Gramáticas de contexto libre y autómatas de pila

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Preguntas pendientes

Gramáticas son una buena forma de describir lenguajes en forma compacta e inteligible a humanos. Pero son terribles para uso en programas. Preferimos algún formalismo de autómatas, más cercano al computador.

Vimos que los autómatas finitos están limitados por tener memoria finita (sus estados), les agregamos una pila para dar los PDA.
Analizamos algunas de sus principales características.
Queda la pregunta de la relación de los PDA con la jerarquía de Chomsky.

CFG a PDA

Daremos una construcción que, dada una gramática de contexto libre, construye un autómata de pila que acepta el lenguaje generado por la gramática. La explicación dada puede extenderse a una demostración formal (a costa de bastantes desvíos que no aportan a nuestro objetivo central).

Idea informal

La idea es construir un autómata que vaya construyendo una derivación de la palabra entre manos. Dada la gramática $G = (N, \Sigma, P, S)$, la idea del autómata $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es que en su pila mantenga un sufijo de una forma sentencial de G, y una movida (si el tope de la pila es un no-terminal) es aplicar una producción en la pila leyendo ε, o (si el tope de la pila es un terminal) verificar que coincide con el terminal siguiente en la entrada.

Formalizando el PDA

Sea la gramática $G = (N, \Sigma, P, S)$. Construimos el PDA $M = (\{q_0\}, \Sigma, \Sigma \cup N, \delta, q_0, S, \varnothing)$ (lo diseñaremos para aceptar por pila vacía), con función de transición:

$$\delta(q_0, a, a) = \{(q_0, \varepsilon)\} \qquad \text{Para todo } a \in \Sigma$$

$$\delta(q_0, \varepsilon, A) = \{(q_0, \alpha^R) : A \to \alpha \in P\} \qquad \text{Para todo } A \in N$$

Entonces $\mathscr{L}(G) = \mathscr{N}(M)$.

Las movidas del primer tipo verifican los terminales generados, las del segundo aplican producciones. (Como debemos considerar los símbolos de izquierda a derecha, y el tope de la pila está a la derecha, almacenamos al revés.)

Un ejemplo

Reusamos nuestra gramática regalona G:

$$E \to E + T$$
$$E \to T$$
$$T \to T * F$$
$$T \to F$$
$$F \to (E)$$
$$F \to a$$

Un ejemplo

Consideremos $a * (a + a) \in \mathscr{L}(G)$.
Para claridad, la pila crece hacia la izquierda (mostramos su reverso).

Entrada	Pila	Operación
$a * (a + a)$	E	$E \to T$
$a * (a + a)$	T	$T \to T * F$
$a * (a + a)$	$T * F$	$T \to F$
$a * (a + a)$	$F * F$	$F \to a$
$a * (a + a)$	$a * F$	✓
$* (a + a)$	$* F$	✓
$(a + a)$	F	$F \to (E)$
$(a + a)$	(E)	✓
$a + a)$	$E)$	$E \to E + T$

Un ejemplo

Entrada	Pila	Operación
$a+a)$	$E+T)$	$E \to T$
$a+a)$	$T+T)$	$T \to F$
$a+a)$	$F+T)$	$F \to a$
$a+a)$	$a+T)$	✓
$+a)$	$+T)$	✓
$a)$	$T)$	$T \to F$
$a)$	$F)$	$F \to a$
$a)$	$a)$	✓
$)$	$)$	✓
ε	ε	Acepta

Gramáticas LL(k)

Lo que hace nuestra construcción es trazar una derivación de extrema izquierda.
Bajo condiciones bastante restrictivas sobre la gramática esta idea se puede extender a un PDA determinista (formalmente, gramáticas LL(1) si solo se considera el símbolo siguiente).

Gramáticas LL(k)

Por suerte las restricciones en gramáticas LL(1) son bastante naturales para uso práctico: debemos estar en condiciones de identificar la producción a aplicar analizando solo el no-terminal y el primer símbolo de lo que resta. Si se revisa un lenguaje de programación como C, vemos que (salvo el caso especial de expresiones) toda construcción mayor se introduce con una palabra o símbolo clave: for, if, ... (Si la estrategia esbozada no permite identificar tempranamente lo que estamos leyendo, el lenguaje nos traerá problemas como programadores.).
Esto da lugar a la popular técnica de *descenso recursivo*, fácil de programar sin herramientas especiales y por eso muy usada en la construcción de compiladores.

Gramáticas $\mathrm{LL}(k)$

El conjunto de lenguajes que son reconocidos por PDAs deterministas son generados por gramáticas mucho menos restrictivas que las gramáticas $\mathrm{LL}(k)$, pueden reconocerse por autómatas que deciden la acción a tomar después de leer la construcción completa. Variantes algo restringidas de gramáticas $\mathrm{LR}(1)$ pueden manejarse en forma muy eficiente mediante programas, son la base de herramientas populares para la construcción de compiladores.

PDA a CFG

Para completar la demostración de que gramáticas de contexto libre y PDAs describen los mismos lenguajes nos falta este paso.
Advertimos nuevamente que este paso *no* es de interés práctico, su interés es puramente teórico.

Idea informal

La idea básica es construir una gramática cuyos no-terminales generen palabras que hagan que el PDA (diseñado para aceptar por pila vacía) consuma un símbolo particular de su pila. Si la pila del PDA en algún instante tiene el símbolo A en el tope, si eventualmente acepta habrá algún instante posterior en que por primera vez la pila sea un símbolo más corta. Es claro que los símbolos debajo de A en la pila no intervienen, no pueden afectar este proceso.

Idea informal

Una complicación al plan esbozado es hacer coincidir estados. Nuestra construcción es simplemente considerar todas las posibilidades. Las gramáticas resultantes suelen ser gigantescas.

PDA a CFG

Suponemos dado un autómata de pila $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$, construiremos una gramática de contexto libre G tal que $\mathscr{L}(G) = \mathscr{N}(M)$.

PDA a CFG

Como no-terminales usaremos el tradicional símbolo de partida S y el conjunto de todos los objetos de la forma:

$$[p, A, q] \qquad p, q \in Q, A \in \Gamma$$

La idea es que si, con $\alpha \in \Sigma^*$:

$$[p, A, q] \Rightarrow^* \alpha$$

entonces M partiendo del estado p consume α de la entrada, elimina A del tope de la pila y queda en el estado q.

PDA a CFG

Formalmente, suponiendo $Q \cap \Sigma = \varnothing$, si:

$$(p\alpha, A) \vdash_M^* (\alpha q, \varepsilon)$$

entonces:

$$[p, A, q] \Rightarrow_G^* \alpha$$

Queremos que si acepta por pila vacía:

$$(q_0\alpha, Z_0) \vdash_M^* (\alpha q, \varepsilon)$$

entonces para algún $q \in Q$ tenemos:

$$S \Rightarrow_G [q_0, Z_0, q] \Rightarrow_G^* \alpha$$

PDA a CFG

Una movida $(q, \alpha) \in \delta(p, x, A)$ de M es ir del estado p al estado q, asumiendo el compromiso de ahora eliminar α que reemplaza a A de la pila comenzando en el estado q. Si $\alpha = \varepsilon$, hemos logrado nuestro cometido. Si $\alpha \neq \epsilon$, serán varias transiciones a estados que desconocemos, y terminaremos en otro estado desconocido. Salida fácil, que siempre funciona, pero para nada elegante, es simplemente ponerse en todos los casos posibles. Seguramente la mayoría de las producciones resulten inútiles, pero eso no importa: nos interesa demostrar que se puede, no obtener una gramática útil.

PDA a CFG

Para claridad, en nuestra construcción consideraremos que el tope de la pila está al comienzo (a la izquierda).

Hay varios casos a considerar al construir las producciones para G:

PDA a CFG

Para $(q, A_1 A_2 \cdots A_n) \in \delta(p, x, A)$:

$$[p, A, q_n] \to x[q, A_1, q_1][q_1, A_2, q_2] \cdots [q_{n-1}, A_n, q_n]$$

Acá $q_i \in Q$ son estados cualquiera, esto resume una posiblemente muy gran colección de producciones.

Para $(q, \varepsilon) \in \delta(p, x, A)$:

$$[p, A, q] \to x$$

Si $x = \varepsilon$, resulta una producción nula; pero sabemos cómo deshacernos de ellas luego.

PDA a CFG

Es simple construir un PDA que acepta ε, cosa que ninguna gramática de contexto libre según la definición formal genera. Demostramos en realidad equivalencia con nuestras gramáticas con producciones de la forma $A \to \alpha$ con $A \in N$, $\alpha \in (N \cup \Sigma)^*$.

Resumen

- Demostramos que PDAs y CFGs describen exactamente el mismo conjunto de lenguajes.
- Obtener un PDA (particularmente uno determinista, simple de simular en código) de una gramática es importante al construir programas que procesan lenguajes no triviales (intérpretes, compiladores, procesar pequeños lenguajes para propósitos específicos, pero también empresas más modestas como lenguajes de configuración). No tenemos espacio para entrar en detalles de las técnicas empleadas.