Lógica Computacional 2023-I, nota de clase 6
Lógica Ecuacional

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca Pilar Selene Linares Arévalo

27 de octubre de 2022

1. Introducción

Una vez que hemos estudiado los términos, sus aspectos sintácticos como la substitución y su significado formal mediante las nociones semánticas de interpretación (modelo), estado e interpretación de términos nos dedicamos ahora a resolver nuestro problema fundamental, el análisis de argumentos lógicos para decidir su corrección. Por ahora nos dedicamos a argumentos cuyas premisas y conclusión son ecuaciones universales de la forma $s = t$. Con universales queremos decir que todas las variables que aparezcan en una ecuación se consideran cuantificadas universalmente de manera implícita y por lo tanto cualquiera de ellas puede sustituirse de manera uniforme en la ecuación por cualquier término y la ecuación seguirá siendo válida. Queremos lograr este objetivo de manera distinta a como lo hicimos en lógica proposicional donde preferimos presentar métodos completamente automáticos (búsqueda de la cláusula vacía en formas normales conjuntivas). Ahora vamos a utilizar un sistema deductivo que captura las principales propiedades de la igualdad y que nos permite verificar la corrección de ecuaciones y argumentos ecuacionales sin recurrir a la semántica. Estos sistemas de reglas de inferencia pueden implementarse y así obtener un sistema interactivo que nos ayude a resolver nuestro problema fundamental.

Veamos un ejemplo de la clase de argumentos que deseamos formalizar en la llamada lógica ecuacional.

Consideremos la implementación en Haskell de árboles binarios de números naturales almacenados sólo en las hojas:

```
data Tree = Leaf Nat | Node Tree Tree
```

El aplanado de árboles es una función que dado un árbol devuelve una lista con todos sus elementos:

```
flatten : Tree -> [Nat]
flatten (Leaf n) = [n]
flatten (Node l r) = flatten l ++ flatten r
```

La función anterior es ineficiente, tiene complejidad cuadrática: si el número de nodos del subárbol izquierdo es m_1 entonces `append` y `flatten` están recorriendo esos m_1 elementos respectivamente, además de que se visita cada hoja del subárbol derecho m_2 para obtener la lista de elementos del árbol original.
Si quisiéramos mejorar la definición de `flatten` debemos eliminar el uso de `append`, para esto usaremos una especificación más general de aplanado de árboles utilizando una lista auxiliar:

```
flatten' t ns = flatten t ++ ns
```

Buscaremos una definición que la cumpla tal especificación por medio de inducción:

- Caso Base:

```
flatten' (Leaf n) ns
        = { especificacion de flatten' }
flatten (Leaf n) ++ ns
        = { def. de flatten }
[n] ++ ns
        = { aplicando ++ }
(n:ns)
```

- Caso Inductivo: [1]

```
flatten' (Node l r) ns
        = { especificacion de flatten' }
flatten (Node l r) ++ ns
        = { def. de flatten }
(flatten l ++ flatten r ) ++ ns
        = { asociatividad de ++ }
flatten l ++ (flatten r ++ ns)
        = { hipotesis de induccion para l }
flatten' l (flatten r ++ ns)
        = { hipotesis de induccion para r }
flatten' l (flatten' r ns)
```

Hemos obtenido una definición mejorada que no utiliza la definición de append y que usa una lista auxiliar que hace las funciones de *acumulador*:

```
flatten' : Tree -> [Nat] -> [Nat]
flatten' (Leaf n) ns = (n : ns)
flatten' (Node l r) ns = flatten' l (flatten' r ns)
```

De este ejemplo quisiéramos referirnos a la equivalencia en ambas definiciones, esto se puede describir como la igualdad de resultados al aplicar cualquiera de ellas al mismo árbol:

para cualquier árbol `t`, *se cumple que* `flatten t = flatten' t []`

El ejemplo anterior es un razonamiento sobre la especificación de estructuras de datos (árboles, listas, números naturales) y funciones sobre estas estructuras. (aplanado, reversa, pertenencia, concatenación, etc.). La lógica ecuacional proporciona una sólida herramienta para la verificación formal de programas funcionales.

1.1. Sistemas deductivos

Para describir las reglas que emplearemos en la Lógica Ecuacional y que permiten *encadenar* razonamientos respecto a ecuaciones, utilizaremos una representación que relaciona cierta clase de expresiones (fórmulas, ecuaciones, secuentes, etc..), a las cuales de manera general llamamos juicios, mediante una línea horizontal.

Estos esquemas son usados en muchas áreas en matemáticas y computación y pueden leerse en dos sentidos: de arriba hacia abajo y viceversa, dependiendo de un propósito en el desarrollo de una demostración: generar, derivar o deducir información.

Un sistema deductivo tiene reglas básicas o axiomas que son reglas que cuentan únicamente con la parte inferior para representar hechos o conclusiones sin requerir información previa:

$$\frac{}{J}$$

[1] Donde la hipótesis de inducción supone verdadera la propiedad para los subárboles `l` y `r`.

También tienen reglas de deducción que cuentan con ambas partes, premisas y conclusión, útiles en la generación de información:

$$\frac{J_1 \; J_2 \; \ldots \; J_m}{J}$$

En sentido de arriba hacia abajo, la regla de deducción anterior se puede leer como sigue: si sucede que se cumplen los juicios $J_1, J_2, \ldots, J_m$ entonces podemos concluir el juicio J. En sentido contrario (de abajo hacia arriba), la regla se puede interpretar como sigue: para demostrar el juicio J basta con demostrar que se cumplen los juicios $J_1, J_2, \ldots, J_n$. Es importante observar que las reglas con las que lidiaremos admiten una única conclusión. Nótese también que las reglas representan argumentos lógicos los cuales pueden ser correctos o incorrectos aunque para que un sistema deductivo sea útil sus reglas deben ser todas correctas, es decir, deben corresponden a argumentos correctos.

Definición 1 *Un sistema deductivo para un lenguaje $\mathcal{L}$ es una colección finita de axiomas y reglas entre enunciados de $\mathcal{L}$.*

Definición 2 (Derivación) *Una prueba o derivación de un juicio $\mathcal{J}$ en un sistema deductivo $\mathcal{S}$ es una secuencia finita de enunciados $\pi = \langle \mathcal{J}_1, \ldots, \mathcal{J}_k \rangle$ tales que $\mathcal{J}_k = \mathcal{J}$ y para cada $\mathcal{J}_i \in \pi$, $1 \leq i \leq k$, se cumple una y sólo una de las siguientes condiciones:*

- *$\mathcal{J}_i$ es instancia de un axioma.*
- *$\mathcal{J}_i$ es la conclusión de una instancia de una regla de inferencia cuyas premisas son $\mathcal{J}_{i_1}, \ldots, \mathcal{J}_{i_m} \in \Pi$ with $i_1, \ldots, i_m < i$.*

Más aún, si Γ es una base de conocimiento dada, decimo que π es una prueba de $\mathcal{J}$ a partir de Γ si, junto con las condiciones mencionadas arriba, también permitimos $\mathcal{J}_i \in \Gamma$. Si existe una derivación de $\mathcal{J}$ en el sistema deductivo $\mathcal{S}$ con base de conocimiento Γ, entonces escribimos $\Gamma \vdash_{\mathcal{S}} \mathcal{J}$ o simplemente $\Gamma \vdash \mathcal{J}$ si el sistema deductivo es conocido.

2. Cálculo de Birkhoff

Un sistema de deducción para razonar respecto a ecuaciones es el llamado Cálculo de Birkhoff, el cual contiene cinco reglas:

$$\frac{A \in \Gamma}{\Gamma \vdash A} \textsc{Hyp} \qquad \frac{}{\Gamma \vdash t = t} \textsc{Refl} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash t = s} \textsc{Sym} \qquad \frac{\Gamma \vdash t = r \qquad \Gamma \vdash r = s}{\Gamma \vdash t = s} \textsc{Trans}$$

$$\frac{\Gamma \vdash t = s}{\Gamma \vdash t\sigma = s\sigma} \textsc{Inst} \qquad \frac{\Gamma \vdash t_1 = s_1 \quad \ldots \quad \Gamma \vdash t_n = s_n}{\Gamma \vdash f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)} \textsc{Congr}$$

Estas reglas hacen énfasis en ciertas propiedades fundamentales de la igualdad:

1. Establecen las características de una relación simétrica y transitiva (Refl, Sym, Trans)
2. Extraen casos particulares por medio de instanciación (Inst)
3. Construyen nuevas ecuaciones por medio de congruencia entre subtérminos (Congr)

En vez de la regla de congruencia se puede usar la regla de Leibniz, la cual permite simplificar derivaciones:

$$\frac{\Gamma \vdash t = s}{\Gamma \vdash r[z := t] = r[z := s]} \textsc{ Leibniz}$$

Veamos algunos ejemplos de derivaciones en la lógica ecuacional, usando las reglas del cálculo de Birkhoff.

Ejemplo 2.1 [Estructura de anillo] Considere el siguiente conjunto de ecuaciones $\mathcal{A}$, que define la estructura de anillo [2]:

$$\begin{array}{rclll} x+0 & = & x & A1 & \qquad x+y = y+x \quad A2 \\ x+(-x) & = & 0 & A3 & \qquad (x+y)+z = x+(y+z) \quad A4 \\ (x\cdot y)\cdot z & = & x\cdot(y\cdot z) & A5 & \qquad x\cdot(y+z) = x\cdot y + x\cdot z \quad A6 \end{array}$$

$$(x+y)\cdot z = x\cdot z + y\cdot z \quad A7$$

la siguiente es una derivación de $\mathcal{A} \vdash x+(y+z) = y+(x+z)$.

(1)	$\mathcal{A} \vdash (x+y)+z = x+(y+z)$	Hyp A4
(2)	$\mathcal{A} \vdash x+(y+z) = (x+y)+z$	Sym (1)
(3)	$\mathcal{A} \vdash z = z$	Refl
(4)	$\mathcal{A} \vdash x+y = y+x$	Hyp A2
(5)	$\mathcal{A} \vdash (x+y)+z = (y+x)+z$	Congr + (4) (3)
(6)	$\mathcal{A} \vdash (y+x)+z = y+(x+z)$	Inst (1)
(7)	$\mathcal{A} \vdash x+(y+z) = (y+x)+z$	Trans (2)(5)
(8)	$\mathcal{A} \vdash x+(y+z) = y+(x+z)$	Trans (7)(6)

La siguiente demostración es acerca de listas.

Ejemplo 2.2 Queremos demostrar que si $y = [\,], w = n, z = \ell'$ y $\ell = (n : \ell')$ entonces $(n : \ell') = y +\!\!+ (w : z)$. Para esto utilizamos la base de conocimiento Γ_{List} que contiene las hipótesis dadas arriba, así como las definiciones de funciones sobre listas (se puede pensar en las implementaciones en Haskell).

(1)	$\Gamma_{\mathsf{List}} \vdash y = [\,]$	Hyp
(2)	$\Gamma_{\mathsf{List}} \vdash w = n$	Hyp
(3)	$\Gamma_{\mathsf{List}} \vdash z = \ell'$	Hyp
(4)	$\Gamma_{\mathsf{List}} \vdash [\,] +\!\!+ u = u$	Hyp
(5)	$\Gamma_{\mathsf{List}} \vdash (w : z) = (n : \ell')$	Congr :: (2)(3)
(6)	$\Gamma_{\mathsf{List}} \vdash (w : z) = (w : z)$	Refl
(7)	$\Gamma_{\mathsf{List}} \vdash y +\!\!+ (w : z) = [\,] +\!\!+ (w : z)$	Congr ++ (1)(6)
(8)	$\Gamma_{\mathsf{List}} \vdash [\,] +\!\!+ (w : z) = (w : z)$	Ins (4)
(9)	$\Gamma_{\mathsf{List}} \vdash y +\!\!+ (w : z) = (w : z)$	Trans (7)(8)
(10)	$\Gamma_{\mathsf{List}} \vdash y +\!\!+ (w : z) = (n : \ell')$	Trans (9)(5)
(11)	$\Gamma_{\mathsf{List}} \vdash (n : \ell') = y +\!\!+ (w : z)$	Sym (10)

Veamos ahora un ejemplo más elaborado que muestra un caso de verificación formal relacionado a la función factorial.

Consideremos el siguiente conjunto de ecuaciones:

[2] Un anillo es una estructura algebráica $(R, +, \cdot, 0)$ donde las operaciones cumplen las ecuaciones presentadas. Siendo $-$ una operación unaria correspondiente al inverso de la operación $+$, es decir, el inverso aditivo.

```
E1. fac 0 = 1              E2.  fac (S x) = (S x) * fac x

E3. prod [ ] = 1           E4.  prod (x:xs) = x * prod xs
```

Es claro que las ecuaciones 1 y 2 definen al factorial mientras que las ecuaciones 3 y 4 definen al producto de una lista de números. Si la lista xs consta de todos los números entre 1 y n entonces prod xs debe devolver fac n. Es decir, la siguiente ecuación es válida:

```
fac x = prod [1,2...,x]
```

Deseamos demostrar esto en la lógica ecuacional pero antes es importante observar que el lado derecho de la ecuación no es un término legal debido al uso de puntos suspensivos. En clase me lavé las manos agregando una ecuación extra (le llame milagro) pero aquí vamos a definirlo de manera formal, para lo cual usamos la siguiente definición de la lista de números consecutivos de 1 a x pero en forma decreciente:

```
E5. dfrom 0 = [ ]
E6. dfrom (S x) = (S x) : dfrom x
```

Nótese que se cumple la especificación siguiente:

```
dfrom x = [x,x-1,x-2,...,2,1]
```

pero esta no es una ecuación legal, debido a los puntos suspensivos.

Resulta que al definir la función `dfrom` no sólo nos deshacemos de la ecuación "milagro" sino que el razonamiento se simplifica de manera que tampoco vamos a necesitar el lema visto en clase (aunque de todos modos lo demuestro abajo como ejemplo extra).

La propiedad deseada es:

```
fac x = prod (dfrom x)
```

que probamos por inducción sobre x.
Para $x = 0$ tenemos

```
fac 0 = 1                (E1)
      = prod []          (E3)
      = prod (dfrom 0)   (E5)
```

Suponemos ahora:

```
HI.  fac x = prod (dfrom x)
```

y demostramos

```
fac (S x) = prod (dfrom (S x))
```

Tenemos que:

```
prod (dfrom (S x)) =  prod ( (S x):dfrom x )     (E6)
                   =  S x * prod (dfrom x)       (E4)
                   =  S x * fac x                (HI)
                   =  fac (S x)                  (E2).
```

A continuación transformamos las pruebas anteriores en pruebas formales en la lógica ecuacional usando como contexto a $\Gamma = \{E1, E2, E3, E4, E5, E6\}$

Primero formalizamos la prueba del paso base:

```
1. Γ ⊢ fac 0 = 1  Hip. E1
2. Γ ⊢ prod [] = 1  Hip. E3
3. Γ ⊢ 1 = prod []  Sim 2.
4. Γ ⊢ fac 0 = prod []   Trans 1, 3.
5. Γ ⊢ dfrom 0 = []  Hip. E5
6. Γ ⊢ prod (dfrom 0) = prod []  Cong 5.
7. Γ ⊢ prod [] = prod (dfrom 0)  Sim 6.
8. Γ ⊢ fac 0  = prod (dfrom 0)  Trans 4,7.
```

Formalicemos ahora el paso inductivo para lo cual necesitamos agregar al contexto la ecuación HI de arriba.

```
 1. Γ, HI ⊢ fac x = prod (dfrom x)   Hip. HI
 2. Γ, HI ⊢ dfrom (S x) =  (S x):dfrom x      (E6)
 3. Γ, HI ⊢ prod (dfrom (S x)) =  prod ( (S x):dfrom x )   Cong 2
 4. Γ, HI ⊢ prod (x:xs ) = x * prod xs        Hip. E4
 5. Γ, HI ⊢ prod ( (S x):dfrom x ) = S x * prod (dfrom x)  Inst 4.
 6. Γ, HI ⊢ prod (dfrom x) = fac x    Sim 1.
 7. Γ, HI ⊢ S x = S x                          Refl.
 8. Γ, HI ⊢ S x * prod (dfrom x) = S x * fac x   Cong 7,6
 9. Γ, HI ⊢ fac (S x) = S x * fac x  Hip. E2.
10. Γ, HI ⊢ S x * fac x = fac (S x)  Sim. 9
11. Γ, HI ⊢ prod (dfrom (S x)) = S x * prod (dfrom x)  Trans 3,5
12. Γ, HI ⊢ S x * prod (dfrom x) = fac (S x)      Trans 8,10
13. Γ, HI ⊢ prod (dfrom (S x)) = fac (S x)       Trans 11,12
```

La prueba anterior puede simplificarse usando la regla de Leibniz en vez de la regla y obviando el uso de las reglas de simetría e instancia, como sigue:

```
1. Γ, HI ⊢ fac x = prod (dfrom x)   Hip. HI
2. Γ, HI ⊢ dfrom (S x) =  (S x):dfrom x      (E6)
3. Γ, HI ⊢ prod (dfrom (S x)) =  prod ( (S x):dfrom x )   Leibniz 2
4. Γ, HI ⊢ prod ( (S x):dfrom x ) = S x * prod (dfrom x)  (E4)
5. Γ, HI ⊢ S x * prod (dfrom x) = S x * fac x  Leibniz 1
6. Γ, HI ⊢ fac (S x) = S x * fac x  Hip. E2.
7. Γ, HI ⊢ prod (dfrom (S x)) = S x * prod (dfrom x)  Trans 4,5
8. Γ, HI ⊢ S x * prod (dfrom x) = fac (S x)      Trans 5,6
9. Γ, HI ⊢ prod (dfrom (S x)) = fac (S x)       Trans 7,8
```

Observe que en cada uso de la regla de Leibniz hemos subrayado los subtérminos de la ecuación que se están reescribiendo.

3. Ejercicios

1. $h(hx) = x, s(hx) = sx \vdash s(s(ha)) = s(h(sa))$

2. $f(fx) = y \vdash x = y$
3. $f(f(fx)) = x, f(fx) = fx \vdash fx = x$
4. $x \cdot y = y \vdash x \cdot (y \cdot z) = (x \cdot y) \cdot z$
5. $x \cdot y = u \cdot v \vdash x \cdot (y \cdot z) = (x \cdot y) \cdot z$
6. $x \cdot y = u \cdot v \vdash y \cdot z = z \cdot y$
7. $x \cdot y = u \cdot v \vdash x \cdot x = y \cdot y$
8. Formalice el razonamiento ecuacional acerca de `flatten` y `flatten'` presentado en la introducción de esta nota.
9. Defina un contexto adecuado Γ y demuestre que $\Gamma \vdash \texttt{flatten t} = \texttt{flatten}' \texttt{ t } [\,]$
10. Considere las ecuaciones $A1 - A7$ del ejemplo 2 más las dos siguientes, las cuales definen un anillo con unidad:

    A8 $x \cdot 1 = x$
    A9 $1 \cdot x = x$

    obsérvese que la operación producto $\cdot$ no es conmutativa. Sea $\mathcal{A}$ la base de conocimiento que consta de estas nueve ecuaciones. Demuestre lo siguiente:

    a) $\mathcal{A} \vdash 0 + x = x$
    b) $\mathcal{A} \vdash (-x) + x = 0$
    c) $\mathcal{A}, x + z = y + z \vdash x = y$
    d) $\mathcal{A}, z + x = z + y \vdash x = y$
    e) $\mathcal{A} \vdash -(-x) = x$
    f) $\mathcal{A} \vdash (-x) + (-y) = -(x + y)$
    g) $\mathcal{A} \vdash 0 \cdot x = 0$
    h) $\mathcal{A} \vdash x \cdot 0 = 0$
    i) $\mathcal{A} \vdash x \cdot y + x \cdot (-y) = x \cdot 0$
    j) $\mathcal{A} \vdash (-x) \cdot y = x \cdot (-y)$
    k) $\mathcal{A} \vdash x \cdot (-y) = -(x \cdot y)$
    l) $\mathcal{A} \vdash (-x) \cdot (-y) = x \cdot y$
11. Defina contextos Γ y construya derivaciones de los siguientes argumentos acerca de listas (algunas de ellas requiren de inducción)

    a) $\Gamma, xs = ys \;\vdash\; xs \;{+\!\!+}\; zs = ys \;{+\!\!+}\; zs$
    b) $\Gamma \vdash rev\,[x] = [x]$
    c) $\Gamma \vdash rev\,[x, x] = [x, x]$
    d) $\Gamma \vdash rev\,[x, y] = [y, x]$
    e) $\Gamma \vdash [x] \;{+\!\!+}\; xs = (x : xs)$
    f) $\Gamma \vdash xs \;{+\!\!+}\; [\,] = xs$