Sistemas de reescritura desde el punto de vista de la programación funcional.

Miguel Ángel Porras Naranjo

Sistemas de reescritura desde el punto de vista de la programación funcional.

Miguel Ángel Porras Naranjo

Memoria presentada como parte de los requisitos para la obtención del título de Grado en Matemáticas por la Universidad de Sevilla.

Tutorizada por

Prof. José A. Alonso
Prof. María José Hidalgo

Índice general

Terminación

Como hemos podido comprobar en los anteriores capítulos, es importante que nuestros sistemas de reescritura tengan la propiedad de la terminación. Sin embargo, en la primera sección de este capítulo demostraremos que el problema de decidir si un sistema de reescritura es terminante, es indecidible. Aunque en casos mas restringidos si podremos decidir sobre la terminación del sistema de reescritura. En la segunda sección definiremos los órdenes de reducción, obteniendo una propiedad con ellos para verificar la terminación. En el resto daremos diferentes maneras de definir los órdenes de reducción.

El problema de decisión

La indecibilidad para el caso general

En esta sección vamos a introducir conceptos generales de las Ciencias de la Computación, aplicados a los sistemas de reescritura. Uno de los problemas generales de las Ciencias de la Computación es afirmar si un problema es decidible. Usaremos algunos de sus resultados sobre máquinas de Turing. Supondremos, sin perdida de generalidad, que el modelo es no determinista de una banda finita en ambas direcciones.

Definición 1.1. *Una máquina de Turing no determinista M viene dada por,*

1. *Un alfabeto* $\Gamma := \{s_0, \ldots, s_n\}$ *de símbolos, donde* s_0 *es considerado el espacio en blanco.*
2. *Un conjunto finito* $Q = \{q_0, \ldots, q_p\}$ *de estados.*
3. *Una relación de transición* $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{l, r\}$.

Una maquina de Turing se puede interpretar de la siguiente manera. Imaginemos que tenemos una tira de papel dividida en infinitos cuadrados en ambas direcciones, y en cada cuadrado se encuentra un símbolo del alfabeto. Además de la tira de papel, se encuentra un marcador en uno de los cuadrados. A partir de la relación de transición que escojamos, el marcador se moverá de un lado a otro y cambiará los símbolos del papel.

Un ejemplo de relación de transición es $\{q_1, s_1, q_2, s_2, l\}$. El marcador empieza por el cuadrado inicial de la tira de papel. Si el marcador posee el estado q_1 y en ese cuadrado de papel se encuentra s_1, procedemos a aplicar el algoritmo. Cambiamos el estado del marcador por q_2 y el símbolo del trozo de papel por s_1. A continuación movemos el marcador a la izquierda (si fuera l) o derecha (si fuera r).

Llamaremos a un estado de la computación de la máquina, configuración. En la configuración de una máquina se incluye, la tira de papel, la posición y el estado del marcador. Podemos expresar $K \vdash_M K'$ si la configuración de K' se puede obtener mediante K con una computación de la máquina de Turing M.

De aquí surge el problema de la parada. Nos interesa saber si dado una maquina de Turing M y una configuración de la máquina K, termina. Es decir, que no ocurre $K \vdash_M K_1 \vdash_M K_2 \ldots$.

Volviendo al tema de la reescritura, en nuestro es caso el problema es un poco más difícil. Como podemos aplicar toda regla en cada momento, no tenemos una configuración K como en el problema de la parada. Pero si planteamos el problema suponiendo que no partimos de una configuración inicial K, la pregunta pasa a ser; ¿qué configuraciones K hace que el problema termine o no? A esto se le denota por el problema de la parada uniforme.

Para extrapolar lo que ya sabemos sobre máquinas de Turing a la reescritura, vamos a codificar las configuraciones como términos con una signatura Σ_M.

Definición 1.2. *Sea M una máquina de Turing, definimos*

$$\Sigma_M := \{\overrightarrow{s_0}, \ldots, \overrightarrow{s_n}, \overleftarrow{s_0}, \ldots, \overleftarrow{s_n}\} \cup \{q_0, \ldots, q_p\} \cup \{\overrightarrow{l}, r\}$$

donde cada función es de aridad 1.

A continuación, enunciamos una definición que relaciona términos con máquinas de Turing.

Definición 1.3. *Sea x_0 una variable fija. Un término de configuración en Σ_M es cualquier término de la forma,*

$$\overrightarrow{l}(\overrightarrow{s_{i_k}}(\dots\overrightarrow{s_{i_1}}(q(\overleftarrow{s_{j_1}}(\dots\overrightarrow{s_{j_h}}(\overleftarrow{r}(x_0))\dots)))\dots))$$

donde $k, h \geq 0$, $\{i_1, \dots, i_k, j_1, \dots, j_h\} \subseteq \{0, \dots, n\}$, y $q \in Q$.

Para entender mejor esta definición, debemos hacer las siguientes consideraciones. El marcador se encuentra en s_{j_1} con el estado q. Los elementos que se encuentran a la derecha del marcador son $s_{j_2}, \dots s_{j_h}$, y los elementos que se encuentran a su derecha, son $s_{i_1}, \dots, s_{i_k}$.

Como la tira es infinita, $\overrightarrow{l}$, $\overleftarrow{r}$ nos ayudan a controlar los espacios vacíos. Si durante la computación de la máquina de Turing hiciese falta un espacio en blanco, estas funciones se lo proporcionan.

Acabamos de transformar una máquina de Turing en un término. En la siguiente definición, adaptaremos un sistema de reescritura para una máquina de Turing.

Definición 1.4. *Un sistema de reescritura R_M consiste en las siguientes reglas de reescritura,*

- *Para cada transición $(q, s_i, q', s_j, r) \in \Delta$, R_M contiene la regla,*

  $$q(\overleftarrow{s_i}(x)) \to \overrightarrow{s_j}(q'(x))$$

  Si $i = 0$, entonces R_M contiene la siguiente regla,

  $$q(\overleftarrow{r}(x)) \to \overrightarrow{s_j}(q'(\overleftarrow{r}(x)))$$

- *Para cada transición $(q, s_i, q', s_j, l) \in \Delta$, R_M contiene la regla,*

  $$\overrightarrow{l}(q(\overleftarrow{s_i}(x))) \to \overrightarrow{l}(q'(\overleftarrow{s_0}(\overleftarrow{s_j}(x))))$$

  y para cada $s_k \in \Gamma$ la regla,

  $$\overrightarrow{s_k}(q(\overleftarrow{s_i}(x))) \to q'(\overleftarrow{s_k}(\overleftarrow{s_j}(x)))$$

  Si $i = 0$, entonces R_M contiene una regla adicional,

  $$\overrightarrow{l}(q(\overleftarrow{r}(x))) \to \overrightarrow{l}(q'(\overleftarrow{s_0}(\overleftarrow{s_j}(\overleftarrow{r}(x)))))$$

  y para cada $s_k \in \Gamma$ la regla

  $$\overrightarrow{s_k}(q(\overleftarrow{r}(x))) \to q'(\overleftarrow{s_0}(\overleftarrow{s_j}(\overleftarrow{r}(x))))$$

Con estas reglas podemos llegar a la conclusión de que para todo par de términos de configuración t, t', que verifiquen $t \rightarrow_{R_M} t'$, implica $K_t \vdash K_{t'}$. E igual ocurre en sentido contrario, si tenemos dos configuraciones K, K' y un término de configuración t, si $K \vdash K'$ y $K \equiv K_t$, esto implica que existe un término de configuración t' tal que $K' \equiv K_{t'}$ y $t \rightarrow_{R_M} t'$.

De aquí podemos razonar una propiedad. Como el problema de la parada de las máquinas de Turing es indecidible, y acabamos de probar que una máquina de Turing es equivalente a un sistema de reescritura (con las reglas que hemos pedido), aseguramos que dado un sistema de reescritura finito R y un término t, el problema de ver si todas las reducciones son terminantes empezando por t es indecidible.

Sin embargo, el problema que acabamos de resolver no es el original que queríamos. El problema de la terminación pide que todas las reducciones desde todos los posibles términos sean terminantes. Puede darse el caso que tengamos una configuración de términos que termine, pero una reducción no terminante que empiece por un término que no este en la configuración inicial. Por tanto el problema no esta resuelto todavía. El siguiente lema asegura que este caso no puede ocurrir.

Lema 1.1. Sea t un término de Σ_M. Si existe una reducción $t \rightarrow_{R_M} t_1 \rightarrow_{R_M} t_2 \rightarrow \ldots$, entonces existe un término de configuración t' y una reducción infinita R_M empezando por t'.

Demostración. Vamos a considerar $\Sigma_M = \overrightarrow{\Gamma} \cup \overleftarrow{\Gamma} \cup Q\{\overleftarrow{r}, \overrightarrow{l}\}$, donde $\overrightarrow{\Gamma} := \{\overrightarrow{s_1}, \ldots, \overrightarrow{s_n}\}$ y $\overleftarrow{\Gamma} := \{\overleftarrow{s_1}, \ldots, \overleftarrow{s_n}\}$. Cualquier elemento w de Σ_M puede ser escrito como composición de funciones tal que, $w = u_1 v_1 u_2 v_2 \ldots u_q v_q u_{q+1}$, ya que u_i, v_j son funciones de aridad 1.

Como todas las reglas de reescritura de R_M contienen un símbolo de Q, cualquier reducción aplicada a w, se hace dentro de las funciones v.

Es decir, que si tomamos $w(x) \rightarrow_{R_M} w'(x)$, entonces existe un índice j, $1 \leq j \leq q$, y una palabra $v'_j \in \overrightarrow{\Gamma}^* Q \overleftarrow{\Gamma}^*$ tal que $w' = u_1 v_1 u_2 v_2 \ldots u_j v'_j u_{j+1} \ldots u_q v_1 u_{q+1}$, y que $\overrightarrow{l} v_j \overleftarrow{r}(x_0) \rightarrow_{R_M} \overrightarrow{l} v'_j \overleftarrow{r}$

Como q es finito, esto implica que la reducción infinita que empieza por $w(x)$ produce una reducción infinita empezando por $\overrightarrow{l} v_j \overleftarrow{r}(x_0)$. Como $\overrightarrow{l} v_j \overleftarrow{r}(x_0)$ es un término de configuración, queda demostrado el lema. ∎

Finalmente, por el lema 1.1, obtenemos el objetivo principal de esta sección, la indecibilidad para el problema de terminación de los sistemas de reescritura.

Sistemas de reescritura básicos hacia la derecha

En esta subsección vamos a analizar un SRT particular R, donde sus reglas hacia la derecha son términos básicos. A estos SRT les llamaremos básicos hacia la derecha, y probaremos que son terminantes.

Lema 1.2. Sea R un SRT finito básico hacia la derecha. Entonces son equivalentes,

1. R no termina.
2. Existe una regla $l \to r \in R$ y un término t de manera que $r \xrightarrow{+}_R t$ y t contiene a r como subtérmino.

Demostración. $(2 \Rightarrow 1)$ es cierto pues obtenemos una reducción finita; $r \xrightarrow{+}_R t = t[r]_p \xrightarrow{+}_R t[t]_p = t[t[r]_p]_p \xrightarrow{+}_R \dots$, donde p es la posición $t|_p = r$.

$(1 \Rightarrow 2)$ se demuestra mediante inducción por el cardinal de R.

Suponemos que $|R| > 0 >$ y consideramos una reducción infinita $t_1 \to t_2 \to t_3 \to \dots$. Tenemos que probar que esta cadena no termina. Sin perdida de generalidad al menos una reducción ocurre en la posición ϵ. Esto significa que existe un índice i, una regla $l \to r \in R$, y una sustitución σ, tal que $t_i = \sigma(l)$ y $t_{i+1} = \sigma(r) = r$. Luego hay una reducción infinita $r \to_R t_{i+2} \to_R t_{i+3} \to_R \dots$ que empieza por r.

Si la regla $l \to r$ no es usada en la reducción, entonces aplicamos la inducción a $R - \{l \to r\}$. En el caso en que sí sea usada, eso significa que existe j tal que r ocurre en t_{i+j} y por tanto, obtenemos la segunda proposición. ∎

Luego si R termina entonces $\to_R$ es globalmente finita, ya que es una ramificación finita por el corolario **??** y el lema **??**. Por tanto todas las reducciones terminarían y, al ser finitas, lo hacen en un numero finito de pasos. Con esta idea, obtenemos el resultado clave de esta subsección.

Teorema 1.1. *Para un SRT finito básico hacia la derecha, el problema de la terminación es decidible.*

Órdenes de reducción

En esta sección relacionaremos la definición de orden bien fundado con el problema de la terminación.

En la sección previa, hemos probado que el problema de la terminación es indecidible. Sin embargo podemos dar varios métodos para resolver el problema. Estos métodos no son totalmente automatizados.

La idea básica para probar la terminación, se hace mediante un orden bien fundado. Supongamos que R es un sistema de reescritura finito, $>$ es un orden estricto bien fundado en $T(\Sigma, X)$. Relacionando los sistemas de reescritura con los órdenes, si R es terminante, diremos que $s \to_R t$ implica $s > t$. En vez de decidir $s > t$, tan solo tendremos que comprobar las reglas de R. Para considerar este nuevo acercamiento al problema necesitamos pedir ciertas propiedades a $>$.

Definición 1.5. *Sea Σ una signatura y V un conjunto numerable de variables. Un orden estricto $>$ es un orden de reescritura syss*

1. *Es compatible con las Sigma-operaciones: para todo $s_1, s_2 \in T(\Sigma, V)$, todo $n \geq 0$ y $f \in \Sigma^{(n)}$, $s_1 > s_2$ implica*

$$f(t_1, \ldots, t_{n-1}, s_1, t_{i+1}, \ldots, t_n) > f(t_1, \ldots, t_{n-1}, s_2, t_{i+1}, \ldots, t_n)$$

   $\forall i, 1 \leq i \leq n$ *y* $\forall t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \in T(\Sigma, V)$.
2. *Es cerrado bajo sustituciones: $\forall \sigma \in Sub\,(T(\Sigma, V))$, si $s_1 > s_2$ entonces $\sigma(s_1) > \sigma(s_2)$.*

Un orden de reducción es un orden de reescritura bien fundado.

El siguiente teorema es de especial importancia para los órdenes de reducción.

Teorema 1.2. *Un sistema de reescritura R termina syss existe un orden de reducción $>$ que satisface $l > r$ para toda regla $l \to r \in R$.*

Demostración. ($\Rightarrow$) Suponiendo que R termina, entonces $\xrightarrow{+}_R$ es un orden de reducción ya que satisface $l \xrightarrow{x}_R r$ para todo $l \to r \in R$.

($\Leftarrow$) Como $>$ es un orden de reescritura, para $l > r$ se verifica $t[\sigma(l)]_p > t[\sigma(r)]_p$ para todo término t, sustitución σ y posición p. Como $>$ esta bien fundada, y para toda regla de R, $s_1 \to_R s_2$ implica $s_1 > s_2$, entonces no puede ocurrir una reducción infinita $s_1 \to_R s_2 \to_R s_3 \ldots$.

A partir de aquí podemos dar varios métodos para dar una repuesta al problema. En las secciones posteriores nos centraremos en los ordenes de simplificación, orden de camino lexicográfico y orden de camino recursivo, y daremos una implementación de ambos.

Órdenes de simplificación

En esta sección definiremos que son los órdenes de simplificación y daremos dos órdenes construidos a partir de estos a modo de ejemplo en las siguientes subsecciones. Empezamos por la definición,

Definición 1.6. *Sea $>$ un orden estricto en $T(\Sigma, V)$ es un orden de simplificación, syss es un orden de reescritura que para todo término $t \in T(\Sigma, V)$ y todas las posiciones $p \in Pos\ (t) - \{\epsilon\}$ entonces $t > t|_p$.*

La definición es equivalente a pedir que, para todo $n \leq 1$, todos los símbolos de funciones $f \in \Sigma^{(n)}$, todas las variables $x_1, \ldots, x_n \in V$, tenemos que $f(x_1, \ldots, x_i, \ldots, x_n) > x_i$. Usando esta nueva definición e introduciendo algunos conceptos de álgebra de homomormismos, se puede demostrar que los órdenes de simplificación son equivalentes a los ordenes de reducción para Σ finito.

Órdenes de caminos lexicográficos

La idea de los caminos recursivos reside en comparar las raíces de los términos, y en comparar recursivamente sus subtérminos. Estos subtérminos se pueden comparar mediante multiconjuntos (orden de caminos de multiconjuntos), tuplas (orden de caminos lexicográficos), o una mezcla de ambos (orden de caminos recursivos). Nos interesaremos por estos dos últimos.

Definición 1.7. *Sea Σ una signatura finita y $>$ un orden estricto de Σ. El orden de caminos lexicográficos $>_{ocl}$, se define como, $s >_{ocl} t$ syss ocurre alguno de los siguientes casos,*

- *(OCL1) $t \in Var(s)$ y $s \neq t$ ó*
- *(OCL2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, y*
    - *(OCL2a) existe i, $1 \leq i \leq m$, con $s_i \geq_{lpo} t$ ó*
    - *(OCL2b) $f > g$ y $s >_{ocl} t_j$ para todo j en $1 \leq j \leq n$ ó*
    - *(OCL2c) $f = g$, $s >_{lpo} t_j$ para todo j en $1 \leq j \leq n$, y exista i, $1 \leq i \leq m$ tal que $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$ y $s_i >_{ocl} t_i$*

La definición es recursiva y esta bien definida.

Teorema 1.3. *Para cualquier orden estricto $>$ en Σ, el orden de camino lexicográfico inducido es un orden de simplificación en $T(\Sigma, V)$.*

A continuación implementaremos el orden de caminos lexicográfico en Haskell. Para realizar los ejemplos de esta función, primero implementaremos `ordenPorLista xs a b`, que es el resultado de comparar a y b, tal que $a > b$ syss a aparece antes en xs que b. Por ejemplo,

```
ghci> ordenPorLista ["a","b","c"] "a" "c"
GT
ghci> ordenPorLista ["a","b","c"] "c" "b"
LT
ghci> ordenPorLista ["a","b","c"] "b" "b"
EQ
```

Su código es,

```
ordenPorLista :: Ord a => [a] -> a  -> a -> Ordering
ordenPorLista [] _ _ =
  error("Ninguno de los dos elementos se encuentran
         en la lista")
ordenPorLista (x:xs) a b
  | a == x = if a == b
             then EQ
```

```
              else GT
  | b == x = LT
  | otherwise = ordenPorLista xs a b
```

Definiremos `ordCamLex ord s t`, que es el resultado de comparar *s* y *t* términos, con el orden de caminos lexicográfico inducido por `ord`. Por ejemplo

```
ghci> let ord = ordenPorLista ["i","f","e"]
ghci> ordenCamLex ord (T "f" [V ("x",1), T "e" []]) (V ("x",1))
GT
ghci> ordenCamLex ord (T "i" [T "e" []]) (T "e" [])
GT
ghci> ordenCamLex ord (T "i" [T "f" [V("x",1),V("y",1)]])
                      (T "f" [T "i" [V("y",1)], T "i" [V("x",1)]])
GT
ghci> ordenCamLex ord (T "f" [V("y",1),V("z",1)])
                      (T "f" [T "f" [V("x",1),V("y",1)], V("z",1)])
LT
```

Su código es,

```
 ordenCamLex:: ([Char] -> [Char] -> Ordering)
               -> Termino -> Termino -> Ordering
ordenCamLex _ s (V x)
  | s == (V x) = EQ
  | ocurre x s = GT --OCL1
  | otherwise = LT
ordenCamLex _ (V _) (T _ _) = LT
ordenCamLex ord s@(T f ss) t@(T g ts) --OCL2
  | all (\x -> ordenCamLex ord x t == LT) ss
    = case ord f g of
      GT -> if all (\x -> ordenCamLex ord s x == GT) ts
            then GT --OCL2b
            else LT
      EQ -> if all (\x -> ordenCamLex ord s x == GT) ts
```

```
            then ordLex (ordenCamLex ord) ss ts --OCL2c
            else LT
      LT -> LT
  | otherwise = GT --OCL1a
```

Órdenes de caminos recursivos

Funciones auxiliares para Haskell

Usaremos las siguientes funciones de la libreria Data.List

- `(xs ++ ys)` es la concatenación de `xs` e `ys`. Por ejemplo,

```
> [2,5] ++ [3,7,6]
[2,5,3,7,6]
```

- `(any p xs)` se verifica si algún elemento de `xs` cumple la propiedad `p`. Por ejemplo,

```
> any even [3,2,5]
True
> any even [3,1,5]
False
```

- `(all p xs)` se verifica si todos los elementos de `xs` cumplen la propiedad `p`. Por ejemplo,

```
> all even [2,6,8]
True
> all even [2,5,8]
False
```

- `(null xs)` se verifica si `xs` es la lista vacía. Por ejemplo,

```
> null []
True
> null [3]
False
```

- `(zip xs ys)` es la lista de pares formado por los correspondientes elementos de `xs` e `ys`.

```
> zip [3,5,2] [4,7]
[(3,4),(5,7)]
> zip [3,5] [4,7,2]
[(3,4),(5,7)]
```

Índice alfabético