Introducción a los Algoritmos - 2do. cuatrimestre 2011

Guía 1: Razonamiento ecuacional, tipado, satisfactibilidad y validez

Docentes: Araceli Acosta, Walter Alini, Luciana Benotti, Ezequiel Orbe

El primer objetivo de esta guía es que te familiarices con la metodología del cálculo ecuacional, que utilizaremos como forma de escribir demostraciones a lo largo de toda la materia. Este formato de prueba, que incluye ***justificaciones exhaustivas para cada paso****, ayuda a evitar cometer errores, en particular cuando las demostraciones son largas y complejas. Discutiremos además, en detalle, las nociones de satisfacibilidad y validez.*

La guía incluye también ejercicios de precedencia y tipado, para que adquieras práctica en la interpretación de fórmulas. Por último, también aprenderás a usar un interprete de `haskell` *, y a desarrollar tus primeros programas.*

Razonamiento ecuacional, *validez* y *satisfactibilidad*

El objetivo de estos ejercicios es introducir el razonamiento ecuacional sobre un dominio familiar: la aritmética. De esta forma podemos centrarnos en el ***método de prueba*** *más que en las propias demostraciones, que esperamos no te presenten demasiadas dificultades.*

Además trabajaremos sobre ecuaciones centrándonos en diferentes aspectos: algunas veces buscando valores que satisfacen las ecuaciones, otras demostrando que las ecuaciones son siempre verdaderas independientemente de los valores que tomen las variables. En el primer caso decimos que la fórmula es ***satisfacible****; en el segundo, decimos que la fórmula es* ***válida****. Es importante tener bien presente estas diferencias. Nuestro principal interés es distinguir cuándo una fórmula es válida y cuándo no. Como la validez es independiente de los valores de las variables involucradas, si queremos demostrar la validez de una fórmula no podemos hacer* ***ninguna suposición*** *sobre los valores de las variables. Por el contrario, cuando queremos demostrar que una fórmula es no valida es suficiente con encontrar al menos un valor que haga que la fórmula sea falsa: esto es un contraejemplo. Algunas veces además es posible demostrar directamente que la fórmula es falsa para* ***todos los valores*** *de las variables. En este caso decimos que la fórmula es* ***no satisfacible****, porque no existe ningún valor posible que haga que sea verdadera.*

Importante: Cuando una fórmula es siempre verdadera la llamamos **válida**, cuando es verdadera para algunos valores la llamamos **satisfactible**, cuando es falsa para algunos valores la llamamos **no válida** y cuando es falsa para todos los valores la llamamos **no satisfactible**.

1. ¿Qué valor de la variable x *satisface* las siguientes ecuaciones, es decir, qué valor hace que las fórmulas sean verdaderas? Por ejemplo para la fórmula $6 * x + 8 = x + 3$:

$$
\begin{array}{ll}
 & 6 * x + 8 = x + 3 \\
\equiv & \{ \text{ restar } x \text{ en ambos miembros } \} \\
 & \underline{6 * x} + 8 \ \underline{-x} = \underline{\underline{x}} + 3 \ \underline{\underline{-x}} \\
\equiv & \{ \text{ aritmética } \} \\
 & 5 * x + 8 = 3 \\
\equiv & \{ \text{ restar 8 en ambos miembros } \} \\
 & 5 * x + \underline{8 - 8} = \underline{\underline{3 - 8}} \\
\equiv & \{ \text{ aritmética } \} \\
 & 5 * x = -5 \\
\equiv & \{ \text{ dividir por 5 en ambos miembros, aritmética } \} \\
 & x = -1
\end{array}
$$

Luego la respuesta es que solamente $x = -1$ hace verdadera la fórmula $6 * x + 8 = x + 3$. En otros casos, puede haber más de un valor para x que satisfaga la fórmula; en ese caso hay que encontrarlos a todos. *Notá que cuando realizamos dos pasos a la vez, utilizamos un subrayado diferente para cada subexpresión involucrada.*

a) $2 * x + 3 = 5$

b) $x = x + 1$

c) $x + x = 2 * x$

Preguntas:

a) ¿Cómo se interpreta cada resultado?

b) ¿qué relación hay entre estos ejercicios y los conceptos de **validez** y **satisfactibilidad**? es decir ¿cómo le llamaríamos a cada fórmula?

2. Un *predicado* sobre `x` es una función que para cada valor de `x` devuelve `Verdadero` o `Falso`. Definí en `haskell` las ecuaciones del ejercicio anterior como predicados sobre `x` y verificá los resultados obtenidos, en los casos que sea posible. Recordá que es necesario utilizar un nombre para cada ecuación y nombrar las variables. Por ejemplo, en `haskell` , en un archivo de texto que llamaremos `Practico1.hs`, se puede definir el predicado `p`

```
p x = (6 * x + 8 == x + 3)
```

y evaluando p en `ghci` con diferentes *argumentos* obtenemos:

```
Prelude> :load Practico1.hs
Main> p (-1)
True
Main> p 5
False
```

Preguntas:

a) ¿te sirven estos programas para verificar satisfactibilidad?

b) ¿cómo convencerías a cualquier persona que una expresión es satisfactible?

c) ¿te sirven estos programas para verificar validez?

d) ¿cómo convencerías a cualquier persona que una expresión es válida?

e) ¿te sirven estos programas para verificar no satisfactibilidad?

f) ¿cómo convencerías a cualquier persona que una expresión es no satisfactible?

g) ¿te sirven estos programas para verificar no validez?

h) ¿cómo convencerías a cualquier persona que una expresión es no válida?

3. Demostrá que las siguientes ecuaciones son *válidas*, es decir que las fórmulas son verdaderas para cualquier valor que tome la variable x. Por ejemplo:

$$
\begin{array}{ll}
 & 4 * x + 14 = \underline{2 * (2 * x + 5)} + 4 \\
\equiv & \{ \text{ distributividad de } * \text{ con } +, \text{ def. de } * \ \} \\
 & 4 * x + 14 = 4 * x + \underline{10 + 4} \\
\equiv & \{ \text{ def. de } + \ \} \\
 & 4 * x + 14 = 4 * x + 14 \\
\equiv & \{ \text{ restar } 4 * x + 14 \text{ en ambos miembros } \} \\
 & \underline{4 * x + 14 - (4 * x + 14) = 4 * x + 14 - (4 * x + 14)} \\
\equiv & \{ \text{ reflexibidad de } = (x = x) \ \} \\
 & \textit{True}
\end{array}
$$

a) $(x - 1) * (x + 1) = x^2 - 1^2$

b) $(a + b)^2 = a^2 + 2 * a * b + b^2$

Preguntas

a) ¿identificás estas fórmulas?

b) ¿tienen nombre?

c) ¿qué expresan?

4. Las siguientes ecuaciones no son válidas, es decir, para al menos algún valor de la variable x cada fórmula es falsa. Justificá dando un *contraejemplo* Además, indicá cuáles son satisfacibles y qué valores las satisfacen.

   Por ejemplo para la ecuación $3 * x + 1 = 3 * (x + 1)$.

   Contraejemplo: $x = 5$ *falsifica* la fórmula ya que

   $$\begin{array}{ll} & 3 * x + 1 = \underline{3 * (x + 1)} \\ \Rightarrow & \{ \text{ tomando } x = 5 \} \\ & \underline{3 * 5} + 1 = \underline{3 * 5} + 3 \\ \equiv & \{ \text{ def. de } * \} \\ & \underline{15 + 1} = \underline{15 + 3} \\ \equiv & \{ \text{ def. de } + \} \\ & 16 = 18 \\ \equiv & \{ \text{ igualdad de números } \} \\ & \textit{False} \end{array}$$

   a) $x + (y * z) = (x + y) * (x + z)$

   b) $x + 100 = x$

   Preguntas

   a) de ser válidas estas expresiones, ¿qué nombre les pondrías a estas fórmulas?

5. Decidí si son *válidas* o *no válidas* y *satisfacibles* o *no satisfacibles* las siguientes (in)ecuaciones. Justificá en cada caso.

   a) $\sqrt{x} + \sqrt{y} = \sqrt{x + y}$

   b) $(a - b) * (a + b) * ((a + b)^2 - 2 * a * b) = a^4 - b^4$

   c) $x^2 + 2 * x + 4 = 0$

   Preguntas:

   a) ¿Qué diferencia existe entre dar un ejemplo y demostrar la validez de una fórmula?

   b) ¿Qué diferencia existe entre dar un contraejemplo y demostrar que una fórmula es equivalente a `Falso`? ¿Cuándo se puede utilizar cada *estrategia*?

6. Da ejemplos y una justificación apropiada de una fórmula:

   a) válida (y por lo tanto satisfacible).

   b) satisfacible pero no válida.

   c) no satisfacible (y por lo tanto no válida).

Precedencia y tipado

*La **precedencia** de los operadores nos permite escribir fórmulas grandes de manera simple y más legible. Cuando un operador tiene mayor precedencia que otro, podemos escribir una fórmula que involucra a ambos sin necesidad de poner paréntesis, y a pesar de esto, la fórmula tiene un sentido único. Un ejemplo conocido por todos es el caso de la suma respecto a la multiplicación. El operador $*$ tiene mayor precedencia que $+$ y por ello es que normalmente interpretamos la expresión $2 + 4 * 3$ como $2 + (4 * 3)$. Esta regla es la que nos permitía en la primaria "separar en términos" una expresión algebraica. Al mismo tiempo, y por la misma regla, sabemos que no es lo mismo la expresión $(2 + 4) * 3$ que $2 + 4 * 3$. No siempre los paréntesis pueden sacarse indiscriminadamente sin cambiar el sentido de la expresión.*

De la misma manera, cuando un operador ***asociativo*** *se aplica múltiples veces es posible eliminar paréntesis, ya que el orden en que se efectúa la operación no altera el resultado. Por ejemplo* $(2+4)+7$ *es igual a* $2+(4+7)$ *y por lo tanto podemos escribir simplemente* $2+4+7$.

Los operadores $+$, $*$, mín, máx *son asociativos.*

tabla de precedencia	*más arriba* $\rightarrow$ *mayor precedencia*
$\sqrt{}, (\cdot)^2$	*raíces y potencias*
$*, /$	*producto y división*
máx, mín	*máximo y mínimo*
$+, -$	*suma y resta*
$=, <, \leqslant, >\geqslant$	*operadores de comparación*

Por otro lado, la noción de ***tipado*** *nos permite distinguir expresiones que están "bien escritas", es decir que tienen sentido, que representan algún valor. El tipo de un operador o función nos dice cual es la clase de valores que toma y cual es la clase del valor que devuelve. En las expresiones algebraicas en general todos los valores son de tipo número, que denotamos con Num (y los* ***subtipos*** *Nat, Int, ...). Como ejemplo tomemos el caso de la suma: tanto a la izquierda del símbolo "+" como a la derecha debe haber expresiones que representen números, y el resultado total también es un número.*

Además del tipo Num utilizaremos el tipo de los valores booleanos, que denotamos con Bool. Este tipo incluye las constantes True y False que representan las nociones de verdadero y falso respectivamente. Los valores booleanos son importantes porque son el resultado de los operadores de comparación $=, <, \leqslant, >, \geqslant$.

Las nociones de precedencia y tipado son importantes para asegurar que escribimos expresiones que tienen un sentido único y bien definido. Los siguientes ejercicios tratan sobre estas nociones en el marco de las expresiones algebraicas.

7. Sacá todos los paréntesis que sean *superfluos* según las reglas de precedencia y asociatividad de los operadores aritméticos. Por ejemplo:

$$\begin{aligned} & (8-6)*x = \underline{(6*(x^2))}+3 \\ \equiv & \{ \text{ precedencia de } * \text{ por sobre } + \} \\ & (8-6)*x = 6*\underline{(x^2)}+3 \\ \equiv & \{ \text{ precedencia de } {}^2 \text{ sobre } * \} \\ & (8-6)*x = 6*x^2+3 \end{aligned}$$

   a) $(-(5+x)+((3*6)/(4*5)))*(8*5)$

   b) $(((2)^2+5)-(4*2)/(4))+1 = ((5*x)/8)+(3*5)^3$

8. Introducí paréntesis para hacer *explícita* la precedencia.

   a) $5*3+4 \geqslant 7-7+3$

   b) $3+4*x=4$

9. Escribí el *tipo* de los siguientes operadores y funciones: $*$, $/$, ^{2}, $\leqslant$, $\geqslant$, $=$. ¿Cuál es el tipo de esos mismos operadores en `haskell`? ¿Qué sucede con el operador $-$?

   Como ejemplo presentamos el caso del operador $+$. Este operador toma dos números y devuelve otro número. Podemos escribir su tipo en *notación funcional* listando el tipo de los parámetros y a continuación el tipo resultado:

$$+ : Num \rightarrow Num \rightarrow Num$$

   Otra forma de escribir el tipo de este operador, útil para armar el árbol de tipado de una expresión, es en *notación de árbol*:

$$\frac{Num + Num}{Num}$$

   En `haskell` el operador `(+)` tiene el mismo tipo. En `ghci` podemos corroborarlo de la siguiente manera:

```
Main> :t (+)
(+) :: Num a => a -> a -> a
```

Esta respuesta se interpreta de la siguiente manera. El tipo del operador `(+)` es lo que está a la derecha de `=>`, es decir $a \to a \to a$, donde la variable a indica cualquier tipo. Por otro lado, lo que que esta a la izquierda de `=>` nos indica que el tipo a es un subtipo de *Num*, es decir, *Nat*, *Int*.

Un árbol de tipado es una herramienta que nos permite verificar que una expresión está bien escrita, es decir, que respeta la sintaxis del lenguaje. Consiste en ir armando los tipos de cada subexpresión de la manera en que se iría resolviendo, es decir teniendo en cuenta la presedencia de los distintos operadores.

10. Completar los siguientes árboles de tipado con los tipos que corresponden a cada operación.

a)

$$\dfrac{\dfrac{(\ x\ +\ (\underline{2}\ /\ \underline{y})) \geqslant (\underline{x}\ +\ \underline{1})}{\dfrac{\dfrac{\underline{\quad}\ /\ \underline{\quad}}{\underline{\quad}\ +\ \underline{\quad}}}{\underline{\quad}\ \geqslant\ \underline{\quad}}}}{}$$

b)

$$\dfrac{\dfrac{(\underline{8}\ -\ \underline{6})\ *\ \underline{x}\ =\ (\underline{6}\ *\ \underline{x}^{\,\underline{2}})\ +\ \underline{3}}{\dfrac{\underline{\quad -\quad}\qquad\qquad \underline{\quad 2\quad}}{\dfrac{\underline{\quad *\quad}\qquad\quad \underline{\quad *\quad}}{\underline{\quad +\quad}}}}}{\underline{\quad =\quad}}$$

11. ¿Están bien escritas las siguientes expresiones? Justificá a través de un árbol de tipado utilizando las reglas que escribiste en el ejercicio 9 y eligiendo el tipo para cada variable. Para evitar errores, hace explícita la precedencia introduciendo paréntesis, y construí una tabla del tipo de las variables. Por ejemplo, para la expresión $x + 2/y \geqslant x + 1$ podemos armar el árbol de tipado de la siguiente manera:

Primero escribimos el tipo de las cosas sobre las que estamos seguros: las constantes.

$$\dfrac{(x + (\underline{2}\ /\ y)) \geqslant (x + \underline{1})}{\underline{\quad}\qquad\qquad \underline{\quad}}$$
$$Num \qquad\qquad Num$$

Luego asignamos tipos a las variables, según las reglas de tipado de los operadores que operan sobre ellas. Luego escribimos el tipo del resultado de estos operadores.

$$\dfrac{(\ x\ +\ (\underline{2}\ /\ \underline{y}))\ \geqslant\ (\underline{x}\ +\ \underline{1})}{\underline{\quad}\qquad \underline{\quad / \quad}\qquad \underline{\quad + \quad}}$$
$$Num \qquad Num\ /\ Num \qquad Num\ +\ Num$$
$$Num \qquad\qquad Num \qquad\qquad Num$$

Var	Tipo
x	*Num*
y	*Num*

Sucesivamente completamos el resto de los operadores, hasta dar el tipo de la expresión completa

$$\dfrac{\dfrac{(\ x\ +\ (\underline{2}\ /\ \underline{y}))\ \geqslant\ (\underline{x}\ +\ \underline{1})}{\dfrac{\underline{\quad}\qquad \underline{\quad / \quad}\qquad \underline{\quad + \quad}}{\underline{\quad +\quad}\qquad\qquad \underline{\quad \geqslant \quad}}}}{\underline{\quad}}$$
$$\dfrac{Num\ /\ Num \qquad Num\ +\ Num}{Num + Num \qquad\qquad Num \qquad\qquad Num \geqslant Num}$$
$$Bool$$

Var	Tipo
x	*Num*
y	*Num*

a) $x + 2 * 3 = 1$

b) $(x - 1) * (x - 2) = x + 6$

c) $y = x = 3$

d) $10 \geqslant x \geqslant 0$

Apéndice

Niveles de Precedencia

$E(x := a)$, .	sustitución y evaluación
$\sqrt{}, (\cdot)^2$	raíces y potencias
$*, /$	producto y división
máx, mín	máximo y mínimo
$+, -$	suma y resta
$=, \leqslant, \geqslant$	operadores de comparación
$\neg$	negación
$\vee \wedge$	disyunción y conjunción
$\Rightarrow \Leftarrow$	implicación y consecuencia
$\equiv \not\equiv$	equivalencia y discrepancia

Tipos de los operadores

$-x$	$- : Num \to Num$
x^y	$(-)^{(-)} : Num \to Num \to Num$
$\sqrt[r]{x}$	$\sqrt{} : Num \to Num \to Num$
$x * y$	$* : Num \to Num \to Num$
x/y	$/ : Num \to Num \to Num$
x máx y	máx : $Num \to Num \to Num$
x mín y	mín : $Num \to Num \to Num$
$x + y$	$+ : Num \to Num \to Num$
$x - y$	$- : Num \to Num \to Num$
$x > y$	$>: Num \to Num \to Bool$
$x \geqslant y$	$\geqslant: Num \to Num \to Bool$
$x < y$	$<: Num \to Num \to Bool$
$x \leqslant y$	$\leqslant: Num \to Num \to Bool$
$x = y$	$=: Num \to Num \to Bool$
$\neg p$	$\neg : Bool \to Bool$
$p \wedge q$	$\wedge : Bool \to Bool \to Bool$
$p \vee q$	$\vee : Bool \to Bool \to Bool$
$head.xs$	$head : [A] \to A$
$tail.xs$	$tail : [A] \to [A]$
$x \triangleright xs$	$\triangleright : A \to [A] \to [A]$
$xs \triangleleft x$	$\triangleleft : [A] \to A \to [A]$
$xs \uparrow n$	$\uparrow: [A] \to Int \to [A]$
$xs \downarrow n$	$\downarrow: [A] \to Int \to [A]$
$xs +\!\!+ ys$	$+\!\!+ : [A] \to [A] \to [A]$
$\#xs$	$\# : [A] \to Int$
$xs.n$	$(-).(-) : [A] \to Int \to A$

Operadores en `haskell`

x^2	`x^2`
x^n	`x^n` con n entero
x^p	`x**p`
$\sqrt{x}$	`sqrt x`
$\sqrt[r]{x}$	`x**(1/r)`
x máx y	`x ‘max‘ y` o bien `max x y`
x mín y	`x ‘min‘ y` o bien `min x y`
$x \geqslant y$	`x >= y`
$x \leqslant y$	`x <= y`
$x = y$	`x == y`
$\neg p$	`not p`
$p \wedge q$	`p && q`
$p \vee q$	`p || q`
$head.xs$	`head xs`
$tail.xs$	`tail xs`
$x \triangleright xs$	`x:xs`
$xs \triangleleft x$	`xs ++ [x]`
$xs \uparrow n$	`take n xs`
$xs \downarrow n$	`drop n xs`
$xs +\!\!+ ys$	`xs ++ ys`
$\#xs$	`length xs`
$xs.n$	`xs !! n`