UNIVERSIDAD DE SEVILLA

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,*

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

donde $k, h \geq 0$, $\{i_1, \ldots, i_k, j_1, \ldots, j_h\} \subseteq \{0, \ldots, n\}$, 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 $s_{j_2}, \ldots s_{j_h}$, y los elementos que se encuentran a su derecha, son $s_{i_1}, \ldots, s_{i_k}$.

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; $r \xrightarrow{+}_R t = t[r]_p \xrightarrow{+}_R t[t]_p = t[t[r]_p]_p \xrightarrow{+}_R \ldots$, 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 $t_1 \to t_2 \to t_3 \to \ldots$. 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 $r \to_R t_{i+2} \to_R t_{i+3} \to_R \ldots$ 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, $s_1 \rightarrow_R s_2$ implica $s_1 > s_2$, entonces no puede ocurrir una reducción infinita $s_1 \rightarrow_R s_2 \rightarrow_R s_3 \ldots$. ∎

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.

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

La implementación de la anterior sección se puede generalizar, a lo que llamaremos órdenes de caminos recursivos. Para mayor claridad crearemos la función `ordenTerminoLex t s`, que es es el resultado de comparar el nombre delelemento de la posición vacía mediante el orden alfabético. Por ejemplo,

```
ghci> ordenTermino (V ("a",2)) (V ("b",1))
LT
ghci> ordenTermino (V ("x",2)) (T "f" [V ("b",1)])
GT
ghci> ordenTermino (T "g" [V("x",2)]) (T "f" [V("b",1)])
GT
```

Su código es,

```
ordenTermino::  Termino -> Termino -> Ordering
ordenTermino (V (a,_)) (V (b,_)) = compare a b
ordenTermino (V (a,_)) (T b _) = compare a b
ordenTermino (T a _) (V (b,_)) = compare a b
ordenTermino (T a _) (T b _) = compare a b
```

La generalización se basa en añadir un argumento adiccional a la función, éste se encarga de especificar que orden se aplica en el apartado (OCL2c). Por ejemplo,

```
ghci> let stat f (ordenTermino) t s = ordLex ordenTermino t s
ghci> let ord = ordenPorLista ["i","f","e"]
ghci> ordenCamRec stat ord (T "f" [V ("x",1), T "e" []]) (V ("x",1))
GT
ghci> ordenCamRec stat ord (T "i" [T "e" []]) (T "e" [])
GT
ghci> ordenCamRec stat 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> ordenCamRec stat 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,

```
ordenCamRec:: ([Char] -> (Termino -> Termino -> Ordering)
                      -> [Termino] -> [Termino] -> Ordering)
           ->  ([Char] -> [Char] -> Ordering)
           -> Termino
           -> Termino
           -> Ordering
ordenCamRec _ _ s (V x)
  | s == (V x) = EQ
  | ocurre x s = GT --OCR1
  | otherwise = LT
ordenCamRec _ _ (V _) (T _ _) = LT
ordenCamRec est ord s@(T f ss) t@(T g ts) --OCR2
  | all (\x -> ordenCamRec est ord x t == LT) ss
    = case ord f g of
      GT -> if all (\x -> ordenCamRec est ord s x == GT) ts
            then GT --OCR2b
            else LT
      EQ -> if all (\x -> ordenCamRec est ord s x == GT) ts
            then est f (ordenCamRec est ord) ss ts --OCR2c
            else LT
      LT -> LT
  | otherwise = GT --OCR2a
```

Confluencia

En este capítulo estudiaremos el problema de determinar si un sistema de reescritura es confluente. En la primera sección vamos a demostrar que este problema es indecidible, sin embargo en las secciones posteriores, estudiaremos que si el sistema es terminante, entonces el problema es decidible. Por último veremos que ocurre para el caso de los sistemas que no terminan.

Estudio sobre el problema de decisión

En esta sección veremos la indecibilidad para comprobar si un sistema es confluente mediante el siguiente resultado,

Teorema 2.1. *El problema de decidir si un sistema de reescritura finito R es confluente, es indecidible.*

Demostración. El objetivo de esta demostración es reducir el problema de las palabras básicas para E (que sabemos que es indecidible), a un sistema de reescritura de términos.

Sea un conjunto de identidades E tal que $\mathrm{Var}\,(l) = \mathrm{Var}\,(r)$ para todo $l \approx r \in E$. Sea $R := E \cup E^{-1}$, entonces al tener $\rightarrow_R = \leftrightarrow_E$ es confluente. Además R es un sistema de reescritura (por $\mathrm{Var}\,(l) = \mathrm{Var}\,(r)$).

Dados dos términos básicos s y t, y una constante a, vamos a probar que $R_{st} := R \cup \{a \rightarrow s, a \rightarrow t\}$ es confluente syss $s \approx_E t$.

- $(\Rightarrow)$ Si R_{st} es confluente, entonces ni s, ni t poseen una constante a, luego $s \downarrow_{R_{st}} t$. Esto significa que las reglas $a \rightarrow s, a \rightarrow t$ no pueden ser usadas. Por tanto $\rightarrow_R = \leftrightarrow_E$ y $s \approx_E t$.

- ($\Rightarrow$) Vamos a probar que para términos u, v, si $u \to_{R_{st}} v$ entonces $v^t \xrightarrow{*}_R u^t$, donde u^t denota el resultado de sustituir las constantes a en u por t. Supongamos que $u \to_{R_st} v$. Distinguimos que reglas se usan.
  - Si usamos $u \to_R v$, reemplazamos a por t para conseguir $u^t \to_R v^t$ y por tanto $v^t \to_R u^t$ al ser R simétrico.
  - Si usamos $a \to s$, entonces $u|_p = a$ y $v = u[s]_p$ para alguna posición p. Como $s \approx_E t$ entonces $s \xrightarrow{*}_R t$. Obtenemos $v \xrightarrow{*}_R u[t]_p$ que conlleva a $v^t \xrightarrow{*}_R (u[t]_p)^t$. Pero $(u[t]_p)^t = u^t[t^t]_p = u^t[t]_p = u^t$.
  - Si usamos $a \to t$ en la posición p, entonces $v^t = (u[t]_p)^t = u^t \xrightarrow{*}_R u^t$.

  Por tanto, si $u \xrightarrow{*}_{R_{st}} u_i$, para $i = 1, 2$, entonces $u_i \xrightarrow{*}_{R_{st}} u_i^t \xrightarrow{*}_R u^t$ y obtenemos $u_1 \downarrow_{R_{st}}$

Pares críticos

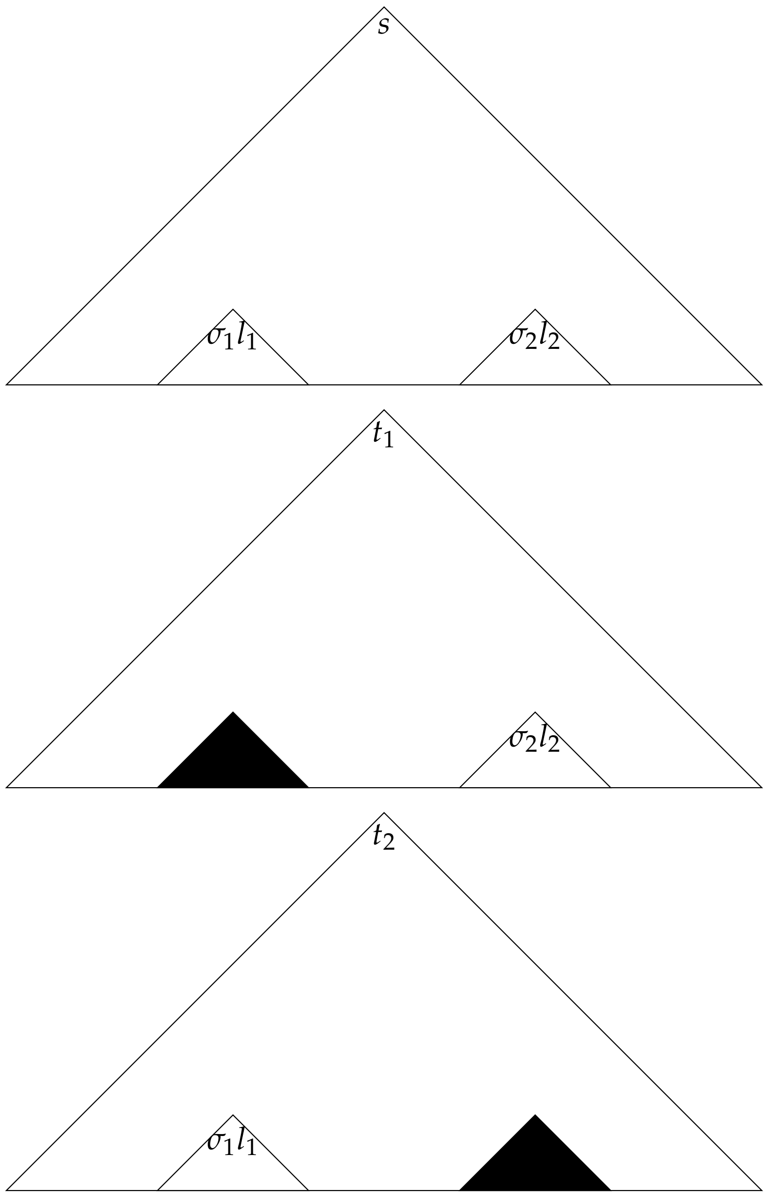
En esta sección estudiaremos la decibilidad para sistemas de reescritura finitos que sean localmente confluentes.

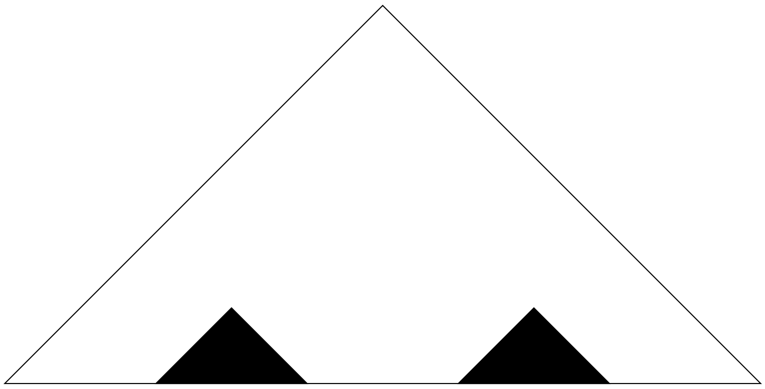
La necesidad de los pares críticos surge de la importancia a la hora de aplicar las reglas de un sistema de reescritura. Para entender mejor este concepto, plantearemos el siguiente ejemplo. Sea s un término, $R := \{l_1 \to r_1, l_2 \to r_2\}$ donde l_1, l_2 son subtérminos de s. Según apliquemos la primera regla o la segunda, obtendremos un nuevo término t.

$$\begin{tikzcd} & s \arrow[dl, "l_1 \to r_1"'] \arrow[dr, "l_2 \to r_2"] & \\ t_1 & & t_2 \end{tikzcd}$$

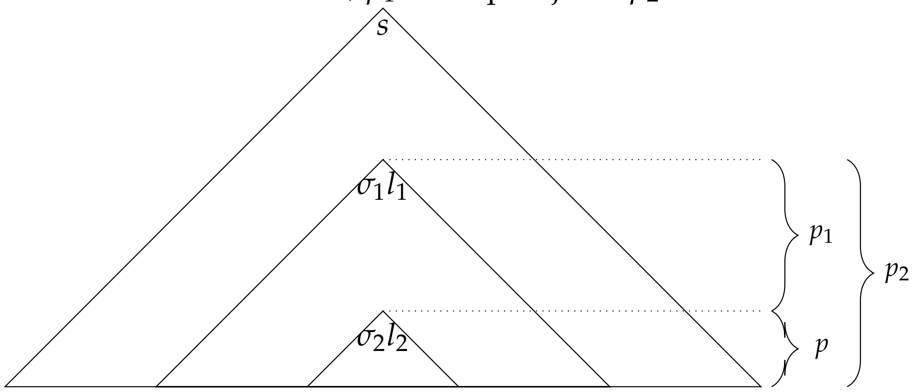
Sea p_i las posiciones y σ_i las sustituciones tal que $s|_{p_i} = \sigma_i l_i$ y $t_i = s[\sigma_i r_i]_{p_i}, i = 1, 2$. Tenemos que estudiar como de relacionados estan p_1 y p_2.

- Caso 1, p_1 y p_2 estan en árboles separados. En este caso, de da la convergencia local.

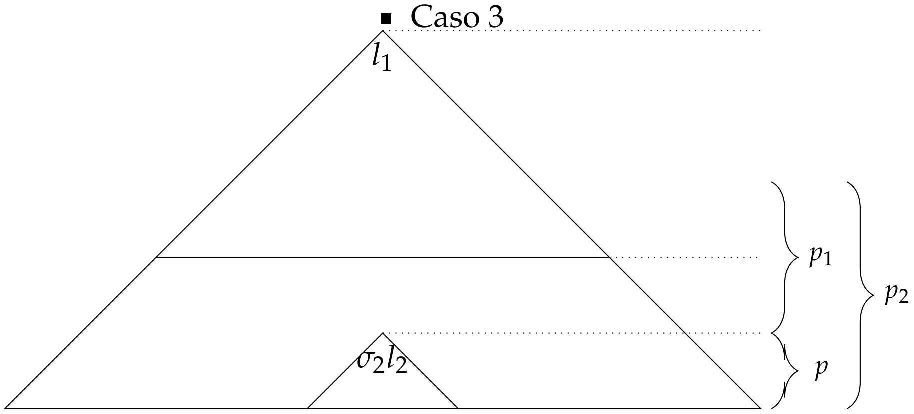

s
$\sigma_1 l_1$
$\sigma_2 l_2$
t_1
$\sigma_2 l_2$
t_2
$\sigma_1 l_1$



- Caso 2, p_1 es un prefijo de p_2



- Caso 3



Definición 2.1. *Sea $l_i \to r_i$, $i = 1, 2$ dos reglas cuyas variables han sido renombradas tal que $Var\ (l_1, r_1) \cap Var\ (l_2, r_2) = \emptyset$. Sea $p \in Pos\ (l_1)$ tal que $l_1|_p$ no es una variable, y θ un umg de $l_1|_p =^? l_2$. Esto determinará el par crítico $\langle \theta r_1, (\theta l_1)[\theta r_2]_p \rangle$. Si dos reglas dan lugar a un par crítico, diremos que se solapan.*

A continuación enunciamos el Teorema de los Pares Críticos.

Teorema 2.2. *Un sistema de reescritura es localmente confluente syss todos sus pares críticos se vuelven a unir.*

Demostración. ($\Leftarrow$) Sabemos que si $s \rightarrow_R t$ para $i = 1, 2$, entonces es localmente confluente, ó $t_i = s[u_i]_p$, donde $\langle u_1, u_2 \rangle$ proviene de algún par crítico $\langle v_1, v_2 \rangle$, es decir $u_1 = \delta v_i$. Entonces $\xrightarrow{*} t$ para algún término, y por tanto $u_i \xrightarrow{\delta} t$ es tambień cierto. Esto implica $t_i \xrightarrow{s} [\delta t]_p$ para $i = 1, 2$.

($\Rightarrow$) Como el sistema de reescritura es localmente confluente, esto significa que cuando lleguemos a un par crítico este volvera a converger a un término. Luego todos sus pares críticos se vuelven a unir. ∎

Algunas propiedades directas de este teorema,

Corolario 2.1. Un sistema de términos terminante es confluente syss todos sus pares críticos se vuelven a unir

Pero si además pedimos que sea un sistema finito, obtenemos el siguiente resultado.

Corolario 2.2. La confluencia de un sistema de reescritura finito y terminante es decidible.

Demostración. El algorítmo que seguiremos será el siguiente:

Para cada par de reglas $l_1 \rightarrow r_1$ y $l_2 \rightarrow r_2$, y para cada $p \in \mathrm{Pos}\,(l_1)$, tal que $l_1|_p$ no es una variable, se genera un par crítico unificando las variables disjuntas de $l_1|_p$ y l_2.

Para cada uno de esos pares críticos $\langle u_1, u_2 \rangle$ reducimos u_i a su forma normal $\tilde{u}_i$. Decimos que el sistema de reescritura es confluente syss $\tilde{u}_1 = \tilde{u}_2$ para todos sus pares críticos.

Si se verifica $\tilde{u}_1 = \tilde{u}_2$, entonces por el corolario 2.1, el sistema es confluente.

Si se da el caso de que existe algún par crítico no $\tilde{u}_1 \neq \tilde{u}_2$, entonces se da la situación $\tilde{u}_1 \xleftarrow{*} u_1 \leftarrow u \rightarrow u_2 \xrightarrow{*} \tilde{u}_2$, que demostraría que no es confluente. ∎

Implementación de los pares críticos

Se implementarán una función para calcular los pares críticos. Antes de poder definirla, se necesitarán varias funciones auxiliares. El código se puede encontrar en el fichero `ParesCriticos.hs`.

Vamos a necesitar la librería de términos.

```
import Terminos
```

- `(indiceMaximo t)` es el mayor índice que aparece en `t`. Por ejemplo,

  ```
  ghci> indiceMaximo (V ("x",3))
  3
  ghci> indiceMaximo (T "w" [V("x",4), V("z", 200), V("y", 1)])
  200
  ```

  Su código es,

  ```
  indiceMaximo :: Termino -> Indice
  indiceMaximo (V (_,i)) = i
  indiceMaximo (T _ ts) = maximum (map indiceMaximo ts)
  ```

- `(renombraTermino n t)` es el termino resultante tras sumar a todos los índices de `t` el entero `n`. Por ejemplo,

  ```
  ghci> renombraTermino 5 (V ("x",3))
  V ("x",8)
  ghci> renombraTermino 3 (T "w" [V("x",4), V("z", 200), V("y", 1)])
  T "w" [V ("x",7),V ("z",203),V ("y",4)]
  ```

  Su código es,

  ```
  renombraTermino :: Int -> Termino -> Termino
  renombraTermino n (V (x,i)) = V(x,i+n)
  renombraTermino n (T f ts) = T f (map (renombraTermino n) ts)
  ```

- `(parCritico c l1 r1 l2 r2)` calcula el par crítico de la regla `l1` $\rightarrow$ `r1` y `l2` $\rightarrow$ `r2`. Por ejemplo,