Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2223 – Teoría de Autómatas y Lenguajes Formales
Segundo semestre de 2024
Profesor: Cristian Riveros
Ayudante: Amaranta Salas

Ayudantia 13

Apiladores

Problema 1

1. Demuestre un autómata apilador para el siguiente lenguaje:

$$L_1 = \{w \in \{0,1\}^* \mid 2 \cdot |w|_0 = |w|_1\}$$

dónde $|w|_a$ representa el número de a-letras en w. Explique su correctitud.

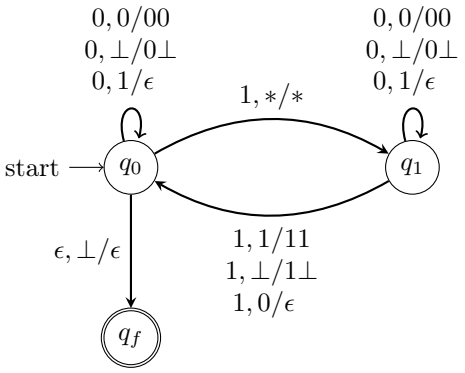
2. Para una palabra $w \in \{0,1\}^+$ sea $bin(w) \in \mathbb{N}$ el número natural que representa la palabra binaria w, donde los bits más significativos son los primeros. Demuestre un autómata apilador para el siguiente lenguaje:

$$L_2 = \{u\#v \mid u, v \in \{0,1\}^+ \text{ y } bin(v^r) = bin(u) + 1\}$$

donde v^r es el reverso de v (esto es, si $v = a_1 \dots a_n$, entonces $v_r = a_n a_{n-1} \dots a_1$). Explique su correctitud.

Solución

1. $L_1 = \{w \in \{0,1\}^* \mid 2 \cdot |w|_0 = |w|_1\}$



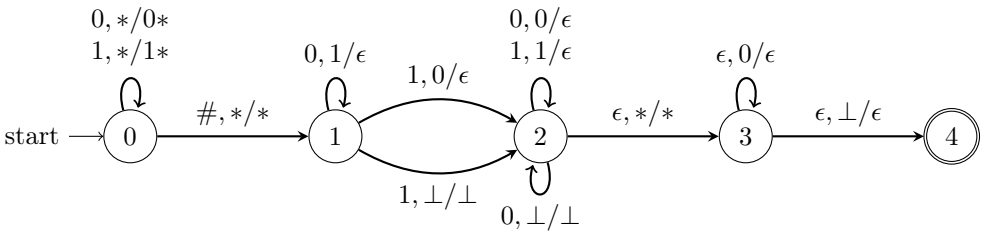
Como se busca aceptar palabras que tengan el doble de 1's que 0's, el autómata tiene los estados q_0 y q_1 que simbolizan que se ha leído una cantidad par o impar de 1's hasta el momento. La idea es que al leer un primer 1 no se realiza ninguna modificación sobre el stack, pero al leer un segundo 1 si se realizan modificaciones.

Cuando se lee un 0, desde q_0 o q_1, se mantiene en el mismo estado. Si hay un 1 en el tope del stack, se elimina del stack. Si hay cualquier otra cosa en el tope del stack, se pushea un 0 al stack.

Al leer un primer 1, se pasa al estado q_1. Ahí, al leer un segundo 1, si hay un 0 en el tope, se elimina y se pasa a q_0. Si hay cualquier otra cosa, se pushea un 1 al stack y se pasa a q_0.

De esta forma, toma dos veces leer un 1 para escribirlo al stack, y al leer un 0 se borra, manteniendo la relación de que por cada 0 deben haber dos 1s en la palabra. Por otro lado, si hay un 0 en el stack, toma leer dos veces un 1 para eliminarlo. Si el stack queda vacío en q_0 luego de leer todo el input, se cumple la condición para aceptar la palabra, y se puede pasar al estado final.

2. $L_2 = \{u\#v \mid u, v \in \{0,1\}^+ \text{ y } bin(v^r) = bin(u) + 1\}$



Para entender el autómata, hay que entender cómo funciona la suma en números binarios. Al sumar un 1 a un número binario, se empieza revisando de los bits menos significativos. Si un bit es un 0, entonces se convierte en un 1 y se termina la suma. Si un bit es un 1, entonces se convierte en un 0, y se pasa un carry al siguiente bit. Por ejemplo, si al número 1010 se le suma un 1, en el primer bit termina la suma, quedando como resultado es 1011. Ahora si a 1011 se le suma un 1, hay carry dos veces, quedando como resultado 1100.

La idea del autómata, es empezar en el estado 0 leyendo solamente la palabra u, y agregando las letras al stack. Cuando se encuentra el # separador, entonces se pasa al estado 1.

En este estado, se empieza a leer la palabra v. Como se espera que $bin(v^r)$ sea igual a $bin(u) + 1$, debemos seguir la idea del algoritmo de la suma. Si leemos una letra 0 en v, entonces esperamos que haya una letra 1 en el stack, y como se sigue con un carry, nos mantenemos en el mismo estado. Solo cuando se lee un 1 en v, y el stack tiene un 0 o está vacío, podemos seguir al estado 2.

En este estado la suma ya terminó de ser ejecutada, por lo que el resto del input debiera ser exactamente igual al resto del stack. Por lo tanto, al leer una letra se espera la misma letra en el stack. En el caso que la palabra u ya haya terminado, pero aún quede v por leer, entonces todas las letras restantes de v deberían ser solamente 0's para mantener el mismo valor. Por esto hay una transición que lee 0, y mantiene el bottom del stack. Una vez se termina de leer v, se pasa al estado 3.

En este estado ya no se lee más input, pero se verifica la posibilidad de que la palabra u haya tenido más bits que la palabra v. En este caso, se espera que todos los elementos restantes en el stack sean letras 0. Una vez se llega al final del stack, se pasa al estado final, completando la ejecución.

Problema 2

La notación polaca de una sentencia φ en lógica proposicional se define recursivamente como:

$$\begin{aligned}
\texttt{np}(0) &= 0 \\
\texttt{np}(1) &= 1 \\
\texttt{np}(\neg\varphi) &= \neg \cdot \texttt{np}(\varphi) \\
\texttt{np}(\varphi_1 \wedge \varphi_2) &= \wedge \cdot \texttt{np}(\varphi_1) \cdot \texttt{np}(\varphi_2) \\
\texttt{np}(\varphi_1 \vee \varphi_2) &= \vee \cdot \texttt{np}(\varphi_1) \cdot \texttt{np}(\varphi_2) \\
\texttt{np}([\varphi]) &= [\cdot \texttt{np}(\varphi) \cdot]
\end{aligned}$$

Donde $\cdot$ representa la concatenación y el alfabeto es $\Sigma = \{0, 1, \neg, \wedge, \vee, [,]\}$. Por ejemplo, la sentencia $\varphi = \neg[0 \vee 1] \wedge 1$ se escribe como $\wedge\neg[\vee 01]1$. Considere el lenguaje de todas las sentencias en lógica proposicional que se evalúan verdadero:

$$L = \{\texttt{np}(\varphi) | \varphi \text{ es una sentencia en lógica proposicional y } \varphi \equiv 1\}$$

Construya un autómata apilador alternativo para L y explique la correctitud de su construcción.

Solución

Para entender mejor el lenguaje, podemos comenzar mirando la fórmula $\varphi = ((0 \wedge 1) \vee 1)$, que en notación polaca se escribe como $\varphi = [\vee[\wedge 01]1]$, y evaluarla de la siguiente manera:

$$\varphi = [\vee[\wedge 01]1] \equiv [\vee[0]1] \equiv [\vee 01] \equiv [1] \equiv 1$$

Luego, como sabemos que todas las fórmulas del lenguaje tendrán valor 1, podemos notar que esto se comporta de forma muy similar a una gramática, pues si vemos el proceso de derecha a izquierda, estamos reemplazando el 1 inicial por fórmulas equivalentes a 1 y repitiendo el proceso hasta llegar a la fórmula completa.
La pseudogramática que describirá este proceso, con variable inicial 1, es:

$$\begin{aligned}
1 &\rightarrow [1] \mid \vee 01 \mid \vee 10 \mid \vee 11 \mid \wedge 11 \mid \neg 0 \\
0 &\rightarrow [0] \mid \wedge 01 \mid \wedge 10 \mid \wedge 00 \mid \vee 00 \mid \neg 1
\end{aligned}$$

Usando lo anterior, podemos seguir la construcción vista en clases para convertir gramáticas en PDA, encontrando $\mathcal{D} = (\{q_0, q_f, 0, 1\}, \Sigma, \Delta, q_0, \{q_f\})$, con:

$$\begin{aligned}
\Delta = &\{(q_0, \varepsilon, 1q_f)\} \cup \\
&\{(a, \varepsilon, \wedge a_1 a_2) \mid a_1, a_2 \in \{0,1\} \wedge a = a_1 \wedge a_2)\} \cup \\
&\{(a, \varepsilon, \vee a_1 a_2) \mid a_1, a_2 \in \{0,1\} \wedge a = a_1 \vee a_2)\} \cup \\
&\{(a_1, \varepsilon, \neg a_2) \mid a_1, a_2 \in \{0,1\} \wedge a_1 = \neg a_2)\} \cup \\
&\{(a, \varepsilon, [a]) \mid a \in \{0,1\}\} \cup \\
&\{(a, a, \varepsilon) \mid a \in \Sigma\}
\end{aligned}$$

Alternativamente, podemos convertir la pseudogramática en una CFG real separando las terminales de las variables, y luego hacer la conversión. El PDA encontrado será ligeramente diferente, pero ambos definirán el mismo lenguaje. La gramática será $\mathcal{G} = (\{U, C\}, \Sigma, P, U)$, con P:

$$U \rightarrow [U] \mid \vee CU \mid \vee UC \mid \vee UU \mid \wedge UU \mid \neg C \mid 1$$
$$C \rightarrow [C] \mid \wedge CU \mid \wedge CU \mid \wedge CC \mid \vee CC \mid \neg U \mid 0$$

Problema 3

Sea $\mathcal{D} = (Q, \Sigma, \Delta, q_0, F)$ un autómata apilador alternativo. Sea R una expresión regular sobre Q, esto es, $\mathcal{L}(R) \subseteq Q^*$. Decimos que D *acepta una palabra* $w \in \Sigma^*$ *por* R*-stack* si existe una ejecución de $\mathcal{D}$ sobre w que empieza en la configuración inicial (q_0, w) y termina en una configuración (γ, ϵ) con $\gamma \in \mathcal{L}(R)$. Se define el lenguaje aceptado por R-stack como:

$$\mathcal{L}_R(\mathcal{D}) = \{w \in \Sigma^* \mid \mathcal{D} \text{ acepta } w \text{ por } R\text{-stack}\}.$$

En otras palabras, $\mathcal{D}$ acepta w por R-stack si existe una ejecución tal que el contenido del stack de la última configuración satisface la expresión regular R. Por ejemplo, si $F = \{p_1, \ldots, p_k\}$ y $R = p_1 + \cdots + p_k$ entonces es fácil ver que $\mathcal{L}(\mathcal{D}) = \mathcal{L}_R(\mathcal{D})$.
Demuestre que para todo autómata apilador alternativo $\mathcal{D}$ y para toda expresión regular R, existe un autómata apilador alternativo $\mathcal{D}'$ tal que $\mathcal{L}_R(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$.

Solución

Sean $\mathcal{D} = (Q, \Sigma, \Delta, q_0, F)$ el autómata apilador alternativo definido en el enunciado y $\mathcal{A}_\mathcal{R} = (Q_R, Q, \Delta_R, q_0^R, F_R)$ el autómata que define el mismo lenguaje que la expresión regular R.
Una posible solución para esta pregunta es el autómata apilador alternativo $\mathcal{D}' = (Q', \Sigma, \Delta', q_0', F')$, donde:

- $Q' = Q \uplus Q_R$
- $q_0' = q_0$
- $F' = F_R$
- $\Delta' = \Delta \cup \{(p, \epsilon, q_0^R q) \mid q \in Q\} \cup \{(p_1 q, \epsilon, p_2) \mid (p_1, q, p_2) \in \Delta_R\}$

Luego se demuestra que $\mathcal{L}_R(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$.

- $\mathcal{L}_R(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{D}')$
  Sea $w \in \mathcal{L}_R(\mathcal{D})$ y sea una ejecución de aceptación de $\mathcal{D}$:

  $$(q_0, w) \vdash^* (\gamma, \epsilon)$$

  Además, sabemos que la pila $\gamma = q_1 ... q_n$ es aceptada por $\mathcal{L}(\mathcal{A}_R)$, por lo que existe una ejecución de aceptación:

  $$q_0^R \xrightarrow{q_1} p_1 \xrightarrow{q_2} ... \xrightarrow{q_n} p_n$$

  Entonces, podemos unir ambas ejecuciones en una ya que $\mathcal{L}_R(\mathcal{D})$ ejecuta ambos autómatas para aceptar a w, lo que nos queda que:

  $$(q_0, w) \vdash^* (q_1 ... q_n, \epsilon) \vdash (q_0^R q_1 ... q_n, \epsilon) \vdash^* (p_n, \epsilon)$$

  Si nos fijamos, la ejecución que queda corresponde a la de la definición de $\mathcal{L}(\mathcal{D}')$, por lo tanto queda demostrado que $\mathcal{L}_R(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{D}')$.

- $\mathcal{L}(\mathcal{D}') \subseteq \mathcal{L}_R(\mathcal{D})$
  Sea la ejecución de aceptación de $\mathcal{D}'$:
  $$(q_0, w) \vdash^* (q_f^R, \epsilon)$$
  Por la construcción del autómata $\mathcal{D}'$, podemos reescribir la ejecución con pasos intermedios, ya que sabemos que en algún momento de la ejecución, cuando se acaba la palabra y deja de ejecutar las transiciones de $\mathcal{D}$, pone un q_0^R al inicio del stack y luego ejecuta las transiciones correspondientes al autómata $\mathcal{A}_R$:
  $$(q_0, w) \vdash^* (q_1...q_n, \epsilon) \vdash (q_0^R q_1...q_n, \epsilon) \vdash^* (q_f^R, \epsilon)$$
  Entonces, como primero ejecuta $\mathcal{D}$ y luego $\mathcal{A}_R$, el lenguaje que acepta w también es $\mathcal{L}_R(\mathcal{D})$ y por lo tanto $\mathcal{L}(\mathcal{D}') \subseteq \mathcal{L}_R(\mathcal{D})$.