第6章 下推自动机

形式化定义

- A PDA is described by:
  1. A finite set of *states* (Q, typically).
  2. An *input alphabet* (Σ, typically).
  3. A *stack alphabet* (Γ, typically).
  4. A *transition function* (δ, typically).
  5. A *start state* (q_0, in Q, typically).
  6. A *start symbol* (Z_0, in Γ, typically).
  7. A set of *final states* ($F \subseteq Q$, typically).

迁移函数

- Takes three arguments:
  1. A state, in Q.
  2. An input, which is either a symbol in Σ or ϵ.
  3. A stack symbol in Γ.
- $\delta(q, a, Z)$ is a set of zero or more actions of the form (p, α).
  - p is a state; α is a string of stack symbols.

注意：在迁移函数中，如果写 $\delta(q, 0, Z_0) = \{(q, XZ_0)\}$，是后面的符号先压栈，即栈顶在左边

瞬时描述ID

A ID is a triple (q, w, α), where:

1. q is the current state.
2. w is the remaining input.
3. α is the stack contents, top at the left.

ID 的 Goes-To关系用 ⊦ 表示

- Theorem 1: Given a PDA P, if $(q, x, \alpha)\vdash^* (p, y, \beta)$, for all the string w in Σ^* and all the string γ in Γ^*, we have $(q, xw, \alpha\gamma)\vdash^* (p, yw, \beta\gamma)$
- Theorem 2: Given a PDA P, if $(q, xw, \alpha)\vdash^* (p, yw, \beta)$, we have $(q, x, \alpha)\vdash^* (p, y, \beta)$

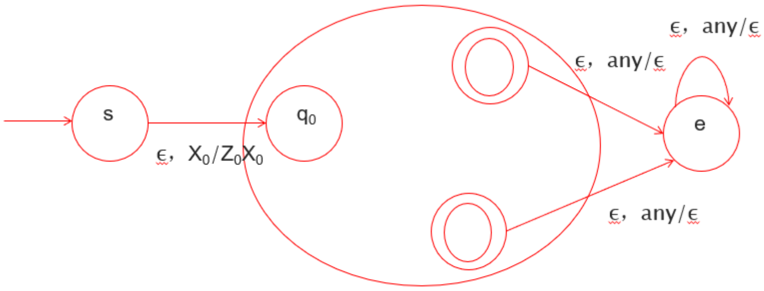
即：栈中符号不能随便去除，因为中途可能用到！

PDA的语言

两种定义方式：接受状态，空栈

在整个语言层面讨论：所有PDA能定义的LP 和所有PDA 能定义的NP 一样，即上下文无关语言

终止状态接收->空栈接收

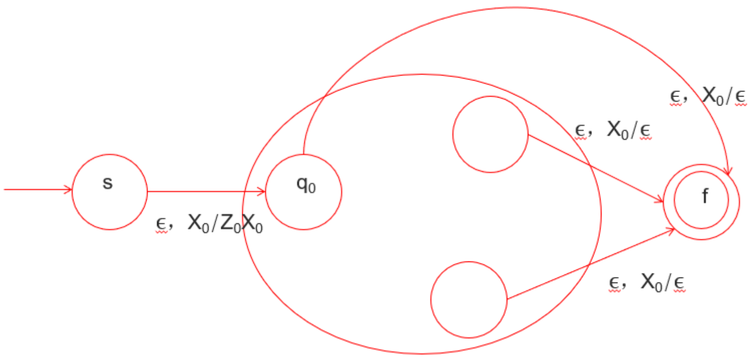


P' has all the states, symbols, and moves of P, plus:

1. Stack symbol X_0 (the start symbol of P'), used to guard the stack bottom.
2. New start state s and "erase" state e.
3. $\delta(s, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$. Get P started.
4. Add $\{(e, \epsilon)\}$ to $\delta(f, \epsilon, X)$ for any final state f of P and any stack symbol X, including X_0.
5. $\delta(e, \epsilon, X) = \{(e, \epsilon)\}$ for any X.

Why add X_0?

空栈接收->终止状态接收



P" has all the states, symbols, and moves of P, plus:

1. Stack symbol X_0 (the start symbol), used to guard the stack bottom.
2. New start state s and final state f.
3. $\delta(s, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$. Get P started.
4. $\delta(q, \epsilon, X_0) = \{(f, \epsilon)\}$ for any state q of P.

DPDA

To be deterministic, there must be at most one choice of move for any state q, input symbol *a*, and stack symbol X.

In addition, there must not be a choice between using input ϵ or real input.

- Formally, $\delta(q, a, X)$ and $\delta(q, \epsilon, X)$ cannot both be nonempty.

用空栈和接受状态定义的DPDA并不等价

DPDA (L(P)) is powerful than DPDA (N(P))

CFG和PDA的等价性

CFG->PDA

空栈接收，输入为终结符串w，如果PDA接收w，则w可由该上下文无关文法推导出：

$\delta(q, a, a) = (q, \epsilon)$. (*Type 1* rules)

- This step does not change the LSF represented, but "moves" responsibility for *a* from the stack to the consumed input.

If A -> α is a production of G, then $\delta(q, \epsilon, A)$ contains (q, α). (*Type 2* rules)

- Guess a production for A, and represent the next LSF in the derivation.

PDA->CFG

- G's variables are of the form [pXq].
- This variable generates all and only the strings w such that

  $(p, w, X) \vdash^* (q, \epsilon, \epsilon)$.
- Also a start symbol S we'll talk about later.

产生式：

G will have variables [pXq] generating exactly the inputs that cause P to have the net effect of popping stack symbol X while going from state p to state q.

- Each production for [pXq] comes from a move of P in state p with stack symbol X.
- Simplest case: $\delta(p, a, X)$ contains (q, ϵ).
  - Note *a* can be an input symbol or ϵ.
- Then the production is [pXq] -> a.
- Here, [pXq] generates *a*, because reading *a* is one way to pop X and go from p to q.

Next simplest case: $\delta(p, a, X)$ contains (r, Y) for some state r and symbol Y.

G has production [pXq] -> a[rYq].

- We can erase X and go from p to q by reading *a* (entering state r and replacing the X by Y) and then reading some w that gets P from r to q while erasing the Y.

- Third simplest case: $\delta(p, a, X)$ contains (r, YZ) for some state r and symbols Y and Z.
- Now, P has replaced X by YZ.
- To have the net effect of erasing X, P must erase Y, going from state r to some state s, and then erase Z, going from s to q.

Since we do not know state s, we must generate a family of productions:

[pXq] -> a[rYs][sZq]

for all states s.

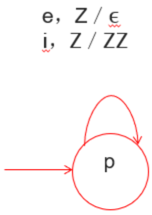
[pXq] =>* auv whenever [rYs] =>* u and [sZq] =>* v.

- We can prove that $(q_0, w, Z_0)\vdash^*(p, \epsilon, \epsilon)$ if and only if $[q_0Z_0p] \Rightarrow^* w$.
  - Proof is two easy inductions.
- But state p can be anything.
- Thus, add to G another variable S, the start symbol, and add productions $S \rightarrow [q_0Z_0p]$ for each state p.

为什么是 any state？——以空栈状态接收！

例子：

Design a PDA, which can handle the if else statement, it stops when the number of else exceeds the number of prefix if



S->[pZp]	S->A
[pZp]->e	A->e
[pZp]->i[pZp][pZp]	A->iAA

S->e|iSS