为剩余的输入，则 PDA 的 ID $(q, y, A \alpha)$ 可以代表最左句型 $x A \alpha$ 。假设下一步用于展开 A 的产生式为 $A \rightarrow \beta$ ，则 PDA 的动作是用 β 替换栈顶的 A ，ID 转换为 $(q, y, \beta \alpha)$ ，该 PDA 中只有一个状态 q ，而现在的 ID 可能不能代表一个最左句型，因为 β 可能有 terminal 作为前缀，因此需要消耗输入，同时从栈中移除对应的 terminal，直至有一个 variable 出现在了栈顶

若最终成功模拟了最左推导，则所有的 variable 都被展开，所有的 terminal 都与输入中的 symbol 匹配，最终栈为空，接受该输入

令 $G = (V, T, P, S)$ 为 CFG，则可以构造 PDA P ，有 $L(G) = N(P)$

$$P = (\{q\}, T, V \cup T, \delta, q, S, F)$$

其中 δ 的定义为

- 对于每个 variable A

$$\delta(q, \epsilon, A) = \{(q, \beta) : A \rightarrow \beta \in P\}$$

- 对于每个 terminal a

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

按照上述方法构造出的 PDA P ，有 $L(G) = N(P)$

Proof.

$$w \in N(P) \iff w \in L(G)$$

(\Leftarrow): 如果 $w \in L(G)$, 则存在一个最左推导序列

$$S = \gamma_1 \xRightarrow{\text{lm}} \gamma_2 \xRightarrow{\text{lm}} \cdots \xRightarrow{\text{lm}} \gamma_n = w$$

则有 $(q, w, S) \vdash_P^* (q, y_i, \alpha_i)$, 其中 (q, y_i, α_i) 代表了最左句型 γ_i (

$\gamma_i = x_i \alpha_i, w = x_i y_i$) , 其证明基于对 i 的归纳

Basis. 当 $i = 1$ 时, 有 $\gamma_1 = S, x_1 = \epsilon, y_1 = w$, 显然有
 $(q, w, S) \vdash^* (q, w, S)$

Induction. 假设 $(q, w, S) \vdash^* (q, y_i, \alpha_i)$, 由于 α_i 是 tail, 其第一个 symbol 是某个 variable A , 在最左推导 $\gamma_i \Rightarrow \gamma_{i+1}$ 中, 使用 A 的一个产生式 $A \rightarrow \beta$ 将其展开, 而根据 PDA 的构造过程, 可以使用 β 替代栈顶的 A , 然后利用输入中的 terminal 将栈中的 terminal 弹出, 直到栈顶为下一个 variable, 达到 $(q, y_{i+1}, \alpha_{i+1})$, 即 $(q, w, S) \vdash^* (q, y_{i+1}, \alpha_{i+1})$

当 $i = n$ 时, $\alpha_n = \epsilon$, 于是有 $(q, w, S) \vdash^* (q, \epsilon, \epsilon)$, P 接受 w

(\Rightarrow): 需要证明: 当 P 进行一系列操作后, 将栈顶的 variable A 弹出, 同时没有涉及到 A 以下的栈内容 (称其**净效应**为弹出 A) , 则 A 可以推导出在此过程中消耗的所有输入, 即

$$(q, x, A) \vdash_P^* (q, \epsilon, \epsilon) \Rightarrow A \xRightarrow{*}_G x$$

证明将基于 P 操作的步数

Basis. 只有一步操作, 则唯一的可能是 $A \rightarrow \epsilon \in P$, 则根据 δ 的定义, 有
 $(q, \epsilon, A) \vdash (q, \epsilon, \epsilon)$, 即 $x = \epsilon$, 显然 $A \Rightarrow x$

Induction. 考虑 P 操作了 n 步, 则第一步一定是用 A 的某个产生式体替换了栈顶的 A , 假设用来替换的产生式是 $A \rightarrow Y_1 Y_2 \dots Y_k$

则接下来的 $n - 1$ 步一定是从输入中消耗了 x , 并且其净效应为逐个弹出 $Y_1, Y_2 \dots$, 则可以将 x 分为 $x_1 x_2 \dots x_k$, 其中 x_i 代表了从弹出 Y_{i-1} 到弹出 Y_i 之间消耗的输入 (即栈顶从 Y_i 变为 Y_{i+1}) , 则对于所有 i 有

$$(q, x_i x_{i+1} \dots x_k, Y_i) \vdash^* (q, x_{i+1} \dots x_k, \epsilon)$$

由于这其中操作步数都不会超过 $n - 1$, 根据 I. H. 有 $Y_i \xRightarrow{*} x_i$

于是有推导

$$A \Rightarrow Y_1 Y_2 \dots Y_k \xRightarrow{*} x_1 Y_2 \dots Y_k \xRightarrow{*} \cdots \xRightarrow{*} x_1 x_2 \dots x_k = x$$

即 $A \xRightarrow{*} x$

令 $A = S, x = w$, 由于 $w \in N(P)$, 有 $(q, w, S) \vdash^* (q, \epsilon, \epsilon)$, 则根据上述结论, 有 $S \xRightarrow{*} w$, 即 $w \in L(G)$

From PDA to Grammars

对任意 PDA P 都存在一个 CFG G 使得 $L(G) = N(P)$

基本思想是文法中的 variable 代表了 PDA 运行中的一个 event, 即 variable $[pXq]$ 代表了

- 净效应为从栈中弹出 X , 即弹出 X 的过程不涉及 X 以下的栈内容
- 当 X 从栈中被弹出的同时, 状态也从开始的 p 转换到了 q

而这个 variable 产生的 string 即是在 event 中被消耗的输入

更为详细的构造为, 令 $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ 为 PDA , 则构造

$$G = (V, \Sigma, R, S)$$

其中 V 包含

- 特殊的开始符号 S
- 其余符号都形如 $[pXq], p, q \in Q, X \in \Gamma$

对于 G 中的产生式

- 对所有状态 p , G 中有产生式 $S \rightarrow [q_0 Z_0 p]$, 即 $[q_0 Z_0 p]$ 产生的 string w 可以将 Z_0 从栈中弹出, 同时状态转移到 p , 即 $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$ 。即 S 将产生所有能让 PDA 清空其栈的 string
- 若 $(r, Y_1 Y_2 \dots Y_k) \in \delta(q, a, X)$, 其中 $a \in \Sigma$ or $a = \epsilon, k = 0, 1, \dots$, 则对于任意状态 r_1, r_2, \dots, r_k , 有产生式

$$[qXr_k] \rightarrow a[rY_1r_1][r_1Y_2r_2] \dots [r_{k-1}Y_kr_k]$$

即从栈中弹出 X 的方法可以是读入 a 后逐个弹出 Y_1, Y_2, \dots, Y_k , 而期间的状态转换可以是任意状态。若 $k = 0$, 则为 $[qXr] \rightarrow a$

可以证明 $[qXp] \xRightarrow{*} w \iff (q, w, X) \vdash^* (p, \epsilon, \epsilon)$

Proof.

(\Leftarrow): 已知 $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$, 则基于 PDA 的行动数归纳

Basis. 行动一步, 则 $(p, \epsilon) \in \delta(q, w, X)$, 而 w 为一个 symbol 或是 ϵ 。根据上述产生式的构造规则, 有 $[qXp] \rightarrow w$, 故 $[qXp] \xRightarrow{*} w$

Induction. 考虑操作了 n 步, 则其形式类似

$$(q, w, X) \vdash (r_0, x, Y_1 Y_2 \dots Y_k) \vdash^* (p, \epsilon, \epsilon)$$

其中 $w = ax, a \in \Sigma$ or $a = \epsilon$

故 $(r, Y_1 Y_2 \dots Y_k) \in \delta(q, a, X)$, 根据产生式构造规则, 有

$$[qXr_k] \rightarrow a[r_0 Y_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$$

其中 $r_k = p$ 而其余 r_1, r_2, \dots, r_{k-1} 可以为任意状态

对任意 i , r_i 是在 Y_i 被弹出时转换到的状态, 可以令 $x = w_1 w_2 \dots w_k$, 其中 w_i 为从 Y_{i-1} 弹出到 Y_i 弹出期间消耗的输入, 则有

$(r_{i-1}, w_i, Y_i) \vdash^* (r_i, \epsilon, \epsilon)$ 。由于这些转换都不可能有多步, 则根据 I. H. 有 $[r_{i-1} Y_i r_i] \xRightarrow{*} w_i$

故有

$$\begin{aligned} [qXr_k] &\xRightarrow{*} a[r_0 Y_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k] \xRightarrow{*} \\ &aw_1[r_1 Y_2 r_2][r_2 Y_3 r_3] \dots [r_{k-1} Y_k r_k] \xRightarrow{*} \\ &aw_1 w_2[r_2 Y_3 r_3] \dots [r_{k-1} Y_k r_k] \xRightarrow{*} \\ &\dots \\ &aw_1 w_2 \dots w_k = w \end{aligned}$$

(\Rightarrow): 证明基于对推导步数的归纳

Basis. 仅有一步推导, 则一定有产生式 $[qXp] \rightarrow w$, 而根据构造规则, 能产生这样的产生式一定说明 $(p, \epsilon) \in \delta(q, a, X)$, 其中 $a = w$, 则有 $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$

Induction. 考虑 $[qXp] \xRightarrow{*} w$ 推导用了 n 步, 则其形式形如

$$[qXr_k] \xRightarrow{*} a[r_0 Y_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k] \xRightarrow{*} w$$

其中 $r_k = p$, 而产生这样的产生式, 根据构造规则, 一定说明 $(r_0, Y_1 Y_2 \dots Y_k) \in \delta(q, a, X)$

则可以将 w 写作 $aw_1 w_2 \dots w_k$, 满足 $[r_{i-1} Y_i r_i] \xRightarrow{*} w_i$, 由于这些推导步数都不会超过 n 步, 则根据 I. H. 有

$$(r_{i-1}, w_i, Y_i) \vdash^* (r_i, \epsilon, \epsilon)$$

由于可以在输入后和栈底添上任意符号串, 即

$$(r_{i-1}, w_i w_{i+1} \dots w_k, Y_i Y_{i+1} \dots Y_k) \vdash^* (r_i, w_{i+1} \dots w_k, Y_{i+1} \dots Y_k)$$

即可得

$$\begin{aligned}
(q, aw_1w_2 \dots w_k, X) &\vdash (r_0, w_1w_2 \dots w_k, Y_1Y_2 \dots Y_k) \\
&\vdash^* (r_2, w_2 \dots w_k, Y_2 \dots Y_k) \\
&\vdash^* \dots \\
&\vdash^* (r_k, \epsilon, \epsilon)
\end{aligned}$$

由于 $r_k = p$, 我们得到了 $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$

则根据上述结论

$$S \xRightarrow{*} w \iff \exists p \in Q, [q_0 Z_0 p] \xRightarrow{*} w \iff (q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon) \iff w \in N(P)$$

Deterministic Pushdown Automata

PDA 默认即是 Nondeterministic 的, 而实际上 deterministic 的 PDA 也有重要的作用

Definition

一个 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ 是 DPDA (Deterministic PDA) 当且仅当满足

- 对于任何 $q \in Q, a \in \Sigma, a \neq \epsilon, X \in \Gamma$, $\delta(q, a, X)$ 只有一个元素
- 若对 $a \in \Sigma$ 有 $\delta(q, a, X)$ 非空, 则 $\delta(q, \epsilon, X)$ 一定是空的

Regular language and DPDA

DPDA 接受的语言集合在正则语言和 CFL 之间, 首先可以证明所有的正则语言都可以被 DPDA 接受

如果 L 是正则语言, 则存在 DPDA P 使得 $L = L(P)$

Proof. 显然, 只需要利用 DPDA 中 DFA 的部分即可

令 DFA $A = (Q, \Sigma, \delta_A, q_0, F)$, 则可以构造一个 DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

对于所有满足 $\delta(q, a) = p$ 的 $p, q \in Q$, 令 $\delta_P(q, a, Z_0) = \{(p, Z_0)\}$ 即可

显然 $(q_0, w, Z_0) \vdash^* (p, \epsilon, Z_0) \iff \delta_A(q_0, w) = p$, 证明基于 w 的长度归纳即可

Prefix Property: 没有两个不同的字符串满足其中一个是另一个的前缀

则有定理: 对某个 DPDA P 来说如果语言 $L = N(P)$ 当且仅当 L 有 prefix property 并且存在 DPDA P' 满足 $L = L(P')$

故可以看出正则语言可以由 acceptance by final state 的 DPDA 描述，但不一定能由 acceptance by empty stack 的 DPDA 描述，即对于 DPDA 来说 $L(P)$ 的描述能力是强于 $N(P)$ 的，两者并不等价

DPDA and Context-Free Language

虽然 DPDA 能接受形如 $\{wcw^R\}$ 这样的非正则的语言，但是也存在 CFL L 满足不存在 DPDA P 使得 $L = L(P)$ ，如 $\{ww^R\}$ ，当读完 w 的时候栈已清空，没有信息用于识别后续的 w^R

故对于 DPDA 来说， $L(P)$ 识别的语言集合是正则语言的超集，CFL 的子集

DPDA and Ambiguous Grammars

如果对 DPDA P 有 $L = N(P)$ ，那么这个语言存在一个无歧义的文法

如果对 DPDA P 有 $L = L(P)$ ，那么这个语言存在一个无歧义的文法

上述定理的证明可以见课本 P.255-256