Autómatas y Lenguajes Formales
Nota 10. Autómatas de Pila*

Noé Salomón Hernández S.

1. Autómatas de Pila

Los *autómatas de pila*, PDA[1], son una clase de máquinas correspondientes a las GLC, es decir, aceptan exactamente los lenguajes libres de contexto. Tales máquinas son útiles al diseñar analizadores sintácticos basados en una GLC. Es menester darles a los PDAs memoria ilimitada para poder manejar lenguajes no regulares. Sin embargo, restringiremos su acceso a la memoria. La configuración se muestra en la siguiente figura

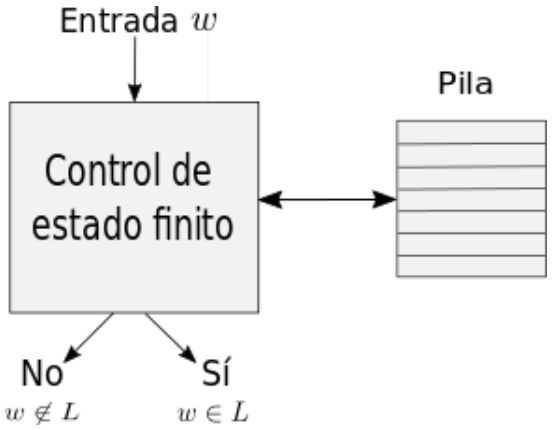

Figura 1: Configuración de una PDA.

Básicamente, un control de estados finito hace referencia a una autómata finito no-determinista con transiciones ε.

Usaremos la siguiente notación para el manejo de la pila. Dada la pila;

A
B
B
A
C

*Esta nota se basa en el libro: J. Martin. *Introduction to Languages and the Theory of Computation*, y en las notas del prof. Rajeev Motwani.

[1]Por sus siglas en inglés *Push Down Automata*.

su contenido se escribe:

$$\begin{array}{ccc} tope & & fondo \\ \searrow & & \swarrow \\ & ABBAC & \end{array}$$

la operación de *pop* regresa A, y el nuevo contenido sería $BBAC$; la operación $push(XYZ)$ añade elementos al tope de la pila, por lo que el contenido sería ahora $XYZBBAC$.

Las transiciones en un PDA están determinadas por:

- el símbolo de entrada o transición ε;
- el estado actual; y
- el símbolo en el tope de la pila.

Así el efecto que produce es

I. llegar a un nuevo estado;

II. *pop* a la pila; y

III. *push* de una nueva cadena a la pila.

Definición 1.1 Una *autómata de pila* (PDA) M se define como una tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, donde

- Q, un conjunto finito de estados – $p, q, r, s, \ldots$;
- Σ, un alfabeto finito de entrada – $a, b, c, \ldots$;
- Γ, un alfabeto finito de la pila, – $A, B, C, \ldots$ En general $\Sigma \subseteq \Gamma$;
- $q_0 \in Q$, estado inicial;
- $Z_0 \in \Gamma$, símbolo inicial de la pila; y
- $F \subseteq Q$, conjunto de estados finales.

La función de transición se define como $\delta(q, a, X) = \{(p_1, \alpha_1), (p_2, \alpha_2), \ldots, (p_k, \alpha_k)\}$, donde

- $q, p_i \in Q$ con $0 \leq i \leq k$,
- $a \in \Sigma$ ó $a = \varepsilon$,
- $X \in \Gamma$, y
- $\alpha_i \in \Gamma^*$ con $0 \leq i \leq k$.

Lo que indica que primero el PDA hace *pop* al tope de la pila para determinar quien es X, también lee la entrada a (a menos que sea una transición ε), y toma su estado actual q; con q, a y X selecciona una de las posibles (p_i, α_i) de manera no determinista. Así, el estado va de q a p_i, el símbolo de entrada a se consume (a menos que sea una transición ε), y la pila pierde el símbolo en su tope X para dar lugar a la cadena α_i que se ingresó al tope de la pila.

Necesitamos a Z_0 como elemento inicial en la pila para permitir a la primera transición hacer *pop*.

Acordamos que las cadenas en Σ^* sean $\ldots, w, x, y, z$ y las cadenas en Γ^* sean $\alpha, \beta, \gamma, \ldots$.

1.1. Configuraciones

Definición 1.2 Una **configuración** es una notación concisa para describir la composición de un PDA en un punto de la ejecución. Se denota como $\langle q, x, \alpha \rangle$, donde:

- $q \in Q$ es el estado actual,
- $x \in \Sigma^*$ es la cadena que actualmente se procesa, y
- $\alpha \in \Gamma^*$ es el contenido actual de la pila.

Podemos escribir una transición en un PDA con configuraciones. Supongamos que $(p_i, \alpha_i) \in \delta(q, a, X)$, entonces escribimos $\langle q, aw, X\beta \rangle \vdash \langle p_i, w, \alpha_i\beta \rangle$, lo que quiere decir que de la configuración $\langle q, aw, X\beta \rangle$ llegamos a $\langle p_i, w, \alpha_i\beta \rangle$ en un paso de transición. En general, escribimos $\mathsf{Conf}_1 \vdash^* \mathsf{Conf}_n$ si $\mathsf{Conf}_1 \vdash \mathsf{Conf}_2 \vdash \ldots \vdash \mathsf{Conf}_n$, es decir, de la configuración Conf_1 llegamos a Conf_n en cero o más pasos de la transición.

Decimos que un PDA acepta la cadena de entrada w si existe al menos una trayectoria de ejecución que llega a un estado final cuando se termina de procesar la cadena de entrada, sin importar el contenido de la pila. Por lo que un PDA rechaza al cumplirse alguna de las siguientes afirmaciones para toda trayectoria de ejecución:

- Si no hay transición posible faltando por procesar parte de la cadena de entrada. Decimos que dicho PDA se estanca.
- Si aún falta por procesar parte de la cadena de entrada, pero la pila está vacía.
- Si se termina de procesar la cadena de entrada, pero no se llega a un estado final.

Definición 1.3 El **lenguaje** de un PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es

$$L(M) = \{w \in \Sigma^* \mid \langle q_0, w, Z_0 \rangle \vdash^* \langle p, \varepsilon, \alpha \rangle \text{ con } p \in F, \text{ y cualquier } \alpha \in \Gamma^*\}.$$

Observamos que $\vdash^*$ indica que $\langle q_0, w, Z_0 \rangle$ nos lleva a $\langle p, \varepsilon, \alpha \rangle$, pero esa es sólo una de muchas posibles trayectorias de ejecución.

Notemos también que $L(M)$ se conoce como **lenguaje aceptado por estado final** ya que hay otra clase de lenguaje definido para PDAs.

Definición 1.4 El **lenguaje aceptado por pila vacía** por un PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es

$$N(M) = \{w \in \Sigma^* \mid \langle q_0, w, Z_0 \rangle \vdash^* \langle p, \varepsilon, \varepsilon \rangle \text{ con } p \text{ cualquier estado}\}.$$

En $N(M)$ ignoramos los estados finales completamente y consideramos la pila vacía como una señal de aceptación. Como de costumbre, decimos que w es aceptada si al menos para una trayectoria de ejecución la pila está vacía cuando w se termina de consumir. Una cadena w se rechaza si para toda trayectoria de ejecución, uno de lo siguiente ocurre:

- Si antes de terminar de procesar a w, el PDA se estanca.
- Si antes de terminar de procesar a w, la pila está vacía.
- Si se termina de procesar la cadena de entrada, pero la pila no está vacía.

Teorema 1.5 *Las definiciones de lenguaje anteriores son equivalentes. Es decir, $L = L(M_1)$ para algún PDA M_1 que reconoce por estados finales si y sólo si $L = N(M_2)$ para algún PDA M_2 que reconoce por pila vacía.*

Demostración.

$\Rightarrow$ Dado $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ vamos a construir M_2 tal que $N(M_2) = L(M_1)$. Tomamos $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, p_0, X_0, \varnothing)$, con

- $Q_2 = Q \cup \{p_0, r\}$,
- $\Gamma_2 = \Gamma \cup \{X_0\}$,
- δ_2 se comporta del siguiente modo sobre la cadena de entrada w:
  1) Comienza en el estado p_0 con X_0 en la pila.
  2) Realiza un transición ε a q_0 con Z_0X_0 en la pila.
  3) Simula a M_1 sobre w.
  4) De cualquier estado final de M_1 agrega una transición ε al estado r, el cual su única función es vaciar la pila sin consumir símbolos de entrada.

  Observe que la función de X_0 es evitar una *aceptación* accidental en la que M_1 termina de procesar la cadena de entrada y rechaza, con la pila vacía.

$\Leftarrow$ Dado $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$ vamos a construir M_1 tal que $L(M_1) = N(M_2)$. Tomamos $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, p_0, X_0, F_1)$, con

- $Q_1 = Q \cup \{p_0, p_f\}$,
- $\Gamma_1 = \Gamma \cup \{X_0\}$,
- $F_1 = \{p_f\}$,
- δ_1 se comporta del siguiente modo sobre la cadena de entrada w:
  1) Comienza en el estado p_0 con X_0 en la pila.
  2) Realiza un transición ε a q_0 con Z_0X_0 en la pila.
  3) Simula a M_2 sobre w.
  4) Cuando M_2 vacía su pila, X_0 se descubre como el único elemento en la pila de M_1.
  5) De cualquier estado de M_2, se agregan transiciones ε al estado p_f siempre que X_0 esté en el tope de la pila.

$\dashv$

Observemos que al simular GLCs o LLCs con PDAs, es más conveniente usar $N(M)$ que $L(M)$, por eso es que se provee de la definición por pila vacía.

2. Equivalencia entre GLC y PDA

Los lenguajes generados por los PDAs corresponden a la clase de lenguajes libres de contexto. Es decir, todo LLC es reconocido por algún PDA, y todo PDA acepta algún LLC.

Teorema 2.1 *Si L es un LLC, entonces $L = N(M)$ para algún PDA M que reconozca por pila vacía.*

Demostración. Supongamos que G es una GLC para L. Deseamos construir un PDA M tal que $\mathcal{L}(G) = N(M)$. Bajo la entrada w, dicho M simulará las derivaciones más a la izquierda en G que producen w, tal que en cualquier paso de la derivación la forma oracional[2] se representa por:

- la secuencia de símbolos de w que ya fueron *consumidos* por M,
- concatenados con el contenido actual de la pila de M.

De modo que si en G tenemos $S \underset{lm}{\Rightarrow}^* abXYcZ \underset{lm}{\Rightarrow}^* abxycz$, donde $abxycz$ es una cadena de símbolos terminales, entonces buscamos que en M la ejecución a partir de la configuración inicial, $\langle q_0, abxycz, S\rangle$, proceda del siguiente modo:

$\langle q_0, abxycz, S\rangle$

$\vdash^* \langle q_0, xycz, XYcZ\rangle$ corresponde a $S \underset{lm}{\Rightarrow}^* abXYcZ$ pues aquí M ha consumido ab de la cadena de entrada y el contenido de su pila es $XYcZ$,

$\vdash^* \langle q_0, \varepsilon, \varepsilon\rangle$ corresponde a $S \underset{lm}{\Rightarrow}^* abxycz$ pues aquí M ha consumido completamente la cadena de entrada y su pila está vacía.

Recordemos que el autómata M reconoce por pila vacía.

Formalmente, dada una GLC $G = (V, T, P, S)$ construimos el PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$ que acepta por pila vacía, donde:

- $Q = \{q\}$,
- $q_0 = q$,
- $\Sigma = T$,
- $\Gamma = V \cup T$,
- $Z_0 = S$.

La definición de δ consta de dos partes:

a) Si $A \in V$ está en el tope de la pila, entonces la reemplazamos por el lado derecho de cualquiera de sus producciones en P. Observemos que no hay cambios en la porción de la cadena de entrada que se ha consumido debido a que efectuamos un movimiento ε. Así, $\forall A \in V,\ \delta(q, \varepsilon, A) = \{(q, \alpha_1), (q, \alpha_2), \ldots, (q, \alpha_k)\}$ con $A \longrightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$ en G.

b) Si $a \in T$ está en el tope de la pila, entonces esperamos encontrar una a en la cadena de entrada y consumimos ambas. Observemos que no se cambia la forma oracional porque el carácter que quitamos del tope de la pila es el que consumimos de la cadena de entrada. Así, $\forall a \in T,\ \delta(q, a, a) = \{(q, \varepsilon)\}$.

Vemos que hay una correspondencia uno a uno entre derivaciones más a la izquierda y trayectorias de ejecución. Es posible que haya más de una trayectoria de ejecución. Si hay dos distintas que acepten a w, entonces existen dos derivaciones más a la izquierda distintas, esto implica que la gramática es ambigua.

Tenemos que demostrar que $N(M) = \mathcal{L}(G)$ siguiendo la idea de que M *adivina*, gracias al no determinismo, cada paso en una derivación más a la izquierda para w, y simula tal derivación.

[2]Recordemos que una forma oracional es cualquier $\alpha \in (V \cup T)^*$ tal que $S \Rightarrow^* \alpha$.

Lema 2.2 *Si $w \in \mathcal{L}(G)$, entonces w es aceptado por el PDA que se describe arriba.*

Demostración. En una derivación más a la izquierda, cualquier forma oracional que no sea una cadena de terminales puede ser escrita como $xA\alpha$, donde A es la variable más a la izquierda, x es la secuencia de terminales que aparecen a su izquierda, y α es la cadena de terminales y variables a la derecha de A. Llamamos a $A\alpha$ la *cola* de la forma oracional. Si la forma oracional consiste únicamente de terminales, entonces su cola es ε.

Supongamos que $w \in \mathcal{L}(G)$. Entonces w tiene una derivación más a la izquierda

$$S = \gamma_0 \underset{lm}{\Rightarrow} \gamma_1 \underset{lm}{\Rightarrow} \gamma_2 \ldots \underset{lm}{\Rightarrow} \gamma_n = w.$$

Mostraremos por inducción sobre i que $\langle q_0, w, S\rangle \vdash^* \langle q_0, y_i, \alpha_i\rangle$, donde la forma oracional γ_i se obtiene a partir de y_i y α_i, pues α_i es la cola de γ_i, así $\gamma_i = x_i\alpha_i$; además, y_i es la cadena tal que $x_i y_i = w$, es decir, y_i es lo que resta de la cadena w una vez que x_i ha sido procesada.

Base Se cumple para $i = 0$, tenemos $S = \gamma_0$. Así, $x_0 = \varepsilon$, $y_0 = w$. Como $\langle q_0, w, S\rangle \vdash^* \langle q_0, w, S\rangle$ en cero pasos, la base está demostrada.

H.I. Suponemos que se cumple para $i = k$, es decir, $\langle q_0, w, S\rangle \vdash^* \langle q_0, y_k, \alpha_k\rangle$.

Paso inductivo Tenemos que argumentar que se cumple para una derivación de tamaño $i = k+1$, es decir, debemos probar que $\langle q_0, w, S\rangle \vdash^* \langle q_0, y_{k+1}, \alpha_{k+1}\rangle$. Por la HI α_k es la cola de γ_k, por lo que comienza con una variable A, la cual también está en el tope de la pila. Vemos ahora que el paso de derivación $\gamma_k \underset{lm}{\Rightarrow} \gamma_{k+1}$ reemplaza A por un lado derecho de sus producciones, digamos β. La regla a) de la construcción de M reemplaza A en el tope de la pila por β, y por la regla b) se emparejan cualesquiera terminales en el tope de la pila con los siguientes símbolos de entrada. Después de esto, llegamos a la configuración $\langle q_0, y_{k+1}, \alpha_{k+1}\rangle$ con α_{k+1} iniciando con una variable, tal configuración representa la siguiente forma oracional γ_{k+1} en la derivación más a la izquierda.

Para terminar la demostración, notamos que $\alpha_n = \varepsilon$, ya que la cola de $\gamma_n = w$ es vacía. Así, $\langle q_0, w, S\rangle \vdash^* \langle q_0, \varepsilon, \varepsilon\rangle$, con lo cual concluimos que M acepta w por pila vacía. ⊣

Lema 2.3 *Si en M tenemos $\langle q, w, S\rangle \vdash^* \langle q, \varepsilon, \varepsilon\rangle$, entonces en G se cumple que $S \underset{lm}{\Rightarrow}^* w$.*

Demostración. Inducción sobre la longitud de la derivación $\vdash$.

Por lo tanto, para toda $w \in \mathcal{L}(G)$, $S \underset{lm}{\Rightarrow}^* w$ implica por el Lema 2.2 que $\langle q, w, S\rangle \vdash^* \langle q, \varepsilon, \varepsilon\rangle$, es decir, $w \in N(M)$. También, por el Lema 2.3 se tiene que para toda $w \in N(M)$, $\langle q, w, S\rangle \vdash^* \langle q, \varepsilon, \varepsilon\rangle$ implica $S \underset{lm}{\Rightarrow}^* w$, es decir, $w \in \mathcal{L}(G)$. Así,

$$w \in \mathcal{L}(G) \Leftrightarrow w \in N(M).$$

⊣

Teorema 2.4 *Si $L = N(M)$ para algún PDA M que reconoce por pila vacía, entonces L es un LLC.*

Demostración. Se puede consultar en el capítulo 5 de J. Martin. *Introduction to Languages and the Theory of Computation*, 3/e. McGrawHill 2004. ⊣

En el Teorema 2.1, nuestra demostración depende fuertemente en el poder del no-determinismo para permitir al autómata *adivinar* la derivación correcta. Pero en la vida real, en los analizadores sintácticos o con la herramienta yacc, no contamos con el poder del no-determinismo. ¿Podemos convertir un PDA a un autómata de pila determinista?

Definición 2.5 Un PDA determinista (DPDA, por sus siglas en inglés) M es una tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ donde cada elemento es idéntico al respectivo que se tiene en un PDA, además satisface las siguientes condiciones:

1. Para toda $q \in Q$, para toda $a \in \Sigma \cup \{\varepsilon\}$ y para toda $X \in \Gamma$, el conjunto $\delta(q, a, X)$ tiene a los más un elemento.

2. Para toda $q \in Q$, para toda $a \in \Sigma$ y para toda $X \in \Gamma$, los dos conjuntos $\delta(q, a, X)$ y $\delta(q, \varepsilon, X)$ no pueden definir ambos transiciones.

Nota: De acuerdo a la definición anterior, las transiciones ε están permitidas en un DPDA, siempre y cuando el DPDA no pueda elegir entre leer un símbolo y realizar un movimiento ε. Es decir, dado un estado q y un símbolo de la pila X, el DPDA puede optar por seguir la transición $\delta(q, a, X)$ definida para algún $a \in \Sigma$, ó realizar el movimiento ε de acuerdo a $\delta(q, \varepsilon, X)$, ó no realizar movimiento alguno si no hay transición definida.

Notemos que la sintaxis para un lenguaje de programación puede ser analizada eficientemente solo si el lenguaje correspondiente tiene un DPDA.

¿Es un DPDA tan expresivo como lo es un PDA? NO, ya que el lenguaje $L = \{ww^{\mathcal{R}} \mid w \in \Sigma^*\}$ no puede ser reconocido por un DPDA porque es imposible conocer de manera determinista donde termina w y comienza $w^{\mathcal{R}}$. Decimos que los lenguajes libres de contexto deterministas (LLCDs) son los generados por los DPDAs. Se tiene que

$$\underset{(\text{como } 0^*1^*)}{\text{Leng. regulares}} \subset \underset{(\text{como } 0^n1^n)}{\text{LLCD}} \subset \underset{(\text{como } ww^{\mathcal{R}})}{\text{LLC}}.$$

Si L es un LLCD, entonces no puede ser inherentemente ambiguo porque al convertir PDAs en GLCs, los DPDAs siempre producen una gramática no ambigua. Por lo que usualmente en lenguajes de programación se trata de definir lenguajes o gramáticas que produzcan LLCDs. Así obtenemos un análisis sintáctico eficiente y evitamos la ambigüedad.

Aceptación de los DPDA's: Los dos tipos de aceptación – por estados finales y por pila vacía – NO son lo mismo para los DPDA's. Se tiene que los lenguajes aceptados por pila vacía son exactamente aquellos aceptados por estados finales que tienen la propiedad de prefijo, la cual estipula que no hay cadena en el lenguaje que sea un prefijo de otra cadena en el lenguaje. De manera que el lenguaje definido por la expresión regular a^* lo reconoce un DPDA que acepte por estados finales; no obstante, tal lenguaje no puede ser reconocido por un DPDA que acepte por pila vacía.

Finalmente, observemos que J. Martin en su libro *Introduction to Languages and the Theory of Computation* afirma que un autómata finito con acceso a una *cola* en lugar de a una pila tiene tanto poder expresivo como una máquina de Turing, formalismo que se estudiará en la Nota 13.