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

IIC2223 – Teoría de Autómatas y Lenguajes Formales – 2' 2024
IIC2224 – Autómatas y Compiladores

Tarea 6

Pregunta 1

Sea $\mathcal{D} = (Q, \Sigma, \Delta, q_0, F)$ un PDA alternativo. Recuerde que $\vdash^*_{\mathcal{D}}$ es la relación entre configuraciones tal que, para todo $\gamma_1, \gamma_2 \in Q^+$ y $w_1, w_2 \in \Sigma^*$, se tiene que $(\gamma_1, w_1) \vdash^*_{\mathcal{D}} (\gamma_2, w_2)$ si, y solo si, desde la configuración (γ_1, w_1) se puede llegar a la configuración (γ_2, w_2) en 0 o más pasos de $\mathcal{D}$.

1. Demuestre que para todo PDA alternativo $\mathcal{D}$, el siguiente lenguaje es libre de contexto:

$$\mathcal{L}_1(\mathcal{D}) = \{w \in \Sigma^* \mid \exists q \in F. \exists \gamma \in Q^*. (q_0, w) \vdash^*_{\mathcal{D}} (q\gamma, \epsilon)\}$$

2. Demuestre que para todo PDA alternativo $\mathcal{D}$, el siguiente lenguaje es libre de contexto:

$$\mathcal{L}_2(\mathcal{D}) = \{w \in \Sigma^* \mid \exists q \in F. \exists v \in \Sigma^*. (q_0, wv) \vdash^*_{\mathcal{D}} (q, \epsilon)\}$$

Solución

Problema 1.1 $\mathcal{L}_1(\mathcal{D})$ es el lenguaje que contiene todas las palabras w que, al ser leídas por $\mathcal{D}$, llegan a un estado final, pero no necesariamente con el stack limpio. Recordemos que para el PDA alternativo $\mathcal{D}$, la relación transición $\Delta \subseteq Q^+ \times (\Sigma \cup \{\epsilon\}) \times Q^*$. Definamos un nuevo PDA a partir de $\mathcal{D}$. Sea $\mathcal{D}_1 = (Q \cup \{q_L\}, \Sigma, \Delta_1, q_0, \{q_L\})$, de modo que agregamos un nuevo estado q_L que actuará como "estado de limpieza" y único estado final. Vamos a ejecutar las transiciones de $\mathcal{D}$ hasta llegar a un estado final, luego vamos a agregar nuevas transiciones para limpiar el stack.

Sea Δ_1 definido de la siguiente manera:

$$\Delta_1 = \Delta \cup \{(q_f, \epsilon, q_L) \mid q_f \in F\} \cup \{(q_L q, \epsilon, q_L) \mid q \in Q\}$$

Es decir, $\mathcal{D}_1$ contiene todas las transiciones de $\mathcal{D}$, además de una ϵ-transición desde todos los estados en F hacia el estado de limpieza q_L. Por último, se incluyen las ϵ-transiciones de limpieza, que mantienen al tope del stack al estado q_L.

Ahora podemos demostrar que $\mathcal{D}_1$ define el mismo lenguaje que $\mathcal{L}_1(\mathcal{D})$.

$\mathcal{L}_1(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{D}_1)$. Sea $w \in \mathcal{L}_1(\mathcal{D})$. Luego, sabemos que

$$(q_0, w) \vdash^*_{\mathcal{D}} (q_f\gamma, \epsilon) \qquad \text{para algún } q_f \in F$$

Como $\Delta \subseteq \Delta_1$, también se cumplirá

$$(q_0, w) \vdash^*_{\mathcal{D}_1} (q_f\gamma, \epsilon) \qquad \text{para algún } q_f \in F$$

Luego, por construcción

$$(q_f\gamma, \epsilon) \vdash_{\mathcal{D}_1} (q_L\gamma, \epsilon) \vdash^*_{\mathcal{D}_1} (q_L, \epsilon)$$

Por lo tanto, $w \in \mathcal{L}(\mathcal{D}_1)$.

$\mathcal{L}(\mathcal{D}_1) \subseteq \mathcal{L}_1(\mathcal{D})$. Sea $w \in \mathcal{L}(\mathcal{D}_1)$. Luego, tenemos que:

$$(q_0, w) \vdash^*_{\mathcal{D}_1} (q_L, \epsilon)$$

Por construcción de $\mathcal{D}_1$, la única manera de llegar a tener el estado q_L en el stack es haber pasado primero por algún estado $q_f \in F$.

$$(q_0, w) \vdash^*_{\mathcal{D}_1} (q_f\gamma, \epsilon) \vdash_{\mathcal{D}_1} (q_L\gamma, \epsilon) \vdash^*_{\mathcal{D}_1} (q_L, \epsilon)$$

Como todas las transiciones antes de q_L provienen de Δ, se cumple también

$$(q_0, w) \vdash^*_{\mathcal{D}} (q_f\gamma, \epsilon)$$

Por lo que $w \in \mathcal{L}_1(\mathcal{D})$. Finalmente, $\mathcal{L}_1(\mathcal{D}) = \mathcal{L}(\mathcal{D}_1)$.

Distribución de puntaje

- 1 punto por definir un nuevo PDA alternativo $\mathcal{D}_1$ a partir de $\mathcal{D}$.
- 1 punto por definir Δ_1 y agregar transiciones hacia un nuevo y único estado final q_L.
- 1 punto por agregar correctamente las ϵ-transiciones de limpieza del stack.
- 1 punto por demostrar que $\mathcal{L}_1(\mathcal{D}) = \mathcal{L}(\mathcal{D}_1)$.

Problema 1.2 $\mathcal{L}_2(\mathcal{D})$ es el lenguaje que contiene todas las palabras w que, con algún sufijo v concatenado, sería aceptada por $\mathcal{D}$. De manera similar, crearemos un nuevo PDA alternativo $\mathcal{D}_2 = (Q \cup \{q_S, q_F\}, \Sigma, \Delta_2, q_0, \{q_F\})$, de forma que $\mathcal{D}_2$ correrá una simulación de la ejecución de $\mathcal{D}$. Aquí agregamos un nuevo estado de simulación q_S y un único estado final q_F. Cuando q_S esté en el top del stack, estaremos en la etapa de simulación de $\mathcal{D}$. Sea Δ_2 definido de la siguiente manera:

$$\Delta_2 = \Delta \cup \{(q, \epsilon, q_S q) \mid q \in Q\} \cup \{(q_S\alpha, \epsilon, q_S\beta) \mid \exists a \in (\Sigma \cup \{\epsilon\}).(\alpha, a, \beta) \in \Delta\} \cup \{(q_S q_f, \epsilon, q_F) \mid q_f \in F\}$$

Nuevamente, mantenemos las transiciones de Δ. Luego, para todos los estados en Q agregamos una ϵ-transición que inicia la etapa de simulación, sumando un q_S al tope del stack. El siguiente grupo de transiciones añadido representan la etapa de simulación. Finalmente, agregamos las transiciones desde estados $q_f \in F$ dentro de la etapa de simulación hacia el único estado final q_F de $\mathcal{D}_2$.

Ahora podemos demostrar que $\mathcal{D}_2$ define el mismo lenguaje que $\mathcal{L}_2(\mathcal{D})$.

$\mathcal{L}_2(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{D}_2)$. Sea $w \in \mathcal{L}_2(\mathcal{D})$. Luego, tenemos que

$$(q_0, wv) \vdash^*_{\mathcal{D}} (q_f, \epsilon) \qquad \text{con } q_f \in F$$

$$(q_0, wv) \vdash^*_{\mathcal{D}} (\gamma, v) \vdash^*_{\mathcal{D}} (q_f, \epsilon) \qquad \text{con } q_f \in F$$

Como $\Delta \subseteq \Delta_2$, también se cumplirá

$$(q_0, w) \vdash^*_{\mathcal{D}_2} (\gamma, \epsilon)$$

Por la existencia de las transiciones de simulación, también será cierto

$$(q_0, w) \vdash^*_{\mathcal{D}_2} (\gamma, \epsilon) \vdash_{\mathcal{D}_2} (q_S\gamma, \epsilon) \vdash^*_{\mathcal{D}_2} (q_S q_f, \epsilon)$$

Llegaremos necesariamente a $q_S q_f$ ya que sabemos que existe v tal que, después de leerse w en $\mathcal{D}$, su lectura llevará a un estado final. Por construcción de las transiciones de la etapa de simulación, como existe este v, deberá existir un camino en la simulación que llegará a $q_S q_f$. Finalmente, por construcción también se cumplirá

$$(q_S q_f, \epsilon) \vdash_{\mathcal{D}_2} (q_F, \epsilon)$$

Por lo que $w \in \mathcal{L}(\mathcal{D}_2)$.

$\mathcal{L}(\mathcal{D}_2) \subseteq \mathcal{L}_2(\mathcal{D})$. Sea $w \in \mathcal{L}(\mathcal{D}_2)$. Se cumplirá entonces

$$(q_0, w) \vdash^*_{\mathcal{D}_2} (q_F, \epsilon)$$

Sabemos que, por construcción, la única forma de llegar al estado q_F es pasando primero por la etapa de simulación, con q_S al tope del stack. Por lo tanto, la ejecución tendrá la siguiente forma:

$$(q_0, w) \vdash^*_{\mathcal{D}_2} (\gamma, \epsilon) \vdash_{\mathcal{D}_2} (q_S\gamma, \epsilon) \vdash^*_{\mathcal{D}_2} (q_S q_f, \epsilon) \vdash_{\mathcal{D}_2} (q_F, \epsilon)$$

Como todas las transiciones que no involucran a q_S, q_F provienen de Δ, sabemos que

$$(q_0, w) \vdash^*_{\mathcal{D}} (\gamma, \epsilon)$$

Y como todos los pasos en la etapa de simulación dependen de la existencia de un camino desde (γ, ϵ) hasta q_f en $\mathcal{D}$, podemos afirmar que existe algún $v \in \Sigma^*$ tal que

$$(q_0, wv) \vdash^*_{\mathcal{D}} (q_f, \epsilon)$$

Por lo que $w \in \mathcal{L}_2(\mathcal{D})$. Finalmente, $\mathcal{L}_2(\mathcal{D}) = \mathcal{L}(\mathcal{D}_2)$.

Distribución de puntaje

- 1 punto por definir un nuevo PDA alternativo $\mathcal{D}_2$ a partir de $\mathcal{D}$.
- 1 punto por definir Δ_2 y agregar transición hacia etapa de simulación con un estado nuevo q_s.
- 1 punto por agregar correctamente las transiciones de la etapa de simulación y un estado final q_F.
- 1 punto por demostrar que $\mathcal{L}_2(\mathcal{D}) = \mathcal{L}(\mathcal{D}_2)$.

Pregunta 2

Una gramática libre de contexto $\mathcal{G} = (V, \Sigma, P, S)$ se dice lineal si todas sus producciones son de la forma:

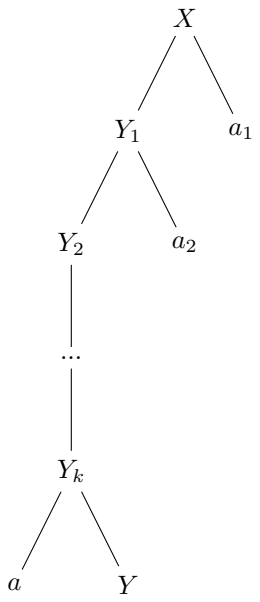
$$X \to aY, \quad X \to Ya \quad \text{o} \quad X \to a$$

con $X, Y \in V$ y $a \in \Sigma$.

1. Demuestre un algoritmo que dado una gramática $\mathcal{G} = (V, \Sigma, P, S)$ y una variable $X \in P$, calcule el conjunto first $_1(X)$ en tiempo $O(|V| + |P|)$.
2. Demuestre un algoritmo que dado una gramática $\mathcal{G} = (V, \Sigma, P, S)$ y una variable $X \in P$, compute el conjunto follow $_1(X)$ en tiempo $O(|V| + |P|)$.

Solución

Problema 2.1. La idea principal del algoritmo la podemos ver en la siguiente figura que muestra la forma que tendrá una derivación de X si la gramática G es lineal, en donde, nos interesa obtener todas las posibles letras que se pueden generar en el lado mas a la derecha de las hojas del arbol. En este ejemplo tenemos que la letra a cumple con estar en $\text{first}_1(X)$ ya que es la primera letra puede generar en la posición en la que se encuentra.



En base a esta idea del arbol, hacemos el siguiente algoritmo:

Algorithm 1: CalculateLinealFirst1(X, G)

Input: $\mathcal{G} = (V, \Sigma, P, S), X \in P$
Output: F : Array con $first_1(X)$

```
1  N = {X} // Cola de no marcados
2  F = ∅ // Array con los elementos de first_1(X)
3  M = ∅ // Array con elementos marcados
4  while Y = N.pop() do
5  |  M = M ∪ {Y};
6  |  foreach Y → aZ ∈ P o Y → a ∈ P do
7  |  |  F = F ∪ {a}
8  |  foreach Y → Za ∈ P do
9  |  |  if Z ∉ M then
10 |  |  |  N = N ∪ {Z}
11 return F;
```

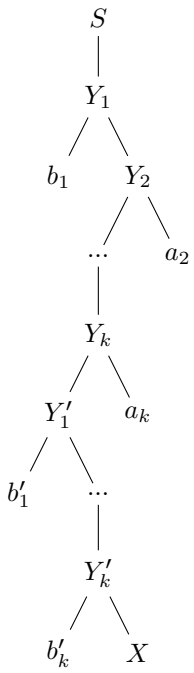
La idea principal es tener una cola N de donde se mantienen todas variables alcanzadas desde X No marcadas. De N se van sacando las variables y se agregan a M como Marcadas (estás variables no volverán a N. Entonces, para cada variable Y en N, si es que se tiene una producción en Y de la forma $Y \to aZ \in P$ o $Y \to a \in P$, quiere decir que la letra a pertenece efectivamente al $\text{first}_1(X)$ ya que pertenece a la parte de más a la izquierda del arbol, por lo que se agrega la letra al array F. En otro caso, si la producción es de la forma $Y \to Za$, si es que no se ha visitado Z (esto es, no está en M) entonces debemos dejarla en la cola N de las variables no marcadas, para seguir explorando dicha variable (en el futuro) y ver si es que encontramos otra letra candidata a pertenecer al conjunto $\text{first}_1(X)$. Finalmente, una vez que se han recorrido todas las variables, el algoritmo termina y se retorna F que contiene el conjunto buscado.

Para la complejidad, como cada variable Y y cada una de sus reglas se revisa a lo más una vez en el peor caso, entonces tenemos que la complejidad de tiempo es $\mathcal{O}(|V| + |P|)$.

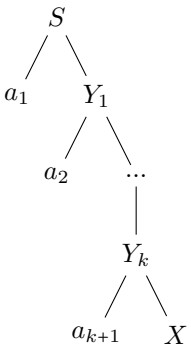
Distribución de puntaje

- 1 punto por considerar la idea de la estructura del arbol de X
- 1 punto por considerar el caso $Y \to aZ \in P$ o $Y \to a \in P$
- 1 punto por considerar el caso $Y \to Za \in P$
- 1 punto por conseguir la complejidad requerida

Problema 2.2. Al igual que para el problema anterior, podemos establecer una idea con el arbol que se genera. En este caso, como buscamos el $\text{follow}_1(X)$, debemos considerar la totalidad del arbol. Sea S la variable inicial de la gramática libre de contexto, entonces podemos ver que un árbol de derivación desde S de una gramática lineal tendrá la siguiente forma. Las letras que son parte del $\text{follow}_1(X)$ son todas las que pueden ir directamente después de la palabra que se genera con X. Si es que vemos el arbol, este caso ocurre para todas las letras a_k, que son las que se generan a la derecha de X.



Otro caso que se puede considerar es cuando ocurre lo del siguiente arbol, en donde se puede dar el caso de que no se generen letras que estén en el lado derecho de X, por lo que en este caso, hay que considerar el "end of file" (#) como parte del conjunto follow$_1(X)$.



En base a esta idea del arbol, hacemos el siguiente algoritmo:

Algorithm 2: CalculateLinealFollow1(X, G)

Input: $\mathcal{G} = (V, \Sigma, P, S), X \in P$
Output: F : Array con follow$_1(X)$

```
1  N = {X} // Cola de no marcados
2  W = ∅ // Array con los elementos de follow_1(X)
3  M = ∅ // Array con elementos marcados
4  while Y = N.pop() do
5  |  M = M ∪ {Y};
6  |  foreach Z → Ya ∈ P do
7  |  |  W = W ∪ {a}
8  |  foreach Z → aY ∈ P do
9  |  |  if Z ∉ M then
10 |  |  |  N = N ∪ {Z}
11 if S ∈ M then
12 |  W = W ∪ {#}
13 return W;
```

Este algoritmo es similar al del problema anterior, solamente que ahora recorremos el árbol desde X hacia arriba, es decir desde X hasta encontrar un nodo como Y_k. Si es que la variable Y que se esta revisando cumple

con que existe una variable Z tal que $Z \to Ya \in P$, entonces sabemos que a pertenece al $\text{follow}_1(X)$ dado que puede venir directamente después de lo que se genera en X. Similarmente, si es que existe una variable Z tal que $Z \to aY \in P$, entonces, si es que Z no se ha visitado, la agregamos al conjunto de variables no marcadas N, para revisarla después y ver si se puede encontrar otra letra que corresponda al conjunto que buscamos. Finalmente, si es que $S \in M$, esto implica a que, para que S haya tenido que ser parte de las variables que se incluyen en el array de los elementos no marcados, entonces se puede establecer una estructura como la del segundo árbol que se dibujó mas arriba, por lo que tiene que incluirse al "end of file" (#) dentro del conjunto.

Similarmente al algoritmo anterior, para la complejidad, como cada variable Y y sus producciones se revisan a lo más una vez en el peor caso, entonces tenemos que la complejidad de tiempo es $\mathcal{O}(|V| + |P|)$.

Distribución de puntaje

- 1 punto por considerar la idea de la estructura del arbol de S
- 1 punto por considerar el caso $Z \to Ya \in P$ y $Z \to aY \in P$
- 1 punto por considerar el caso $S \in M$
- 1 punto por conseguir la complejidad requerida

Evaluación y puntajes de la tarea

Cada item de cada pregunta se evaluará con un puntaje de 0, 1, 2, 3 o 4 puntos. Todas las preguntas tienen la misma ponderación en la nota final y cada item tiene la misma ponderación en cada pregunta.