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

Miguel Ángel Porras Naranjo

Tutorizado por José Antonio Alonso y María José Hidalgo

16 de diciembre de 2016

Índice

Introducción

La **reescritura** es una técnica que consiste en reemplazar términos de una expresión con **términos equivalentes**.

Introducción

Ejemplo

Dadas las reglas $x * 0 \Rightarrow 0$ y $x + 0 \Rightarrow x$, aplicadas a la siguiente expresión,

$$x + (x * 0) \longrightarrow x + 0 \longrightarrow x$$

Ejemplo

Dadas las reglas $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$ y $\neg(\neg P) \Rightarrow P$, reescribimos el siguiente término,

$$\neg(\neg(\neg((\neg P) \wedge Q))) \longrightarrow \neg((\neg P) \wedge Q) \longrightarrow \neg(\neg P) \vee \neg Q \longrightarrow P \vee \neg Q$$

Introducción

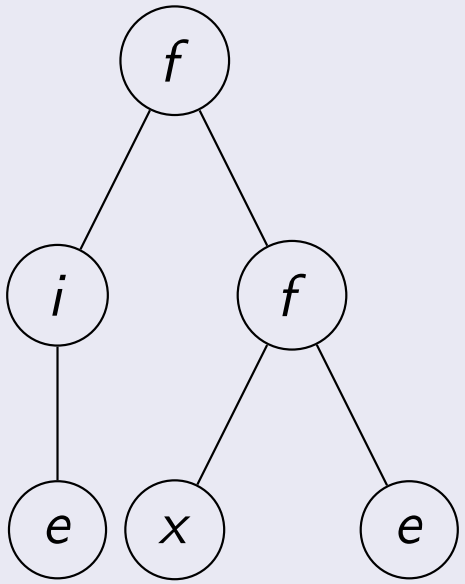
Los **sistemas de reescritura** se pueden aplicar a cualquier tipo de expresión (ya sea de lógica, ecuacional, etc) y podemos decidir qué reglas se pueden usar. Pero antes debemos formalizar como deben ser estas expresiones.

Definición

Un **término** está construido por **variables**, **símbolos constantes** y **símbolos de funciones**.

Introducción

El término $f(i(e), f(x, e))$ se representa por,



Donde i, f representa un símbolo de función de aridad 1 y 2 respectivamente, e representa un símbolo constante y x representa una variable.

Introducción

Los ejemplos anteriores se pueden escribir como términos,

$$x + (x * 0) \longrightarrow +(x, *(x, 0))$$

$$P \vee \neg Q \longrightarrow \vee(P, \neg(Q))$$

Introducción

Los nombres de variables son pares formados por un nombre y un índice.

```
type Nvariable = (String, Int)
```

Por ejemplo, x_3 se representa como `("x",3)`.
Definimos el tipo de dato `Termino` de la siguiente manera,

```
data Termino = V Nvariable
             | T String [Termino]
             deriving (Eq, Show)
```

Por ejemplo,

- x_2 se representa por `(V ("x", 2))`
- $f(x_2, g(x_2))$ se representa por `(T "f"[V ("x", 2), T "g"[V ("x", 2)]])`

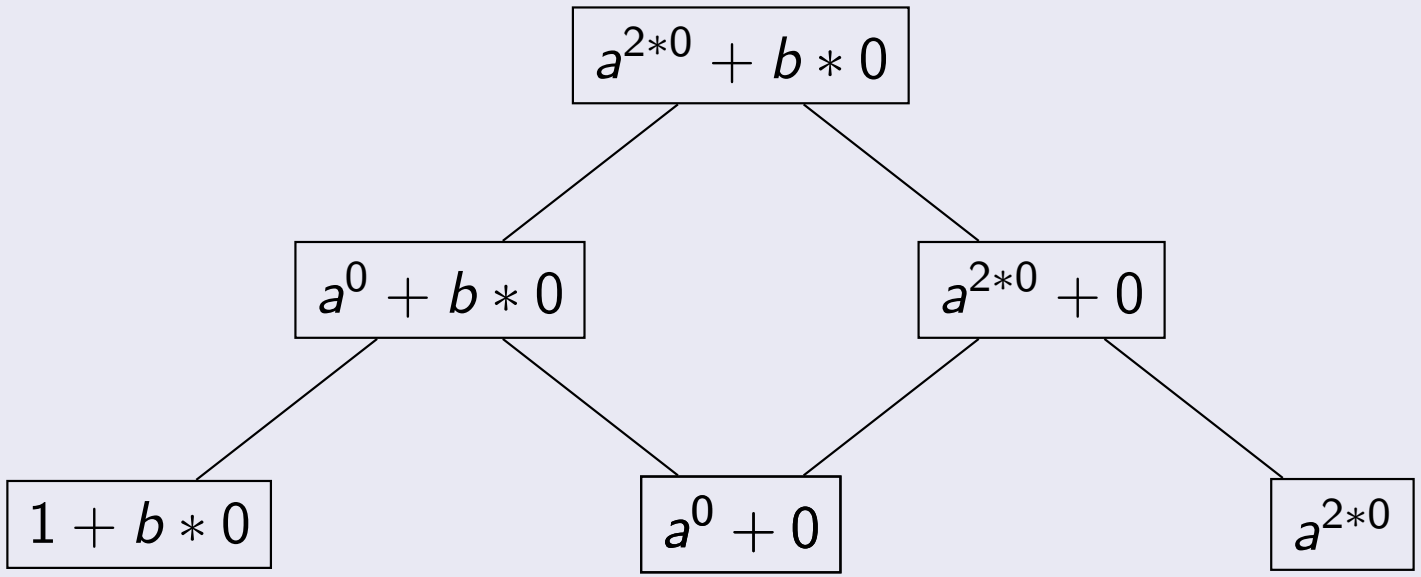
Sistemas de reescritura

Definición

Un **sistema de reescritura de términos** es un conjunto de sustituciones termino a término. A estas sustituciones las llamaremos **reglas**.

Sistemas de reescritura

Estos conjuntos de reglas determinan ciertas propiedades según los elementos que contengan. Por ejemplo,



Sistemas de reescritura

Problemas

- ¿El árbol es finito?
- ¿Todas las hojas del árbol son la misma?

Terminación

Definición

Diremos que un sistema de reescritura **termina**, si no podemos aplicar ninguna regla más.

Tener un mayor número de reglas no siempre es mejor. Por ejemplo para las reglas, $x + 0 \Rightarrow x$ y $x \Rightarrow x + 0$,

$$x + 0 \longrightarrow x \longrightarrow x + 0 \longrightarrow x \longrightarrow x + 0 \ldots$$

Terminación

Teorema

El problema de la terminación es **indecidible** para el caso general.

Se puede conseguir la decibilidad con varias hipótesis de partida.

Confluencia

Definición

Diremos que un sistema de reescritura es **confluente**, si sea cual sea el orden en el que apliquemos las reglas, llegamos al mismo resultado.

Para analizar si un sistema es confluente o no, debemos estudiar el momento en el que se crea una ramificación del árbol. La idea intuitiva es ver si la sustitución que se hace en cada rama se solapa entre sí. A estos puntos los llamaremos **pares críticos**.

Implementación de los pares críticos

```
parCritico :: (Termino -> Termino)
           -> (Termino, Termino)
           -> (Termino, Termino)
           -> [(Termino, Termino)]
parCritico c (l1, r1) (l2, r2)
    | sigma1 == Left UNIFICACION = []
    | otherwise = [(sigma r1, sigma (c r2))]
    where sigma1 = unificacion l1 l2
          sigma = aplicaTerm (elimR(sigma1))
          elimR (Right a) = a
```

Completación

Definición

Podemos modificar el conjunto reglas de los reescritura para que se verifiquen estas dos propiedades. A este proceso se le llama **completación**. Los cambios que se realizan al conjunto de reglas van desde eliminar alguna regla, hasta añadir propias.

Algoritmo de completación

Input: Un conjunto finito E de Σ-identidades y un orden de reducción $>$ en $T(\Sigma, V)$.
Output: Si el procedimiento termina; un sistema de reescritura R finito y convergente que es equivalente a E. En caso contrario devuelve un `Error`.

Algoritmo de completación

Inicialización: Si existe una identidad $(s \equiv t) \in E$ tal que $s \neq t, s \not> t$ y $t \not> s$, entonces termina con `Error`. En otro caso, $i := 0$ $R_0 := \{l \to r | (l \equiv r) \in E \cup E^{-1} \wedge l > r\}$.

`Mientras` $R_i \neq R_{i-1}$

$R_{i+1} := R_i$;

Para todo $\langle s, t \rangle \in$ `ParesCriticos`(R_i):

$>$ Reducir s, t a forma normal s', t'

$>$ Si $s' \neq t'$, $s' \not> t'$ y $t' \not> s'$ terminar con `Error`

$>$ Si $s' > t'$ entonces $R_{i+1} := R_{i+1} \cup \{s' \to t'\}$

$>$ Si $t' > s'$ entonces $R_{i+1} := R_{i+1} \cup \{t' \to s'\}$

$i := i + 1$

Devuelve R_i

Conclusiones del algoritmo de completación

Este algoritmo no es muy eficiente pues calcula los pares críticos cada vez que añade o elimina una regla.

Un mejor algoritmo para la completación es el de **Huet**.

Sistemas utilizados

`http://github.com/migpornar/SRTenHaskell`

Bibliografía I


J.A. Alonso.
Temas de programación funcional.
Technical report, Univ. de Sevilla, 2015.


A. Bundy.
The Computer Modelling of Mathematical Reasoning.
Academic Press Professional, 1990.


T. Nipkow F. Baader.
Term Rewriting and All that.
Cambridge University Press, 1998.


J. Fleuriot.
Automated Reasoning course.
Technical report, University of Edinburgh, 2016.

Bibliografía II


J. Harrison.
Handbook of Practical Logic and Automated Reasoning.
Cambridge University Press, 2009.


G. Hutton.
Programming in Haskell.
University of Nottingham, 2016.