NOMBRE: Benjamín Farías Valdés

N.ALUMNO: 17642531

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2223 — Teoría de Autómatas y Lenguajes Formales — 2' 2021

Tarea 5

Pregunta 1

El **PDA alternativo** se construye a continuación:

$$\mathcal{D} := (Q, \Sigma, \Delta, q_0, F)$$
$$Q = \{\, S, 0, 1, \neg, \wedge, \vee, [\,\}$$
$$\Sigma = \{\, 0, 1, \neg, \wedge, \vee, [,]\,\}$$
$$q_0 = S$$
$$F = \{1\}$$
$$\Delta_0 = \{(S, x, x) \mid x \in \Sigma - \{\,]\,\}\}$$
$$\Delta_1 = \{(0\neg, \epsilon, 1), (1\neg, \epsilon, 0)\} \cup \{(\neg, x, x\neg) \mid x \in \Sigma - \{\,]\,\}\}$$
$$\Delta_2 = \{(11\wedge, \epsilon, 1), (01\wedge, \epsilon, 0), (10\wedge, \epsilon, 0), (00\wedge, \epsilon, 0)\} \cup \{(0\wedge, x, x0\wedge), (1\wedge, x, x1\wedge), (\wedge, x, x\wedge) \mid x \in \Sigma - \{\,]\,\}\}$$
$$\Delta_3 = \{(11\vee, \epsilon, 1), (01\vee, \epsilon, 1), (10\vee, \epsilon, 1), (00\vee, \epsilon, 0)\} \cup \{(0\vee, x, x0\vee), (1\vee, x, x1\vee), (\vee, x, x\vee) \mid x \in \Sigma - \{\,]\,\}\}$$
$$\Delta_4 = \{(0[,], 0), (1[,], 1)\} \cup \{(\,[, x, x[\,) \mid x \in \Sigma - \{\,]\,\}\}$$
$$\Delta = \bigcup_{i=0}^{4} \Delta_i$$

La construcción de este autómata se basa en que toda fórmula en notación polaca se puede ir evaluando de **izquierda a derecha**, ya que los operadores se colocan siempre a la izquierda de sus operandos. La idea es utilizar transiciones que simulan las definiciones de cada operador, haciendo uso del stack para acumular todas las operaciones que aún no se pueden procesar (debido a que no se han leído todos sus operandos o están anidadas). Eventualmente, esto permitirá ir reemplazando todos los estados que representan a operadores por estados con valor 0 o 1, obteniendo así la evaluación final de la fórmula proposicional. Bajo esta idea, los estados representarán a cada símbolo del alfabeto (para poder abarcar las operaciones y valores de verdad), excepto al símbolo], ya que siempre se corresponderá con un [que lo precede.

El estado inicial será S, que simplemente permite comenzar a almacenar otros estados en el stack **(este no es un estado final, ya que la fórmula vacía se asumirá como falsa)**.

El **único estado final** corresponderá a 1, ya que significa que toda la fórmula pudo ser reducida a un valor de verdad de 1 (esto por la definición de configuración final, donde sólo queda 1 estado en el stack).

A continuación se explican los componentes de la relación de transición:

- Δ_0: Este conjunto contiene a las transiciones base desde el estado inicial S, simplemente reemplazándolo por el primer símbolo leído desde el input (si dicho símbolo es un] se rechaza, ya que la fórmula estaría mal escrita).
- Δ_1: Este conjunto se encarga de procesar las operaciones de **negación.** Si en cualquier momento se tiene la negación de un número en el tope del stack, se reemplaza directamente por el número negado.
- Δ_2: Este conjunto se encarga de procesar las operaciones de **conjunción.** Si en cualquier momento se tiene una conjunción de 2 valores de verdad en el tope del stack, se reemplaza directamente por el resultado.
- Δ_3: Igual al anterior, pero procesando las operaciones de **disyunción.**
- Δ_4: Este conjunto se encarga de procesar los **paréntesis.** Si el tope del stack contiene un valor de verdad con un [y se lee un], se sabe que el interior del paréntesis ya fue resuelto, por lo que se eliminan ambos símbolos de paréntesis.

La relación de transición es la unión de todos los componentes descritos arriba, logrando acumular operaciones pendientes o anidadas en el stack hasta poder resolverlas (gracias a las tuplas que simplemente agregan el símbolo leído sin reemplazar el actual). De esta forma se logra modelar la gramática correctamente por medio del **no determinismo del autómata apilador.**

NOMBRE: Benjamín Farías Valdés

N.ALUMNO: 17642531

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2223 — Teoría de Autómatas y Lenguajes Formales — 2' 2021

Tarea 5

Pregunta 2

La definición del conjunto $\text{follow}_k(X)$, para una variable $X \in V$ es la siguiente:

$$\text{follow}_k(X) = \{w \mid S \overset{*}{\Rightarrow} \alpha X \beta \wedge w \in \text{first}_k(\beta\#)\}$$

De aquí se puede notar que lo que se pide calcular es justamente un conjunto que contenga todas las palabras del conjunto $\text{follow}_k(X)$, pero para cada X tal que $X \overset{*}{\Rightarrow} \gamma$ y X es alcanzable desde S. De hecho, **no es necesario asegurar que X sea alcanzable,** ya que si no lo es entonces su conjunto follow será vacío, debido a que depende de la concatenación con los conjuntos follow de las variables que la derivan, y la única variable que por sí sola tiene el conjunto follow^0 no vacío es la inicial ($\text{follow}_k^0(S) = \{\#\}$). En resumen, necesitamos calcular la unión de cada conjunto $\text{follow}_k(X)$ tal que $X \overset{*}{\Rightarrow} \gamma$, ya que así se considerarán todas las posibles palabras que le siguen a las palabras generadas por γ.

Para lograr esto, **se seguirán estos pasos:**

1. Calcular el conjunto $\text{follow}_k(X)$ para toda variable $X \in V$.
2. Determinar las variables $X \in V$ que satisfacen $X \overset{*}{\Rightarrow} \gamma$.
3. Obtener el conjunto $\text{follow}_k(\gamma)$ como la unión de los $\text{follow}_k(X)$ para cada $X \in V$ obtenida del paso anterior.

Para el primer paso necesitaremos tener previamente definido el algoritmo de first_k visto en clases, de forma que podamos **acceder siempre al conjunto $\text{first}_k(X)$ para cualquier $X \in V \cup \Sigma$ sin preocuparnos de calcularlo:**

Algorithm 1 $First_k(\mathcal{G}, k)$

for each $a \in \Sigma$ **do**
  $\text{first}_k^0(a) = \{a\}$
end for
for each $X \in V$ **do**
  $\text{first}_k^0(X) = \bigcup_{X \to w \in P} w|_k$
end for
$i = 0$
repeat
  $i = i + 1$
  for each $a \in \Sigma$ **do**
    $\text{first}_k^i(a) = \{a\}$
  end for
  for each $X \in V$ **do**
    $\text{first}_k^i(X) = \bigcup_{X \to X_1 ... X_n \in P} \text{first}_k^{i-1}(X_1) \odot_k ... \odot_k \text{first}_k^{i-1}(X_n)$
  end for
until $\text{first}_k^i(X) = \text{first}_k^{i-1}(X)$. $\forall X \in V \cup \Sigma$
return $\{\text{first}_k(X)\}_{X \in V \cup \Sigma}$

Además, es de nuestro interés obtener el $\text{first}_k(\beta)$, con $\beta \in (V \cup \Sigma)^*$, para lo que definimos el siguiente algoritmo auxiliar **aprovechando la propiedad vista en clases para calcular el first_k de una palabra formada por variables y terminales:**

Algorithm 2 $FirstBeta(\mathcal{G}, \beta, k)$

$\beta = X_1 ... X_n$
return $\text{first}_k(X_1) \odot_k ... \odot_k \text{first}_k(X_n)$

Ahora podemos definir el algoritmo para calcular el conjunto $\text{follow}_k(X)$ para toda $X \in V$ según el programa recursivo visto en clases, logrando así el **primer paso:**

Algorithm 3 $Follow_k(\mathcal{G}, k)$

$\text{follow}_k^0(S) = \{\#\}$
for each $X \in V \setminus \{S\}$ **do**
  $\text{follow}_k^0(X) = \varnothing$
end for
$i = 0$
repeat
  $i = i + 1$
  for each $X \in V$ **do**
    $\text{follow}_k^i(X) = \bigcup_{Y \to \alpha X \beta} FirstBeta(\mathcal{G}, \beta, k) \odot_k \text{follow}_k^{i-1}(Y)$
  end for
until $\text{follow}_k^i(X) = \text{follow}_k^{i-1}(X)$. $\forall X \in V$
return $\{\text{follow}_k(X)\}_{X \in V}$

Para el **segundo paso,** se necesita un algoritmo que reciba una palabra $\gamma \in (V \cup \Sigma)^*$ y entregue el conjunto de todas las variables $X \in V$ tal que $X \overset{*}{\Rightarrow} \gamma$. Para esto se define el siguiente algoritmo, basado en el algoritmo CKY visto en clases (que es aplicable ya que $\mathcal{G}$ está en CNF):

Algorithm 4 $CKY_\gamma(\mathcal{G}, \gamma)$

for $i \leftarrow 1$ **to** n **do**
  $X_{ii} = \varnothing$
  if $\gamma_i \in V$ **then**
    $X_{ii} = X_{ii} \cup \{\gamma_i\}$
  else
    for $X \to c \in P$ **do**
      if $c = \gamma_i$ **then**
        $X_{ii} = X_{ii} \cup \{X\}$
      end if
    end for
  end if
end for
for $k \leftarrow 1$ **to** $n - 1$ **do**
  for $i \leftarrow 1$ **to** $n - k$ **do**
    $X_{ii+k} = \varnothing$
    for $j \leftarrow i$ **to** $i + k - 1$ **do**
      for $X \to YZ \in P$ **do**
        if $Y \in X_{ij} \wedge Z \in X_{j+1\ i+k}$ **then**
          $X_{ii+k} = X_{ii+k} \cup \{X\}$
        end if
      end for
    end for
  end for
end for
return X_{1n}

El algoritmo de arriba es muy parecido al CKY, pero **difiere en lo siguiente:**

- En la primer iteración revisa si es que el elemento en la posición i de γ es una **variable,** en cuyo caso simplemente deja $X_{ii} = \gamma_i$. En otro caso, procede al igual que en el algoritmo original (cuando γ_i es terminal).
- Al final del algoritmo, se retorna el conjunto X_{1n}, que contiene justamente a **todas las variables** X **que satisfacen** $X \overset{*}{\Rightarrow} \gamma$**.**

Finalmente, teniendo todo lo anterior, podemos definir el algoritmo que calcula el conjunto $\text{follow}_k(\gamma)$, completando así el **tercer paso:**

Algorithm 5 $Follow_\gamma(\mathcal{G}, \gamma, k)$

$\text{follow}_k = Follow_k(\mathcal{G}, k)$
$\text{gen} = CKY_\gamma(\mathcal{G}, \gamma)$
$\text{follow}_k(\gamma) = \varnothing$
for each $X \in \text{gen}$ **do**
  $\text{follow}_k(\gamma) = \text{follow}_k(\gamma) \cup \text{follow}_k(X)$
end for
return $\text{follow}_k(\gamma)$

El algoritmo de arriba es el propuesto para resolver el problema, y **su correctitud fue explicada a medida que se fue construyendo.**