Discreta s2e04

Hasta ahora hemos estado viendo propiedades de los lenguajes regulares y diferentes maneras de caracterizar qué cosas se pueden hacer con ellos. Una vez que vimos la definición de autómata finito determinista y su equivalencia con autómata finito no determinista, descubrimos todo un mundo de posibilidades con los lenguajes regulares. Ahora, de repente, casi todas las operaciones interesantes entre conjuntos, si estos conjuntos son lenguajes regulares, resultan ser regulares. Tenemos la unión, la intersección, el complemento y muchas operaciones más. Veremos que casi cualquier forma algorítmica que tengamos para combinar una cantidad finita de lenguajes regulares nos dará un lenguaje regular como resultado.

Por lo tanto, esto nos lleva a una pregunta interesante: si estamos ampliando el conjunto de lenguajes regulares de adentro hacia afuera, ¿podemos acotar de afuera hacia adentro el conjunto de lenguajes regulares y establecer un límite? Decir que definitivamente un lenguaje regular no puede ser expresado por un autómata finito determinista.

Intuitivamente, debe haber limitaciones para los lenguajes regulares, ya que sabemos que cada lenguaje es equivalente a un problema en ciencia de la computación, y los autómatas finitos tienen una restricción fundamental: tienen una cantidad de memoria finita. Sabemos que hay programas que no pueden resolverse con una cantidad de memoria finita, e intuitivamente, uno de los problemas más simples que no se puede resolver con memoria finita es contar hasta un número específico. Entonces, cualquier lenguaje equivalente al problema de contar hasta un número N, donde N es un valor arbitrario, intuitivamente no debe ser resoluble por un autómata finito determinista, ya que no tendríamos suficiente memoria para contar hasta un número arbitrariamente alto.

El objetivo de esta conferencia es formalizar esta idea y hacerla lo más rigurosa posible. Veremos efectivamente un primer resultado limitante en teoría de la computabilidad que será central en todo el curso. Este resultado nos dirá que hay ciertas cosas que ciertos modelos de cómputo no pueden hacer. Descubriremos que existe un conjunto de problemas que un modelo de cómputo específico no puede resolver.

La forma en que abordaremos esto para los lenguajes regulares no solo nos proporcionará una herramienta práctica para demostrar que ciertos lenguajes que presentan ciertos problemas no pueden resolverse con autómatas finitos deterministas, y que se necesita un modelo de cómputo más potente, sino que también nos brindará una forma de pensar sobre los modelos de cómputo. Esto nos permitirá, más adelante, realizar demostraciones similares para modelos de cómputo más poderosos, incluida la máquina de Turing.

Nuestro primer lenguaje no regular

"Comencemos buscando un problema que, intuitivamente, no debería ser un lenguaje regular. El lenguaje más simple y evidente que no debe ser regular es el que consiste en cadenas de la forma $a^n b^n$.

En otras palabras, este lenguaje cuenta la cantidad de 'a' y asegura que sea igual a la cantidad de 'b' en la cadena. La primera pregunta que debemos hacernos es: ¿por qué es interesante este lenguaje? ¿Por qué es relevante conocer la complejidad de problemas como este? La respuesta básica radica en que en prácticamente cualquier problema de computación interesante, nos encontraremos con la necesidad de contar cosas: hacer coincidir variables, contar caminos en un grafo que cumplan ciertas propiedades, contar cadenas de texto en una base de datos que satisfagan ciertos criterios y contar iteraciones, entre otros.

Contar la cantidad de veces que algo ocurre es un problema fundamental en la computación. Este lenguaje es una forma directa de codificar esa

tarea: 'Cuéntame cuántas veces aparece la letra 'a' en una cadena'. Esto es equivalente a decir, el lenguaje de todas las cadenas en las que la cantidad de 'a' sea igual a la cantidad de 'b'.

Cualquier máquina capaz de verificar que la cantidad de un elemento es igual a la cantidad de otro elemento debe ser capaz de contar. Estoy afirmando esto sin proporcionar una demostración formal, pero debería resultar intuitivamente evidente para ustedes.

Por lo tanto, el objetivo de esta conferencia es, en primer lugar, demostrar que este lenguaje no es regular. Para lograrlo, primero intentaremos entender intuitivamente por qué este lenguaje no puede ser regular y cómo podemos obtener una demostración lo suficientemente sólida como para convencernos de que es así. Una vez que tengamos una comprensión clara de esta idea intuitiva, la formalizaremos al máximo.

> Sea $L = \{a^n b^n \mid n \geq 0\}$

¿Es L regular? ¿Cómo podemos demostrar que no existe un autómata A tal que $L_A = L$?

Idea intuitiva

Vamos a comenzar por explicar la demostración intuitiva de que $a^n b^n$ no es un lenguaje regular. Utilizaremos un concepto fundamental en matemáticas discretas, que es el principio del palomar: *Si tengo una cantidad finita de lugares para asignar cosas y tengo más cosas para asignar que lugares disponibles, entonces al menos dos cosas deben asignarse al mismo lugar.*

La idea de la demostración será la siguiente: supongamos que el lenguaje es regular. Esto significa que existe un autómata finito determinista que lo reconoce. Ahora, dime cuál es ese autómata finito determinista que lo reconoce. Este autómata tendrá una cantidad finita de estados, es decir, K estados en total.

Ahora, consideremos una cadena de forma $a^K b^K$, que tiene más 'a' que la cantidad de estados que tiene el autómata. Según el principio del palomar, cuando esta cadena se reconoce en el autómata, debe haber al menos un estado que se visita dos veces durante el proceso de reconocimiento de las á ' antes de ver la primera 'b'. Esto se debe a que hay más 'a' que estados en el autómata, y por lo tanto, debe haber repeticiones de estados.

Ahora, ese estado se encuentra en dos momentos diferentes: una vez cuando procesa el prefijo a^i y otra vez cuando procesa el prefijo a^j. Entre a^i y a^j hay una subcadena no vacía porque el autómata determinista consume un carácter cada vez que cambia de estado.

Dado que el autómata tiene una cantidad finita de estados, no tiene suficiente memoria para llevar un registro preciso de la cantidad exacta de 'a'. Lo que puede hacer el autómata es agrupar las cadenas en familias basadas en las cantidades específicas de 'a' que ha visto en esos estados. No puede identificar una cadena específica en ese estado, solo una familia de posibles cadenas con ciertas cantidades de 'a'.

Por lo tanto, habrá al menos dos cadenas con cantidades diferentes de 'a' para las cuales el autómata hace exactamente lo mismo antes de llegar a la 'b'. El autómata no podrá diferenciar entre estas dos cadenas porque sus estados no pueden identificar con precisión la cantidad de 'a' que se ha visto. Puedo completar ambas con la misma cantidad de 'b' de manera que el autómata haga lo mismo mientras reconoce las 'b'. Esto significa que si el autómata reconoce una de las cadenas, también debe reconocer la otra.

En resumen, puedo construir una cadena que pertenezca al lenguaje $a^n b^n$ de manera que termine en un estado final y luego construir otra cadena que el autómata trate de la misma manera, ya que hace lo mismo al reconocer todas las 'a' y luego todas las 'b'. Esto demuestra que el autómata acepta ambas cadenas, y sin embargo, por construcción, una de ellas no es parte del lenguaje.

Por lo tanto, el autómata no reconoce exactamente el lenguaje, sino que debe reconocer cadenas adicionales. De cierta forma la idea es que el autómata no es suficientemente "preciso" para distinguir exactamente este lenguaje, sino que a lo sumo puede distinguir un lenguaje

Formalizando

- Supongamos que existe A.
- Sea $k = |Q|$ la cantidad de estados de A.
- Analicemos la cadena $a^k b^k$
- Existe un estado q_i que se usa 2 veces.
  - $< q_0, a > \ldots < q_i, a > \ldots < q_i, a >< q_j, w_j > \ldots < q_f, \$ >$
- Piquemos la cadena en la forma:
  - $a^p a^q a^{n-p-q} b^n$
  - donde a^q es el fragmento de cadena que se reconoce entre las dos primeras veces que se pasa por q_i
- Entonces la cadena $w' = a^p a^{n-p-q} b^n$ tiene la siguiente secuencia de transiciones:
  - $< q_0, w_0 > \ldots < q_i, w_p >< q_j, w_{q+1} > \ldots < q_f, \$ >$
- Por tanto, $w' \in L(A)$, pero $w' \notin L$.
- Contradicción, A no reconoce el lenguaje L.

Generalizando

Ahora, se darán cuenta de que si aplicamos este enfoque a cualquier otro lenguaje específico, la idea será la misma. Examinaremos un lenguaje en particular, solicitaremos un autómata y buscaremos una cadena con un formato conveniente para que la parte relevante de esa cadena quede en un ciclo. Luego, podremos tomar ese ciclo y repetirlo 0, 1, 2, 3, 4 veces, todas las veces que deseemos, y construir una cadena que no pertenezca al lenguaje. A este proceso le llamaremos "bombeo". Tomaremos una cadena específica, abriremos un autómata cualquiera y bombearemos esa

cadena cierta cantidad de veces para hacer que crezca una parte de la cadena que debe estar en correspondencia con otra parte. Al bombear una de las partes, logramos que la cadena ya no cumpla con la correspondencia entre ambas partes.

En lugar de tener que seguir todo este proceso para cualquier lenguaje, convertiremos esta forma de demostración, esta plantilla de demostración, en un teorema. De esta manera, ya no será necesario mencionar el autómata ni la cantidad de estados, ni llevar a cabo todo ese análisis del principio del palomar, y demás. Simplemente, podremos extraer todo ese proceso y derivar un teorema que nos permitirá tomar una cadena, bombearla repetidamente, y encontrar el contraejemplo que nos saque del lenguaje. A esto le llamaremos el "lema del bombeo".

Lema del bombeo (regular)

Sea L un lenguaje regular, existe un n (que depende de L), tal que, para toda cadena ω con $|\omega| \geq n$, ω se puede escribir como xyz, tal que:

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. Para todo k, $xy^kz \in L$

> Ver la demostración en pizarra.

Usando el lema del bombeo

Quiero demostrar que un lenguaje no es regular.

- Escojo una cadena ω en función de n.
- Escojo una forma de picar ω en xyz que *no dependa* de n.
- Escojo un k tal que xy^kz no esté en L.

Ejemplos

- Cantidad igual de a y b.
- Paréntesis balanceados.
- 0^n1^m con $n \leq m$.