Ciencias computacionales

Propedeutico: Autómatas

Propiedades de los Lenguajes Regulares

Contents

1 Propiedades de los lenguajes regulares

Este capítulo explora las propiedades de los lenguajes regulares. Entre los puntos importantes se encuentra el lema de bombeo para proar que ciertos lenguajes no son regulares y la propiedad de cerradura, que nos permiten reconocer lenguajes que han sido construídos a través de otros lenguajes mediante ciertas operaciones. Otros puntos importantes son las propiedades de decisión, que nos permiten decidir si dos autómatas definen el mismo lenguaje. Empezaremos por cómo probar qe un lenguaje no es regular.

1.1 Demostración de lenguajes no regulares

Hasta ahora sabemos que un lenguaje regular se puede describir de 4 formas diferentes: DFA, NFA ϵ-NFA y expresiones regulares. No todos los lenguajes son regulares, y ahora mostraremos como el lema del bombeo permite definir si un lenguaje es regular.

1.2 Lema de bombeo para lenguajes regulares

Teorema 1 *El lema del bombeo Si L es un lenguaje regular, entonces existe una constante n, que depende de L tal que para cada cadena w en L si $|w| \geq n$, entonces podemos separar w en tres cadenas, $w = xyz$, tal que:*

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. *Para todas las $k \geq 0$, la cadena xy^kz está también en L.*

Lo anterior quiere decir que siempre podemos encontrar una cadena y no my lejos del inicio de w que puede ser "bombeada" (o sea repetir y k veces o borrarla en el caso de que $k = 0$, y la cadena resultante sigue estando en el lenguaje L.

1.3 Aplicaciones del lema de bombeo

El lema del bombeo se puede ver como un juego de dos adversarios de la siguiente forma:

1. El jugador 1 elige el lenguaje L que quiere probarse como no regular.
2. El jugador 2 elige n, pero no revela al jugador 2 cuál n es; el jugador 1 debe idear una manera de jugar para todos los posibles valores de n.
3. El jugador 1 elige w, que puede depender de n y que debe de ser mínimo de tamaño n.
4. El jugador 2 divide w en x, y z, obedeciendo las restricciones del lema del bombeo.
5. El jugador 1 gana escogiendo el valor de k, que puede ser una función de n, x, y y z, de tal forma que xy^kz no está en L.

Un ejemplo de lo anterior. Demostraremos que un lenguaje L_{eq} consistente de todas las cadenas con n número igual de 0 y 1 sin importar el órden no es un lenguaje regular. En términos del lema del bombeo como un juego de dos adversarios seremos el jugador 1. Supongamos que n es la constante que debe existir si L_{eq} es regular, de acuerdo al lema del bombeo; el jugador 2 elige n. Nosotros elegimos $w = 0^n1^n$ (n ceros seguido de n unos). Ahora el jugador 2 separa w en xyz. Todo lo que sabemos es que $y \neq \epsilon$, y que $|xy| \leq n$. Sin embargo esta información es muy útil y ganamos de la siguiente forma. Ya que $|xy| \leq n$ y $|xy|$ está al frente de w, sabemos que x y y consiste sólo de ceros. El lema del bombeo nos dice que xz está en L_{eq} si L_{eq} es regular. Esta conclusión es cierta si $k = 0$. SIn embargo xz tiene n unos, ya que todos los unos de w están en z. Pero también xz tiene menos de n ceros porque perdimos los ceros de y. Como $y \neq \epsilon$ sabemos que no hay más de $n - 1$ ceros entre x y z. Por tanto, después de asumir que L_{eq} es un lenguaje regular hemos probado un hecho falso, que xz está en L_{eq}. Tenemos una prueba por contradicción de que L_{eq} no es regular.

2 Propiedades de cerradura de los lenguajes regulares

En esta sección se mostrarán algunos teoremas de la forma "si ciertos lenguajes son regulares, y un lenguaje L está formado a partir de ellos mediante ciertas peraciones, entonces L es regular también".

- Unión. Si L y M son lenguajes regulares, entonces $L \cup M$ también lo es.
- Complemento. Si L es un lenguaje regular sobre el alfabeto Σ, entonces $\bar{L} = \Sigma * -L$ es también un lenguaje regular.
- Intersección. Si L y M son lenguajes regulares, entonces $L \cap M$ es regular tambíen.
- Diferencia. Si L y M son lenguajes regulares, entonces $L - M$ es regular.
- Inversión. Una cadena L se invierte al escribirla al revés para dar lugar a L^R. Si L es regular, L^R también lo es.

2.1 Homomorfismos

Un homomorfismo para una cadena es una función sobre los elementos de la cadena que funciona sustituyendo cada símbolo por una otra cadena. Por ejemplo, si tenemos la cadena 0011 y le aplicamos el homomorfismo descrito por $h(0) = ab$ y $h(1) = \epsilon$ la cadena se transforma en $abab$. Si L es un lenguaje regular sobre un alfabeto Σ y h es un homomorfismo en Σ, entonces $h(L)$ es tambien regular.

2.2 Homomorfismos inversos

Los homomorfismos tambien pueden ser aplicados en sentido inverso y preservar sus lenguajes regulares. Supongamos qe h es un homomorfismo para algún alfabeto Σ de otro alfabeto T posiblemente similar. Suponer que L es el lenguaje del alfabeto T. Entonces $h^{-1}L$, que se lee como "h inversa de L", es el conjunto de cadenas w en $\Sigma*$ tal que $h(w)$ está en L.

Si h es un homomorfismo del alfabeto Σ al alfabeto T, y L es un lenguaje regular sobre T, entonces $h^{-1}L$ es un lenguaje regular también.

3 Propiedades de decisión de los lenguajes regulares

En esta sección se considerará la manera de responder preguntas sobre lenguajes regulares, pero antes hay que notar que un lenguaje puede ser infinito, por lo que responder la pregunta no debe requerir analizar un conjunto infinito de carácteres. En su lugar se plantea la pregunta usando una representación del lenguaje: DFA, NFA, ϵ-NFA o una expresión regular. Antes de comenzar a estudiar las técnicas y algoritmos para respoder preguntas sobre lenguajes regulares con un repaso de las formas en las que se pueden hacer conversiones entre representaciones para un mismo lenguaje regular, poniendo especial atención en la complejidad temporal de los algoritmos que realizan las conversiones.

3.1 Conversión entre representaciones

Sabemos que podemos convertir cualquiera de las cuatro representaciones de lengujes regulares a cualquiera de las otras tres. Si bien hay algoritmos para cualquiera de las conversiones es necesario considerar el tiempo que requieren para procesar una entrada, dado que no nos resultarán útiles, en la mayoría de los casos, algoritmos que requieren tiempos que crecen exponencialmente con el número de entradas. Los algoritmos que requieren tiempos lineales o cuadráticos son realistas si consideramos que requeriremos procesar problemas con un gran número de entradas.

- Convertir NFA a DFA. Esta conversión es simple y requiere un tiempo $O(n)$ para un DFA de n estados. Todo lo que tenemos que hacer es modificar la tabla de transición del DFA colocando los símbolos "" y "" alrededor de los estados (para enunciarlos como conjuntos) y, si la salida es un ϵ-NFA, agregar otra columna para ϵ.

- Convertir autómata a Expresión Regular. La complejidad en tiempo de esta conversión es de $O(n^3 4^n)$ en el peor de los casos.
- Convertir Expresión Regular a Autómata. Esta conversión requiere un tiempo lineal. Para hacer la conversión necesitamos *parsear* la expresión eficientemente para obtener un árbol de expresiones con un nodo para cada símbolo de la expresión regular. Una vez que tenemos el árbol de expresiones podemos subir por una de las ramas construyendo el ϵ-NFA para cada nodo.

3.2 Prueba de vacuidad de lenguajes regulares

Responder a la pregunta de si el lenguaje regular L esta vacío puede parecer trivial, sin embargo, tal como se discutió anteriormente la información que se tiene para responder esta pregunta no es un conjunto explícito de los elementos de L, sino una representación de L, sobre la que tenemos que decidir si denota al lenguaje $\emptyset$. Si la representación es un autómata la respuesta a la pregunta depende de si hay algún camino desde el estado inicial a un estado final. Esta es una tarea de análisis de grafos qe puede resumirse mediante el siguiente algoritmo recursivo.

BASE. el estado inicial es alcanzable desde el estado inicial.

INDUCCIÓN. Si el estado q es alcanzable desde el estado inicial y hay un arco desde q a p con cualquier etiqueta, entonces p es alcanzable.

3.3 Prueba de pertenencia en lenguajes regulares

La siguiente pregunta importante es saber si w (una cadena) está contenida en un lenguaje L. Mientras que w se representa de forma explícita L se representa mediante un autómata o un lenguaje regular. Si L es un DFA el algoritmo es simple, pues sólo hay que simular el DFA procesando los símbolos de entrada w a partir del estado inicial. Si el DFA termina en un estado final la respuesta es "si", de lo contrario es "no". Si el DFA está representado por una estructura de datos eficiente, como un arreglo bidimensioal, entonces cada transición requiere tiempo constante y el proceso entero tiene una complejidad de tiempo de $O(n)$.

Si L está representado de cualquier otra forma que no sea un DFA, podríamos convertirlo a un DFA y correr el proceso anterior. Esta aproximación requeriría de tiempo exponencial en el tamño de la representación, aunque sería lineal en $|w|$. Si la representación es un NFA o un ϵ-NFA es más eficiente simular el NFA directamente, proceso que tendría una complejidad en tiempo de $O(ns^2)$. Si el NFA tiene transiciones epsilon estas deben de ser procesadas antes de comenzar la simulación, lo que nos genera una complejidad de $O(s^2)$.

4 Equivalencia y minimización de autómatas

Esta sección ahora tratará el problema de decidir si dos descripciones de dos lenguajes regulares deifnen realmente el mismo lenguaje.

4.1 Prueba de equivalencia de estados

Decimos que dos estados p y q son equivalentes si para todas las entradas w, $\hat{\delta}(p, w)$ es un estado final si y solo si $\hat{\delta}(q, w)$ es un estado final también. Si dos estados no son equivalentes entonces decimos que sin *distinguibles*. Lo que quiere decir que el estado p es distinguible del esado q si exite al menos una cadena w tal que uno de $\hat{\delta}(p, w)$ y $\hat{\delta}(q, w)$ es un estado final y el otro no.

Para encontrar los estados que son equivalentes es necesario hacer un esfuerzo por encontra r los pares de estados que son distinguibles. Si esta tarea se realiza correctamente mediante el algoritmo llamado "llenado de tabla", entonces cualquier par de estados que no encontremos es equivalente. Este algoritmo consiste en un descubrimiento recursivo de pares distinguibles en un DFA $A = (Q, \Sigma, \delta, q_0, F)$.

BASE: si p es un estado final y q es no final, entonces el par $\{p, q\}$ es distinguible.

INDUCCIÓN: p y q son estados que ara algún símbolo de entrada a, $r = \hat{\delta}(p, a)$ y $s = \hat{\delta}(q, a)$ son un par de estados que se sabe son distinguibles. Entonces $\{p, q\}$ son un par de estados distinguibles. La razón por la que esta regla tiene sentido es que debe de existir alguna cadena w que distingue entre r y

s; lo que significa que uno de $\hat{\delta}(r, w)$ y $\hat{\delta}(s, w)$ es final. Entonces la cadena aw debe distinguir p de q ya que $\hat{\delta}(p, aw)$ y $\hat{\delta}(q, w)$ son el mismo par de estados que $\hat{\delta}(r, w)$ y $\hat{\delta}(s, w)$. De lo anterior se desprende el siguiente teorema.

Teorema 2 *Si dos estados no se distinguen por el algoritmo de llenado de tabla, entonces los estados son equivalentes.*

La figura 2 muestra el resultado del algoritmo de llenado de tabla sobre el autómata de la figura 1. En la tabla las x indican los pares de estados distinguibles, y los cuadros en blanco indican que ese par es equivalente. Inicialmente no hay x en la tabla.

Para lel caso base, ya que C es el único estado final marcamos con x cada par que incya a C. Ahora que tenemos algunos estados distinguibles podemos descubrir otros. Por ejemplo, ya que $\{C, H\}$ es distinguible, y los estados E y F van a H y C respectivamente al recibi la entrada 0, entonces sabemos que $\{E, F\}$ es también un par distinguible. Todos los pares de la tabla se pueden encontrar de manera similar a excepción de $\{A, G\}$. Para encontrar este par hay que observar que al recibir la entrada 1 pasan a ser F y E respectivamente, y hemos establecido con anterioridad que $\{E, F\}$ son distingibles.

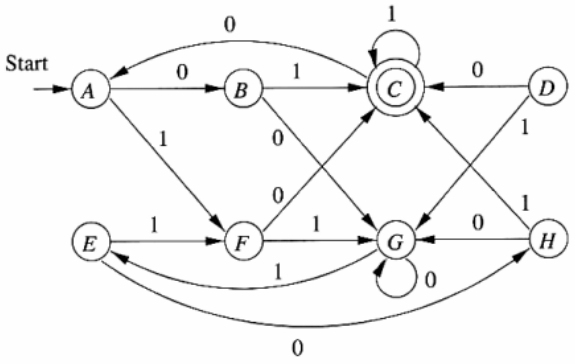

Figure 1: Un autómata con estados equivalentes.

	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

Figure 2: Tabla de inequivalencias de estados.

4.2 Prueba de equivalencia de lenguajes regulares

El algoritmo de llenado de tabla nos provee de una forma sencilla de saber si dos lenguaje regulares son el mismo. Supongamos que los lenguajes L y M se representan de alguna forma y se convierten a un DFA. Ahora imaginemos que uno de los DFA cuyos estados son la unión de los estados de los DFA para L y M. Técnicamente este DFA tiene dos estados iniciales, pero realmente el estado inicial es irrelevante desde el punto de vista de la eqivalencia, así qe podemos hacer a cualquiera el estado inicial. El siguiente

paso es probar si los estados iniciales de los dos DFA son equivalentes usando el algoritmo de llenado de tabla. Si lo son, entonces $L = M$, de lo contrario $L \neq M$.

4.3 Minimización de DFA

Otra consecuencia importante de la prueba de equivalencia de estados es qe podemos minimizar los DFA. Esto significa que por cada DFA podemos encontrar otro DFA equivalente que tenga tan pocos estados como cualquier otro DFA que acepte el mismo lenguaje. Los siguientes teoremas nos permitirán definir un algoritmo para minimizar DFA.

Teorema 3 *La equivalencia de estados es transitiva, lo que quiere decir que si en algún DFA* $A = (Q, \Sigma, \delta, q_0, F)$ *encontramos que los estados p y q son equivalentes, y encontramos tambien que los estados q y r son equivalentes, entonces p y r tienen que ser equivalentes.*

Teorema 4 *Si creamos para cada estado q de un DFA un bloque qe consista de q y de todos sus estados equivalentes, entonces los diferentes loques de estados dorma una partición del conjunto de estados. Esto es, cada etado está exactamente en un bloque. Todos los miembros del bloque son equivalentes, y ningún par de los estados escogidos de diferentes bloques son equivalentes.*

Con esta información el algoritmo de minimización de n DFA $A = (Q, \Sigma, \delta, q_0, F)$ es como sigue:

1. Usar el algoritmo de llenado de tabla para encontrar todos los pares de estados equivalentes.
2. Particionar el conjunto de estados Q en bloques de estados mutuamente equivalentes.
3. Construir el DFA minimizado usando los bloqes como estados.

4.4 Porque no se puede vencer a un DFA minimizado

Supongamos que tenemos un DFA A y lo minimizamos para construir un DFA M, sando el método de participnado del teorema 4. Este teorema indica que no podemos agrupar los estados de A en menos grupos y mantener un DFA equivalente. Sin embargo, ¿podría haber otro DFA N, no relacionado con A, que acepte el mismo lenguaje que A y M y que tenga menos estados que M)? El siguiente teorema nos indica que no es posible.

Teorema 5 *Si A es un DFA y M es un DFA construido a partir de A mediante el algoritmo descrito en el teorema 4, entonces M tiene menos estados que cualquier DFA equivalente a A.*

[1]

References

[1] George B. Dantzig. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1(2):43–48, 1982.