MAC: Entrega relación 5

David Cabezas Berrido

1. **Demostrar que NL es cerrado para la clausura de Kleene.**

Sea A un lenguaje de NL, probaré que A* (clausura de Kleene de A), el lenguaje formado por concatenaciones de 0 o más palabras de A, también pertenece a NL.

Sea pues M una Máquina de Turing no determinista que acepta A usando espacio O(log(n)), construimos otra máquina no determinista M' que acepte A* y también use espacio logarítmico sobre la longitud de la entrada.

Dada una entrada w, tenemos $w \in A^*$ si y solo si, o bien w es vacía, o existe una descomposición de w en un número finito de subcadenas $w=w_1 \ldots w_m$ tales que $w_i \in A$ para cada i=1,…,m. M' elegirá de forma no determinista una descomposición y comprobará si cada subcadena pertenece a A simulando la ejecución de M sobre la subcadena, cada una de estas simulaciones se hace usando espacio logarítmico porque $A \in NL$.

Si M posee k cintas, M' poseerá k+3: k para simular M (incluye la entrada), una para almacenar la posición donde empieza la subcadena que está comprobando en el momento (N1), otra para almacenar la posición donde acaba (N2) y otra para almacenar en todo momento la posición del cabezal (N3), si intenta leer fuera de la subcadena, debe leer blancos y no modificar las otras subcadenas. Sabemos que M no modifica la cinta de entrada, puesto que usa espacio logarítmico.

El comportamiento de M' sería el siguiente:

1. Si w=ε, **acepta.**
2. N1=N2=1.
3. Se decide de forma no determinista si avanzar el cabezal de la cinta de entrada e incrementar N2 o empezar a comprobar. Si se llega al final, se retrocede hasta la última posición y se comprueba.
   3.1. Si se incrementa, se vuelve al paso 3.
   3.2. Si se decide comprobar, se coloca el cabezal de la cinta de entrada en N1 y se simula el comportamiento de M leyendo la cinta de entrada entre las posiciones N1 y N2. Si en algún momento N3<N1 o N3>N2, M debe comportarse como si leyese un blanco, pero sin modificar lo que hay realmente en la cinta (caracteres de w fuera de la subcadena que estamos comprobando). Si M rechaza, se **rechaza** la entrada.
4. Se borran las cintas auxiliares de M.
5. N1=N2=N2+1
6. Si en N1 no hay un blanco, vuelta al paso 3.
7. Se **acepta** la entrada.

La cinta de entrada no se modifica (ni M ni M' lo hacen). En las tres cintas auxiliares de M' sólo se almacenan posiciones que ocupan O(log(|w|)) casillas. En el resto de cintas (k-1) sólo se utilizan $O(\log(\max(|w_1|,\ldots,|w_m|))) \subset O(\log(|w|))$ casillas, puesto que M usa espacio logarítmico. Por tanto M' usa espacio logarítmico.

Si $w \in A^*$, existirá una descomposición $w=w_1\ldots w_m$ tales que $w_i \in A$ para cada $i=1,\ldots,m$. En la rama no determinista en la que M' selecciona dicha descomposición y M acepta todas las $w_i \in A$ (M también es no determinista) se aceptará w.

Si $w \notin A^*$, no importa la descomposición que seleccione M', ya que siempre existirá una subcadena que no pertenece a A y M la rechazará seguro. Por tanto siempre se rechazará w.

Dado cualquier lenguaje A de NL, hemos probado que A* está en NL. Por tanto NL es cerrado para la clausura de Kleene.

Demostrar que L es cerrado para la clausura de Kleene si y sólo si L=NL

La implicación a la derecha es trivial, puesto que acabamos de probar que NL sí es cerrado para la clausura de Kleene.

La implicación hacia la izquierda es un teorema de Burkhard Monien de 1975. El artículo original es este, pero incluiré una prueba más sencilla que hace uso de que el siguiente problema es NL-completo:

P1: Sea G un grafo dirigido en el que cada arista (i,j) cumple i<j (esto evita por ejemplo que el grafo pueda tener ciclos) y s y t dos de sus nodos. Determinar si existe un camino del nodo s al nodo t.

Al ser un problema NL-completo, si probamos que está en L tendremos NL=L.

Definimos el lenguaje:

B={G#i+1#G#i+2#…#G#j# | hay una arista (i,j) en G}

Claramente B ∈ L, para reconocer si una entrada está en B basta almacenar i+1 y j, decrementar (i+1)-1=i y recorrer G para comprobar si existe una arista (i,j), esto se hace en espacio logarítmico.

Por construcción, tenemos

B*={G#i+1#G#i+2#…#G#j# | hay un camino de i a j en G}, que reconoce el conjunto de instancias positivas del problema P1. Por tanto si L es cerrado para la clausura de Kleene, B* ∈ L y por tanto P1 ∈ L como queríamos demostrar.

2. **Demostrar que todo lenguaje regular está en L.**

Dado cualquier lenguaje regular, existe un autómata finito determinista que lo acepta. Sólo hay que simular cualquier AFD por una MT que sólo use para cada entrada de longitud n espacio O(log(n)).

Dado cualquier autómata finito determinista (Q, A, δ, q_0, F) construimos una máquina de Turing $M = (Q \cup \{q_f\}, A, A \cup \{\#\}, \delta', q_0, \#, q_f)$ donde para cada transición $\delta(p,a)=q$ del autómata tendríamos $\delta'(p,a)=(q,a,D)$, simulamos cómo el autómata va leyendo la palabra de izquierda a derecha sólo cambiando de estado, sin modificar nada en la cinta.

El único inconveniente es que el autómata acepta cuando termina en un estado final y la máquina cuando pasa por uno. Por tanto, debemos añadir las siguientes transiciones a la máquina, que contemplan cuando se termina de leer la palabra en un estado final.

$\delta'(p,\#)=(q_f,\#,D)$ para cada $p \in F$

Como no modificamos la cinta (sólo la leemos), la Máquina de Turing usa espacio O(log(n)), por tanto todo lenguaje regular está en L.

3. **MULT={a✱b✱c tales que a, b y c son números en binario y a × b = c}. Demostrar MULT ∈ L.**

Me basaré en el algoritmo:
https://en.wikipedia.org/wiki/Multiplication_algorithm#Optimizing_space_complexity
Que permite calcular cada uno de los dígitos de una multiplicación usando espacio logarítmico. Primero pondré un ejemplo para ilustrar su funcionamiento:
(denoto a_i, b_i, c_i al dígito i-ésimo empezando por el final de a, b, c, si alguna vez se accede a un dígito posterior a la longitud del número se toma 0)

```
a            1011
b      ×      111
       ----------
            1011      a4b1=1
           10110      a3b2=0
     +    101100      a2b2=1
       ----------
c       1001101      i=4, ci=1, dígito que queremos calcular (hay que coger
                     parejas ajbk tal que j+k=5=i+1)
```

Calcularemos la suma de esas parejas a_jb_k y el acarreo (r_{i-1}) proveniente de las sumas anteriores (de hacer la misma operación con a_jb_k con j+k<i+1), c_i será el último dígito de esa suma, y se generará un acarreo r_i que habrá que tener en cuenta para calcular c_{i+1}. Estas operaciones corresponden a las fórmulas recursivas

Denotando $s_i = \sum_{j+k=i+1} a_j b_k$ tenemos:

$$r_i = E[(r_{i-1} + s_i)/2] \qquad r_0 = 0$$
$$c_i = (r_{i-1} + s_i) \bmod 2$$

Dividir entre 2 y hacer parte entera es simplemente eliminar el último dígito.
Hacer módulo 2 es quedarse con el último dígito.

¿Podemos almacenar los valores necesarios en espacio O(log(n))?

Cada valor s_i puede ser como máximo i+1 (todos los a_j y b_k valen 1), c no puede tener más dígitos que la longitud de la entrada (n): $s_i \leq n$. Por tanto para almacenar un s_i necesitamos espacio O(log(n)). Usemos este hecho para probar por inducción que cada r_i puede almacenarse en espacio O(log(n)) comprobando que $r_i \leq 2n$ para todo i.

$r_0 \leq 2n$ obviamente. Si $r_i \leq 2n$, $r_i + s_{i+1} \leq 3n$ y al dividir entre dos y tomar parte entera obtenemos $r_{i+1} \leq 3n/2 \leq 2n$, por tanto cada r_i se puede almacenar en espacio O(log(n)). Obviamente tampoco hay problema para almacenar un c_i, ya que cada uno consta de un dígito.

El algoritmo que acepta MULT es el siguiente:

1. Para i=1,...,long(c):
   1.1 Calcular s_i
   1.2 Calcular $d=r_{i-1}+s_i$
   1.3 Calcular $c_i=d\%2$
   1.4 Comprobar que c_i coincida con el i-ésimo dígito de c: en caso negativo **rechaza**
   1.5 Calcular $r_i=E[d/2]$
2. Si no ha recorrido por completo a y b: **Rechaza** (Salvo que sólo queden ceros a la izquierda)
3. **Acepta**

Este algoritmo detecta si una entrada pertenece a MULT. Sólo necesita almacenar algunos índices auxiliares como el valor de i y el valor de j para calcular s_i (k=i+1-j), o el dígito de c que compara. Necesita almacenar un sólo valor de s_i a la vez, d, un sólo valor de c_i a la vez (hasta que lo compara) y un sólo valor de r_i a la vez. Todos ellos ocupan O(log(n)) espacio. Como sólo leemos de la entrada, el algoritmo usa espacio O(log(n)) y concluimos que MULT $\in$ L.