Material de estudio

Araceli Acosta, Renato Cherini, Miguel Pagano, Leticia Losano

Enero de 2014

Índice general

Capítulo 1

Revisando la aritmética

En este primer capítulo comenzaremos trabajando con los conceptos matemáticos con los que más familiarizado estás: los números y la aritmética. Pero vamos a estudiarlos desde una perspectiva formal, es decir, vamos a analizarlos como componentes de un sistema formal. Esto implicará que hagamos explícitas muchas reglas que utilizás implícitamente desde la primaria así como también introducir un vocabulario específico para referirse a algunas propiedades y conceptos.

1.1. Elementos sintácticos y su semántica *intuitiva*

A continuación vamos a presentar la pieza inicial de nuestro sistema formal: la aritmética. Con el paso de las unidades lo iremos complejizando, agregando nuevas piezas y formalizando gradualmente las ya presentadas.

En primer lugar, presentaremos la **sintaxis** que nos permitirá construir expresiones. La sintaxis se ocupa de definir qué símbolos podremos utilizar y de la forma correcta en que deben estar dispuestos para armar expresiones.

Conceptos teóricos

Nuestras expresiones podrán tener la siguiente estructura:

- Un número, por ejemplo: 1, 35, 8. A estas expresiones les llamaremos **constantes**.
- Una letra, por ejemplo: x, y, z. A estas expresiones les llamaremos **variables**. Como en la secundaria, las variables pueden ser reemplazadas por cualquier otra expresión.
- Si ya construimos dos expresiones, por ejemplo 3 y x, podemos construir expresiones más complicadas combinándolas con **operadores**. Por el momento los únicos operadores que utilizaremos serán:
  - La suma. Por ejemplo, combinando las variables x e y a través del operador $+$ se puede formar la expresión: $x + y$
  - La resta. Por ejemplo, combinando la variable z con la constante 6 a través del operador $-$ se puede formar la expresión: $z - 6$
  - La multiplicación. Por ejemplo, combinando las variables a y b a través del operador $*$, se puede formar la expresión $a * b$
  - La división. Por ejemplo, combinando las constantes 6 y 3 a través del operador / se puede formar la expresión: $6/3$
  - La potenciación. Por ejemplo, combinando la constante 2 y la variable x se puede formar la expresión: 2^x

  A su vez, cada una de estas expresiones puede volver a combinarse con otras formando expresiones más complejas, por ejemplo: $\frac{(x+y)}{2^x} + 6/3$.

Actividad

A partir de las tres reglas anteriores:

1. Construí tres expresiones con al menos dos operadores. ¿Cómo podrías justificar que tus expresiones están bien construidas?
2. Presentá dos expresiones que estén mal construidas y da razones para justificar porqué considerás que están mal construidas.

La sintaxis nos permite, entonces, construir expresiones. Pero necesitamos también pensar en qué es lo que éstos símbolos y expresiones *representan*, es decir, en el significado de nuestras expresiones. De esto se ocupa la **semántica**. Diremos que todos los símbolos que hemos presentado en el listado anterior representarán **valores de tipo numérico**. Así, el símbolo 1 representa al valor uno, el 5 al valor cinco, etc. Así, haremos una distinción entre el símbolo utilizado para representar a un valor y el valor en sí mismo. Esta distinción no suele hacerse durante la secundaria pero aquí, visto que a medida que vayamos avanzando no sólo trabajaremos con números sino también con otros tipos de datos, la distinción nos será muy útil.

Es importante notar que un mismo valor puede representarse a través de distintas expresiones. Por ejemplo, el valor siete se puede representar a través de la constante 7 o a través de las expresiones $5 + 2$, $8 - 1$, $2 * 3 + 1$, etc.

Retomando la discusión generada a partir de la actividad anterior, podemos decir que otra forma de analizar si una expresión está bien construida es mostrar que representa a un determinado valor. De esta forma, la expresión $1^5 + 5$ está bien construida porque representa al valor 6 mientras que la expresión $7*$ no lo está porque no representa a ningún valor.

1.1.1. Relaciones: su semántica *intutiva* y su sintaxis

Contando con constantes, variables y operadores sólo podemos escribir expresiones pero no disponemos, en nuestro lenguaje formal, de herramientas para expresar relaciones o propiedades sobre las expresiones. Así, no podemos decir que la expresión $2 + 1$ es igual a la expresión 3 o que x es menor a $x + 1$.

Para superar esta limitación, a nuestro sistema formal vamos a agregarle **relaciones**. Cuando combinamos dos expresiones usando un símbolo de relación decimos que construimos una **fórmula**. Una fórmula es un tipo particular de expresión.

Los símbolos de relaciones que consideraremos por el momento serán: $=$, $\neq$, $<$, $\leqslant$, $>$ y $\geqslant$. La semántica de cada uno de ellos será la siguiente:

- Igualdad ($=$): Cuando decimos que $x = y$ estamos diciendo que x representa el mismo valor que y y viceversa.
- Diferencia ($\neq$): Cuando $x \neq y$ significa que x representa un valor distinto al de y y viceversa.
- Menor ($<$): Al escribir $x < y$ estamos diciendo que el valor que representa x es estrictamente menor al valor que representa y.
- Menor o igual ($\leqslant$): Cuando decimos que $x \leqslant y$ estamos afirmando que el valor que representa x es menor o igual que el valor que representa y.
- Mayor ($>$): Cuando $x > y$ significa que x representa un valor estrictamente mayor a y.
- Mayor o igual ($\geqslant$): Al escribir $x \geqslant y$ estamos afirmando que el valor que representa x es mayor o igual al valor que representa y.

Aquí hemos explicitado la semántica de cada una de las relaciones que utilizaremos usando variables, esto nos permite enunciarla de manera general. Por supuesto, al igual que los operadores $+$, $-$, etc, las relaciones también pueden utilizarse para relacionar dos expresiones cualesquiera, por ejemplo la fórmula: $3 * x \leqslant 45 - 7$.

Actividad

Conociendo la semántica de cada una de las relaciones podemos comenzar a pensar en interrelaciones entre ellas:

1. ¿Qué relación es la "contraria" u "opuesta" a la igualdad?
2. ¿Qué relación es la "contraria" u "opuesta" al menor? ¿Y al mayor o igual?

Actividad

¿Qué valor representa la fórmula $2 + 3 = 5$?

Con la introducción de las relaciones buscábamos tener herramientas con las cuales comparar o relacionar dos expresiones dadas. Ahora bien, el resultado de esa comparación evidentemente no es de tipo numérico. Al comparar dos expresiones podemos obtener dos respuestas: verdadero, por ejemplo en el caso $6 \neq 4$, o falso, como en el caso $5 > 1$. Es decir, la semántica de las fórmulas es representar dos posibles valores: verdadero o falso. Así, diremos que las fórmulas no representan valores de tipo numérico como las expresiones sino que representan **valores de tipo booleano**.

Actividad

Pensemos ahora acerca de un aspecto de la sintaxis de los operadores de relación. Considerá las siguientes fórmulas:

1. $2^3 + 2 = 8 + 2 = 10$
2. $4 = 3 + 1 = 5$

¿Son expresiones bien construidas? ¿Por qué?

Dentro de este curso consideraremos que cada operador de relación sólo puede tomar dos expresiones y devolver un valor booleano. Estas serán las únicas fórmulas que juzgaremos como bien construidas.

1.1.2. Reglas de precedencia

Actividad

¿Te parece que las siguientes expresiones representan el mismo valor?

1. $(2 - 3) * 5$
2. $2 - (3 * 5)$
3. $2 - 3 * 5$

Para saber cómo evaluar una expresión compleja es necesario contar con reglas que establezcan qué operación debe realizarse en primer lugar, en segundo lugar, etc. Estas reglas son llamadas **reglas de prescendencia**. Podemos resumir estar reglas para los operadores incluidos en nuestro sistema formal en la siguiente tabla:

Conceptos teóricos

Reglas de precedencia

+ precendencia		
	$\sqrt{\cdot}, (\cdot)^2$	raíces y potencias
	$*, /$	producto y división
$\downarrow$	$+, -$	suma y resta
	$=, \leqslant, \geqslant$	relaciones
- precedencia		

Esta tabla nos dice que si queremos evaluar una expresión debemos resolver, en primer lugar, las raíces y potencias —si es que están presentes en la expresión—, luego los productos y divisiones, etc.

Los símbolos de paréntesis son los que nos permiten alterar el orden dado por las reglas de precedencia. Por ejemplo, en la conocida expresión del binomio al cuadrado:

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

Se hace necesario utilizar el paréntesis en el lado izquierdo de la igualdad porque si no la potencia sólo se aplicaría a la variable b.

Dada una expresión, el orden dado por las reglas de precedencia y los paréntesis presentes se puede marcar a partir de subrayados. Por ejemplo, supongamos que queremos evaluar la siguiente expresión para conocer qué valor representa:

$$(\frac{6}{2} + 3)^2 - 1 = 3 * 11$$

En primer lugar deben resolverse la división y el producto, por lo que subrayamos estas partes de la expresión:

$$(\underline{\frac{6}{2}} + 3)^2 - 1 = \underline{3 * 11}$$

Luego debe resolverse la suma dentro del paréntesis, a continuación la potencia cuadrada, luego la resta y, finalmente, la igualdad. Cada uno de estos pasos corresponde a un subrayado:

$$(\underline{\frac{6}{2}} + 3)^2 - 1 = \underline{3 * 11}$$

$$\underline{\quad}$$

$$\underline{\quad\quad}$$

$$\underline{\quad\quad\quad\quad}$$

$$\underline{\quad\quad\quad\quad\quad\quad\quad\quad}$$

Actividad

1. En las siguientes expresiones marcá cada etapa de reescritura dada por las reglas de precedencia y los paréntesis con un subrayado diferente.

   a) $3 + 4 * x = 4$

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

   c) $(a^3 + b)^2 * 23$

   ¿Cambiaría la expresión si encerraras en paréntesis cada parte subrayada?

2. ¿Son correctos los siguientes subrayados?

   a) $\underline{(5 * 4)} * \underline{(2 - y)} + 20 \geqslant 0$

   b) $(\underline{7 * x} + \underline{8 * z}) * \underline{y - 9} \neq \underline{4 * (y} - 2)$

Actividad

Agregá los paréntesis que sean necesarios para hacer que las siguientes formulas sean verdaderas:

- $2 * 3 + 1 \geqslant 8$
- $5 * 4 - 2\ ^2 + 7 = 27$

1.1.3. Variables

Actividad

Utilizando solo las constantes, los operadores y las relaciones podemos resolver una enorme cantidad de cálculos, tan complicados como se nos ocurran. Una calculadora básicamente funciona en base a estos elementos. Pudiendo hacer todo esto es válido preguntarnos: ¿para qué es necesario contar con variables? ¿qué novedades aportan? ¿qué respuestas podrías darles a estas preguntas?

Las variables nos permiten, al menos, dos cosas:

- En primer lugar, **descubrir** cosas. Seguramente en la secundaria y en el Curso de Nivelación has resuelto numerosas veces problemas como el siguiente: En diciembre Carlos cobró $1000 de aguinaldo de los cuales gastó $340 en el supermercado. El resto lo dividió en partes iguales entre sus tres hijos. ¿Cuánto dinero le tocó a cada hijo?

  Al formalizar este problema con la ecuación: $x = \frac{(1000-340)}{3}$ y realizar los cálculos típicos para "despejar" la variable, lo que finalmente conseguimos es descubrir para qué valores de la variable x la ecuación es verdadera.

- En segundo lugar, **generalizar** cosas. A lo largo de la historia los científicos han arribado a un conjunto de expresiones que consideran válidas, muchas de ellas muy conocidas. Por ejemplo, en física la segunda ley de Newton establece que $F = m * a$, en matemática la regla para elevar al cuadrado un binomio es: $(a + b)^2 = a^2 + 2ab + b^2$ Estas reglas establecen que una propiedad es válida para todos los elementos de un determinado conjunto, tarea que no podría llevarse a cabo sin la ayuda de las variables.

Si no tuviéramos un símbolo para denotar "a cualquier número" deberíamos escribir infinitas reglas, una para cada constante. Así, uno de los rasgos más importantes de las variables es que pueden ser sustituidas por otras expresiones.

Actividad

¿Podés establecer una relación entre las dos fórmulas?

1. $(a^2 - b^2) = (a + b) * (a - b) \longrightarrow (4^2 - y^2) = (4 + y) * (4 - y)$
2. $(a * b) * c = a * (b * c) \longrightarrow (\frac{2*x}{y^3} * 350) * 23 = \frac{2*x}{y^3} * (350 * 23)$
3. $a * (b + c) = a * b + a * c \longrightarrow 8 * (z^2 + w) = 4 * z^2 + 8 * w$
4. $a^2 + 2 * a * b + b^2 = (a + b)^2 \longrightarrow a^2 + 2 * a * a + a^2$

Cuando reemplazamos las variables de una regla dada decimos que estamos creando una **instancia** de esa regla. Así,

$$3^2 - 7^2 = (3 + 7) * (3 - 7)$$

es una instancia de la regla general que podría enunciarse como:

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

En este caso particular, la variable a de la regla ha sido sustituida por la constante 3 y la variable b por la constante 7.

Se pueden crear instancias de reglas reemplazando las variables de la regla por expresiones tan complejas como queramos —por ejemplo, que contengan otras variables como en el caso de la actividad anterior—. Pero sí es necesario, para no alterar el significado de la regla original que todas las ocurrencias de una variable sean reemplazadas por la misma expresión.

1.2. Simplificación de expresiones numéricas

Actividad

Simplificá las siguientes expresiones aritméticas.

1. 3^{2*2}
2. $(27^2 - 25^2)$
3. $(5 + 2 * (-4))^2/(-3) - (5 * (-4) + (-6)) - (-1)^2$

Cuando simplificamos una expresión como las de la actividad anterior lo que hacemos es ir reescribiéndola en términos más simples de modo que sea más fácil reconocer qué valor está representando. Para las expresiones numéricas la reescritura más simple a la que podemos arribar es un número que es el resultado de aplicar las operaciones en el orden adecuado. Por ejemplo, 81 en la primer expresión. A este número le llamaremos **forma canónica** de la expresión.

Las formas canónicas de las expresiones que involucren constantes numéricas será siempre un número. Veamos cómo encontrarla para algunos casos particulares:

- La forma canónica de una expresión que sólo involucre a una constante es la misma constante. Así, la forma canónica de la expresión 7 es 7.

- Si tenemos una expresión que contenga a una suma, si los dos sumandos están expresados en forma canónica, entonces la forma canónica de la expresión completa será el número que se obtiene sumarlos. Por ejemplo, la forma canónica de la expresión $5 + 8$ es 13. Si ambos o alguno de ellos no está expresado en forma canónica primero es necesario expresarlos de esta manera y luego calcular la forma canónica de la suma de ambos. Por ejemplo, en la expresión $3*2+6$, $3*2$ no está expresado en forma canónica. Primero se encuentra esta forma canónica —6— y luego se calcula la forma canónica de la expresión completa que es 12.

En general, la manera de llegar a las formas canónicas es aplicar las reglas semánticas de los operadores que vimos anteriormente hasta que ya no quede ningún operador sin evaluar. Para expresiones que incluyen variables la semántica de los operadores no se puede aplicar (porque desconocemos el valor de las variables) y por lo tanto su forma canónica será aquella en donde todos los restantes operadores hayan sido reducidos.

Actividad

¿Son correctas las siguientes simplificaciones?

1. $\frac{\frac{2}{7}+\frac{1}{13}*(-\frac{1}{5}+\frac{3}{2})}{(-2)*\frac{1}{5}+\frac{3}{5}} = \frac{\frac{20+7}{70}}{\frac{1}{5}} = \frac{27}{14}$
2. $(\frac{1-\frac{5}{4}}{\sqrt[3]{\frac{11}{8}+2}} + \sqrt{(\frac{1}{2})^4})^2 = (-\frac{2}{36} + \frac{1}{4})^2 = \frac{7}{36}$

Actividad

Simplificá usando el evaluador de Equ las siguientes expresiones:

1. $5 * 27 + 2 * (5 + 7 * 25 + (4 - 27^3)) * 2$
2. $\frac{2*4}{2^2-16}$
3. $(2 + 3)^4$

1.2.1. Simplificando fórmulas

Hasta ahora hemos trabajado mayormente simplificando expresiones que contengan sólo constantes numéricas y para ello hemos introducido la noción de forma canónica. En esta sección vamos a comenzar a manipular fórmulas. La pregunta que intentaremos responder será: ¿como simplificarlas? Para comenzar a pensar cuáles deberían ser las formas canónicas para fórmulas te proponemos la siguiente actividad.

Actividad

Simplificá las siguientes fórmulas en Equ: $2^3 + 2 = 5 + 5$ y $2^3 + 2 = 2^4$.

Cuando trabajábamos con expresiones numéricas, las relaciones disponibles eran $=, <, >, \ldots$, y principalmente utilizamos la relación de igualdad para escribir la simplificación de esta clase de expresiones. Así, cuando escribimos:

$$\begin{array}{l} \quad 2 + 3 \\ = \{ \text{ Definición de suma } \} \\ \quad 5 \end{array}$$

el símbolo $=$ indica que la expresión de arriba $2+3$ tiene el mismo valor que la expresión de abajo, 3.

Cuando trabajamos con fórmulas, que son expresiones de tipo booleano, necesitamos entonces una relación que nos permita comparar sus valores, una relación de igualdad pero para valores de verdad, en lugar de valores numéricos. La "igualdad" de booleanos se denomina "equivalencia" y se denota con el símbolo $\equiv$. Así, si escribimos

$$
\begin{array}{ll}
 & 2+3=5 \\
\equiv & \{ \text{ Definición de suma } \} \\
 & 5=5 \\
\equiv & \{ \text{ reflexividad } \} \\
 & True
\end{array}
$$

estamos indicando que la primer fórmula representa al mismo valor de verdad que la tercera. Tal como sucedía con la igualdad numérica, la equivalencia también tiene la propiedad de transitividad, por lo que normalmente vamos a escribir simplificaciones de muchos pasos también para fórmulas.

Hasta ahora $True$ y $False$ no son expresiones, porque no son números, ni son variables, ni expresiones complicadas construidas con operadores o relaciones. Para poder usarlas necesitamos extender nuestro lenguaje formal; lo haremos integrándolas a la categoría de las constantes. Entonces ahora además de los números $1, 2, 3, \ldots$, tenemos $True$ y $False$ como constantes.

Actividad

¿Pódríamos considerar a $True + 3$ como una expresión?

1.3. ¿Cómo asegurarnos que una expresión tiene sentido? El problema del tipado

Actividad

Considerá las siguientes expresiones:

1. $4 + 3$,
2. $True + 3$,
3. $x + 3$.

¿Pueden simplificarse? ¿Por qué constantes podrías sustituir la x para que la expresión se pueda simplificar?

La introducción de $True$ y $False$ implicó extender la idea de expresión. Así por ejemplo, tenemos expresiones, sin variables, a las que no les podemos encontrar forma canónica; es decir que tenemos expresiones que no tienen sentido porque no representan ningún valor que podamos interpretar. Ahora tenemos en la bolsa de expresiones cosas que no tienen sentido, por eso necesitamos herramientas que nos permitan distinguirlas de las que sí tienen sentido.

El concepto clave para esta distinción es el concepto de **tipo**. Los tipos son categorías que se les asignan a las expresiones. Intuitivamente las expresiones pueden ser interpretadas como elementos que pertenecen al conjunto del tipo. Tendremos dos tipos: Nat, categoría a la que pertenecen todas las constantes numéricas, y $Bool$, a la que pertenecen las constantes $True$ y $False$.

Las constantes ya tienen tipos; ahora podemos preguntarnos cómo calcular el tipo de una expresión compleja. Esto es interesante porque nos puede ahorrar la tarea de embarcarnos en la simplificación de una expresión que no tenga forma canónica y, por lo tanto, que no tenga sentido. Esto puede no parecer muy importante a primera vista, pero imaginá que necesitás reducir una expresión que sea tan larga que ocupe toda una hoja de tu cuaderno. Ponerte en el trabajo de simplificarla para darte cuenta, media hora después de que ni siquiera tiene sentido puede ser frustrante. Ahora imaginá la misma situación pero para una expresión diez veces más larga. Aquí la cuestión ya parece más grave. Los programas son expresiones —en general, sumamente largas— y asegurarse de que tengan el tipo apropiado es parte de verificar que estén bien definidos y sean correctos.

¿Qué herramienta podemos usar para saber si podremos encontrar la forma canónica de una expresión compleja?

La herramienta fundamental que vamos a usar es el **tipado** de expresiones; esto significa asignarle un tipo a una expresión. Ese tipo no puede ser cualquiera; para averiguarlo construimos **árboles de tipado**. Tenemos la gran ventaja de que nuestro programa, EQU , construye esos árboles de tipado por nosotros.

Actividad

Nuestro primer objetivo es descubrir cómo EQU construye el árbol de tipado que justifique el tipo que le asigna a la expresión:

$$2 * (3 + 5) + (1 + (2 - 4 * 5))$$

Equ dice que Nat es el tipo de la expresión completa; y que los árboles de las subexpresiones $2 - 4 * 5$ y de $3 + 5$ son respectivamente:

$$2 \; - \; 4 \; * \; 5$$

$$\frac{Nat * Nat}{\quad}$$

$$\frac{Nat - \quad Nat}{Nat}$$

$$3 \; + \; 5$$

$$\frac{Nat + Nat}{Nat}$$

Observando estos árboles, intentá construir en papel el árbol para la expresión entera.

Actividad

Al construir el árbol de la actividad anterior te encontraste con situaciones parecidas en diferentes ramas. Ahora te proponemos que construyendo los árboles de tipo de las siguientes expresiones, sistematices en un esquema general la forma en que se construyen árboles para expresiones con operadores aritméticos.

1. *a*) $8 * 24$
   b) $2 * 16$
   c) $98 + 90$
   d) $3 + 17$
2. ¿Sería diferente el árbol de la resta? ¿Y el árbol de los operadores de relación?
3. Usando las conclusiones de la respuesta anterior, construí en papel el árbol para $(3 + 5) * 2$. Contrastalo con el árbol que construye EQU . Si es necesario revisa tu sistematización de la forma de construir árboles para estos operadores.
4. Construí usando EQU el árbol de tipado para $(2 = 3) + 1$. Discutí con tus colegas la situación.

Actividad

Dadas las siguientes expresiones:

1. $(-(5 + x) + ((3 * 6)/(4 * 5))) * (8 * 5)$
2. $(((2)^2 + 5) - (4 * 2)/(4)) + 1 = ((5 * x)/8) + (3 * 5)^3$
   a) Construí usando EQU el árbol de tipo de cada una.
   b) Utilizando EQU sacá todos los paréntesis que sean *superfluos*, es decir, que su eliminación no modifique ni su semántica ni su árbol de tipo.

1.3.1. Tipando expresiones con variables

Ahora pasaremos a tipar expresiones que contengan variables. Para ello, será necesario utilizar los esquemas generales de tipado para los operadores que construiste en la actividad anterior.

Actividad

Utilizá EQU para averiguar qué tipo tiene que tener cada variable para que la expresión esté bien tipada.

1. $x + 2 * 3 = 1$
2. $(x - 1) * (x - 2) = x + 6$
3. $y + x = 3$
4. $x \geqslant 0$

1.4. Validez y satisfactibilidad

Actividad

Simplificá las siguientes fórmulas utilizando EQU :

1. $6 * x + 8 = x + 3$
2. $(a + b)^2 = a^2 + 2 * a * b + b^2$

¿Como interpretarías la solución en cada caso? ¿Ademas de estas dos situaciones, se te ocurre otra situación a la que puedas llegar cuando simplifiques una formula?

1.4.1. Validez

Cuando trabajamos con expresiones que contengan variables podemos hacerlo centrándonos en diferentes aspectos. Algunas veces buscamos encontrar los valores que la(s) variables deben adoptar para que la expresión sea verdadera. Otras veces manipulamos expresiones para demostrar que son siempre verdaderas independientemente de los valores que tomen las variables. En este curso nos interesa particularmente este segundo tipo de trabajo, es decir demostrar que ciertas fórmulas expresan verdades generales.

Conceptos teóricos

Validez y no validez de una fórmula:
Cuando el valor que representa una fórmula es verdadero, independientemente de los valores que tomen las variables que ocurren en ella, decimos que es **válida**. Cuando esto no ocurre decimos que la fórmula es **no válida**.

Los teoremas, lemas y proposiciones que podés haber visto en materias como Álgebra son ejemplos de fórmulas válidas. Así, cuando se enuncian teoremas o reglas como el que sigue:

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

en realidad se quiere decir que

$$\textit{Para todo } a \textit{ y } b \in \mathbb{N} \textit{ vale que } a^2 - b^2 = (a + b) * (a - b)$$

En este caso se dice que las variables a y b son **variables libres** porque pueden reemplazarse por cualquier otro símbolo y la regla seguirá teniendo el mismo significado. Por ejemplo, el mismo teorema podría haber sido enunciado utilizando las letras x e y:

$$\textit{Para todo } x \textit{ e } y \ \in \mathbb{N} \textit{ vale que } x^2 - y^2 = (x + y) * (x - y)$$

y el significado sería el mismo. Volveremos a retomar el concepto de variable libre más adelante en el curso.

Actividad

Demostrá que las siguientes fórmulas son válidas:

- $4 * x + 14 = 2 * (2 * x + 5) + 4$
- $(x - 1) * (x + 1) = x^2 - 1^2$

Actividad

Decidí si las siguientes fórmulas son válidas o no:

1. $x + x = 2 * x$
2. $x = x + 1$

¿Cómo interpretas el último resultado? ¿Se puede obtener mas información a partir de ese resultado?

Actividad

Simplificá la siguiente formula: $2 * x - 18 = 4$
¿Cómo podrías dar una prueba fehaciente de que no es válida?

Actividad

Se le pidió a Pedro que demostrara que $x^2 - 1 = (x - 1) * (x + 1)$ es una fórmula válida. Pedro presentó lo siguiente:

Hipótesis: $x = 1$

$$\begin{aligned}
& (x - 1) * (x + 1) \\
= & \{ \text{ Hipótesis: Reemplazo } x \text{ por su valor } \} \\
& (1 - 1) * (1 + 1) \\
= & \{ \text{ Distributividad } \} \\
& 1^2 + 1^2 - 1^2 - 1^2 \\
= & \{ \text{ Opuesto de la suma } \} \\
& 1^2 - 1^2 \\
= & \{ \text{ Definición de potencia } \} \\
& 1^2 - 1 \\
= & \{ \text{ Por hipótesis } \} \\
& xr - 1
\end{aligned}$$

¿qué podes decir acerca de la resolución de Pedro?

1.4.2. Satisfactibilidad

Como ya dijimos, cualquier valor posible de las variables de una fórmula dada constituye una *solución* para la misma, decimos que es **válida**. Caso contrario la fórmula es no válida. Pero existen más posibilidades, puede suceder que la fórmula sea verdadera para algunos casos, es decir, para alguna asignación de valores a las variables mientras que sea falsa para otras asignaciones. De este caso se ocupa el siguiente concepto:

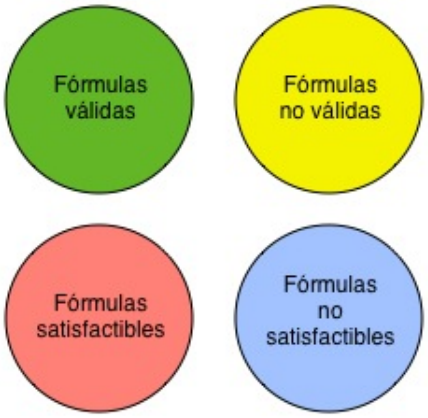
Conceptos teóricos

Satisfactibilidad y no satisfactibilidad de una fórmula:
La fórmulas que tienen al menos una solución, es decir, que se hacen verdaderas para al menos una asignación de valores a sus variables, se llaman **satisfactibles**. Cuando no existe ninguna asignación que haga válida la fórmula, diremos que es una fórmula **no satisfactible**.

Actividad

¿Se te ocurre cómo utilizar EQU para demostrar rápidamente que la formula $3 * x - 16 = x$ es satisfactible?

Hasta aquí tenemos cuatro categorías para clasificar las fórmulas: valida, satisfactible, y sus negaciones: no válida y no satisfactible. Cada una de estas categorías determina un conjunto de fórmulas, esquemáticamente se podría plantear así:



Actividad

¿Qué relaciones podés establecer entre estos cuatro conjuntos de fórmulas? ¿Creés que alguno de ellos está incluido en otro? ¿Creés que algunos de ellos son disjuntos? Justificá tu respuesta.

Actividad

Las siguientes fórmulas 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 satisfactibles y qué valores las satisfacen.

1. $3 * x + 1 = 3 * (x + 1)$
2. $x + (y * z) = (x + y) * (x + z)$
3. $x + 100 = x$

Actividad

Dadas las siguientes fórmulas que involucran variables:

1. $6 * x + 8 = x + 3$
2. $2 * x + 3 = 5$
3. $x + (y * z) = (x + y) * (x + z)$
4. $x + 100 = x + 1$
5. $x * (y + z) = x * y + x * z$
   a) Dale valor a las variables y, utilizando el comando Evaluar, observá la reducción paso a paso.
   b) ¿Sirve este mecanismo para verificar satisfactibilidad? ¿Sirve para verificar la validez? ¿y con respecto a la no satisfactibilidad? ¿y con respecto a la no validez?

Como hemos visto a través de las actividades, el proceso de simplificación de una formula se puede utilizar para diferentes fines. Cada una de estas simplificaciones es una **demostración** de algún hecho: que una ecuación tiene una o mas soluciones, que no las tiene, o que cualquier valor es solución.

Al mismo tiempo, hemos visto algunas **estrategias** para llevar adelante estas demostraciones: simplificando la fórmula hasta llegar a una equivalente cuyo valor es obvio, a través de ejemplos y contraejemplos.

Se debe tener presente que, en matemática, cuando se habla de demostración se está haciendo referencia a mostrar que **cierta formula es válida**.

Actividad

¿Qué estrategias se puede utilizar, en cada caso, para demostrar que una formula pertenece a alguna de las cuatro categorías: válida, no válida, satisfactible y no satisfactible?

Actividad

Decidí si son *válidas* o *no válidas* y *satisfactibles* o *no satisfactibles* las siguientes fórmulas. Justificá en cada caso.

1. $\sqrt{x} + \sqrt{y} = \sqrt{x + y}$
2. $(a - b) * (a + b) * ((a + b)^2 - 2 * a * b) = a^4 - b^4$
3. $x^2 + 2 * x + 4 = 0$

Actividad

Da ejemplos y una justificación apropiada de una fórmula:

1. válida (y por lo tanto satisfactible).
2. satisfactible pero no válida.
3. no satisfactible (y por lo tanto no válida).

1.5. Resumiendo

Actividad

1. Construí un listado en donde aparezcan los principales conceptos que desarrollamos en el capítulo.
2. Con el objetivo de establecer relaciones entre los conceptos del listado anterior elaborá oraciones en donde figuren al menos dos de ellos.
3. A partir del punto 1 y 2 construí un mapa conceptual en donde aparezcan resumidos y relacionados los conceptos tratados en este capítulo.

Capítulo 2

Funciones y recursión

Este capítulo está enteramente dedicado a dos conceptos claves en programación: las funciones y la recursión. En primer lugar, vamos a retomar las nociones vinculadas con las funciones que viste en la secundaria y en el cursillo. Luego pasaremos a estudiar las distintas formas de definirlas. Incluir funciones dentro de nuestro lenguaje nos permitirá ampliarlo, teniendo capacidad para expresar muchas más cosas. A continuación trabajaremos sobre el concepto de recursión para finalizar aprendiendo a definir funciones recursivas.

2.1. Un breve repaso

Actividad

Para revisar el concepto de función te proponemos que leas el siguiente texto y respondas a las preguntas a continuación:

> En matemáticas, se dice que una magnitud o cantidad es función de otra si el valor de la primera depende exclusivamente del valor de la segunda. Por ejemplo, supongamos que queremos viajar desde Córdoba a Buenos Aires en auto. La duración del viaje T dependerá de la distancia d entre Córdoba y Buenos Aires y de la velocidad v que lleve el auto. Del mismo modo, el área A de un círculo es función de su radio r. Estas magnitudes a veces pueden vincularse a través de la proporcionalidad directa o ser inversamente proporcionales. Así, el área A de un círculo es proporcional al cuadrado de su radio, lo que se expresa con una fórmula del tipo $A = \pi * r^2$ y la duración de un viaje T es inversamente proporcional a la velocidad del vehículo ($T = d/v$).
>
> A la magnitud que se encuentra a la izquierda de estas fórmulas (el área, la duración) se la denomina *variable dependiente*, y a la magnitud de la que depende (el radio, la velocidad) se la llama *variable independiente*.
>
> De manera más abstracta, el concepto general de función se refiere a una regla que asigna a cada elemento de un primer conjunto un único elemento de un segundo conjunto. Por ejemplo, cada número entero posee un único cuadrado, que resulta ser un número natural (incluyendo el cero):
>
> $$\begin{array}{ccccccccc} \ldots & & -2 & -1 & 0 & 1 & 2 & 3 & \ldots \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & & +4 & +1 & 0 & +1 & +4 & +9 & \end{array}$$
>
> Esta asignación constituye una función entre el conjunto de los números enteros y el conjunto de los naturales.

Aunque las funciones que manipulan números son las más conocidas, no son el único ejemplo: puede imaginarse una función que a cada palabra le asigne su letra inicial:

...	estación	museo	arrollo	rosa	avión	...
	↓	↓	↓	↓	↓	
	E	M	A	R	A	

Esta es una función entre el conjunto de las palabras en español y el conjunto de las letras del alfabeto español.

- ¿Qué es una función? Da algunos ejemplos de funciones que se te ocurran.
- ¿Para qué servirán las funciones en programación?

Intentemos ahora formalizar un poco nuestras reflexiones. Diremos que una **función** le asigna a los elementos de un conjunto, llamado **dominio**, elementos de otro conjunto, llamado **codominio**. La notación:

$$\texttt{f} : A \to B$$

indica que `f` es una función con dominio A y codominio B.

Para denotar a la aplicación de una función utilizaremos la siguiente notación:

$$\texttt{f}.a = b$$

para indicar que `f` le asigna el elemento $b \in B$ a a. Aquí, b se llamará el **valor** de f en el **argumento** a.

La notación habitual utilizada para denotar a la aplicación de funciones, $f(x)$, si bien es útil en Análisis Matemático y Álgebra, no lo es en el contexto en el que trabajaremos. Por eso hemos decidido utilizar el símbolo "."

Es importante que siempre distingas claramente entre una función `f` y la aplicación de ésta al valor x que se escribe como $\texttt{f}.x$. El símbolo `f` representa el conjunto de todas las asignaciones de elementos del dominio a elementos del codominio. En cambio una aplicación de la función, como $\texttt{f}.5$ representa sólo el valor que la función le asigna a ese argumento.

2.2. ¿Cómo definir funciones?

Dentro de este curso estaremos particularmente interesados en definir funciones. Para hacerlo, el primer paso que tenemos que dar es otorgarle un nombre a la función que queremos crear. Retomando el ejemplo de la actividad anterior si quisiéramos definir una función que dado un número entero devuelva su cuadrado, debemos, en primer lugar, nombrarla. Llamemos a esta función `cuadrado`.

Utilizaremos un símbolo especial para definir funciones que será $\doteq$. La introducción de este símbolo nos permitirá distinguirlo del símbolo de la igualdad ($=$).

Inicialmente podríamos intentar definir la función `cuadrado` enumerando cada uno de sus casos, de la siguiente manera:

$$\begin{aligned}
&\texttt{cuadrado}.0 \doteq 0\\
&\texttt{cuadrado}.1 \doteq 1\\
&\texttt{cuadrado}.(-1) \doteq 1\\
&\texttt{cuadrado}.2 \doteq 4\\
&\ldots
\end{aligned}$$

Actividad

Es claro que esta forma de definir funciones, además de poco práctica, es inviable visto que el dominio de la función es infinito, por lo que nunca terminaríamos de escribir su definición. ¿Cómo definirías de manera general a la función *cuadrado*?

Generalmente las funciones son definidas usando expresiones, como en los siguientes ejemplos:

$\texttt{f}.x \doteq 2x^2 + 4$

$\texttt{g}.y \doteq 3$

$\texttt{h}.z \doteq 7z$

Al definir una función usando una fórmula se pueden utilizar constantes, como 2 y 4 en el caso de `f` y también variables de distinto tipo. En los casos anteriores las variables x, y, z representan números naturales.

Una función también puede tener más de un argumento, por ejemplo:

$\texttt{f}.x.y \doteq x + y$

toma dos números naturales y devuelve como resultado la suma de ambos.

Actividad

En las siguientes definiciones identificá las variables, las constantes y el nombre de la función:

1. $\texttt{f}.x \doteq 5 * x$
2. $\texttt{duplica}.a \doteq a + a$
3. $\texttt{por2}.y \doteq 2 * y$
4. $\texttt{multiplicar}.zz.tt \doteq zz * tt$

2.3. Tipado de funciones

Además de dar una fórmula para definir a una función en este curso pediremos que el **tipo** de la misma sea declarado explícitamente. En esta sección introduciremos la notación y los conceptos necesarios para tipar funciones.

Como ya vimos, las funciones toman uno o más argumentos, todos ellos con un tipo asociado. Así, la función `f` definida por la siguiente fórmula:

$\texttt{f}.x \doteq x + 4$

toma un argumento de tipo Num.

A su vez, las funciones devuelven un valor que también tiene su tipo. En el caso de la función `f` definida anteriormente, el valor que esta función devuelve al ser evaluada es de tipo Num.

Para capturar esta característica de las funciones, en la declaración de su tipo utilizaremos el símbolo $\rightarrow$. Así, el tipo de f se declarará de la siguiente forma:

$\texttt{f} :: Num \rightarrow Num$

Veamos otro ejemplo. Tomemos la función `g` definida así:

$\texttt{g}.x.y \doteq 3x - x * y > 0$

Esta función toma dos argumentos de tipo Num y devuelve un valor de tipo $Bool$. Así, el tipo de `g` se declara de la siguiente forma:

$\texttt{g} :: Num \rightarrow Num \rightarrow Bool$

Sintetizando: Los tipos de los argumentos se listan primero, siguiendo el orden en el que serán llamados por la función, y en último lugar se coloca el tipo del resultado de evaluar la función.

Actividad

Declará el tipo de las siguientes funciones:

1. $\texttt{g}.y \doteq 8y$
2. $\texttt{h}.z.w \doteq z + w$
3. $\texttt{j}.x \doteq x \leqslant 0$

Cuando la definición de una función se vuelve más compleja puede ser necesario recurrir a un árbol de tipado para descubrir qué tipos deben tener las variables en los argumentos y el tipo final. Para ello podemos construir un árbol de tipo para el **cuerpo** de la función, es decir, para lo que está a la derecha del símbolo $\doteq$ utilizando EQU .

Actividad

Con la ayuda de EQU construí los árboles de tipo para las siguientes funciones. Usando esta herramienta descubrí el tipo que deben que tener sus argumentos y, finalmente, declará el tipo de cada una de las funciones:

1. $\texttt{g}.x \doteq 6 * x + 8 = x + 3$
2. $\texttt{h}.y.z. \doteq (y - 1) * (z - 2) - z + 6$
3. $\texttt{f}.w.z. \doteq (\frac{3w+1}{z+3} - 40)^{w.z}$

2.4. Expresiones y funciones

Una vez que tenemos definida una función esta pasa a ser parte de nuestro lenguaje de expresiones. Si una función tiene tipo resultado Num la aplicación de la función se puede utilizar para construir expresiones combinándola con los operadores de tipo numérico y otras funciones que tengan el tipo adecuado. Por ejemplo, se puede construir la expresión: $\texttt{duplica.(4+2)} + multiplicar.9.7 = 75$

Del mismo modo, los argumentos de una función pueden ser cualquier expresión del tipo adecuado. En particular posible evaluar una función en un argumento que a su vez sea la evaluación de otra función, siempre y cuando el tipado sea correcto.

Actividad

Dadas las siguientes definiciones de funciones:

$$\begin{array}{rcl} \texttt{g} & :: & Num \to Num \\ \texttt{g}.x & \doteq & 6 * x + 8 = x + 3 \\ \texttt{duplica} & :: & Num \to Num \\ \texttt{duplica}.a & \doteq & a + a \\ \texttt{por5} & :: & Num \to Num \\ \texttt{por5}.z & \doteq & 5 * y \\ \texttt{multiplicar} & :: & Num \to Num \to Num \\ \texttt{multiplicar}.x.y & \doteq & x * y \end{array}$$

Evalualas en los siguientes argumentos:

1. $\texttt{g}.7$
2. $\texttt{g}.(\texttt{multiplicar}.3.2)$
3. $\texttt{por5}.(\texttt{duplica}.8)$

2.5. Distintas formas de definir funciones

En esta sección nos ocuparemos de introducir la notación necesaria para definir funciones de maneras variadas según sea más conveniente.

2.5.1. Definiciones de funciones por casos

Hay ocasiones en las que una sóla fórmula no alcanza para definir una función. Así, existen funciones que para un conjunto de argumentos requieren una definición y para otro conjunto de argumentos necesitan de otra definición diferente. Es el caso, por ejemplo, de la función `espar` que dado un natural devuelve *true* si el número es par o $false$ si el número es impar. En este caso tendremos que definir una función que a todos los números pares les asigne un valor booleano y a todos los impares les asigne otro valor booleano.

Un mecanismo muy útil para definir a este tipo de funciones es el análisis por casos. Cuando se usa esta construcción, el valor que tomará la función dependerá de que ciertas expresiones booleanas sean ciertas o no.

Una **definición por casos** de una función tendrá la siguiente forma general:

$$\begin{array}{lcll} \texttt{f}.x & \doteq & (& B_0 \to f_0 \\ & & \square & B_1 \to f_1 \\ & & & \vdots \\ & & \square & B_n \to f_n \\ & &) & \end{array}$$

donde las B_i son expresiones de tipo booleano, llamadas **guardas** y las f_i son expresiones del mismo tipo que el resultado de f. Para un argumento dado el valor de la función se corresponde con la expresión cuya guarda es verdadera para ese argumento. Para que esto tenga sentido es requerimiento para que una función por casos esté bien definida que las guardas sean *disjuntas* entre sí, esto quiere decir que no puede ocurrir que un argumento haga verdadera dos o mas guardas al mismo tiempo.

Veamos algunos ejemplos. Definamos la función `maxino`, que toma dos números y devuelve el más grande de los dos. Una definición para esta función es la siguiente:

$$\begin{array}{lcll} \texttt{maximo}.x.y & :: & & Nat \to Nat \to Nat \\ \texttt{maximo}.x.y & \doteq & (& x \leqslant y \to y \\ & & \square & x > y \to x \\ & &) & \end{array}$$

Para comprender mejor cómo funciona la función la evaluemos en el caso en que x sea 2 e y sea 5. Como para estos valores sucede que $x \leqslant y$ entonces la primera guarda es verdadera. La definición de la función nos dice que, para este caso, `maximo` está definida como y y, por lo tanto, su valor será 5.

Por último, definamos la función `anterior` que toma un número y devuelve 0 si el argumento es 0, y en cualquier otro caso el valor anterior al argumento.

$$\begin{array}{lcll} \texttt{anterior}.n & :: & & Nat \to Nat \\ \texttt{anterior}.n & \doteq & (& n = 0 \to 0 \\ & & \square & n \neq 0 \to n - 1 \\ & &) & \end{array}$$

Actividad

Definí y evaluá en FUN las siguientes funciones:

1. Función `signo` que toma un número y devuelve 1 si el número es mayor que 0 y devuelve 0 si el número es menor o igual que 0
2. Función `valorabsoluto` que toma un número x y devuelve x si $x \geqslant 0$ o $-x$ si $x < 0$
3. Función `espar` que dado un número devuelve $True$ si el mismo es par o $False$ si el mismo es impar.

2.6. Pattern matching

Existe otra manera de definir funciones cuando las mismas deben tomar diferentes valores según determinados casos. Esta manera alternativa se denomina **pattern matching**[1] y utiliza fuertemente patrones que permiten describir y distinguir un conjunto de números.

Para comprender las ideas detrás de este concepto veamos un ejemplo. Retomemos la función `anterior` que en la sección precedente hemos definido por casos. Para definirla usando pattern matching necesitamos una *forma* de escribir el argumento en dos casos: cuando el argumento es cero y cuando es mayor que cero. Ahora bien, cuando el argumento es cero tenemos un símbolo apropiado y podemos definir la función directamente como:

$$\texttt{anterior}.0 \doteq 0$$

Para el caso en que el argumento es mayor que cero, el patrón que utilizaremos el siguiente: $n+1$ con $n \in \mathbb{N}$ —notá que $n+1$ es siempre mayor que cero, sin importar cuál sea el valor de n—. La función se definiría para este caso como sigue:

$$\texttt{anterior}.(n+1) \doteq n$$

De esta forma la función se define a través de dos declaraciones que aprovechan el hecho de que un número natural es o bien cero o puede escribirse con el patrón $(n+1)$.

La definición completa de la función será:

$$\begin{array}{rcl} \texttt{anterior} & :: & Num \to Num \\ \texttt{anterior}.0. & \doteq & 0 \\ \texttt{anterior}.(n+1) & \doteq & n \end{array}$$

Una importante ventaja del pattern matching es que en la misma definición de la función podemos acceder a un valor relacionado con el parámetro. Así, el caso del pattern 0 y $n+1$ nos permite acceder al valor anterior del parámetro (n). Esto hace que la definición de `anterior` quede mucho más compacta sin tener que recurrir a la resta, como lo hacíamos cuando la definíamos por casos.

Otro patrón para los naturales puede ser 0, 1 y $n+2$, el cual está asociado a tres casos: el primero considera el caso en que el argumento sea 0, el segundo considera el caso en que el argumento sea 1 y el último se ocupa del caso en que el argumento sea 2 o más. Una definición que utilice este patrón tendrá la siguiente forma:

$$\begin{array}{rcl} \texttt{f}.0 & \doteq & f_0 \\ \texttt{f}.1 & \doteq & f_1 \\ \texttt{f}.(n+2) & \doteq & f_3 \end{array}$$

Un último patrón para los naturales es el que separa los números en pares e impares, aprovechando que los números pares se pueden escribir de la forma $2 * n$ y los impares como $2 * n + 1$. Una función que utilice este patrón tendría la siguiente forma:

[1] Este término podría traducirse al español como comparación de patrones.

$$\begin{array}{rcl} \texttt{f}.(2*n) & \doteq & f_1 \\ \texttt{f}.(2*n+1) & \doteq & f_2 \end{array}$$

Actividad

Descubrí qué hace la función definida de la siguiente forma. Para ello podés ayudarte evaluándola en distintos casos.

$$\begin{array}{rcl} f.(2*n) & \doteq & n \\ f.(2*n+1) & \doteq & n \end{array}$$

2.6.1. Definiciones locales

Una herramienta muy útil en algunas circunstancias es utilizar **definiciones locales** dentro de la definición de una función. Como su nombre lo indica, estas definiciones sólo tienen sentido dentro de la definición de la función a la que están asociadas.

Como ejemplo tomemos la función `raiz` que dados tres números, a, b y c, calcula una de las raíces de la ecuación $ax^2 * bx * c$. Una posible definición de esta función, utilizando definiciones locales, es la siguiente:

$$\begin{array}{lcl} \texttt{raiz} & :: & Num \to Num \to Num \\ \texttt{raiz}.a.b.c & \doteq & \frac{-b+\sqrt{disc}}{2*a} \end{array}$$

$$\|\texttt{disc} \doteq b^2 - 4 * a * c\|$$

En esta definición se incluye a la función `disc`, la cual está localmente definida es decir, *disc* sólo tiene sentido dentro de la definición de *raiz*. Notá que, en este caso la definición de *disc* hace referencia a nombres (a, b, c) que sólo tienen sentido dentro de la definición de *raiz*.

En caso de haber más de una definición local separaremos las mismas con comas, dentro de los corchetes. Las definiciones locales pueden servir para mejorar la legibilidad de un programa pero también para mejorar la eficiencia del mismo, como veremos más adelante.

Actividad

Se quiere construir una función que calcule el área de un prisma rectangular. El prisma tiene una altura h, un ancho a y una profundidad p. Completá la siguiente definición utilizando definiciones locales:

$$\begin{array}{lcl} \texttt{area} & :: & Num \to Num \to Num \to Num \\ \texttt{area}.h.a.p & \doteq & 2 * \texttt{frente} + 2 * \texttt{lado} + 2 * \texttt{tapa} \\ & & \|\texttt{frente} \doteq \ldots, \\ & & \ \ \texttt{lado} \doteq \ldots, \\ & & \ \ \texttt{tapa} \doteq \ldots \| \end{array}$$

2.7. Algunas funciones más complejas

Actividad

Definir la función `bisiesto` que toma un *Nun* y devuelve un *Bool*. Esta función determina si un año es bisiesto. Recordá que los años bisiestos son aquellos que son divisibles por 4 pero no por 100 a menos que también lo sean por 400. Por ejemplo, 1900 no es bisiesto pero 2000 sí lo es.

Actividad

Suponé que querés jugar al siguiente juego: m jugadores en ronda comienzan a decir los números naturales consecutivamente. Cuando toca un múltiplo de 7 o un número con algún dígito igual a 7, el jugador debe decir "pip" en lugar del número. Definí la función `pip` que toma un Nat y devuelve un $Bool$ que sea $true$ cuando el jugador debe decir "pip" y $false$ en caso contrario. Resolvé este problema para $0 \leqslant n < 10000$.

- Primero te recomendamos definir una función `unidad` que devuelva la cifra de las unidades de un número.
- Hacé lo mismo para las funciones `decenas`, `centenas`, `unidadesdemil`.
- Podés usar las funciones `div` y `mod`.

Sintetizando...

Una función se puede, entonces, definir de alguna de estas formas:

1. Una definición que establece el nombre de la función, el nombre para sus parámetros seguida del símbolo $\doteq$, seguida de una expresión.
2. Una definición que establece el nombre de la función, el nombre para sus parámetros seguida del símbolo $\doteq$, seguida de una expresión por casos.
3. Una función definida localmente utilizando cualquiera de las dos formas anteriores.
4. Una o más definiciones que establecen el nombre de la función, patrones para los parámetros, seguida del símbolo $\doteq$ seguida de sendas expresiones.

2.8. Recursión

Actividad

¿Qué tienen en común las figuras 2.1, 2.2 y 2.3 y el siguiente texto?

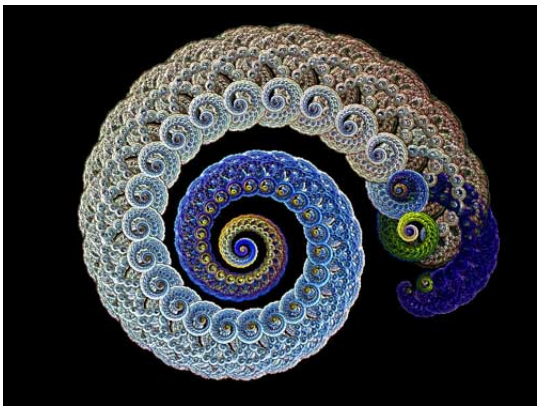


Figura 2.1:

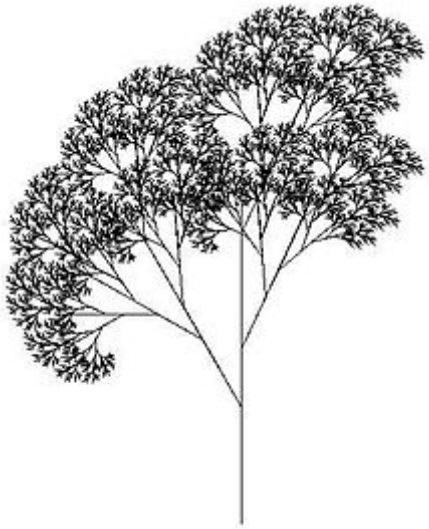


Figura 2.2:

> Un ejecutivo cuenta con un aparato telefónico muy versátil, por el que recibe muchas llamadas. Está hablando con A, y es llamado por B; dice entonces a A: "¿tendría inconveniente en esperar un momento?". Por supuesto que en realidad no le preocupa si A tiene inconveniente o no; aprieta el botón, y se pone en comunicación con B. Ahora es C quien llama, y la conversación con B queda en suspenso. Esto podría extenderse indefinidamente, pero no nos vamos a dejar llevar por el entusiasmo. Digamos entonces que la comunicación con C finaliza; nuestro ejecutivo, pues, se "saca" de vuelta para arriba hasta B, y continúa con su conversación con él. En tanto, A sigue sentado al otro extremo de la línea, haciendo tamborilear sus uñas sobre la superficie de algún escritorio, y probablemente escuchando espantosa música funcional que la línea telefónica le transmite para aplacarlo ... La alternativa más sencilla sería que la comunicación con B termine y el ejecutivo reanude finalmente su conversación con A. Pero *podría* suceder que, después de continuar la comunicación con B, haya una nueva llamada, ahora de D. En tal caso, B sería otra vez puesto en la pila de los que esperan y sería atendido D.
>
> Extraído de *Göedel, Escher, Bach* de D. H. Hofstadter (1979).

La recursión, además de estar presente en nuestras vidas cotidianas es también una herramienta sumamente importante en ciencias de la computación. Usualmente la idea de recursión se utiliza para definir **funciones recursivas**.

2.8.1. Funciones recursivas

Definir una función recursivamente implica construir una definición de forma tal que la función se contiene a sí misma en su propia definición. Un ejemplo bastante conocido de función recursiva es la función `factorial` que, dado un natural devuelve el factorial del mismo. Así:

`factorial`.$0 = 1$

`factorial`.$3 = 3 * 2 * 1$

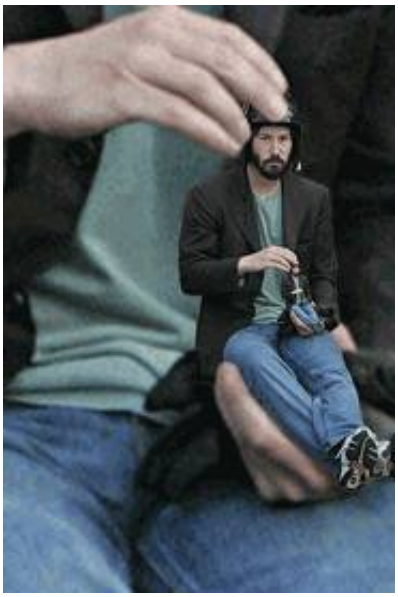


Figura 2.3:

Esta función se puede definir de manera recursiva de la siguiente forma:

$$\texttt{factorial}.0 = 1$$

$$\texttt{factorial}.(n+1) = (n+1) * \texttt{factorial}.n$$

Notá que esta definición utiliza pattern matching. Visto que esta forma de definir funciones nos permite acceder rápidamente al valor anterior del parámetro suele ser una manera muy apropiada para hablar de funciones recursivas.

En el segundo renglón de la definición la función se llama a sí misma. Este es el caso que denominaremos **caso recursivo**. El otro caso se llama **caso base**.

Las definiciones recursivas pueden, en una primer instancia, dar la impresión de que se está cayendo en un ciclo infinito conduciendo a una regresión que nunca termine. En realidad, una definición recursiva de una función —si está correctamente formulada— jamás conduce a una regresión infinita porque una definición recursiva nunca define una cosa en función de esa misma cosa sino, siempre, en función de las interpretaciones más simples de la misma.

Así, en la definición de la función `factorial` su caso inductivo llama a la función pero no para $n+1$ sino para n. Genéricamente la resolución de la aplicación de esta función podría describirse como sigue:

$$\begin{array}{rcl}
\texttt{factorial}.(n+1) & = & (n+1) * \texttt{factorial}.n \\
& = & (n+1) * n * \texttt{factorial}.(n-1) \\
& = & (n+1) * n * (n-1)\texttt{factorial}.(n-2) \\
& \dots & \\
& = & (n+1) * n * (n-1) * (n-2) * \dots \texttt{factorial}.0 \\
& = & (n+1) * n * (n-1) * (n-2) * \dots 1
\end{array}$$

Al intentar calcular $\texttt{factorial}.n$ la función llamará a $\texttt{factorial}.(n-1)$ lo que requerirá, a su vez, que se calcule $\texttt{factorial}.(n-2)$ y así sucesivamente hasta que se llegue a $\texttt{factorial}.0$. En el penúltimo paso se aplica el caso base, que es el que nos asegura que el cálculo en algún momento acaba. Como para este caso la definición de la función no es recursiva sino que da como resultado 1 estamos seguros de que el cálculo termina.

Actividad

Ahora que conocemos esta manera de definir funciones podemos preguntarnos, entre otras cosas, por qué las funciones recursivas son útiles en la programación. Para responder a esta pregunta te proponemos que leas el siguiente texto y que discutas con tus compañeros las respuestas de la preguntas que te planteamos:

> Usualmente la función `factorial` se define a través de la siguiente notación:
>
> $\texttt{factorial}.n = 1 * 2 * 3 * \ldots * n$
>
> Esta definición es correcta pero la notación de puntos suspensivos confía en que el lector comprenda su significado. Una definición informal como esta no es útil para demostrar teoremas y no es un programa ejecutable en muchos lenguajes de programación.
>
> La definición recursiva que presentamos de la función `factorial` no tiene estos problemas.
>
> Las definiciones recursivas de funciones consisten en una colección de ecuaciones que establecen las propiedades de la función que está siendo definida. Hay un número de propiedades algebraicas de la función `factorial`, y una de ellas es utilizada en el caso recursivo de la función. De hecho, la definición no consiste en un conjunto de comandos a ser obedecidos; consiste de un conjunto de ecuaciones verdaderas que describen las propiedades sobresalientes de la función definida.
>
> Los lenguajes de programación que permiten este estilo de definición de funciones son llamados *lenguajes declarativos*, porque el programa consiste en un conjunto de declaraciones de propiedades. El programador debe encontrar un conjunto de propiedades adecuadas a declarar, usualmente via ecuaciones recursivas, y la implementación del lenguaje de programación encuentra una forma de resolver esas ecuaciones.
>
> Adaptación libre de *Discrete Mathematics using a computer* de O'Donnell, Hall & Page (2006) . Págs: 47-48.

- ¿Qué vínculos hay entre definciones recursivas y la programación?
- ¿Qué es un programa según la perspectiva del autor del texto?

Actividad

Definí y evaluá en FUN las siguientes funciones recursivas:

1. Función `sumatoria` que, dado un número Natural n, devuelve la sumatoria de los primeros naturales hasta n.
2. Función `potencia` que toma dos números naturales, x e y, y devuelve x^y.

Resumiendo

Actividad

De manera de sintetizar las ideas que hemos desarrollado en este capítulo te proponemos que agregues al mapa conceptual que construiste en el capítulo anterior los conceptos más importantes tratados aquí.

Capítulo 3

Listas

En los capítulos 1 y 2 nos hemos dedicado a trabajar, principalmente, en el dominio de la aritmética. Nuestro sistema formal consta hasta ahora de constantes y variables numéricas y de operadores y relaciones entre estas constantes y variables. Además, hemos introducido dos constantes de tipo booleano: True y False. También contamos con un formato de demostración que nos permite saber si una fórmula es válida o no.

En este capítulo extenderemos nuestro sistema formal a un nuevo tipo de datos: las **listas**. Así, además de números y constantes booleanas tendremos a nuestra disposición las listas. Esto nos permitirá ganar poder expresivo, es decir, poder expresar en nuestro lenguaje más cosas que las que podíamos decir hasta el momento.

Agregar las listas a nuestro sistema formal involucrará desarrollar la notación necesaria para definirlas y construirlas, definir funciones que se puedan aplicar a las listas, analizar el tipado de expresiones que contengan números, booleanos y listas y presentar herramientas que nos permitan demostrar propiedades de las funciones definidas sobre listas. De todo esto nos ocuparemos a continuación.

3.1. Definiendo las listas

En este curso, consideraremos que una **lista** es una colección de valores ordenados, los cuales deben ser todos del mismo tipo. Además, sólo vamos a ocuparnos de las listas que contengan un número finito de elementos.

Notación. Denotaremos a las listas entre corchetes, separando cada uno de sus elementos por una coma.

Los siguientes son ejemplos de listas:

$[3, 2, 1, 1]$

$[True, True, False, False, True]$

$[[8, 9], [4], [6, 6]]$

$[3]$

$[\,]$

El penúltimo ejemplo es una lista con un solo elemento y el último es un caso bastante particular: la **lista vacía**.

A diferencia de los conjuntos, las listas pueden contener elementos repetidos. Así, $[8, 8]$ es una lista de dos elementos.

Diremos que dos listas son iguales si y sólo si tienen los mismos elementos en el mismo orden. Por lo tanto:

$[3, 6, 9] \neq [9, 3, 6]$

$[4, 4, 4,] \neq [4, 4]$

Tipos de las listas. A partir de los ejemplos anteriores podemos deducir que las listas pueden ser de distintos tipos, así tenemos listas de números, listas de booleanos, listas de listas de números, etc. La restricción es que todos los elementos de una lista dada sean todos del mismo tipo.

Siguiendo estas ideas, notamos que necesitamos dos cosas para determinar el tipo de una lista: debemos ser capaces de expresar el tipo de los elementos que la componen y tenemos que establecer que estamos hablando de una lista. Así, diremos que el tipo de una lista es $[A]$:

- Los corchetes nos indican que se trata de una lista
- A es el tipo de los elementos de la lista

Veamos los tipos de las listas del ejemplo anterior:

- $[3, 2, 1, 1]$ es de tipo $[Num]$
- $[True, True, False, False, True]$ es de tipo $[Bool]$
- $[[8, 9], [4], [6, 6]]$ es de tipo $[[Num]]$
- $[3]$ es de tipo $[Num]$

La lista vacía puede ser de diversos tipos, por eso le asignaremos, en general, el tipo $[A]$ donde A puede ser cualquiera de los tipos que ya conocemos. Será por el contexto el que determina cuál es el tipo de la lista vacía. En el caso $[[3, 2], [\,], [6]]$ la lista vacía es de tipo $[Num]$ porque el tipo de la lista completa es $[[Num]]$.

Variables en listas. Supongamos ahora que queremos hablar sobre listas de manera general, sin necesidad de referirnos a una lista específica. Poder llevar a cabo esta tarea implicará utilizar una variable que represente a una lista, cosa que es perfectamente válida. Convencionalmente:

- Utilizaremos los símbolos xs, ys, zs, etc. para representar con variables a una lista;
- Usaremos xss, yss, etc, para representar con variables a una lista de listas.
- Por último, si queremos representar a un elemento de una lista, lo que será de muchísima utilidad, utilizaremos las variables x, y, etc.

Es importante recordar que esto es una convención; es la manera en la que generalmente se escriben las cosas en la mayoría de los libros, no es algo que le otorgue un sentido particular a las variables sobre listas. Por lo tanto, podrían ser presentadas con otras letras o de otra manera. Nosotros seguiremos la convención.

3.1.1. ¿Cómo construir una lista?

Hasta aquí hemos introducido la notación y el tipo de nuestro nuevo tipo de dato. Así, ya contamos con las herramientas para escribir una lista particular, que nos venga dada. Una pregunta que surge a continuación es aquella que da nombre a este apartado: ¿cómo podemos construir una lista?

Conceptos teóricos

Los **constructores de listas** serán dos:

- La lista vacía: $[\,]$
- El constructor $\triangleright$: este constructor agrega un elemento a una lista por la izquierda.

Analicemos como funciona este último constructor: $\triangleright$ tomará un elemento de tipo A y una lista de tipo $[A]$. Al aplicar este constructor obtendremos una nueva lista cuyo primer elemento es el que

acabamos de agregar y cuyos elementos restantes son los que ya estaban en la lista. Por ejemplo, tomemos el elemento 3 y la lista $[5, 1, 2]$. Aplicar el constructor nos dará como resultado:

$$3 \triangleright [5, 1, 2] = [3, 5, 1, 2]$$

La potencia de estos dos elementos —la lista vacía y $\triangleright$ — radica en que podemos construir todas las listas posibles valiéndonos sólo de ellos dos. ¿Cómo es este proceso? Veamos un ejemplo: Supongamos que queremos construir la lista $[False, False, True]$. En primer lugar, tomaremos la lista vacía y le agregaremos por la izquierda el elemento $True$ utilizando nuestro constructor $\triangleright$. A ese resultado le agregaremos por izquierda el elemento $False$ y, finalmente, al resultado le volveremos a agregar el elemento $False$. En fórmulas este proceso se escribe de la siguiente manera:

$$[False, False, True] = False \triangleright (False \triangleright (True \triangleright [\,]))$$

De manera más general, si x es de tipo A y xs es de tipo $[A]$, entonces $x \triangleright xs$ es una lista cuyo primer elemento es x y el resto es xs.

Al contar con estos dos constructores, debe ser claro que la notación introducida en los párrafos anteriores es sólo eso, una notación. Por lo tanto, cada vez que la utilicemos, por ejemplo para escribir la lista $[7, 4, 1]$, en realidad estamos haciendo uso de una manera más corta y cómoda de describir esta lista en lugar de escribir $7 \triangleright (4(\triangleright(1 \triangleright [\,])))$.

Actividad

Utilizando los dos constructores de listas, indicá cómo se construyen las siguientes listas:

1. $[7, 6, 5, 4]$
2. $[[3], [9, 2], [3, 8]]$
3. $[[[5]]]$

3.2. Funciones sobre listas

Ahora que tenemos un nuevo tipo de datos nos ocuparemos de construir funciones —es decir, programas— que utilicen o manipulen listas. Así, nos interesa definir funciones que calculen el largo de una lista, que «peguen» dos listas, que devuelvan su primer elemento, que devuelvan todos los elementos menos el primero, que multipliquen por dos todos sus elementos, etc., etc. Muchas de ellas serán recursivas, otras no.

3.2.1. Función cardinal

En primer lugar, elaboremos la definición de la función `cardinal`, habitualmente denotada por el símbolo $\#$, que, dada una lista devuelve la cantidad de elementos que ésta lista contiene.

Actividad

A partir de la descripción anterior:

1. Calculá vos mismo los cardinales de las siguientes listas:
   a) $\#[3, 4, 7, 1] = \ldots$
   b) $\#[2, 9] = \ldots$
   c) $\#[2, 2, 2, 2, 2, 2] = \ldots$
   d) $\#[\,] = \ldots$
2. Indicá cuál es el tipo de la función $\#$.

Actividad

En el capítulo anterior vimos que un buen pattern matching para los números naturales surgía a partir de pensar que un número cualquiera es o bien cero o es distinto de cero. Esto permitía escribir a todos los números como: 0 o $n + 1$ con $n \in Nat$. Si queremos definir una función sobre todos los naturales, entonces podemos hacerlo definiendo la función para el caso 0 y para el caso $n + 1$.
Pensando ahora en las listas y en sus constructores:

1. ¿De qué maneras podemos representar a todas las listas? ¿Cómo es posible cubrir todos los casos posibles de listas?

2. A partir de tus respuestas a las preguntas anteriores determiná qué patrón puede ser útil para definir funciones sobre listas.

Notemos ahora que los constructores de listas que aparecen en el patrón que utilizaremos nos permiten armar listas más largas a partir de listas más pequeñas —agregando elementos a través de $\triangleright$. Es por esto que parece razonable intentar una *definición recursiva* de la función $\#$.

Recordá que las funciones recursivas eran aquellas que en su definición se llaman a sí mismas pero siempre para valores «más chicos» del dominio en cuestión. Las definiciones de estas funciones suelen tener uno o más casos bases y un caso recursivo en donde la función se llama a sí misma.

Para la función $\#$, el **caso base** trabaja con la lista vacía. Como esta lista no contiene ningún elemento entonces el cardinal debe valer 0.

El **caso inductivo** trabajará con una lista que tenga al menos un elemento, denotándola $x \triangleright xs$. Esta lista tiene exactamente un elemento más que la lista xs. Entonces, si aplicamos la llamada recursiva en xs, el cardinal de la lista $x \triangleright xs$ se puede calcular sumando 1 al cardinal de la lista xs.

Traduciendo todas estas ideas a fórmulas, la definición completa de la función quedará de la siguiente forma:

$$\begin{array}{rcl} \# & :: & [A] \to Nat \\ \#[\,] & \doteq & 0 \\ \#(x \triangleright xs) & \doteq & 1 + \#xs \end{array}$$

Actividad

Evaluá paso a paso en FUN la función `cardinal` para distintas listas particulares, de forma que puedas ganar intuición sobre cómo funciona.

3.2.2. Función concatenar

Esta función, denotada por el símbolo $+\!\!+$, captura la idea de «pegar» dos listas dadas. Así, toma dos listas del mismo tipo y devuelve otra lista del mismo tipo, que consiste en las dos anteriores, puestas una inmediatamente después de la otra. Por ejemplo:

$$[3, 2] +\!\!+ [6] = [3, 2, 6]$$

$$[True, True] +\!\!+ [False] = [True, True, False]$$

$$[[7, 3], [8, 0]] +\!\!+ [[1, 1], [0]] = [[7, 3], [8, 0], [1, 1], [0]]$$

Si bien esta función toda dos listas, que pueden denotarse como xs y ys, para definirla es suficiente con hacer recursión en una de ellas, por ejemplo en xs.

Al igual que en la definición de la función $\#$, el caso base considerará que xs sea la lista vacía y para el caso inductivo pediremos que esa misma lista tenga al menos un elemento denotándola como $x \triangleright xs$. Como no vamos a hacer recursión sobre la otra lista, en ambos casos la denotaremos como ys. Esta notación permite que esta lista sea vacía o no.

Actividad

Las siguientes son anotaciones que un estudiante elaboró para darse una idea de cómo definir a la función. Leelas con atención:

> **Caso base:** La primer lista es vacía. Entonces el resultado de «pegar» esta lista con otra cualquiera debe ser la segunda lista completa.
> **Caso inductivo:** Aquí debo trabajar con una expresión del tipo $(x \triangleright xs) +\!\!+ ys$. El primer elemento del resultado de aplicar la concatenación debe ser x. Luego deben venir todos los elementos xs seguidos de todos los elementos de ys. Esto último se puede escribir, usando la llamada recursiva, como $xs +\!\!+ ys$.

1. Utilizando estas intuiciones, completá la definición de la función $+\!\!+$:

$$\begin{array}{rcl} +\!\!+ & :: & \ldots \\ [\,] +\!\!+ ys & \doteq & \ldots \\ (x \triangleright xs) +\!\!+ ys & \doteq & \ldots \end{array}$$

2. Evaluá la función en distintas listas, para ganar confianza en la corrección de tu definición.

3.2.3. Función agregar por derecha

Supongamos que tenemos una lista dada, por ejemplo $[3, 2, 7]$ y queremos agregarle al final de la lista el elemento 8 para obtener la lista $[3, 2, 7, 8]$. De esta función, que se denota con el símbolo $\triangleleft$, nos ocuparemos en esta sección.

Actividad

1. ¿Cuál debería ser el resultado de aplicar la función a los siguientes casos?
   a) $[9] \triangleleft 0$
   b) $[[1, 1]] \triangleleft [1]$
   c) $[] \triangleleft 4$
   d) $[] \triangleleft True$
2. Definí cuál es el tipo de la función $\triangleleft$
3. ¿Cuál será el caso base y el caso inductivo?
4. Escribí en una nota de FUN cuál debería ser el resultado de aplicar la función a la lista vacía y a la lista con al menos un elemento. Considerá que si el caso inductivo trabaja con $(x \triangleright xs) \triangleleft y$, la llamada recursiva se expresa para esta función como $xs \triangleleft y$.
5. Usando tus ideas anteriores da una definición de la función.

3.2.4. Funciones `cabeza` y `cola`

Vamos a ocuparnos en esta sección de dos funciones: en primer lugar, aquella que devuelve el primer elemento de una lista dada. Esta función, que llamaremos `cabeza`, toma una lista de tipo $[A]$ y devuelve un elemento de tipo A. En segundo lugar, definiremos una función que devuelve todos los elementos de la lista menos el primer elemento. Esta función, que llamaremos `cola`, toma una lista de tipo $[A]$ y devuelve una lista de tipo $[A]$

Actividad

1. ¿Cuáles serán las condiciones que deberá cumplir una lista para que las funciones `cabeza` y `cola` puedan ser definidas? Para dar tu respuesta te puede ser útil pensar cuál debería ser el resultado de aplicar la función a distintas listas.
2. ¿Es necesario que las funciones sean recursivas? ¿por qué?
3. A partir de tus respuestas a las preguntas anteriores construí una definición para ambas funciones.

3.3. Expresiones que contienen listas, números y booleanos

En las secciones anteriores hemos definido distintas funciones sobre listas. Para evaluar expresiones más complejas que combinen estas funciones con los otros operadores que ya conocemos sobre los números necesitamos determinar la **precedencia** de estas nuevas funciones. Como ya vimos, la precedencia nos provee de reglas para saber qué operación deben resolverse en primer lugar, en segundo lugar, etc. y también nos permite eliminar paréntesis.

A la lista de precedencia que ya teníamos le agregamos, entonces, los operadores sobre listas:

Conceptos teóricos

Reglas de precedencia

+ precedencia

	$\sqrt{}, (\cdot)^2$	raíces y potencias
	$*, /$	producto y división
	$+, -$	suma y resta
$\downarrow$	$=, \leqslant, \geqslant$	conectivos aritméticos
	., #, **`cabeza` y `cola`**	aplicación de función, cardinal, `cabeza` y `cola`
	$\triangleright, \triangleleft$	pegar a derecha y pegar a izquierda
	$+\!\!+$	concatenar dos listas

- precedencia

Como siempre, los operadores que están más arriba tienen mayor precedencia. Cuando hay más de un operador en un nivel de precedencia, es necesario poner paréntesis para evitar la ambigüedad. Por ejemplo, $x \triangleright xs +\!\!+ ys$ se interpreta como $(x \triangleright xs) +\!\!+ ys$, pero la expresión $x \triangleright xs \triangleleft y$ no tiene sentido si no se ponen paréntesis: o bien es $(x \triangleright xs) \triangleleft y$ o bien $x \triangleright (xs \triangleleft y)$.

El objetivo de las siguientes actividades es que te familiarizases con expresiones que involucren todas estas operaciones de manera que podamos extender el método que teníamos para justificar el tipado de expresiones, considerando expresiones más complejas que las que veníamos trabajando.

Actividad

Evaluá en FUN las siguientes expresiones. Interpretá los resultados a partir de tu conocimiento de las funciones:

1. $[23, 45, 6] \triangleleft (\texttt{cabeza}.[1, 2, 3, 10, 34])$
2. $[0, 11, 2, 5] \triangleright [\,]$
3. $\texttt{cabeza}.(0 \triangleright [1, 2, 3])$
4. $([3, 5] \triangleright [[14, 16], [1, 3]]) \triangleleft [4 + 8, 3^3]$
5. $([1, 2] +\!\!+ [3, 4]) \triangleleft (2 + 3)$
6. $(([[True]] +\!\!+ [[True]]) \triangleleft [False])$
7. $(([[\,]] +\!\!+ [[\,]]) \triangleleft ([\,] +\!\!+ [\,]))$
8. $[\#[False, True, True]] \triangleright [[3, 6] +\!\!+ [4, 9]]$

Actividad

Cuando nos ocupamos de tipar expresiones que involucraban operadores sobre números elaboramos esquemas de tipado para cada uno de ellos. Por ejemplo, para la multiplicación la regla de tipado era:

$$\frac{Nat \quad * \quad Nat}{Nat}$$

1. Elaborá esquemas de tipado similares para las funciones que hemos definido sobre listas: $\#$, $\triangleright$, $\triangleleft$, $+\!\!+$, `cabeza` y `cola`.
2. Usando EQU construí el arbol de tipado de todas las expresiones de la actividad anterior
3. ¿Qué ocurre en los siguientes casos?
   a) $[1, 5, False]$
   b) $0 \triangleleft [1, 2, 3]$
   c) $([1] +\!\!+ [2]) \triangleright [3]$
   d) $[[0, 1], [False, False]]$

Actividad

Utilizá EQU para ver si es posible asignarles tipo a las variables para hacer que las expresiones queden bien tipadas:

1. $x \triangleright (y \triangleright z)$
2. $x \triangleright (y \triangleright x)$
3. $(x \triangleright y) \triangleright z$
4. $(x \triangleright y) \triangleright y$
5. $x \triangleright (y \triangleleft z)$
6. $(x +\!\!+ y) \triangleright (z +\!\!+ w)$
7. $x \triangleright [x]$
8. $x \triangleright [[x]]$
9. $x \triangleright ([[True]] \triangleright y)$

3.4. Más funciones sobre listas

Al contar con cada vez más tipos de datos el sistema formal que vamos construyendo se va volviendo más y más rico, ampliando el rango de funciones —es decir, de programas— que se pueden definir. En esta sección nos ocuparemos de definir distintas funciones que tomen o devuelvan listas, números o booleanos.

Actividad

1. Definí la función `duplicar` que toma una lista de números y devuelve otra lista de números. El resultado de aplicar esta función es una lista que contiene duplicados cada uno de los elementos de la lista original. Para ello:

   a) Pensá en el resultado que debería devolver la función en algunos casos particulares. Esto te permitirá ganar intuición sobre cómo definir la función.

   b) Definí cuál será el tipo de la función.

   c) Definí sobre qué variable se realizará la recursión y cuál será el caso base y el caso recursivo.

   d) Definí la función para ambos casos. Recordá que el caso recursivo debe llamar a la función para un valor «más chico» de la variable y que el caso base debe asegurar que el cómputo termine. Podés usar también una nota de FUN para escribir con tus palabras lo que debería hacer la función en cada caso.

   e) Evaluá, usando FUN la función para distintos casos. Si es necesario revisá tu definición.

2. Definí la función `agregar0` que toma una lista de listas de números y devuelve otra lista de listas de números. El resultado de aplicar la función es agregar el elemento 0 al inicio de cada una de las listas que componen la lista de listas original. Para poder hacerlo puede serte útil seguir los pasos mencionados en el ítem anterior.

3. ¿Qué hace la siguiente función?

$$\begin{array}{rcl} \texttt{f} & :: & Num \to [Num] \to [Num] \\ \texttt{f}.n.[\,] & \doteq & [\,] \\ \texttt{f}.n.(x \triangleright xs) & \doteq & n * x \triangleright \texttt{f}.n.xs \end{array}$$

4. ¿Qué hace la siguiente función?

$$\begin{array}{rcl} \texttt{g} & :: & [[Num]] \to [[Num]] \\ \texttt{g}.[\,] & \doteq & [\,] \\ \texttt{g}.(xs \triangleright xss) & \doteq & [\#xs] \triangleright \texttt{g}.xss \end{array}$$

   Si te hace falta podes evaluar "paso a paso" la función en FUN.

5. ¿Podés encontrar similitudes entre estas cuatro funciones? ¿Cuáles? Proponé otra función que comparta estas características.

Las funciones que definimos en la actividad anterior pertenecen a una clase de funciones denominadas comúnmente **mapeos**. Los mapeos son funciones que toman listas de algún tipo, pueden ser $[Num]$, $[Bool]$, $[[Num]]$, etc. y devuelven otra lista del mismo tipo que la original. Ellas le aplican a cada elemento de la lista original una operación o función, como multiplicar por dos o agregar un 0 al comienzo de una lista. Los elementos de la lista que devuelven son el resultado de esta operación.

Actividad

1. Trabajemos ahora con la función `sum` que toma una lista de números y devuelve un número que es el resultado de sumar todos los elementos de la lista.

   a) En primer lugar, pensá cuál debería ser el resultado para algunos casos particulares.

   b) Definí el tipo de la función.

   c) Definí sobre qué variable se realizará la recursión y cuál será el caso base y el caso recursivo.

   d) Las siguientes son las notas que un estudiante construyó para ayudarse a definir la función en los dos casos:

   Caso base: La lista vacía no contiene elementos, por lo tanto no hay nada para sumar. Por eso definiré la función para el caso base como 0. Esto probablemente sea bueno para asegurar que la función termina. Además, como el cero es el elemento neutro de la suma, cuando aplique el caso recursivo una cantidad de veces y llegue a la lista vacía, que la función este definida en este caso como cero me asegurará que el valor de `sum` que ya vengo calculando no se altere.

   Caso inductivo: Para este caso lo que `sum` tiene que hacer es sumar la cabeza de la lista al resultado de aplicar la función a la lista con un elemento menos.

   e) Ayudándote de estas notas definí la función para ambos casos

   f) Evaluá tu definición en distintos casos. Si es necesario revisá tu definición.

2. Definí la función `concat` que toma una lista de listas y devuelve una lista. El resultado de aplicar esta función es una lista que consiste en todos los elementos de la lista de listas original concatenados. Para ello:

   - Tomá en cuenta que `concat`.$[[1],[2,3],[4,5,6]]$ debe dar como resultado la lista $[1,2,3,4,5,6]$. Pensá en otros resultados para casos particulares.
   - Para definir a la función podés utilizar la función $+\!\!+$ que ya definimos.

3. ¿Qué hace la siguiente función?

$$\begin{array}{rcl} \texttt{h} & :: & [Num] \to Num \\ \texttt{h}.[\,] & \doteq & 1 \\ \texttt{h}.(x \triangleright xs) & \doteq & x * \texttt{h}.xs \end{array}$$

4. La función `g` definida a continuación utiliza la función `maximo` que desarrollamos en la sección 2.5.1, en la página 22. Revisá la definición de `maximo` y luego evaluá y analizá la definición de `f` para averiguar qué hace:

$$\begin{array}{rcl} \texttt{g} & :: & [Num] \to Num \\ \texttt{g}.[\,] & \doteq & 0 \\ \texttt{g}.(x \triangleright xs) & \doteq & \texttt{maximo}.x.(\texttt{g}.xs) \end{array}$$

5. ¿Qué semejanzas hay entre estas cuatro funciones? Imaginá otra función que también posea estas características.

Las funciones de la actividad anterior pertenecen a una clase de funciones llamada **acumuladores**. Todas las funciones de esta clase operan sobre todos los elementos de una lista, acumulando este resultado a medida que se realiza el cómputo. Por este motivo el caso base siempre debe ser el neutro de la operación que se está realizando.

Actividad

1. Trabajemos ahora con la función `quita0` que toma una lista de números y devuelve otra lista de números. Esta función remueve todos los ceros presentes en una lista, devolviendo sólo los restantes.

   a) Como siempre, pensá en el resultado que debería dar la función en algunos casos particulares. Luego, definí cuál será el tipo de la función.

   b) La siguiente es una definición que dió un estudiante de la función:

$$\begin{array}{rcl} \texttt{quita0} & :: & [Num] \to Num \\ \texttt{quita0}.[\,] & \doteq & [\,] \\ \texttt{quita0}.(x \triangleright xs) & \doteq & (\quad x = 0 \to 0 \triangleright \texttt{quita0}.xs \\ & & \square \quad x \neq 0 \to \texttt{quita0}.xs \\ & &) \end{array}$$

   c) ¿Por qué te parece que el estudiante decidió utilizar una definición por casos en el caso recursivo?

   d) Evaluá en FUN esta definición para ver si está bien hecha. De ser necesario modificala.

2. Definí la función `solopares` que toma una lista y devuelve otra lista donde se encuentran sólo los elementos de la lista original que son pares. Para hacerlo podés seguir la estrategia que venimos desarrollando y tener en cuenta la forma de la definición de la función anterior.

3. ¿Qué hace la siguiente función? ¿Sobre qué variable se está haciendo recursión? Si es útil, podés evaluarla en FUN para descubrir qué hace.

$$\begin{array}{rcl} \mathsf{g} & :: & [[Bool]] \to [[Bool]] \\ \mathsf{g}.[\,] & \doteq & [\,] \\ \mathsf{g}.(xs \triangleright xss) & \doteq & (\quad xs = [\,] \to \mathsf{g}.xss \\ & & \square \quad xs \neq [\,] \to xs \triangleright \mathsf{g}.xss \\ & &) \end{array}$$

4. ¿Qué semejanzas hay entre estas tres funciones? Imaginá otra función que también posea estas características.

Las funciones de la actividad anterior pertenecen a una clase de funciones llamada **filtros**. Como su nombre lo indica, todas ellas eliminan o filtran algunos de los elementos de la lista que toma la función de acuerdo a una condición —por ejemplo, que elemento sea 0 o que sea impar—, devolviendo una lista que contiene sólo los elementos que no satisfacen dicha condición.

Actividad

1. Definamos ahora una función que comparte algunas características con los filtros pero no es exactamente uno. Se trata de la función `esta` que, dado un número y una lista de números, determina si el número está o no en la lista.

   a) ¿De qué tipo será el resultado de aplicar la función? ¿Cuál será el tipado completo de la función? Para ello puede serte útil pensar qué resultado debería devolver la función en algunos casos particulares.

   b) Esta función toma dos parámetros: un número, llamémosle n y una lista, xs. ¿Sobre cuál de estos parámetros se realizará la recursión? ¿Por qué?

   c) Lee la siguiente nota de FUN que escribió un estudiante sobre el caso inductivo de la función:

   Caso inductivo: Si el primer elemento de la lista es igual a n entonces la función debe terminar, devolviendo el valor booleano que indique que el elemento está en la lista. Si el primer elemento no es igual a n entonces la función debe continuar analizando si el elemento está o no en la lista xs.

   d) Escribí una nota para el caso base.

   e) Formalizá estas ideas dando la definición de la función.

2. Trabajemos ahora con la función `numerodeveces` que toma un número y una lista de números y devuelve el número de veces que dicho número aparece en la lista.

   a) ¿Cuál debería ser el resultado de aplicar la función a los siguientes parámetros? 0 y $[0, 3, 4, 0, 7]$; 5 y $[\,]$; 9 y $[1]$; 4 y$[2, 2, 2]$

   b) La siguiente es una definición que dio un estudiante de la función. Analizala para decidir si es correcta o no. En caso de no serlo, corregila para que quede bien definida:

$$
\begin{array}{rcl}
\texttt{numerodeveces} & :: & Num \to [Num] \to Num \\
\texttt{numerodeveces}.n.[\,] & \doteq & 0 \\
\texttt{numerodeveces}.n.(x \triangleright xs) & \doteq & (\quad x = n \to 1 + \texttt{numerodeveces}.n.xs \\
 & & \square \quad x \neq n \to \texttt{numerodeveces}.n.(x \triangleright xs) \\
 & &)
\end{array}
$$

Actividad

Existen funciones en las que el patrón que venimos utilizando —lista vacía y lista con al menos un elemento— no es útil y es preciso utilizar uno distinto. Por ejemplo, la siguiente función utiliza el patrón que distingue la lista vacía, la lista con un elemento y la lista con al menos dos elementos.

$$\begin{array}{rcl} \mathtt{g} & :: & [Num] \to Bool \\ \mathtt{g}.[\,] & \doteq & True \\ \mathtt{g}.[x] & \doteq & True \\ \mathtt{g}.(x \triangleright y \triangleright xs) & \doteq & (\quad x \leqslant y \to \mathtt{g}.xs \\ & & \square \quad x > y \to False \\ & &) \end{array}$$

1. Descubrí qué hace esta importante función y explicá por qué es necesario utilizar este patrón.
2. Definí la función `removeralternada` que, dada una lista de números remueve algunos de sus elementos de forma alternada, comenzando con el primero. Por ejemplo, `removeralternada`.$[1, 2, 3, 4, 5, 6, 7]$ da como resultado la lista $[2, 4, 6]$. Para ello:
   a) Definí el tipo de la función
   b) Decidí qué patrón utilizarás para definirla y justificá tu decisión.
   c) Escribí en una nota de FUN con tus palabras qué debería hacer la función en cada caso.
   d) Formalizá tus ideas dando una definición de la función. Probala para distintos casos y si es necesario revisá la definición.

Una de las prácticas más usuales y útiles en programación es el reutilizar código, es decir, hacer uso de funciones que ya han sido definidas para construir nuevas funciones. Así, cada vez que se necesita dar una definición de una función compleja esta puede componerse a través de la combinación de funciones ya definidas previamente. De estos temas nos ocuparemos en la siguiente actividad:

Actividad

1. Trabajemos con la función `promedio` que dada una lista de números devuelve el valor promedio de los números que componen la lista.
   a) Usá tus palabras para describir qué tendría que hacer la función.
   b) ¿Qué funciones que ya definimos pueden utilizarse para definirla?
   c) Usando estas ideas definí, en una sola línea, la función `promedio`.
2. ¿Qué hace la siguiente función definida en términos de la función `concat` y de #?

   $$\mathtt{f}.xss = \#(\mathtt{concat}.xss)$$

3. Definí usando funciones que ya construimos la función `invertir` que, dada una lista devuelve la lista invertida, es decir: `invertir`.$[1, 6, 4, 2] = [2, 4, 6, 1]$

3.5. Principio de inducción: demostrando propiedades de funciones sobre listas

Ocupémonos ahora del problema de demostrar propiedades de funciones definidas sobre listas. Esta es una cuestión bastante importante y común en ciencias de la computación, porque las propiedades que satisfaga una función serán propiedades que satisfaga un programa.

Por ejemplo, podría ser útil demostrar que la función cardinal es siempre mayor que cero, o que la lista vacía es el elemento neutro de la concatenación. Es importante resaltar que buscamos

una manera de demostrar que estas propiedades son válidas para todas las listas, es decir debemos probar que:

$\#xs \geqslant 0$ *para toda lista* xs

$xs +\!\!+ [\,] = xs$ *para toda lista* xs

Así, buscamos demostrar que una propiedad es satisfecha por todos los elementos del conjunto de las listas.

Una posibilidad que tenemos es construir una demostración para cada caso particular. Así, podríamos ir probando que la primer propiedad vale para la lista vacía, para la lista [1], para la lista $[1, 2]$, para la lista $[1, 2, 3]$, etc., etc. Pero esta estrategia es inviable porque las listas son un conjunto infinito y nunca terminaríamos de probar con todos los casos posibles.

Una segunda estrategia que podemos aplicar es demostrar que la propiedad vale para todas las listas de un largo dado. Así, podríamos construir una demostración para la lista vacía, otra para todas las listas que poseen un elemento, otra para todas las listas que poseen dos elementos y así sucesivamente. Si llamamos a P a la propiedad a demostrar, entonces esta estrategia requeriría elaborar una demostración para $P([\,])$, otra para $P([x])$, otra para $P([x, y])$, etc. Esta estrategia tiene el mismo problema que la estrategia anterior, pero trae a primer plano una intuición importante.

Tal como definimos a las listas, sabemos que con los dos constructores podemos armar todas las listas posibles. Dichos constructores permiten vincular muy fácilmente a una lista con la lista que posee un elemento más a través de la expresión $x \triangleright xs$.

Ahora bien, supongamos que podemos demostrar que la propiedad vale para la lista vacía, es decir, construimos una prueba que demuestra que $P([\,])$ vale. Si además logramos construir una demostración en la que, suponiendo que la propiedad vale para una lista de un largo cualquiera, probamos que la propiedad vale para la lista que tiene un elemento más, entonces habremos cubierto todos los casos posibles. Esta última demostración puede formalizarse de la siguiente manera:

Si $P(xs)$ *vale, entonces* $P(x \triangleright xs)$ *vale*

Esta es la esencia del **principio de inducción sobre listas**.

Veamos cómo funciona. En primer lugar demostramos que $P([\,])$ es válida. Además, hemos demostrado que si $P(xs)$ vale, entonces $P(x \triangleright xs)$ vale para una lista xs cualquiera. En particular, xs puede ser la lista vacía, lo que nos permite afirmar que si $P([\,])$ vale, entonces $P(x \triangleright [\,])$ también vale. Como la primer demostración nos asegura que $P([\,])$ es válida, entonces ya estamos en condiciones a afirmar que propiedad vale para la lista con un elemento. Haciendo un razonamiento similar se puede concluir que la propiedad vale para una lista con dos elementos, luego para la lista con tres elementos y así sucesivamente. De manera esquemática:

Vale $P([\,])$				
Vale $P([\,])$	y	si vale $P([\,])$ entonces vale $P([x])$	luego	$P([x])$ es válida
Vale $P([x])$	y	si vale $P([x])$ entonces vale $P([x, y])$	luego	$P([x, y])$ es válida
Vale $P([x, y])$	y	si vale $P([x, y])$ entonces vale $P([x, y, z])$	luego	$P([x, y, z])$ es válida
...				

El principio de inducción sobre listas se enuncia formalmente de la siguiente manera:

Conceptos teóricos

Principio de inducción sobre listas: Sea $P(xs)$ una propiedad de una lista xs. Si

- $P([\,])$ es válida, y
- **si** $P(xs)$ vale para una lista arbitraria xs **entonces** $P(x \triangleright xs)$ vale,

  entonces $P(xs)$ vale para cualquier lista xs.

La demostración del primer ítem de este principio suele llamarse **caso base**. La demostración del segundo ítem se llama **caso inductivo**. Para este último siempre es importante no perder de vista que lo que hay que demostrar es que **si** una propiedad vale **entonces** otra también lo hace;

no hay que demostrar que vale $P(xs)$, ni que vale $P(x \triangleright xs)$ sino que **si** $P(xs)$ vale, **entonces** $P(x \triangleright xs)$ vale.

Para llevar a cabo una demostración de este tipo lo que se hace generalmente es usar la propiedad $P(xs)$ como **hipótesis** en la demostración de que $P(x \triangleright xs)$ es equivalente a $True$. A esta hipótesis se la llama **hipótesis inductiva**.

Pongamos todas estas ideas en práctica para demostrar que la lista vacía es el neutro a derecha de la concatenación. Para construir esta prueba haremos uso de la definición de la función $+\!\!+$. La misma era:

$$\begin{array}{rcl} +\!\!+ & :: & [A] \to [A] \to [A] \\ [\,] +\!\!+ ys & \doteq & ys \\ (x \triangleright xs) +\!\!+ ys & \doteq & x \triangleright (xs +\!\!+ ys) \end{array}$$

En primer lugar, debemos enunciar formalmente la propiedad que vamos a demostrar. En este caso es:

$$P(xs) : xs +\!\!+ [\,] = xs$$

Realizaremos una demostración por inducción en la variable xs. Construyamos la demostración para el **caso base**. Debemos probar que $P(xs)$ vale para la lista vacía, es decir que:

$$P([\,]) : [\,] +\!\!+ [\,] = [\,]$$

es válida. Hagamos uso de la definición de la función para construir esta demostración:

$\underline{[\,] +\!\!+ [\,]} = [\,]$
$\equiv$ { Definición de $+\!\!+$ caso base }
$[\,] = [\,]$
$\equiv$ { Reflexividad }
$True$

Demostremos ahora el **caso inductivo**. Para ello vamos a suponer que

$$P(xs) : xs +\!\!+ [\,] = xs$$

es válida y deberemos demostrar que la propiedad vale para la lista que posee un elemento más, es decir, probar que:

$$P(x \triangleright xs) : (x \triangleright xs) +\!\!+ [\,] = (x \triangleright xs)$$

Nuevamente, utilicemos la definición de $+\!\!+$ para elaborar la prueba:

$\underline{(x \triangleright xs) +\!\!+ [\,]} = (x \triangleright xs)$
$\equiv$ { Definición de $+\!\!+$ }
$x \triangleright \underline{(xs +\!\!+ [\,])} = (x \triangleright xs)$
$\equiv$ { Hipótesis inductiva: $xs +\!\!+ [\,] = xs$ }
$x \triangleright xs = (x \triangleright xs)$
$\equiv$ { Reflexividad }
$True$

Como hemos demostrado que la propiedad vale para estos dos casos, entonces el principio de inducción nos asegura que la propiedad vale para todas las listas.

Analizando la demostración que acabamos de realizar, determinemos el formato que utilizaremos en general para construir demostraciones por inducción de propiedades sobre listas:

1. **Establecer que la demostración utiliza el principio de inducción**. Esto inmediatamente determina la estructura global de la prueba, lo que le ayuda al lector a entender tus argumentos.

2. **Definir una propiedad P(xs)**. La conclusión de la demostración será que $P(xs)$ vale para toda lista xs. En algunos casos la propiedad puede involucrar a varias variables. En ese caso debés indicar sobre cual de ellas se aplicará la inducción, es decir, cuál de ella jugará el papel de xs.

3. **Demostrar que** $P([\,])$ **es verdadera**. Esta parte de la prueba se llama caso base.

4. **Demostrar que si** $P(xs)$ **vale, entonces** $P(x \triangleright xs)$ **también vale**. Esta parte de la demostración se llama caso inductivo. La estrategia a utilizar en esta parte de la demostración es usar a $P(xs)$ como hipótesis para demostrar que $P(x \triangleright xs)$ es verdadera.

5. **Invocar el principio de inducción**. Dadas las subpruebas anteriores, el principio de inducción nos permite afirmar que $P(xs)$ es válida para toda lista

Actividad

Utilicemos el principio de inducción para demostrar un teorema útil acerca de la relación entre dos funciones: $+\!\!+$ y `sum`. Si tenemos dos listas, por ejemplo xs e ys, entonces hay dos maneras de computar la suma de los elementos de ambas listas combinadas. Se puede concatenar primero ambas listas y luego realizar la suma de todos los elementos o es posible calcular independientemente la suma de los elementos de ambas listas y luego sumar estos resultados.
Este tipo de teoremas son útiles para hacer más eficientes a los programas. Por ejemplo, suponé que tenés que sumar los elementos de una lista muy larga y que para ello contás con dos computadoras. Lo que puede hacerse es partir la lista a la mitad, hacer que cada computadora compute en paralelo la suma de una mitad de la lista original y luego, con una simple suma obtener el resultado. Esto permite acortar el tiempo de ejecución del programa en casi la mitad.

1. A partir de los párrafos anteriores formalizá la propiedad $P(xs)$ que debemos demostrar

2. Decidí sobre qué variable se va a hacer inducción y justificá tu respuesta.

3. Haciendo uso de las definiciones de las funciones `sum` y $+\!\!+$ construí la demostración para el caso base y el caso inductivo.

4. Utilizando esta demostración como base, enunciá y demostrá un teorema que me permita calcular la longitud de una lista formada por la concatenación de dos listas como la suma de las longitudes de las dos listas originales.

Usemos el principio de inducción para demostrar algunas propiedades muy buenas de las funciones que hemos definido. Para ello te proponemos la siguiente actividad:

Actividad

1. Explicá con tus palabras cuál es la propiedad que establece el siguiente teorema: $\#(\texttt{duplica}.xs) = \#xs$
2. Demostralo en FUN utilizando inducción.
3. Enunciá y demostrá una propiedad que estalezca que duplicar una lista formada por la concatenación de dos listas es igual a duplicar cada una de ellas y luego concatenarlas.
4. ¿Estos dos teoremas podrían generalizarse para demás funciones que definimos en la actividad de la página 36? ¿Por qué?
5. Demostrá el siguiente teorema: $\texttt{sum}.(\texttt{duplica}.xs) = 2 * \texttt{sum}.xs$
6. Definí la función `sumauno` que dada una lista de números le suma 1 a cada elemento de la misma devolviendo otra lista de números.
   a) Utilizando esta definición justificá con tus palabras por qué la siguiente propiedad es válida: $\texttt{sum}.(\texttt{sumauno}.xs) = \#xs + \texttt{sum}.xs$
   b) Demostrá la propiedad utilizando inducción.
7. Enunciá y demostrá por inducción que la concatenación es asociativa.
8. Nos ocuparemos ahora de probar una propiedad similar a la que demostramos entre `sum` y $+\!\!+$ en la actividad anterior. La propiedad que demostraremos vincula la función `concat` con la función $+\!\!+$ de manera tal que nos permitirá hacer más efectivos algunos de nuestros programas:

$$\texttt{concat}.(xss +\!\!+ yss) = \texttt{concat}.xss +\!\!+ \texttt{concat}.yss$$

Resumiendo

Para cerrar este capítulo y como forma de sintetizar los conceptos y herramientas desarrollados aquí te proponemos la siguiente actividad:

Actividad

Construí un mapa conceptual que relacione las siguientes palabras: listas, mapeos, funciones sobre listas, constructores de listas, propiedades de funciones sobre listas, filtros, principio de inducción, acumuladores, recursión, precedencia, tipado.

Capítulo 4

Inducción y recursión sobre los números naturales

> *"Aunque esta proposición pueda tener un número infinito de casos, daré una prueba muy corta de ella asumiendo dos lemas. El primero [...] es que la proposición es válida para la segunda fila. El segundo es que si la proposición es válida para cualquier fila entonces ella debe ser necesariamente válida para la fila siguiente"*
>
> Blaise Pascal (1654)

En este capítulo daremos un paso importante en la formalización del sistema formal que venimos construyendo lo que implicará definir explícita y rigurosamente muchos conceptos, ideas y herramientas que siempre damos por sentado. En primer lugar, vamos a dar una definición precisa de los números naturales que nos permitirá construir todos los números a partir de ciertos constructores. Esta tarea será similar a la que llevamos a cabo con las listas cuando presentamos a los constructores [ ] y ▷. Luego, vamos a definir las operaciones sobre los naturales, poniendo particular atención a la definición recursiva de las operaciones de suma y multiplicación.

La segunda mitad del capítulo está dedicada a desarrollar una de las herramientas de demostración más potente de la matemática y más útil para la programación: el principio de inducción para números naturales cuyas ideas claves están mencionadas en la cita de Pascal con la que comienza este capítulo. Utilizando esta estrategia de demostración construiremos pruebas para las propiedades más básicas de las funciones suma y multiplicación: conmutatividad, asociatividad, elemento neutro, etc.

Así, todo este capítulo está focalizado en introducir las herramientas matemáticas necesarias para demostrar propiedades que has venido utilizando desde hace muchísimo tiempo. Este no es un desafío menor e implica un esfuerzo de formalización que será recompensado ganando estrategias de prueba muy potentes.

4.1. Definición inductiva de los números naturales

En la escuela primaria todos aprendimos a escribir los números usando el sistema de numeración decimal. Excepto en ciertas culturas, es el sistema usado habitualmente en todo el mundo y en todas las áreas que requieren de un sistema de numeración. Dicho sistema posee diez símbolos diferentes: $1, 2, 3, 4, 5, 6, 7, 8, 9, 0$. Este es un sistema de numeración en el cual el valor de cada dígito depende de su posición dentro del número. Al primero corresponde el lugar de la unidades, el siguiente las decenas, centenas, etc. Para construir un número el procedimiento a seguir es multiplicar el dígito correspondiente a las unidades por 10^0, luego sumarle el dígito correspondiente a las decenas multiplicado por 10^1, luego sumarle el dígito correspondiente a las centenas multiplicado por 10^2, etc. Así, para construir el número 743, debemos hacer:

$$3 * 10^0 + 4 * 10^1 + 7 * 10^2$$

$= 3 + 40 + 700$

$= 743$

De esta forma,el sistema de numeración decimal nos permite, siguiendo ciertas reglas, definir a todos los números naturales.

Además de esta manera de construir los números naturales también contamos y, de hecho venimos trabajando, con muchas funciones: por ejemplo, la suma, la multiplicación, la resta. Hasta aquí las hemos utilizado para definir otras funciones, por ejemplo la función `sumatoria` en el capítulo anterior, pero nunca hemos dado definiciones de ellas.

Actividad

¿De qué forma definirías la función `suma` que calcule la suma de dos números naturales? No está permitido hacerlo caso por caso.

Ahora bien, si nuestro objetivo es construir programas necesitamos ser sumamente precisos porque queremos instruir a una computadora. Esto plantea la necesidad de dar una definición de todos los números y de todas las operaciones, incluidas también la suma, la multiplicación, la resta, etc. Ese será el desafío de este capítulo.

Existen otras definiciones para los naturales aparte de la que se presenta en el sistema de numeración decimal. En particular, este conjunto puede ser definido **inductivamente**. ¿Qué significa esta afirmación? Pues que definiremos dos objetos matemáticos que nos permitirán **construir** todos los números. La siguiente definición captura esta idea:

Conceptos teóricos

Definición inductiva de $\mathbb{N}$:

1. $0 \in \mathbb{N}$
2. Existe un constructor, llamado *suc*, que, dado un natural devuelve el natural siguiente.

Con la definición que acabamos de dar, podemos escribir todos los naturales sólo a partir de dos símbolos: el símbolo 0 y el símbolo *suc*. De esta forma, un número es 0 o es el sucesor de un número más pequeño.

Por supuesto, es posible "traducir" números escritos en un sistema al otro. Así, el valor unidad, que acostumbramos a denotar con el símbolo 1, según la definición inductiva de los naturales se escribe como $suc(0)$. El valor dos, en un sistema se escribe como 2 y en el otro como $suc(suc(0))$.

Actividad

Completá la siguiente tabla que permite traducir algunos valores de un sistema a otro:

Sistema decimal	Definición inductiva de $\mathbb{N}$
5	
	$suc(suc(suc(0)))$
	$suc(suc(suc(suc(suc(suc(0))))))$
8	

Si bien esta definición de los naturales puede parecer extraña en un primer momento y poco práctica para escribir cifras grandes también presenta grandes ventajas, que iremos explorando a lo largo de todo este capítulo.

4.2. Construyendo funciones sobre los naturales definidos inductivamente

En esta sección nos enfrentaremos al desafío de definir las operaciones sobre los naturales — particularmente la suma y la multiplicación— a partir de la definición inductiva de los $\mathbb{N}$. ¿Cómo

llevar a cabo esta tarea? Un primer intento es definir, por ejemplo, la suma para cada número. Así, podríamos intentar dar una definición del tipo:

$$0 + 0 = 0$$
$$0 + suc(0) = suc(0)$$
$$suc(0) + suc(0) = suc(suc(0))$$
$$suc(0) + suc(suc(0)) = suc(suc(suc(0)))$$
$$...$$

Es claro que este tipo de definición, además de ser poco práctica, nunca podría terminar de escribirse porque los naturales son infinitos. Necesitamos una mejor manera de definir la suma.

En un segundo intento, podemos dar una regla general para la suma, es decir, definir una **función** que dados dos números naturales nos devuelva la suma de ambas. Esta función tendrá, entonces, dos argumentos, que llamaremos a y b y será de la forma:

$$\begin{array}{rcl} + & :: & Nat \to Nat \to Nat \\ a + b & \doteq & \text{Definición de la función} \end{array}$$

Ahora bien, la pregunta que surge a continuación es ¿qué tipo de función utilizaremos? En este punto es donde echamos mano a nuestra definición inductiva de los $\mathbb{N}$. La misma nos permite, a partir de un número, construir el siguiente utilizando el constructor *suc*. Esto hace viable intentar dar una definición recursiva de la suma. Mas aún, en este punto no disponemos de otra cosa que no sean los constructores 0 y *suc*. Así, nuestra definición solo podrá escribirse como una combinación de ellos.

Una buena estrategia cuando tenemos que definir una función que toma dos argumentos es intentar, en primer lugar, definirla recursivamente para sólo uno de ellos. Si cuando, después de intentarlo, concluimos que no es posible hacerlo en términos recursivos de esta manera, consideraremos definirla recursivamente para ambos argumentos.

Definamos la función + en términos recursivos para el primer argumento a. Al segundo argumento lo denotaremos con la letra b. Esta es una notación que nos permite expresar que b es 0 o es el resultado de aplicar *suc* un número desconocido de veces.

Actividad

Decidamos ahora cuáles van a ser el caso base y el caso inductivo de nuestra definición de +. Hacer esto involucra distinguir dos patrones del argumento de la función que nos permitan describir a todos los naturales. Mirando la definición inductiva de $\mathbb{N}$ proponé cuál debería ser el argumento de la función para el caso base y cuál debería ser el argumento para el caso inductivo.

Para poder darnos una idea de qué es lo que debe hacer esta función utilicemos una analogía entre la suma y los movimientos y las operaciones posibles en un ábaco. Supongamos que tenemos un ábaco con sólo tres varas. En la primera de ella colocaremos las fichas correspondientes al primer argumento de la función +; en la segunda, las fichas correspondientes al segundo argumento; y en la tercera, el resultado de sumar ambos parámetros (ver figura 4.2)

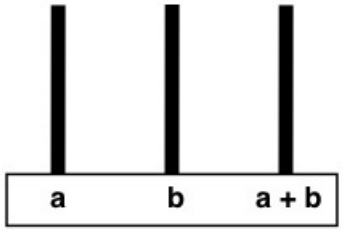


Figura 4.1: Abaco

Visto que sólo contamos con el 0 y el constructor *suc*, intentemos describir en estos términos el significado de tener fichas en una vara del ábaco:

- No tener ninguna ficha en una vara se representa como 0
- Tener una ficha en una vara se representa como $suc(0)$
- Tener dos fichas en una vara se representa como $suc(suc(0))$
- Tener tres fichas en una vara se representa como $suc(suc(suc(0)))$
- De modo general, tener n fichas en una vara se representa como aplicar n veces el suc al 0: $\underbrace{suc(suc(suc(\ldots(suc}_{n}(0)))))$

Ahora bien, pensemos qué significaría sumar el valor cuatro al valor tres con el ábaco. La situación inicial sería la siguiente:

- En la primer vara tengo tres fichas que se representan por: $suc(suc(suc(0)))$
- En la segunda vara tengo cuatro fichas que se representan por: $suc(suc(suc(suc(0))))$ (ver figura 4.2)

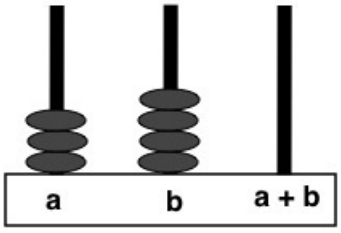


Figura 4.2: Situación inicial de la suma del valor tres con el valor cuatro

Sumar utilizando un ábaco implica mover todas las fichas de la primera y la segunda vara a la tercera vara que representa a la suma. Movamos, entonces, todas las fichas de la segunda vara a la tercera y luego todas las fichas de la primer vara a la tercera. Por último, contemos cuántas fichas hay en la tercera vara (ver figura 4.2):

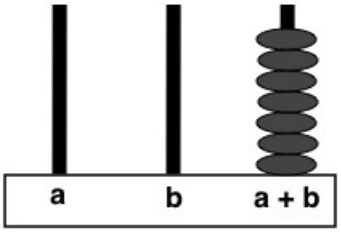


Figura 4.3: Situación final de la operación $3 + 4$

En la última vara quedaron siete fichas, lo que se representa como:

$$suc(suc(suc(suc(suc(suc(suc(0)))))))$$

Analizando este resultado podemos notar que la cantidad de suc que contiene son iguales a los suc que tenía el primer argumento y el segundo:

$$\underbrace{suc(suc(suc}_{3}(\underbrace{suc(suc(suc(suc}_{4}(0)))))))$$

Este trabajo nos ha permitido, entonces, ganar una intuición importante respecto a qué queremos que haga la función. Ahora podemos enunciar una propiedad que queremos que la función + satisfaga:

La función + será una función que aplique suc tantas veces como se aplica suc al primer argumento y al segundo.

En fórmulas, la propiedad que queremos que suma del valor n con el valor m satisfaga puede representarse como sigue:

$$\underbrace{suc(suc(\ldots(suc}_{n\ veces}(0)))) + \underbrace{suc(suc(\ldots(suc}_{m\ veces}(0)))) = \underbrace{suc(suc(\ldots(suc}_{n}(\underbrace{suc(suc(\ldots(suc}_{m}0)))))))$$

Esta será la idea que seguiremos para construir el caso base y el inductivo.

En primer lugar, construyamos una definición para el **caso base**: $0 + b$. En el ábaco esta situación se representa así:

- Primera vara: no hay ninguna ficha: 0
- Segunda vara: hay b fichas: $\underbrace{suc(suc(\ldots(suc}_{b\ veces}(0)))$ (ver figura 4.2)

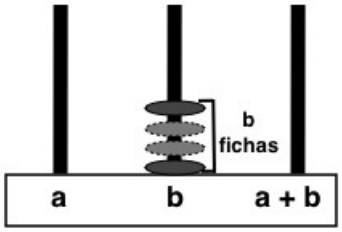


Figura 4.4: Situación inicial del caso base

Realizar la suma $0 + b$ implica mover las fichas de la primera y la segunda vara a la tercera. En este caso, no tenemos ninguna ficha de la primera vara para mover y tenemos b fichas en la segunda vara para mover a la tercera. Trasladémolas, entonces, y contemos cuántas fichas quedan en la tercera vara: evidentemente quedarán b fichas. Por lo tanto el caso base para la función puede definirse de la siguiente forma:

$$0 + b \doteq b$$

Esta definición satisface la propiedad que habíamos definido para la función +: el resultado aplica *suc* tantas veces como se aplica al primer argumento (0 veces) y tantas veces como se aplica al segundo argumento (b veces). Esta es, además, una conocida propiedad de los naturales: el cero es el elemento neutro. Haber llegado a esta propiedad nos alienta a pensar que estamos en la dirección adecuada.

Pasemos ahora al caso inductivo: $suc(a) + b$. Esta situación se representa en el ábaco así:

- Primera vara: Hay $suc(a)$ fichas. El patrón que escogimos para definir la función nos permite distinguir la última ficha: tenemos a fichas, que se representan como $\underbrace{suc(suc(\ldots(suc}_{a\ veces}(0)))$ y una última ficha.
- Segunda vara: hay b fichas: $\underbrace{suc(suc(\ldots(suc}_{b\ veces}(0)))$ (Ver figura 4.2)

Ahora bien, traslademos todas las fichas de la segunda vara a la tercera y luego todas las fichas de la primera vara a la tercera. Como resultado, en la tercera vara tendremos, en primer lugar, b fichas, luego a fichas encima y, finalmente, la última ficha de la primera vara.

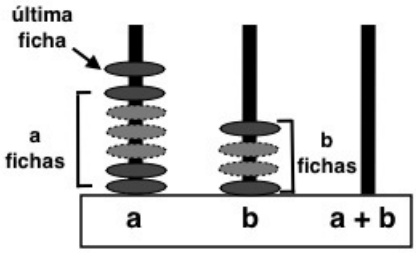


Figura 4.5: Situación inicial del caso inductivo

Al contar cuantas fichas quedaron en la tercera vara tendremos algo del estilo (ver figura 4.2):

$$suc(a) + b = \underbrace{suc}_{ultima\ ficha}(\underbrace{suc(suc(\ldots(suc}_{a\ veces}(\underbrace{suc(suc(\ldots(suc}_{b\ veces}(0)))))))))$$

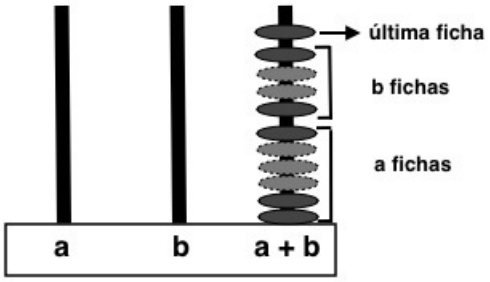


Figura 4.6: Situación final del caso inductivo

La propiedad que establecimos para la función establecía que la misma sería una función tal que aplique *suc* tantas veces como se aplica en el primer argumento y en el segundo. Observando la expresión a la que arribamos podemos notar que la segunda parte de ella puede ser reescrita en términos de la llamada recursiva —$a + b$— a partir de nuestra propiedad:

$$suc(a) + b = \underbrace{suc}_{ultima\ ficha}\underbrace{(\underbrace{suc(suc(\ldots(suc}_{a\ veces}(\underbrace{suc(suc(\ldots(suc}_{b\ veces}(0)))))))))}_{a+b}$$

Reemplazando:

$$\text{suc}(a) + b = \underbrace{suc}_{ultima\ ficha}(a + b)$$

Así, utilizar la propiedad que descubrimos nos ha permitido construir un caso inductivo recursivo para la función + donde la función se llama a sí misma en un caso más sencillo.

La definición completa de la función será:

$$\begin{array}{rcl} + & :: & Nat \to Nat \to Nat \\ 0 + b & \doteq & b \\ suc(a) + b & \doteq & suc(a + b) \end{array}$$

Para convencernos de que esta definición realmente captura nuestras ideas en torno a la suma, la evaluemos para el caso en que quiera sumarse el valor dos —representado por $suc(suc(0))$— con el valor cinco —representado por $suc(suc(suc(suc(suc(0)))))$—:

$$
\begin{aligned}
& suc(suc(0)) + suc(suc(suc(suc(suc(0))))) \\
= & \{ \text{ Definición de suma, caso inductivo } \} \\
& suc(\ suc(0)) + suc(suc(suc(suc(suc(0))))) \ ) \\
= & \{ \text{ Definición de suma, caso inductivo } \} \\
& suc(\ suc\ (0 + suc(suc(suc(suc(suc(0)))))\ )\ ) \\
= & \{ \text{ Definición de suma, caso base } \} \\
& suc(\ suc\ (suc(suc(suc(suc(suc(0)))))\ )\ )
\end{aligned}
$$

Seguiremos pasos similares para construir la definición de la función multiplicación, que denotaremos por $*$. Esta función también tomará dos argumentos, a y $b \in \mathbb{N}$ y devolverá otro natural que sea el resultado de mutiplicar ambos argumentos.

Actividad

Al igual que lo hicimos con la suma, decidamos ahora cuáles van a ser el caso base y el caso inductivo de nuestra definición de $*$. Proponé cuál debería ser el argumento de la función para el caso base y cuál debería ser el argumento para el caso inductivo.

Actividad

Para pensar en cómo vamos a definir cada caso recordemos cómo aprendimos a multiplicar en la escuela primaria: la multiplicación se define en términos de la suma. Multiplicar a por b es igual a sumar a veces el valor de b. En fórmulas esto se escribe como:

$$a * b = \underbrace{b + b + \ldots + b}_{a \ veces}$$

Esta será la propiedad que utilizaremos para guiarnos en la construcción de la definición de nuestra función en cada caso.

1. Definí la función para el caso base y comprobá que la propiedad enunciada se satisface.
2. Definí la función para el caso inductivo utilizando la propiedad para hacer la llamada recursiva de la función $*$ en un caso más simple.
3. Evaluá la función en distintos valores para los argumentos de manera de comprobar que la definición captura nuestras ideas respecto a la multiplicación.

Actividad

Definí y evaluá en FUN la siguiente función para descubrir qué hace:

$$
\begin{aligned}
\texttt{f} &:: Nat \to Nat \\
\texttt{f}.0 &\doteq 0 \\
\texttt{f}.suc(a) &\doteq a
\end{aligned}
$$

Actividad

La siguiente función es una versión de la resta en los naturales. La misma toma dos números y devuelve el resultado de restarle el segundo al primero. Para que la función pertenezca a $\mathbb{N}$ en el caso en el que el primero sea cero la función devuelve cero (de otra manera devolvería un número negativo que no pertenece a $\mathbb{N}$). Te damos listo el tipado, los casos y una de las definiciones. Observá que para esta función la recursión se realiza en los dos argumentos a y b hay más de una llamada recursiva posible: $resta.a.suc(b)$, $resta.suc(a).b$ y $resta.a.b$.

$$\begin{array}{rcl} resta & :: & Nat \to Nat \to Nat \\ resta.0.0 & \doteq & \ldots \\ resta.suc.(a).0 & \doteq & \ldots \\ resta.0.suc(b) & \doteq & 0 \\ resta.suc(a).suc(b) & \doteq & \ldots \end{array}$$

1. Escribíla en FUN , completá las definiciones para los otros dos casos y evaluala para chequear que esté bien definida:
2. ¿Se te ocurre una forma más compacta de definir esta función? En particular, ¿podrías definirla utilizando menos casos?

Actividad

Construí una definición para la función `igualdad`. Dicha función tomará dos números naturales y devolverá un booleano: *true* si los números son iguales y $False$ cuando sean distintos. Para ello:

1. Decidí cuáles van a ser los casos de la función, teniendo en cuenta que la misma posee dos argumentos
2. Describí con tus palabras lo que la función debería hacer en cada caso en una nota de FUN .
3. Definí, transformando tus palabras en fórmulas, la función para cada caso.
4. Evaluala para varios números.

Actividad

Definí y evaluá en FUN la siguiente función para descubrir qué hace:

$$\begin{array}{rcl} \mathsf{g} & :: & Nat \to Nat \to Bool \\ \mathsf{g}.0.0 & \doteq & True \\ \mathsf{g}.suc(a).0 & \doteq & False \\ \mathsf{g}.0.suc(b) & \doteq & True \\ \mathsf{g}.suc(a).suc(b) & \doteq & \mathsf{g}.a.b \end{array}$$

Actividad

Definí la función `mayor` que dado dos números, n y m, devuelve *true* si n es mayor que m o $false$ en caso contrario.

A lo largo de esta sección hemos ido construyendo todas las operaciones y relaciones básicas sobre los naturales que dimos por sentadas en el capítulo 1. Definimos la suma, la multiplicación, la resta, la igualdad, el menor o igual, etc. La siguiente sección estará dedicada a aprender una técnica que nos permita demostrar propiedades sobre estas funciones.

4.3. Principio de inducción para los números naturales

En el capítulo anterior desarrollamos el principio de inducción sobre listas. Esta es una herramienta muy poderosa para demostrar propiedades de funciones definidas para listas. En esta sección vamos a introducir un principio análogo para los números naturales. Comencemos con una analogía:

> *Una hilera de fichas de dominó, uno detrás del otro, se extiende hasta el horionte. Las fichas están colocadas los suficientemente cerca para que si una cae las siguientes caerán también. Alguien empuja la primera ficha. Como resultado, todas las fichas caen* (Ver figura).

Esta imagen de los dominós en fila cayendo uno a uno describe la esencia del principio de inducción para los naturales.

Este principio de utiliza cuando se quiere demostrar que una propiedad $P(n)$ vale para todos los números naturales. La tarea con la que nos enfrentamos es demostrar que son válidas las siguientes propiedades:

$$P(0), P(suc(0)), P(suc(suc(0))), P(suc(suc(suc(0)))), \ldots$$

Cada una de estas propiedades juega el rol de una ficha de dominó. Demostrar que esa proposición es verdadera puede interpretarse como que la ficha es golpeada por la anterior y cae. Si podemos demostrar que si la propiedad vale para un número natural cualquiera, por ejemplo n, entonces se puede demostrar que la propiedad vale para el natural siguiente $P(suc(n))$ probaríamos que cada ficha está lo suficientemente cerca como para voltear la siguiente. Si también podemos demostrar que la propiedad vale para el cero,$P(0)$ —siguiendo nuestra analogía, si podemos demostrar que la primera ficha cae—, entonces todas las proposiciones son verdaderas y nuestra fila de dominós se irá derrumbando en cadena.

En este capítulo no sólo pretendemos que comprendas este principio. Al igual que como lo hicimos en el capítulo anterior, también buscamos que puedas construir demostraciones por inducción, ya que este método es fundamental para demostrar propiedades de los programas, como veremos más adelante. Una ventaja que tendrás para enfrentarte al desafío de construir estas demostraciones es que las pruebas por inducción tienen una forma específica: siempre tendrás que demostrar que la primer ficha del dominó cae; y luego que cada ficha golpea y hace caer a la siguiente. Recordá que para el caso de las listas la situación era muy similar. Esto no quiere decir que las demostraciones serán siempre iguales y que todas serán fáciles. Algunas de ellas requerirán más de un caso previo para llegar al siguiente caso y otras necesitarán de más de un caso base.

Como primer ejemplo demostremos que el 0 es el elemento neutro a derecha de la suma. ¿Por qué demostrar algo tan obvio? Recordemos que la función + se definía de la siguiente forma:

$$\begin{array}{rcl} + & :: & Nat \to Nat \to Nat \\ 0 + b & \doteq & b \\ suc(a) + b & \doteq & suc(a + b) \end{array}$$

Esta definición establece que el 0 es el neutro de la suma a izquierda. Pero todavía no hemos construido una demostración rigurosa de que el 0 es el neutro a derecha, y si queremos utilizar esta última propiedad primero debemos demostrarla rigurosamente.

Entonces nuestro conjunto de fichas de dominó vendrán a representar al siguiente conjunto infinito de propiedades:

$$P(0) : 0 + 0 = 0,$$

$$P(suc(0)) : suc(0) + 0 = suc(0),$$

$$P(suc(suc(0))) : suc(suc(0)) + 0 = suc(suc(0)), ...$$

El primer paso que debemos dar es establecer específicamente qué es lo que queremos demostrar. Esto suele involucrar construir una fórmula que establezca en lenguaje matemático la propiedad que vamos a probar —$P(n)$—. En nuestro caso, que el 0 sea el neutro a derecha de la suma se puede escribir así:

$$P(n) : \quad n + 0 = n \;\; \textit{para todo } n \in \mathbb{N}$$

El principio de inducción nos permite construir la demostración de que una propiedad vale para todos los naturales ocupándonos sólo de dos casos: continuando con la analogía del dominó, tenemos que asegurarnos de que la primera ficha cae, es decir, tenemos que demostrar que $P(0)$ vale; y luego debemos demostrar que si una ficha cualquiera cae entonces hace caer a la que está inmediatamente detrás de ella. Vista nuestra definición inductiva de los naturales, dado un número a su sucesor se escribe como $suc(a)$. Por lo tanto, lo que debo demostrar es que si $P(n)$ vale, entonces $P(suc(n))$ vale también.

Ocupémonos, entonces, del primer caso, también llamado **caso base**. Debo demostrar que:

$$P(0) : \quad 0 + 0 = 0 \;\; \textit{para todo } n \in \mathbb{N}$$

Para construir esta demostración utilizaremos fuertemente la definición de la función $+$:

$$\begin{array}{l} \quad \underline{0 + 0} = 0 \\ \equiv \{ \text{ Definición de } + \} \\ \quad 0 = 0 \\ \equiv \{ \text{ Reflexividad } \} \\ \quad True \end{array}$$

Esta prueba establece entonces que $P(0)$ es válida.

Pasemos ahora a demostrar que si $P(n)$ vale, entonces $P(suc(n))$ vale, es decir, demostremos el **caso inductivo**. Al igual que en el principio de inducción sobre listas, es importante no perder de vista que lo que debe probarse es que una implicación es válida; no hay que demostrar que vale $P(n)$, ni que vale $P(suc(n))$ sino que **si** $P(n)$ vale, **entonces** $P(suc(n))$ vale.

Para llevar a cabo una demostración de este tipo usaremos la propiedad $P(n)$ como hipótesis en la demostración de que $P(suc(n))$ es equivalente a $True$. A esta hipótesis se también se la llama **hipótesis inductiva**.

Con estas ideas en mente, construyamos la demostración para el caso inductivo. Hay que probar que:

$$P(n) : \quad n + 0 = n \quad \Rightarrow \quad P(suc(n)) : \quad suc(n) + 0 = suc(n)$$

Partamos de $P(suc(n))$ y demostremos que es equivalente a $True$ utilizando como hipótesis $P(n)$. Nuevamente en esta demostración haremos uso de la definición de la función suma:

$$\begin{array}{l} \quad \underline{suc(n) + 0} = suc(n) \\ \equiv \{ \text{ Definición de } + \} \\ \quad suc(\underline{n + 0}) = suc(n) \\ \equiv \{ \text{ Hipótesis inductiva: } n + 0 = 0 \} \\ \quad suc(n) = suc(n) \\ \equiv \{ \text{ Reflexibidad } \} \\ \quad True \end{array}$$

Esta demostración nos asegura que si una ficha cualquiera del dominó cae, entonces también cae la que está inmediatamente detrás. Notá que hemos demostrado que $P(n) \Rightarrow P(suc(n))$ para un n genérico que puede tomar cualquier valor en los naturales. En otras palabras, hemos probado que $P(0) \Rightarrow P(suc(0))$, $P(suc(0)) \Rightarrow P(suc(suc(0)))$, $P(suc(suc(0))) \Rightarrow P(suc(suc(suc(0))))$ y así sucesivamente.

Junto con la prueba de que $P(0)$ válida, el principio de inducción nos permite concluir que $P(n)$ es válida para cualquier $n \in \mathbb{N}$

Formalicemos, entonces, las ideas desarrolladas hasta aquí, enunciando el principio de inducción:

Conceptos teóricos

Principio de inducción para $\mathbb{N}$: Sea P(n) una propiedad. Si:

1. P(0) es verdadera y
2. Si para todo $n \in \mathbb{N}$ $P(0), P(suc(0)), \ldots, P(n)$ implican $P(suc(n))$

Entonces $P(n)$ es verdadera para todo número natural.

Es importante resaltar que en la segunda condición del enunciado del principio se establece que podemos utilizar la validez de todas las propiedades anteriores para demostrar la validez de la propiedad siguiente. Esto algunas veces será necesario. Otras veces, sólo deberemos hacer uso de la validez de algunas de las proposiciones anteriores y en muchas ocasiones solo necesitaremos la validez de la proposición inmediatamente anterior —como lo hicimos cuando demostramos que el 0 es neutro a derecha de la suma. Lo que este enunciado del principio de inducción hace es permitirnos usar la validez de todas las proposiciones anteriores, pero no nos obliga a utilizarlas a todas ellas.

El formato que utilizaremos par construir demostraciones por inducción será análogo al utilizado para listas:

1. **Establecer que la demostración utiliza el principio de inducción**. Esto inmediatamente determina la estructura global de la prueba, lo que le ayuda al lector a entender tus argumentos.
2. **Definir una propiedad P(n)**. La conclusión de la demostración será que $P(n)$ vale para todo $n \in \mathbb{N}$. En algunos casos $P(n)$ puede involucrar a varias variables. En ese caso debés indicar sobre cual de ellas se aplicará la inducción, es decir, cuál de ella jugará el papel de n.
3. **Demostrar que P(0) es verdadera**. Esta parte de la prueba se llama caso base.
4. **Demostrar que** $P(0), P(suc(0)), \ldots, P(n) \Rightarrow P(suc(n))$. Esta parte de la demostración se llama caso inductivo. La estrategia a utilizar en esta parte de la demostración es usar a $P(n)$, o a la cantidad de proposiciones anteriores que sean necesarias, como hipótesis para demostrar que $P(suc(0))$ es verdadera.
5. **Invocar el principio de inducción**. Dados las subpruebas anteriores, el principio de inducción nos permite afirmar que $P(n)$ es válida para todo número natural.

4.3.1. Demostraciones de propiedades de la suma

Utilicemos el principio de inducción para demostrar propiedades de la función suma (+) definida recursivamente en las secciones anteriores.

Actividad

Completá la siguiente demostración por inducción de la asociatividad de la suma:

1. **Demostración utilizando el principio de inducción.**
2. **Definición de la propiedad a demostrar:**

   $$P(a): \quad (a+b)+c = a+(b+c) \text{ para todo } a,\ b,\ c \in \mathbb{N}$$

   En este caso la propiedad depende de tres variables, a, b y c. La demostración se hará haciendo inducción sólo en la variable a
3. **Demostrar que $P(0)$ es verdadera:**

   $$P(0): \quad (0+b)+c = 0+(b+c)$$

   $$\begin{array}{l} (0+b)+c = 0+(b+c) \\ \equiv \{\dots\} \\ \vdots \\ True \end{array}$$

4. **Demostar que $P(n) \Rightarrow P(suc(a))$:**

   $$P(suc(a)): \quad (suc(a)+b)+c = suc(a)+(b+c)$$

   $$Hipotesis\ inductiva: \quad (a+b)+c = a+(b+c)$$

   $$\begin{array}{l} (suc(a)+b)+c = suc(a)+(b+c) \\ \equiv \{\dots\} \\ \vdots \\ \equiv \{Hipotesis\ inductiva\} \\ \vdots \\ True \end{array}$$

5. Por el principio de inducción entonces vale que $P(a)$ es verdadera para todo $a \in \mathbb{N}$

Actividad

Utilizando FUN y un formato de demostración similar al de la actividad anterior, demostrá por inducción el siguiente lema:

$$suc(a)+b = a+suc(b)$$

Reflexioná primero sobre qué variable vas a hacer inducción, si sobre a o b.

Actividad

Completá la siguiente demostración por inducción de la conmutatividad de la suma:

1. **Demostración utilizando el principio de inducción.**
2. **Definición de la propiedad a demostrar:**

   $$P(a): \quad a+b=b+a \text{ para todo } a \text{ y } b \in \mathbb{N}$$

   En este caso la propiedad depende de dos variables, a y b. La demostración se realizará haciendo inducción en: ...
3. **Demostrar que $P(0)$ es verdadera:**

   $$P(0): \ldots$$

   Demostración: ...
4. **Demostar que $P(n) \Rightarrow P(suc(a))$:**

   $$P(suc(a)): \ldots$$

   $$Hipotesis\ inductiva: \ldots$$

   Demostración: ...
5. Por el principio de inducción entonces vale que $P(a)$ es verdadera para todo $a \in \mathbb{N}$

Actividad

Demostrá, por inducción, la siguiente propiedad:

$$suc(n) = n + suc(0)$$

4.3.2. Demostraciones de propiedades de la multiplicación

Ahora que tenemos demostradas las propiedades básicas para la suma, pasemos a construir pruebas para las propiedades de la multiplicación definida recursivamente en las secciones anteriores. Notá que como la multiplicación se define en términos de la suma, puede ser necesario utilizar las propiedades de la suma en las demostraciones.

Actividad

Demostrá utilizando inducción, que el cero es el elemento absorbente de la multiplicación también a derecha, es decir:

$$b * 0 = 0 \text{ para todo } b \in \mathbb{N}$$

Para hacerlo utilizá alguna de las propiedades que ya demostramos para la suma.

Actividad

Demostrá que $suc(0)$ es el elemento neutro de la multiplicación, es decir, que:

$$b * suc(0) = b \text{ para todo } b \in \mathbb{N}$$

Actividad

Visto que nuestra multiplicación está definida en términos de la suma será de gran utilidad demostrar una propiedad que vincule a ambas, en este caso, la distributividad de la multiplicación con la suma:

$$a * (b + c) = (a * b) + (a * c) \text{ para todo } a,\ b \text{ y } c \in \mathbb{N}$$

Construí una demostración por inducción de esta propiedad.

Actividad

Construí una demostración por inducción de la propiedad conmutativa de la suma. Para ello será necesario utilizar varias de las propiedades que ya hemos demostrado:

$$a * b = b * a$$

Actividad

Demostrá la asociatividad de la multiplicación:

$$a * (b * c) = a * (b * c) \text{ para todo } a \text{ y } b \in \mathbb{N}$$

De nuevo, puede ser necesario utilizar propiedades ya demostradas.

4.3.3. Una «demostración» falaz por inducción

Hasta aquí hemos venido construyendo demostraciones de propiedades de la suma y el producto explorando la potencia del principio de inducción. Pero existen sutilezas que pueden llevarnos a construir pruebas falsas utilizando este principio. En esta sección exploraremos una de ellas, con la idea de que puedas distinguir mejor lo que es una demostración por inducción correcta y lo que no es una prueba correcta.

Actividad

Lee con atención la siguiente «demostración»:

«**Teorema**»: Todas las mujeres son rubias.

En primer lugar, como nuestra proposición no incluye explícitamente ninguna variable sobre la que hacer inducción, es necesario reformularla. Demostremos, entonces, que:
$P(n)$: Para cualquier conjunto de n mujeres, si una de ellas es rubia, entonces todas las demás mujeres del conjunto son rubias.

No es posible usar como caso base el 0 porque en un conjunto de cero mujeres la proposición no tiene sentido. Comencemos la inducción en $suc(0)$ ya que un conjunto con un elemento es el mínimo conjunto posible a partir del cual nuestra proposición comienza a ser útil.
Caso base: $P(suc(0))$ afirma que en un conjunto de una sola mujer, si una de ellas el rubia, entonces todas las restantes son rubias. Esta es sólo una manera retorcida de decir "una rubia es rubia". Por lo tanto $P(suc(0))$ es verdadera.
Caso inductivo: Ahora debemos demostrar que, dado un conjunto arbitrario S de $suc(n)$ mujeres (m) que incluya a una rubia, todas las restantes son rubias. Para poder aplicar nuestra hipótesis inductiva, $P(n)$, debemos buscar conjuntos de n mujeres, incluyendo a una rubia (r), dentro de las $suc(n)$ mujeres de S. Entonces, ordenemos a las $suc(n)$ mujeres con la rubia en alguna posición:

$$m_1, m_2, \ldots, r, \ldots, m_n, m_{suc(n)}$$

A partir de este conjunto podemos definir dos subconjuntos:

$$A = m_1, m_2, \ldots, r, \ldots, m_n$$

y

$$B = m_2, \ldots, r, \ldots, m_n, m_{suc(n)}$$

Estos dos subconjuntos contienen n elementos y contienen a una rubia. Entonces, por hipótesis inductiva todas las mujeres de A y de B son rubias. Por lo tanto, en el conjunto $A \cup B$ todas las mujeres son rubias. Pero $A \cup B = S$. Por lo tanto $P(suc(n))$ es válida.

Versión libre de *Discrete Algorithmic Mathematics* de Stephen, Maurer & Anthony Ralston (1998).

1. Discutí con tus compañeros para determinar en dónde se encuentra el error de esta demostración.

4.3.4. Distinguiendo algunos conceptos

La palabra "inducción" es un término bastante corriente en el lenguaje de la ciencia, y es probable que ya la hayas escuchado algunas veces. Para poder realizar algunas distinciones importantes te proponemos la siguiente actividad:

Actividad

Lee el siguiente texto:

En ciencia, existen dos enfoques fundamentales opuestos: inducción y deducción. De acuerdo al Diccionario abreviado Oxford, la inducción consiste en "inferir una ley general a partir de ejemplos particulares" mientras que la deducción es una "inferencia de lo general a lo particular".

En general, no podemos confiar en el resultado de un razonamiento inductivo. Un ejemplo es la conjetura de Euler, formulada en 1769, que interrogaba acerca de si es posible encontrar cuatro enteros positivos a, b, c y d tal que:

$$a^4 + b^4 + c^4 = d^4$$

Después de fallar en encontrar un sólo ejemplo de este comportamiento, Euler conjeturó que esta ecuación no puede ser satisfecha. Más de dos siglos transcurrieron antes de que Elkies en 1987 descubriera el primer conjunto de números que satisfacen la ecuación, todos ellos de entre 7 y 8 dígitos. Usando cientos de horas de computación en varias computadoras conectadas, Frye mostró que el único ejemplo con d menor que un millón es:

$$95800^4 + 217519^4 + 414560^4 = 422481^4$$

En contraste, el razonamiento deductivo no está sujeto a errores de este tipo. Siempre que la regla invocada sea correcta y que se aplique a la situación bajo discusión, la conclusión alcanzada es necesariamente correcta.

Esto no significa que no podamos inferir algo falso usando el razonamiento deductivo. Desde una premisa falsa, podemos deductivamente derivar una conclusión falsa; este es el principio que subyace a las demostraciones por el absurdo. Por ejemplo, si es correcto que $P(x)$ es verdadero para todo x en un conjunto X, pero somos descuidados y aplicamos esta regla a un y que no pertenece a X entonces podemos creer erróneamente que $P(y)$ vale. De manera similar, si la creencia de que $P(x)$ es verdadera para todo x en X está basada en un razonamiento inductivo descuidado, entonces $P(y)$ puede ser falso aún si y pertenece a X. En conclusión, el razonamiento deductivo puede llevar a resultados erróneos, pero sólo si las reglas que se aplican son incorrectas o si no se aplican apropiadamente.

¿Qué rol juega el razonamiento inductivo en el descubrimiento científico? Si fueras un físico cuyo objetivo es determinar las leyes fundamentales que gobiernan el universo, tendrías que usar un enfoque inductivo: las reglas que inferirías deben reflejar datos reales obtenidos por experimentos. Por ejemplo, fue por un razonamiento inductivo que Halley predijo el regreso del famoso cometa. ¿Y qué sucede en matemática? En esta disciplina no es raro que se descubran verdades matemáticas considerando muchos casos especiales e infiriendo a partir de ellos, por inducción, que una regla general parece plausible. Sin embargo, no importa cuán convincente se vuelva la evidencia, una regla general de este tipo no puede ser afirmada sólo sobre la base del razonamiento inductivo. La diferencia entre la matemática y las ciencias experimentales es que una vez que una ley matemática general ha sido descubierta por inducción, debemos demostrarla rigurosamente aplicando el enfoque deductivo.

Traducción libre de *Fundamental of Algorithmics* de Brassard & Bratley (1996), págs: 16-18.

1. Discutí con tus compañeros para decidir si los siguientes razonamientos son inductivos o deductivos:

   "Todos los seres humanos son mortales. Lady Gaga es humana. Entonces, Lady Gaga es mortal"

   "Hemos llevado a cabo catorce experimentos en los cuales hemos dividido a los pacientes en dos grupos de pacientes, siete tratados con el medicamento y siete con un placebo. Entre los siete pacientes tratados con placebo, solamente en uno disminuyó el dolor gástrico y el dolor de cabeza, continuando la fiebre; mientras que los otros seis continuaron con la sintomatología. De los pacientes tratados con el medicamento, los siete presentaron mejoría en los síntomas gástricos, dolor de cabeza y fiebre. De estos pacientes, tres presentaron efectos secundarios consistentes en entumecimiento de dedos de las manos y mareo por la mañana; síntomas que desaparecieron tres días después de terminar la administración del medicamento. Por lo que podemos concluir que la administración de este medicamento es efectiva y segura para los pacientes."

   "La fuerza de gravedad es una constante, que hace que los objetos caigan a una velocidad de 9.8 metros por segundo cada segundo, por lo tanto, si vas a tirar mi computadora desde la azotea, como nos encontramos a 5 metros de altura, y la computadora tiene una masa de 5 kilogramos; como para ese momento alcanzará una aceleración de 4.9 metros por segundo, entonces el impacto será de aproximadamente 24,5 kilogramos, lo que significa que causarán abolladuras en el gabinete y daños en sus componentes, y si cae por su costado derecho, la destrucción de la placa principal. Así que por favor, no lances mi computadora."

2. ¿Creés que la estrategia de demostración que hemos llamado "demostración por inducción" pertenecen al grupo de los razonamientos inductivos o deductivos? ¿Por qué?

Resumiendo

Actividad

1. Releé el capítulo con la finalidad de construir un listado de los conceptos y herramientas más importantes desarrollados.

2. Escribí un texto de no más de una carilla en donde todos ellos aparezcan relacionados.

Capítulo 5

Funciones recursivas mas complejas

5.1. Funciones que construyen listas

En esta sección vamos a trabajar con funciones recursivas que, de distintas maneras, construyen una lista. Así, comenzamos a combinar en la definición de funciones recursivas diferentes tipos, haciendo nuestro lenguaje cada vez más expresivo.

Actividad

Trabajemos con la función `repetir` que toma dos naturales x y n y devuelve una lista con x repetido n veces. Por ejemplo, $repetir.3.5 = [3, 3, 3, 3, 3]$

1. ¿Cuál será el tipo de la función?
2. Un punto clave para definir `repetir` es decidir sobre qué variable se va a hacer la recursión. Discutí con tus compañeros para tomar esta decisión dando razones que la justifiquen.
3. Escribí con tus palabras en una nota de FUN qué debería hacer la función en cada uno de los casos. En el caso recursivo intentá dilucidar cómo aparece la llamada recursiva.
4. Formalizá tus ideas dando la definición de la función.
5. Demostrá la siguiente propiedad que define en parte cómo debe comportarse la funcion `repetir`:

   $$\#\texttt{repetir}.n.m = n$$

   ¿Qué dice esta propiedad?

Actividad

Ocupémonos de la función `desdehasta` que toma dos naturales, n y m, y devuelve una lista cuyos elementos son los naturales desde n hasta m. Para que la función pueda ser definida es preciso que $n \leqslant m$.

1. Pensá cuál debería ser el resultado de aplicar la función a los siguientes parámetros: 6 y 12; 0 y 1 y 5 y 5.
2. ¿Cuál será el tipo de la función?
3. ¿Sobre qué parámetro se va a realizar la recursión?
4. La siguiente es una nota de FUN escrita por un alumno, leela con atención:

   Cuando los dos números sean iguales la función debe devolver la lista con sólo un elemento, m o n. Cuando n sea menor que m, el primer elemento de la lista debe ser m. Luego debe venir $m - 1$, luego $m - 2$ y así sucesivamente hasta que se llegue al caso en que los dos parámetros son iguales. Esto se puede ver como la llamada recursiva pero aplicada a n y a $m - 1$.

   a) ¿Qué ocurre con el caso base en esta función? ¿Qué valores tomarán los parámetros para este caso? ¿Qué debe devolver la función?

   b) Definí la función para el caso recursivo formalizando las ideas discutidas hasta ahora.
5. Existe una relación entre los parámetros de `desdehasta` y el tamaño de la lista resultado. ¿Cuál es esa relación? Escribila formalmente y demostrala.

5.2. Recursión heterogénea

Hasta el momento veníamos analizando y construyendo funciones que aplicaban la recursión a sólo uno de sus argumentos, como por ejemplo la función $+\!\!+$ que aunque toma dos listas se define recursivamente sobre sólo uno de sus parámetros; o funciones que recurrían sobre mas de un parámetro, pero siempre del mismo tipo como la función `resta`.

Pero existen funciones que combinan parámetros de diferentes tipos y precisan, para su correcta definición, que la recursión se realice sobre varios de ellos. En estos casos decimos que la recusión es **heterogénea**. De este tipo de funciones nos ocuparemos en esta sección.

Actividad

Considerá la siguiente función que se denota con el símbolo !:

$$\begin{array}{rcl} ! & :: & [A] \to Num \to A \\ (x \triangleright xs)!0 & \doteq & x \\ (x \triangleright xs)!(n+1) & \doteq & xs!n \end{array}$$

1. Escribí con tus palabras qué toma la función y qué devuelve.
2. Evaluá la función en distintos casos particulares.
3. ¿Sobre qué variables se está haciendo recursión?
4. ¿Por qué se consideran estos casos?
5. A partir de tus respuestas anteriores ¿qué hace esta función?

La función de la actividad anterior se denomina comúnmente `indexar` y se denota con el símbolo "!". Suele ser de gran utilidad porque nos permite acceder a un elemento determinado de una lista.

Así podemos utilizarla para construir otra función que, por ejemplo, multiplique cada elemento de una lista de números por el tercer elemento de dicha lista.

La función `indexar` sólo está definida para listas no vacías visto que no es posible devolver un elemento de una lista que no contiene elementos. Por esta razón en la definición anterior no están considerados los casos $[\,].0$ ni $[\,].(n+1)$.

Como ya hemos visto, cuando utilizamos recursión doble generalmente deben considerarse mas de dos casos. Así, si se quiere definir una función `h` que aplique simultáneamente la recursión a una lista y a un número, los casos a considerar inicialmente serán los siguientes:

$\texttt{h}.[\,].0 \doteq \ldots$

$\texttt{h}.[\,].(n+1) \doteq \ldots$

$\texttt{h}.(x \triangleright xs).0 \doteq \ldots$

$\texttt{h}.(x \triangleright xs).(n+1) \doteq \ldots$

Estos cuatro casos surgen de combinar de todas las maneras posibles el hecho de que un número es 0 o es distinto de 0 y que una lista es vacía o tiene al menos un elemento. Cuando el resultado de aplicar la función a dos casos es el mismo, es posible que estos se reduzcan a uno. Por ejemplo si `h` devuelve el mismo resultado para los dos primeros casos se puede definir a la función usando solamente tres renglones:

$\texttt{h}.[\,].n \doteq \ldots$

$\texttt{h}.(x \triangleright xs).0 \doteq \ldots$

$\texttt{h}.(x \triangleright xs).(n+1) \doteq \ldots$

El primero de ellos considera tanto el caso en que n sea cero o distinto de cero. Te recomendamos que cuando te enfrentes a una recursión doble, en primer lugar, definas los cuatro casos para asegurarte de que no olvidas ninguno. Luego, si notás que el resultado es el mismo para algunos de ellos intentes disminuir el número de casos.

Todas las consideraciones que hemos mencionado para definir una función con recursión heterogéna son también válidas para el momento de hacer demostraciones por inducción sobre estas mismas funciones. Es muy probable que al intentar hacer una demostración de cierta propiedad sobre una función de este tipo necesitemos utilizar su definición. Para poder hacer esto efectivamente, va a resultar necesario poder distinguir el caso de la definición correspondiente. De esta manera estaremos casi obligados a hacer en nuestra demostración por inducción tantos casos (y los mismos!) como existan en la definición de las funciones involucradas.

Actividad

Las funciones ! y $+\!\!+$ están relacionadas de una manera relativamente compleja. ¿Se te ocurre cómo? Completá la siguiente definición y demostrá la propiedad haciendo inducción:

$(xs +\!\!+ ys)!n = \ldots$

Ayuda: pensá la relación entre n y las longitudes ambas listas.

Actividad

Definí la función $\uparrow$ que toma una lista $[A]$ y un natural n y devuelve otra lista. El resultado de aplicar esta función es una lista que contiene los primeros n elementos de la lista original.

1. Pensá cuál debería ser el resultado de aplicar esta función para algunos casos particulares.
2. Determiná cuál es el tipo de $\uparrow$.
3. En primer lugar, decidí sobre qué variables vas a aplicar la recursión y en función de esta decisión establecé qué casos vas a tratar. Escribí con tus palabras en una nota de FUN qué debería devolver la función en cada uno de ellos.
4. Definí $\uparrow$ para cada caso y probá tu definición con ejemplos particulares. Si es necesario revisá tu definición.
5. ¿Qué relación existe entre la $\uparrow$ y `cabeza`? ¿Cómo demostrarías esa relación?

Actividad

Definí la función $\downarrow$, análoga a $\uparrow$, que toma una lista $[A]$ y un natural n y devuelve otra lista, que es el resultado eliminar los primeros n elementos de la lista original.

1. Pensá cuál debería ser el resultado de aplicar esta función para algunos casos particulares.
2. Determiná cuál es el tipo de $\downarrow$.
3. En primer lugar, decidí sobre qué variables vas a aplicar la recursión y en función de esta decisión establecé qué casos vas a tratar. Escribí con tus palabras en una nota de FUN qué debería devolver la función en cada uno de ellos.
4. Definí $\downarrow$ para cada caso y probá tu definición con ejemplos particulares. Si es necesario revisá tu definición.
5. ¿Qué relación existe entre la $\downarrow$ y `cola`?

Actividad

A uno de tus compañeros se le ocurrió la siguiente propiedad que relaciona la función $\uparrow$, $+\!\!+$, $\#$ y `resta`:[a]

$$(xs +\!\!+ ys) \uparrow n = xs \uparrow n \ +\!\!+ \ ys \uparrow \texttt{resta}.n.(\#xs)$$

1. Tomate un tiempo para intentar comprender lo que dice esta propiedad. Puede ser útil instanciarla para algunos casos particulares de xs, ys y n.
2. ¿La propiedad es correcta? Si no lo es corregila.
3. Demostrá la propiedad (posiblemente corregida) utilizando los siguientes casos de inducción:
   - 0 para el parámetro n,
   - [ ] para el parámetro xs, y
   - $n + 1$ y $x \triangleright xs$.
4. ¿Se te ocurre una propiedad similar, pero para $\downarrow$?

[a] Aunque usamos simbolos especiales para las funciones !, $\uparrow$ y $\downarrow$ y las utilizamos como si se trataran de operadores, son simples funciones y por lo tanto su aplicación tienen la misma precedencia que $\#$, `cabeza` y `cola`

5.3. Recursión doble sobre listas

Trabajemos ahora con funciones recursivas dobles en dos listas. Para ello te proponemos la siguiente actividad:

Actividad

Definamos la función `iguales` que dadas dos listas devuelve *True* si ambas son iguales y *False* en caso contrario.

1. Teniendo en cuenta que precisamos hacer recursión en las dos listas porque necesitamos ir comparando uno a uno los elementos de ambas, ¿cuáles serán todos los casos a considerar en la definición de la función?
2. Da una definición para cada uno de ellos. Puede serte útil escribir en una nota tus ideas iniciales sobre los mismos.

Vamos a ocuparnos ahora de una función muy importante y bastante frecuente dentro de la programación. Es aquella función que se ocupa de «mezclar» ordenadamente dos listas previamente ordenadas. Ordenar es una actividad de gran relevancia cuando se construyen programas y para ello se han desarrollado una gran variedad de métodos.

En el caso de la función `mezclaordenada` que queremos definir nuevamente no basta con hacer recursión en una sola de las listas que toma la función porque, al igual que en la función `iguales` necesitamos ir comparando uno a uno los elementos de ambas listas. Además será preciso tener en cuenta los casos en los que una de ellas sea vacía y la otra no.

Te proponemos la siguiente actividad para construir una definición de la función:

Actividad

1. Pensá qué debería devolver la función en algunos casos particulares para poder ganar intuición sobre la definición de *mezclaordenada*.
2. Definí el tipo de la función.
3. Considerando que no es necesario que ambas listas tengan la misma cantidad de elementos y que es preciso hacer recursión en las dos listas, ¿cuáles serán los casos en los que definir la función?
4. Escribí en una nota de FUN qué debería hacer *mezclaordenada* en los casos en que una de las listas sea vacía y la otra no.
5. Esta es una nota de FUN que elaboró un estudiante para el caso en que las dos listas son no vacías:

   Aquí la función debe comparar las cabezas de ambas listas. Si la cabeza de la primera lista (x) es menor que la cabeza de la segunda (y) entonces la lista resultado debe comenzar con x y luego debo seguir ordenando las listas xs y ($y \triangleright ys$). Una cosa similar ocurre cuando la cabeza de la primera lista es mayor que la cabeza de la segunda.
6. Usando todas estas notas, da una definición de *mezclaordenada* para cada uno de los casos.
7. Evaluá tu definición usando distintas listas y, de ser necesario, modificá tu definición.

Capítulo 6

Lógica Proposicional

En los capítulos anteriores hemos ido desarrollando nuestro sistema formal a partir de dos tipos de datos: los números y las listas. En este capítulo centramos nuestra atención en los **razonamientos**, lo que nos llevará a introducir un nuevo tipo de datos: las **proposiciones**.

La rama de la matemática que estudia los razonamientos se denomina lógica. A continuación te proponemos una actividad en donde trabajarás con algunos ejemplos de razonamientos son muy conocidos en el estudio de la lógica:

Actividad

¿Crees que estos razonamientos son correctos? ¿Cómo justificarías tus respuestas?

- Si tuviera pelo sería feliz. No tengo pelo. Entonces, no soy feliz.
- Todos los hombres son mortales. Sócrates es hombre. Por lo tanto, Sócrates es mortal.
- Si Dios fuera incapaz de evitar el mal no sería omnipotente; si no quisiera hacerlo sería malévolo. El mal sólo puede existir si Dios no puede o no quiere impedirlo. El mal existe. Si Dios existe, es omnipotente y no es malévolo. Por lo tanto, Dios no existe.

Uno de los conceptos básicos de lógica es el de **proposición**. Suele definirse una proposición como una sentencia declarativa de la cual puede decirse que es verdadera o falsa. Por ejemplo, en el primer razonamiento de los anteriores, "Sócrates es mortal" es una proposición.

Las proposiciones serán usadas esencialmente en **razonamientos**.

Conceptos teóricos

Razonamiento: Entendemos por razonamiento a un conjunto de proposiciones de las cuales se afirma que una de ellas se deriva de las otras. En este sentido, un razonamiento no es cualquier conjunto de proposiciones sino que tiene una estructura. La **conclusión** de un razonamiento es una proposición que se deriva de las otras, llamadas **hipótesis**. Se supone que las hipótesis sirven como justificación o evidencia de la conclusión. Si el razonamiento es válido, no puede aceptarse la verdad de las hipótesis sin aceptar también la validez de la conclusión.

Se suele definir a la lógica como el estudio de los métodos y principios usados para distinguir los razonamientos correctos de los incorrectos. En este sentido, la lógica estudia básicamente la estructura de estos razonamientos, y determina, a partir de este análisis, si un razonamiento es correcto o no. Así, La lógica examina la validez de los argumentos en términos de su estructura lógica, independientemente del contenido específico del discurso y de la lengua utilizada en su expresión y de los estados reales a los que dicho contenido se pueda referir. Esto es exactamente lo que quiere decir que la lógica es una ciencia «formal». Por lo tanto, lo único que la lógica puede afirmar de un razonamiento correcto es que si se partió de hipótesis verdaderas la conclusión será verdadera, pero en el caso que alguna de las hipótesis sea falsa nada sabremos del valor de verdad de la conclusión.

Desde hace ya 2.500 años la filosofía busca y propone respuestas a las siguientes preguntas vinculadas con problemas que involucran proposiciones y razonamientos:

- ¿Cómo puede determinarse si un razonamiento es correcto?
- ¿Cómo puede determinarse si una conclusión es consecuencia de un conjunto de hipótesis?, y de ser así, ¿cómo puede demostrarse que lo es?
- ¿Qué características de las estructuras del mundo, del lenguaje y de las relaciones entre palabras, cosas y pensamientos hacen posible el razonamiento deductivo?

Las herramientas con las que contamos hasta ahora dentro de nuestro sistema formal no son suficientes para realizar este tipo de análisis. En este capitulo extendemos nuestro formalismo para poder dar cuenta de ello.

6.1. Elementos sintácticos y su semántica *intuitiva*

Los razonamientos presentados más arriba fueron hechos en castellano. Cuando se usa un lenguaje natural, es sencillo caer en ambigüedades y confusiones que no tienen que ver con problemas lógicos. Para evitar este tipo de confusiones y concentrarnos en los problemas centrales de la lógica, se creará un lenguaje artificial libre de este tipo de ambigüedades, al cual puedan después traducirse los razonamientos anteriores.

Conceptos teóricos

Una **proposición** se puede pensar como una afirmación, es decir, una oración de la cuál se puede afirmar que es verdadera o que no lo es, aunque no sepamos la respuesta en ese momento.

Por ejemplo, sobre la oración "la pizza es un vegetal" puede afirmarse que es **falsa** y sobre la expresión "$2 + 4 = 6$" puede afirmarse que es **verdadera**, por lo tanto ambas son proposiciones. Por otro lado la oración "lunes" no es una proposición, porque es imposible asignarle un *valor de verdad*.

Es importante aclarar que, en muchos casos, aunque no conozcamos su valor de verdad (verdadero o falso) las proposiciones siguen siendo tales; por ejemplo, la afirmación"mañana lloverá" puede ser verdadera o falsa, pero no lo sabremos hasta que el día de mañana llegue. Sin embargo, es considerada como una proposición bien formulada.

Para poder referirnos a estas proposiciones y poder combinarlas las representaremos con un nombre. Por ejemplo, con la letra p podemos representar la afirmación "me gusta la palta" y con la letra q podemos representar la afirmación "me gusta el queso". Para definir a estas proposiciones utilizaremos el mismo símbolo que introdujimos para definir funciones ($\doteq$). Las dos proposiciones anteriores se definen, entonces, de la siguiente manera:

$$p \doteq \text{me gusta la palta}$$
$$q \doteq \text{me gusta el queso}$$

Para realizar afirmaciones más complejas, en lenguaje natural, utilizamos conectores. La lógica proposicional toma algunos de ellos y los representa de forma simbólica. Por ejemplo si queremos afirmar "me gusta la palta y el queso" podemos hacerlo utilizando las letras p y q definidas anteriormente y un nuevo símbolo, en este caso $\wedge$, que represente el conector "y"

$$p \wedge q$$

En la expresión anterior, las letras p y q funcionan como variables ya que su valor puede variar dependiendo de la situación. Cada una de ellas puede ser verdadera o falsa en función de los gustos que cada persona tenga y esto hará que la expresión completa $p \wedge q$ también cambie su valor de verdad. Por lo tanto, p y q serán llamadas **variables proposicionales**. Además, muchas veces cuando trabajemos con estos símbolos nos vamos a olvidar del significado específico que le dimos, es decir de su *semántica*, y sólo nos vamos a concentrar en la relación que hay entre las expresiones.

Así, nos ocuparemos principalmente de la forma de las expresiones y no del significado que le hemos asignado a cada variable.

En el capítulo 1 introdujimos el concepto de **formula**. Recordá que, en ese caso, una fórmula estaba formada por una combinación de expresiones a través de un símbolo de relación, por ejemplo $6+7*2 > 21$. La característica principal de las fórmulas es que representan valores de tipo booleano, es decir, pueden ser verdaderas o falsas. Las proposiciones que acabamos de introducir en nuestro sistema formal también comparten esta característica. Por lo tanto, extender nuestro sistema formal implicará extender también la idea de fórmula incorporando a este concepto nuevas variables, las constantes *True* y *False* y **operadores booleanos** que nos permiten construir fórmulas complejas a partir de otras mas sencillas.

De ahora en más una fórmula será alguna de las siguientes, y la llamaremos **fórmula proposicional**:

Conceptos teóricos

Serán **formulas proposicionales**:

- una constante como *True* y *False*;
- una variable proposicional, como $p, q, r, \ldots$;
- una fórmula númerica como las que definimos en la seccion 2.1.1;
- una combinación de fórmulas proposicionales con los operadores booleanos $\wedge, \vee, \neg, \Rightarrow, \Leftarrow, \equiv, \not\equiv$.

A continuación describiremos cada uno de estos operadores booleanos y su significado. Más adelante los usaremos para interpretar oraciones del lenguaje natural y consideraremos los aspectos más sutiles o controversiales del uso de la lógica matemática para razonar acerca de argumentos presentados en el lenguaje corriente.

Conjunción. El operador $\wedge$ representa el 'y' del lenguaje. Es un operador binario, ya que toma dos fórmulas (que llamaremos subfórmulas) y construye una nueva fórmula. Una fórmula de esta clase es verdadera cuando ambas subfórmulas son también verdaderas. Ya hemos visto un ejemplo de este operador en funcionamiento en los párrafos anteriores cuando formalizamos la afirmación "me gusta la palta y el queso" a través de la fórmula $p \wedge q$. Para que esta fórmula sea verdadera es preciso que efectivamente me guste el queso y la palta. Si sólo me gusta el queso pero no la palta entonces sólo una de las subfórmulas es verdadera y por lo tanto la fórmula completa es falsa.

Disyunción. El operador binario $\vee$ representa el 'o' del lenguaje. Se lo suele llamar disyunción inclusiva, dado que una formula compuesta por $\vee$ es verdadera cuando al menos una de las subfórmulas es verdadera. Siguiendo el ejemplo anterior, la afirmación "me gusta la palta o el queso" se formaliza a través de la fórmula $p \vee q$ y para que sea verdadera solo alcanza con que una de las subfórmulas sea verdadera.

Negación. El operador $\neg$ es unario, es decir toma una sola fórmula y construye otra cuyo valor de verdad es el opuesto. Así, si F_1 formaliza la afirmación "me gusta la palta" entonces $\neg F_1$ simboliza la afirmación "no me gusta la palta".

Implicación. El operador binario $\Rightarrow$ captura la idea de consecuencia lógica. Si F_1 y F_2 son subfórmulas, la fórmula $F_1 \Rightarrow F_2$ es verdadera siempre que siendo F_1 verdadera, F_2 también lo sea. La fórmula anterior también es verdadera cuando la *hipótesis* F_1 es falsa. Como veremos mas adelante, este operador es uno de los más controversiales respecto de su relación con el lenguaje.

Consecuencia. El operador $\Leftarrow$ es el converso de $\Rightarrow$, esto significa que $F_1 \Leftarrow F_2$ es igual a $F_2 \Rightarrow F_1$.

Equivalencia. El operador $\equiv$ simboliza la igualdad de valores de verdad. Ya hemos trabajado con este operador cuando simplificamos fórmulas numéricas en la sección 1.2.1 del capítulo 1.

Actividad

Usando la definición de fórmula, construí cinco fórmulas proposicionales, con al menos dos operadores *booleanos*.
¿Cómo podes justificar que tus fórmulas están bien construidas?

Actividad

¿Son fórmulas proposicionales las siguientes?

1. p
2. $2 + 3$
3. $2 + 3 = 4$
4. $(p \Rightarrow q) \wedge r$
5. $\wedge\ p\ \vee$
6. $5 * 7 \wedge p$
7. $2 < 3 \vee 3 < 4 \Rightarrow r \equiv (2 < 3 \Rightarrow r) \wedge (2 < 4 \Rightarrow r)$

6.1.1. Reglas de precedencia

Para las expresiones numéricas habíamos introducido las reglas de precedencia que nos permitían desambigüar el significado de una expresión compleja, aun cuando no contaba con todos los paréntesis.

Para el caso de las fórmulas proposicionales es necesario definir la precedencia de cada operador y su relación con las reglas de las fórmulas númericas:

$\sqrt{}, (\cdot)^2$	raíces y potencias
$*, /$	producto y división
$+, -$	suma y resta
$=, \leqslant, \geqslant$	conectivos aritméticos
$\neg$	negación
$\vee\ \wedge$	disyunción y conjunción
$\Rightarrow\ \Leftarrow$	implicación y consecuencia
$\equiv\ \not\equiv$	equivalencia y discrepancia

Por ejemplo, la fórmula

$$(((2 < 3) \vee (3 < 4)) \Rightarrow r) \equiv (((2 < 3) \Rightarrow r) \wedge ((2 < 4) \Rightarrow r))$$

puede simplificarse como

$$2 < 3 \vee 3 < 4 \Rightarrow r \equiv (2 < 3 \Rightarrow r) \wedge (2 < 4 \Rightarrow r)$$

Actividad

Al igual que como lo hacíamos con fórmulas numéricas, indicá cómo se fue construyendo la expresión marcando cada etapa de la construcción con un subrayado diferente:

1. $p \vee q \Rightarrow r \equiv (p \Rightarrow r) \wedge (q \Rightarrow r)$.
2. $p \Rightarrow q \equiv p \vee q \equiv q$.
3. $p \Rightarrow q \equiv \neg p \vee q$.

Actividad

¿Son correctos los siguientes subrayados?

1. $p \Rightarrow \underline{(q \Rightarrow p)} \equiv p \wedge q \Rightarrow q$.
2. $\neg a * b + c = d \vee p \Rightarrow q \equiv r \Leftarrow s \wedge j = k + l * m$.
3. $(a > b \wedge c \Rightarrow d) \wedge (x > 0) \vee p \Leftarrow s$

6.1.2. Tipado de fórmulas proposicionales

En los capítulos anteriores vimos como la herramienta de tipado nos permitía distinguir entre fórmulas que estaban bien escritas y tenían pleno sentido de las que no. Mientras trabajabamos sólo con la aritmética (y las constantes *True* y *False*) las posibilidades de escribir expresiones mal formadas se reducían casi a no respetar la aridad de los operadores. Ahora con la introducción de los operadores booleanos, nos preguntamos como tipar formulas que los involucren.

Actividad

Utilizando EQU y a partir de fórmulas proposicionales sin variables, deducí los esquemas de tipado para los nuevos operadores booleanos.

Actividad

¿Están bien escritas las siguientes fórmulas? Justificá construyendo el arbol de tipado en EQU .

1. $6 \geqslant 4 \wedge 3 + 2 < 4 \;\Rightarrow\; \mathit{True} \equiv 1 + 1 = 2$.
2. $1 + 2 \geqslant 6 \Rightarrow 3 + 2 < 1 \;\equiv\; \mathit{False} = 11 = 2 * 5$.

Ahora pasaremos a tipar fórmulas que involucran variables. Para tiparlas será necesario utilizar los esquemas de tipado que construiste en la actividad anterior.

Actividad

Utilizá EQU para averiguar que tipo debe asignarse a cada variable para que las siguientes fórmulas estén bien tipadas:

1. $(\mathit{True} \equiv a) \vee 5$.
2. $b \vee 3 = 4 \vee a * a + 2 \leqslant b + 7$.
3. $(x \wedge y \equiv a) \wedge z \leqslant w$.
4. $a \vee b = 3 + y$.
5. $x = (y = z)$.
6. $(x = y) \equiv z$.
7. $x = (y \equiv z)$.
8. $a \wedge b \leqslant a$.
9. $\neg a * b + c = d \vee p \Rightarrow q \equiv r \Leftarrow s \wedge j = k + l * p$.

6.2. Semántica formal: tablas de verdad

Enfrentemonos ahora a la tarea de evaluar una fórmula proposicional, es decir, determinar si su valor es verdadero o falso. En la sección anterior, introdujimos el significado intuitivo de los diferentes operadores. Para poder determinar el valor de una fórmula formalmente, es necesario disponer del significado de cada uno de los operadores, en todos los casos posibles. Esto se denomina la semantica formal.

La semántica de las fórmulas proposicionales mas simple es la siguiente:

- Las constantes *True* y *False* tienen los valores verdadero y falso respectivamente.
- El valor de una variable proposicional está dado por su significado en el contexto. Cuando la variable está libre, entonces su valor es desconocido, y se asume que puede tomar tanto el valor verdadero como el valor falso.
- El valor de una fórmula numérica se obtiene a partir de la semántica que vimos en los capítulos anteriores.

Para las fórmulas mas complejas que involucran operadores booleanos, el valor de verdad se obtiene a partir de los valores de las subfórmulas y la sémantica de los operadores involucrados.

Para los operadores binarios las posibles combinaciones de los valores de las subfórmulas son cuatro, mientras que para el operador unario de la negación son dos. Una forma esquemática de describir estos valores son las llamadas tablas de verdad. En estas tablas se colocan en la primera columna las posibles combinaciones de valores de las subfórmulas y debajo de cada expresión el valor que corresponde al estado descripto por una fila dada. En lo que sigue vamos a utilizar $F, F_1, F_2, \ldots$ para referirnos a una fórmula cualquiera.

El caso de la negación es el más simple dado que sólo son necesarias dos filas:

F	$\neg F$
verdadero	falso
falso	verdadero

En la siguiente tabla se resumen las tablas de verdad de los operadores $\wedge, \vee, \equiv$:

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \equiv F_2$
verdadero	verdadero	verdadero	verdadero	verdadero
falso	verdadero	falso	verdadero	falso
verdadero	falso	falso	verdadero	falso
falso	falso	falso	falso	verdadero

Actividad

EQU nos permite evaluar facilmente fórmulas proposicionales. Utilizandolo, obtene el valor de verdad de las siguientes fórmulas y deducí la tabla de verdad del operador $\Rightarrow$ (y por lo tanto tambien la tabla de $\Leftarrow$):

- $2+3=5 \;\Rightarrow\; 2+3+1=5+1$
- $2+3=5 \;\Rightarrow\; 0 \neq 0$
- $2+3 \neq 5 \;\Rightarrow\; 0 \neq 0$
- $2+3 \neq 5 \;\Rightarrow\; 2+3+1=5+1$

Actividad

Simplifica las siguientes fórmulas, utilizando las definiciones de cada operador booleano dadas por sus tablas de verdad:

1. $((True \wedge True) \vee False) \equiv False \vee True$
2. $7 > 4 \wedge 4 > 7$
3. $7 > 4 \vee 4 > 7$
4. $5 > 3 \wedge 3 > 1 \Rightarrow 5 > 1$
5. $False \Rightarrow 2 + 2 = 5$
6. $2 + 2 = 5 \Rightarrow True$

Actividad

Evaluá en EQU las siguientes fórmulas proposicionales, suponiendo que la variable p tiene valor verdadero, q falso, y r falso.

1. $p \Rightarrow (q \equiv r)$
2. $\neg p \equiv (\neg q \vee (q \vee r))$
3. $(q \Rightarrow (p \equiv \neg r)) \Rightarrow ((p \wedge q) \Rightarrow (r \equiv r))$
4. $\neg(p \vee \neg q) \Rightarrow (\neg(r \equiv (p \wedge q)) \Rightarrow \neg(p \vee \neg q))$

6.3. Lenguaje y lógica

La lógica matemática surgió como una manera de analizar los razonamientos escritos en lenguaje natural. Los operadores presentados en la sección anterior fueron pensados como contrapartidas formales de operadores del lenguaje. Si se tiene cierto cuidado puede traducirse una proposición escrita en lenguaje natural a lenguaje simbólico. Esta traducción nos permitirá dos cosas: por un lado resolver ambigüedades del lenguaje natural; por otro, manipular y analizar las expresiones así obtenidas usando las herramientas de la lógica.

La idea básica de traducción consiste en identificar las proposiciones elementales de un enunciado dado y componerlas usando los operadores booleanos asociados a los conectivos del lenguaje que aparecen en el enunciado. Los conectivos del lenguaje son traducidos a su interpretación literal, aunque veremos algunas sutilezas de dicha traducción.

Actividad

Discutí con tus compañeros cómo traducir la siguiente oración del lenguaje natural en una fórmula proposicional:

Hamlet defendió el honor de su padre pero no la felicidad de su madre.

Supongamos que complejizamos la sentencia anterior de la siguiente manera:

Hamlet defendió el honor de su padre pero no la felicidad de su madre o la suya propia.

Podemos analizarla como compuesta por tres proposiciones elementales:

p : Hamlet defendió el honor de su padre.
q : Hamlet defendió la felicidad de su madre.
r : Hamlet defendió su propia felicidad.

Usando estas variables proposicionales podemos traducirla como:

$$p \wedge \neg(q \vee r)$$

Notá que la palabra "pero" se traduce como una conjunción (dado que afirma ambas componentes).

6.3.1. Negación, conjunción y disyunción

Actividad

¿Cómo traducis la sentencia "No todos los unicornios son azules"? ¿y la sentencia "Algunos unicornios no son azules"?

La operación de negación aparece usualmente en el lenguaje insertando un "no" en la posición correcta del enunciado que se quiere negar; alternativamente, puede anteponerse la frase "es falso que" o la frase "no se da el caso que" o hacer construcciones algo más complejas. Por ejemplo el enunciado

Todos los unicornios son azules.

también puede negarse de las siguientes maneras:

No se da el caso de que todos los unicornios sean azules.

Es falso que todos los unicornios sean azules.

Si simbolizamos con p a la primera proposición, usaremos $\neg p$ para cualquiera de las variantes propuestas.

Actividad

Si tomamos como proposiciones simples las siguientes

p : Llueve.

q : Hace frío.

¿cuáles de las siguientes sentencias del lenguaje natural están representadas por la fórmula $p \wedge \neg q$?

1. Llueve y no hace frío.
2. No hace frío pero llueve.
3. Es falso que llueve y hace calor.
4. Aunque llueve hace frío.

La conjunción se representa por la palabra "y" uniendo dos proposiciones. Otras formas gramaticales de la conjunción se interpretan del mismo modo, por ejemplo las palabras "pero" o "aunque". Así, interpretamos las siguientes oraciones:

Llueve y no hace frío.

Llueve pero no hace frío.

Aunque llueve no hace frío.

como representadas por la proposición

$$p \wedge \neg q$$

Hay que tener en cuenta, sin embargo, que la palabra "y" no siempre representa una conjunción. Si bien lo hace en la frase

Joyce y Picasso fueron grandes innovadores en el lenguaje del arte.

no es este el caso en:

Joyce y Picasso fueron contemporáneos.

donde simplemente se usa para expresar una relación entre ambos. En este caso la sentencia no puede dividirse en dos sentencias relacionadas por el "y" ya que la relación "ser contemporáneo" necesariamente involucra al menos dos sujetos.

Actividad

- Si tenemos las siguientes proposiciones elementales

  p : Para ser emperador hace falta el apoyo de la nobleza.
  q : Para ser emperador hace falta el apoyo del pueblo.

  ¿cómo formalizas la sentencia "Para ser emperador hay que tener el apoyo de la nobleza o del pueblo. "?

- Si consideramos la sentencia "El consejero del emperador pertenece a la nobleza o al pueblo". ¿cuáles son las sentencias elementales involucradas? ¿utilizarías el mismo operador para formalizar la sentencia completa?

La palabra usual para representar la disyunción es "o", aunque existen variantes del estilo "o bien . . . o bien". El caso de la disyunción es un poco más problemático que los anteriores, dado que existen en el lenguaje dos tipos de disyunción, la llamada *inclusiva* y la *exclusiva*. Ambos tipos difieren en el caso en que las proposiciones intervinientes sean ambas verdaderas. La disyunción inclusiva considera a este caso como verdadero, como por ejemplo en:

Para ser emperador hay que tener el apoyo de la nobleza o del pueblo.

obviamente, teniendo el apoyo de ambos se está también en condiciones de ser emperador. Esta proposición se simboliza como

$$p \vee q$$

donde p y q son las proposiciones definidas en la actividad anterior.

La sentencia

El consejero del emperador pertenece a la nobleza o al pueblo.

es un caso de disyunción exclusiva, ya que se se interpreta que el consejero no puede pertenecer a la nobleza y al pueblo al mismo tiempo. Esta proposición se simboliza como

$$p \not\equiv q$$

donde las proposiciones elementales son

p : El consejero del emperador pertenece a la nobleza.

q : El consejero del emperador pertenece al pueblo.

En latín existen dos palabras diferentes para la disyunción inclusiva y exclusiva. La palabra "vel" se usa para la disyunción inclusiva mientras que para la exclusiva se usa la palabra "aut". El símbolo usado en lógica para la disyunción proviene precisamente de la inicial de la palabra latina.

6.3.2. Implicación y equivalencia

La implicación lógica suele representarse en el lenguaje natural con la construcción "si ... entonces ...". Es uno de los conectivos sobre los que menos acuerdo hay y al que más alternativas se han propuesto. Casi todo el mundo está de acuerdo en que cuando el antecedente p es verdadero y el consecuente q es falso, entonces $p \Rightarrow q$ es falsa. Por ejemplo, el caso de la afirmación

> Si se reforman las leyes laborales entonces bajará el desempleo.

Sin embargo existen ciertas dudas acerca del valor de verdad (o del significado lógico) de frases como

> Si dos más dos es igual a cinco, entonces yo soy el Papa.

Para la lógica clásica que estamos presentando aquí, la frase anterior es verdadera (mirando la tabla de verdad del $\Rightarrow$, vemos que si ambos argumentos son falsos entonces la implicación es verdadera).

Normalmente se usa una implicación con un consecuente obviamente falso como una manera elíptica de negar el antecedente. Por ejemplo decir

> Si la economía mejoró con esos ajustes, yo soy Gardel.

es una manera estilizada de decir que la economía no mejoró con esos ajustes. Otra forma de representar la implicación (y la consecuencia) en el lenguaje es a través de las palabras "suficiente" y "necesario". Por ejemplo la siguiente proposición:

> Es suficiente que use un preservativo para no contagiarme el VIH

puede simbolizarse con $p \Rightarrow \neg q$ donde las proposiciones simples son:

> p : Uso un preservativo. q : Me contagio el VIH.

Algunas veces la construcción lingüística "si ... entonces ... o si ..." no se traduce como una implicación sino como una equivalencia. En matemática es común presentar definiciones como

> Una función es biyectiva si es inyectiva y suryectiva.

lo cual podría en primera instancia traducirse como $p \wedge q \Rightarrow r$, donde p dice que "una función es inyectiva", q que "es suryectiva" y r que "es biyectiva". Si bien esa afirmación es indudablemente cierta, la definición matemática está diciendo en realidad que $p \wedge q \equiv r$. Obviamente si tengo una función biyectiva puedo asumir que es tanto inyectiva como suryectiva, lo cual no podría inferirse de la primer formalización como implicación.

Una alternativa en el lenguaje es usar la construcción "si y sólo si" para representar la equivalencia. De todas maneras no existe ninguna construcción del lenguaje que represente fielmente la equivalencia lógica.

Actividad

Escribí una oración en leguaje natural que pueda ser modelada con las siguientes fórmulas. Describí el significado que le asignás a cada variable.

1. $p \vee q$
2. $p \wedge q$
3. $p \Rightarrow q$
4. $p \equiv q$
5. $\neg p$

Actividad

Representá con una fórmula de lógica proposicional los siguientes enunciados. Usá el símbolo de predicado p para representar $(a < b)$, usá q para representar $(b < c)$, y r para $(a < c)$.

1. si a es menor que b y b es menor que c entonces a es menor que c"
2. $a < b < c$
3. Si $a < b$ entonces no es el caso que $(a \geq c)$.
4. $(a \geq b$ y $b < c)$ o $(a \geq c)$
5. No es el caso que $(a < b$ y $a < c)$

Actividad

Escribí una fórmula proposicional para cada una de las siguientes frases, utilizando una variable proposicional para cada sentencia atómica, aclarando siempre el significado escogido para cada variable. En lo posible utilizá un mismo símbolo de proposición para la misma sentencia atómica a lo largo de todo el ejercicio, así podés comparar las fórmulas.

1. Hoy es martes y hay sol.
2. Hoy es martes y no hay sol.
3. Hoy no es ni miércoles ni viernes.
4. Mañana, llueve o no llueve.
5. Mañana será jueves y no será jueves.
6. Llueve pero no hace frío.
7. Si Dios existe nos está mirando.
8. Yo no voy de vacaciones y Juan y Pedro tampoco.
9. Juan vendrá a clase, y seguro que vendrán también María o Pedro.
10. Santa Fe o Rosario deberían ser la capital de Santa Fe, pero Rosario es claramente más grande.
11. No es verdad que si tienes menos de 16 años y consentimiento paterno te puedas casar.

6.4. Análisis de razonamientos y su corrección

Ya hemos visto como traducir sentencias del lenguaje natural en formulas proposicionales de mayor o menos complejidad. Ocupémonos ahora de formalizar un razonamiento expresado en lenguaje natural.

Actividad

Retomá los razonamientos 1 y 3 de la primera actividad del capitulo e intenta expresarlos como una formula proposicional. Respecto al razonamiento 2 ¿Es posible capturarlo como una fórmula proposicional?

Una vez que sabemos como formalizar un razonamiento, estamos en condiciones de analizar su corrección. Hemos dicho que un razonamiento es correcto si suponiendo que las hipótesis en

conjunto son verdaderas, la conclusión también lo es. Cuando trabajamos con razonamientos formalizados, verificar su corrección se corresponde con analizar el valor de verdad de la fórmula que lo representa.

Un razonamiento del estilo "*hipótesis* 1 y *hipótesis* 2, por lo tanto *conclusión*", puede formalizarse como

$$(F_1 \wedge F_2) \Rightarrow F$$

donde F_1 es una fórmula para la hipótesis 1, F_2 es una fórmula para la hipótesis 2, y F es una fórmula que representa la conclusión. Normalmente hay una relación entre F y F_1, F_2, en el sentido que en F ocurren algunas de las variables proposicionales que también se utilizan para formalizar las hipótesis. Así el razonamiento "Si tuviera pelo sería feliz. No soy feliz. Entonces no tengo pelo", se puede formalizar como

$$(p \Rightarrow q \wedge \neg q) \Rightarrow \neg p$$

donde

p : Tengo pelo
q : Soy feliz

Analizar su corrección equivale a analizar el valor de verdad de la implicación $(p \Rightarrow q \wedge \neg q) \Rightarrow \neg p$, cuando las hipótesis en conjunto $p \Rightarrow q \wedge \neg q$ son verdaderas. Esta tarea se puede traducir en dos preguntas:

1. ¿en qué casos, es decir, para qué valores de verdad de p y q la fórmula $p \Rightarrow q \wedge \neg q$ es verdadera?
2. en esos casos anteriores, ¿$\neg p$ es verdadera?

El razonamiento anterior sería correcto si fueramos capaces de completar las incógnitas en la siguiente tabla de valores de verdad

p	q	$p \Rightarrow q \wedge \neg q$	$\neg p$	$(p \Rightarrow q \wedge \neg q) \Rightarrow \neg p$
?	?	verdadero	?	verdadero

Pero resolver esas incógnitas no es una tarea sencilla. Para que la conjunción de las hipótesis sea verdadera, todas ellas también deben serlo. Y $p \Rightarrow q$ es verdadera cuando p es verdadera y q es verdadera, así como también cuando p es falsa sin importar el valor de q. Pero $\neg q$ es verdadera cuando q es falsa. Por lo tanto sólo debemos considerar el caso en que tanto p como q son falsas. Como se puede verse, cuanto mayor sea el número de hipótesis, y mas complejas sean mas difícil resulta que no se nos escape ningún caso relevante en nuestro análisis

En el único caso que deducimos es relevante, la conclusión $\neg p$ es verdadera. Por lo tanto, la fórmula completa es verdadera, y asi podemos decir el razonamiento es correcto. Pero la forma de análisis parece mas compleja que el análisis intuitivo del razonamiento informal!

Por fortuna, un razonamiento formalizado siempre tiene una estructura fija. Y podemos valernos de algunas propiedades de los operadores lógicos para simplificar considerablemente su análisis, hasta el punto de obtener un procedimiento mecánico, una *receta* para analizar su corrección.

Si recordamos las tablas de verdad de la implicación y la conjunción

F_1	F_2	$F_1 \Rightarrow F_2$	$F_1 \wedge F_2$
verdadero	verdadero	verdadero	verdadero
falso	verdadero	verdadero	falso
verdadero	falso	falso	falso
falso	falso	verdadero	falso

podemos ver, por un lado, que en el caso en que la hipótesis sea falsa, la implicación es siempre verdadera. Por otro lado, cada vez que una hipótesis es falsa, la conjunción de ellas también lo es. Por lo tanto, la tarea de verificar que cuando las hipótesis en conjunto son verdaderas, la implicación

es verdadera, no se ve modificada si contemplamos también los casos en que las hipótesis son falsas. Esto nos permite analizar la corrección de un razonamiento armando una gran tabla que resuma el valor de verdad de cada una de las subfórmulas que aparecen en el razonamiento (hipótesis y conclusión), considerando todas las posibles combinaciones de los valores de verdad de cada una de las variables proposicionales involucradas. Si bien de este modo analizamos una serie de casos que son irrelevantes, estamos seguros de no dejar fuera ninguno de los casos que si son relevantes.

Para nuestro ejemplo, la tabla es la siguiente:

p	q	$p \Rightarrow q$	$\neg q$	$p \Rightarrow q \wedge \neg q$	$\neg p$	$(p \Rightarrow q \wedge \neg q) \Rightarrow \neg p$
verdadero	verdadero	verdadero	falso	falso	falso	verdadero
falso	verdadero	verdadero	falso	falso	verdadero	verdadero
verdadero	falso	falso	verdadero	falso	falso	verdadero
falso	falso	verdadero	verdadero	verdadero	verdadero	verdadero

El valor de verdad de la fórmula completa es verdadero en todos los casos, como era esperable. De esta manera, y sin necesidad de razonar intuitivamente ni analizar los casos relevantes, podemos deducir que el razonamiento es correcto.

Actividad

Analizá la corrección de los razonamientos formalizados en la actividad anterior.

Si bien este método para analizar razonamientos que presentamos, es efectivo para razonamientos con pocas variables proposicionales, a medida que la cantidad de variables crece, el tamaño de las tablas de verdad crece de manera exponencial (hay 2^n filas en una tabla de verdad con n variables proposicionales). Esta limitación práctica al uso de tablas de verdad no ocurre sólo para analizar razonamientos, pero en este caso es más obvio el problema dado que en los razonamientos suelen ser necesarias una gran cantidad de variables proposicionales.

En los capítulos siguientes presentamos métodos alternativos para resolver la validez de razonamientos a través de la simplificación de las fórmulas proposicionales que los representan.

Capítulo 7

Demostraciones en lógica propocicional

En el capítulo anterior hemos visto que, al formalizar razonamientos, las tablas de verdad se vuelven herramientas poco adecuadas cuando las fórmulas son muy largas.

Además, si tenemos que trabajar con una fórmula proposicional que, por ejemplo, especifique un programa que tenemos que escribir, todavía no contamos con un método para simplificar esa fórmula, como si lo tenemos para las expresiones numéricas. Esto es un punto sumamente importante, porque muchas veces poder simplificar una fórmula nos ayuda a comprender mejor la información que contiene.

En este capítulo desarrollaremos, entonces, métodos algebraicos para simplificar fórmulas proposicionales, lo que nos permitirá construir demostraciones formales de que algunas de nuestras fórmulas son teoremas. Hacerlo implicará un paso importante en la formalización del lenguaje que venimos construyendo desde que comenzamos a trabajar. Así, en este capítulo haremos explícitas muchas reglas e intuiciones que veníamos utilizando —y que utilizas desde la escuela secundaria— les vamos a poner un nombre y daremos, para cada una de ellas, un conjunto de reglas explícitas de utilización.

7.1. Sistemas formales

El primer concepto de vamos a terminar de formalizar es el de *sistema formal*, con el que venimos trabajando desde el inicio de este material.

El objetivo de un sistema formal es explicitar un lenguaje en el cual se realizarán demostraciones y las reglas para construirlas. Esto permite tener una noción muy precisa de lo que es una demostración, así también como la posibilidad de hablar con precisión de la sintaxis y la semántica de las expresiones que escribimos en este lenguaje.

Un sistema formal consta de cuatro elementos:

- Un conjunto de símbolos llamado *alfabeto*, a partir del cual las expresiones son construidas.
- Un conjunto de *expresiones bien formadas*, es decir aquellas palabras construidas usando los símbolos del alfabeto que serán consideradas correctas. Siempre es importante resaltar que una expresión bien formada no necesariamente es verdadera. Los términos «bien formada» se refieren sólo a la sintaxis de la expresión y no a su semántica.
- Un conjunto de *axiomas*, los cuales son las fórmulas básicas a partir de las cuales todos los teoremas se derivan.
- Un conjunto de *reglas de inferencia*, las cuales indican cómo derivar fórmulas a partir de otras ya derivadas.

En función de estos elementos pueden definirse una gran variedad de sistemas formales, cada uno adecuado para trabajar en diferentes problemas. Por ejemplo, Peano definió un sistema formal

a partir del cual se puede deducir toda la aritmética de los números naturales. Ese sistema es el que has estudiado en Matemática Discreta I.

En base a esta definición de sistema formal podemos notar que hasta aquí nosotros hemos establecido un alfabeto para trabajar con fórmulas proposicionales: en el capítulo anterior trabajamos con constantes, variables y operadores proposicionales. También nos hemos ocupado de dar reglas para determinar si una expresión está bien formada, particularmente cuando trabajamos con tipado de fórmulas proposicionales. Aquí, vamos a retomar estas dos ideas que ya veníamos trabajando y vamos a centrarnos en los dos aspectos que todavía no habíamos definido: los axiomas y las reglas de inferencia.

7.1.1. Nuestro sistema formal

Utilizando la definición de sistema formal dada anteriormente, presentemos a nuestro sistema formal.

Alfabeto. Consta de los siguientes elementos:

- **constantes:** Las constantes $True$ y $False$ que se usarán para denotar los valores verdadero y falso respectivamente; y las constantes $1, 2, 3, \ldots$ para denotar los valores númericos.
- **variables:** Se usarán típicamente las letras p, q ,r como variables proposicionales; y las letras $x, y, z, \ldots$ como variables númericas.
- **operadores:** Serán aquellos que presentamos en el capítulo anterior. Se dividen en:
  - **operadores unarios:** negación $\neg$.
  - **operadores binarios:** equivalencia $\equiv$, disyunción $\vee$, conjunción $\wedge$, discrepancia $\not\equiv$, implicación $\Rightarrow$, consecuencia $\Leftarrow$
- **signos de puntuación:** Paréntesis $'('$ y $')'$. Como lo hemos visto hasta ahora, los paréntesis serán los símbolos que nos permitan escribir fórmulas en donde se altere el orden dando por las reglas de precedencia.

Fórmulas. La *expresiones numéricas* bien formadas, serán las que se puedan construir a partir de las siguientes reglas:

1. Una constante numérica es una expresión numérica.
2. Una variable numérica es una expresion numérica.
3. Si E es una expresión numérica y $\oplus$ es un operador aritmético unario ($-$, ^{1}, ^{2},...), $\oplus E$ también lo es.
4. Si E_1, E_2 son expresiones numéricas y $\oplus$ es un operador aritmético binario ($+, -, *, /$), entonces $E_1 \oplus E_2$ también lo es.
5. Si E es una expresión numérica, (E) también lo es.

A partir de las expresiones numéricas se construyen las *fórmulas aritmética* utilizando los signos de relación según las siguientes reglas:

1. Si $\approx$ es una relación aritmética ($=, \not\equiv, <, >, \leqslant, \geqslant$), y E_1 y E_2 son expresiones numéricas, entonces $E_1 \approx E_2$ es una fórmula.
2. Si F es una fórmula aritmética, entonces (F) también lo es.

Las expresiones bien formadas o *fórmulas proposicionales* serán las que se puedan construir de acuerdo a las siguientes prescripciones:

1. Las variables proposicionales y las constantes son fórmulas.
2. Las fórmulas aritméticas son también fórmulas proposicionales.
3. Si F es una fórmula, entonces $\neg F$ también lo es.
4. Si F_1 y F_2 son fórmulas y $\oplus$ es un operador binario ($\equiv$, $\vee$, etc.), entonces $F_1 \oplus F_2$ también es una fórmula.
5. Si F es una fórmula proposicional, entonces (F) también lo es.

Es importante resaltar que el sistema formal, así como está enunciado, engloba todo nuestro trabajo realizado hasta aquí en relación con los números y las expresiones booleanas.

Actividad

Retomá las fórmulas proposicionales que escribiste en la primer actividad de la sección 6.1 y analizá a través de cuáles de estas reglas te permiten decir que es una expresión bien formada.

Axiomas. Iremos presentado gradualmente los axiomas para cada operador. Es importante comprender que, al igual que muchas de las propiedades que estudiaste en álgebra, los axiomas están escritos en términos de variables que pueden reemplazarse por otras expresiones más complejas —siempre manteniendo los tipos correspondientes y siendo coherente, es decir, si en una parte del axioma reemplazo una variable por la fórmula $p \vee q$ entonces en todas las demás ocurrencias de esa variable debo hacer el mismo reemplazo. Cuando hacemos estor reemplazos decimos que estamos *instanciando* una regla o axioma.

Por ejemplo, la propiedad algebraica conocida como diferencia de cuadrados establece que

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

puede instanciarse de la siguiente manera

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

donde se a reemplazado a la variable a por otra variable (x) y a la variable b por la constante 2

Lo mismo sucede con los axiomas de los operadores proposicionales, tomemos el caso del axioma de la conmutatividad de la diyunción. El mismo establece que:

$$P \vee Q \equiv Q \vee P$$

Este axioma puede instanciarse, por ejemplo, de la siguiente manera:

$$2 > 1 \vee (p \wedge r) \equiv (p \wedge r) \vee 2 > 1$$

Donde se ha reemplazado a la variable P por la fórmula $2 > 1$ y a la variable Q por la fórmula $(p \wedge r)$.

Para dar mejor la idea de que nuestros axiomas pueden instanciarse de maneras múltiples según nuestra conveniencia siempre los enunciaremos con letras mayúsculas (P, Q, R, etc.)

Actividad

Descubrí por qué fórmulas se han reemplazado las variables de la primer columna para obtener las fórmulas de la segunda columna:

$P \vee Q \equiv Q \vee P$	$(p \wedge q) \vee (p \vee q) \equiv (p \vee q) \vee (p \wedge q)$
$P \vee (Q \equiv R) \equiv (P \vee Q) \equiv (P \vee R)$	$(q \vee q) \vee (p \equiv \neg r) \equiv (p \vee (q \vee q)) \equiv (p \vee \neg r)$
$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$	$False \wedge (r \wedge (p \Rightarrow p)) \equiv (False \wedge p) \wedge (p \Rightarrow p)$
$P \wedge True \equiv P$	$(p \equiv r) \wedge True \equiv (p \equiv r)$

Cada axioma o teorema nos habilita a reescribir una expresión de diversas maneras. Por ejemplo la Regla Dorada, cuya formulaciónn es

$$P \wedge Q \equiv P \equiv Q \equiv P \vee Q$$

nos permite reescribir la expresión $P \wedge Q$ por $P \equiv Q \equiv P \vee Q$, pero también:

$P \wedge Q \equiv P \;\; por \;\; Q \equiv P \vee Q$

$Q \equiv P \vee Q \;\; por \;\; P \wedge Q \equiv P$

$P \wedge Q \equiv P \vee Q \;\; por \;\; P \equiv Q$

$P \wedge Q \equiv Q \equiv P \vee Q \;\; por \;\; P$

$P \equiv P \vee Q \;\; por \;\; P \wedge Q \equiv Q$

etc...

Reglas de inferencia. Vamos a trabajar con dos reglas de inferencia. En primer lugar, la *transitividad de la equivalencia*, que establece que si la fórmula $F1$ es equivalente a la fórmula F_2 y a su vez la fórmula F_2 es equivalente a la fórmula F_3, entonces la fórmula F_1 es equivalente a la fórmula F_3. Esta es la regla que nos permite realizar varios pasos de demostración y concluir que la primer expresión es equivalente a la última.

En segundo lugar, la *Regla de Leibnitz* que es útil cuando queremos reemplazar parte de una fórmula por otra dentro de una fórmula mayor, es decir, dentro de un contexto. Esta regla la has utilizado una innumerable cantidad de veces cuando estabas en la secundaria y resolvías ejercicios combinados en matemática. Tanto la equivalencia y la igualdad cumplen con esta regla. La misma establece para la equivalencia que si sabemos que dos fórmulas son equivalentes entre sí, por ejemplo $F_1 \equiv F_2$ entonces podemos reemplazar F_1 por F_2 en un contexto más complejo sin cambiar el valor de verdad de la fórmula como un todo. Por ejemplo, si se cumple $p \vee p \equiv p$, se puede transformar la fórmula $p \wedge q$ a la fórmula $(p \vee p) \wedge q$ donde se reemplazó p por otra expresión equivalente, $p \vee p$.

Actividad

Descubrí las dos expresiones equivalentes entre sí, que fueron utilizadas para transformar la primera expresión en la última y escribíla entre las llaves. Subrayá la parte de la fórmula que está siendo modificada en el primer renglón.

1. $p \vee q$
   $\equiv \{ \quad \}$
   $(p \equiv True) \vee q$

2. $p \Rightarrow q$
   $\equiv \{ \quad \}$
   $(p \equiv True) \Rightarrow q$

3. $p \vee (q \equiv r) \equiv q \vee \neg r$
   $\equiv \{ \quad \}$
   $p \vee q \equiv p \vee r \equiv q \vee \neg r$

4. $q \Rightarrow (p \vee q)$
   $\equiv \{ \quad \}$
   $q \Rightarrow (q \vee p)$

7.1.2. Demostraciones y estrategias de demostración

Una *demostración* dentro de nuestro sistema formal consistirá en probar la **validez** de una fórmula mediante una serie de pasos justificados con **axiomas** y **teoremas** ya demostrados.

Recordemos que una fórmula es válida si para toda asignación posible de las variables es equivalente a *True*; por lo tanto una demostración será una serie de fórmulas, equivalentes entre sí, donde la primera fórmula es la que queremos demostrar válida y la última es *True*.

El formato de demostración que utilizaremos será el mismo que venimos utilizando hasta el momento. El operador que utilizaremos para "unir" un paso con el otro será la equivalencia porque las fórmulas que manipularemos serán fórmulas proposicionales. En términos generales, si queremos demostrar que la fórmula F_1 es un teorema la demostración que construiremos tendrá la siguiente forma:

$$
\begin{array}{ll}
& F_1 \\
\equiv & \{ \texttt{Razón 1} \} \\
& F_2 \\
\equiv & \{ \texttt{Razón 2} \} \\
& F_3 \\
\equiv & \{ \texttt{Razón 3} \} \\
& \mathit{True}
\end{array}
$$

Esta demostración se puede leer de la siguiente manera: Debido a la `Razón 1` F_1 es equivalente a F_2; debido a la `Razón 2` F_2 es equivalente a F_3, y debido a la `Razón 3` F_3 es equivalente a *True*. Por lo tanto, como el equivalente es transitivo, se concluye que F_1 es equivalente a *True*.

Cada paso de la demostración consiste en "reescribir" la fórmula que estamos manipulando o una parte de ella en otra equivalente dada por uno de los Axiomas o Teoremas ya demostrados.

Por supuesto que este ejemplo de tres pasos se puede generalizar a cualquier cantidad necesaria de pasos.

Veamos un ejemplo particular de la aritmética que utiliza en el último paso uno de los axiomas de la disyunción: el tercero excluido. Este axioma será presentado en las secciones siguientes, pero básicamente establece que la disyunción de una proposición y su negación es siempre verdadera. Así, es verdad que "es de día o no es de día" y que "el sol está ardiendo o no está ardiendo". La fórmula que corresponde a este axioma es $P \vee \neg P \equiv True$. Demostremos, usando estas herramientas, que un número es mayor o igual que cero o que es menor que cero:

$$
\begin{array}{ll}
& (x > 0) \vee (x \leqslant 0) \\
\equiv & \{ \text{ Aritmética } \} \\
& (x > 0) \vee \neg(x > 0) \\
\equiv & \{ \text{ Tercero excluido } (P \vee \neg P \equiv True) \} \\
& True
\end{array}
$$

En el primer paso de la demostración hemos "reescrito" parte de la primera fórmula —$(x \leqslant 0)$— utilizando la definición aritmética del operador menor o igual y el operador negación. Esto nos permite transformar la subfórmula $(x \leqslant 0)$ en $\neg(x > 0)$. El resto de la fórmula no fue transformada en el primer paso.

El segundo paso consiste en aplicar el axioma del tercero excluido reemplazando a la variable P por la fórmula $(x > 0)$. Esto nos permite reescribir la disyunción con la que veníamos trabajando como $True$.

Estrategias de demostración. Cuando la fórmula que queremos demostrar es de la forma $P \equiv Q$ tendremos dos estrategias de prueba posibles.

En primer lugar, podremos seguir la estrategia planteada anteriormente transformando la fórmula completa en *True*. Una segunda estrategia será partir de la subfórmula P y transformarla en la subexpresión Q (o viceversa). Esta estrategia tendrá, entonces, el siguiente esquema:

$$
\begin{array}{ll}
& P \\
\equiv & \{ \text{ Justificación de } P \equiv P_1 \} \\
& P_1 \\
\equiv & \{ \text{ Justificación de } P_1 \equiv P_2 \} \\
& P_2 \\
& \vdots \\
\equiv & \{ \text{ Justificación de } P_{n-1} \equiv P_n \} \\
& P_n \\
\equiv & \{ \text{ Justificación de } P_n \equiv Q \} \\
& Q
\end{array}
$$

¿Cómo podemos garantizar que esta estrategia de prueba es equivalente a la primera? Esto puede lograrse transformando la demostración anterior en una que utilice la otra estrategia haciendo uso de las mismas justificaciones y de la regla de Leibnitz:

$$
\begin{array}{ll}
 & P \equiv Q \\
\equiv & \{ \text{ Justificación } P \equiv P_1 \ \} \\
 & P_1 \equiv Q_1 \\
\equiv & \{ \text{ Justificación } P_1 \equiv P_2 \ \} \\
 & P_2 \equiv Q_2 \\
 & \vdots \\
\equiv & \{ \text{ Justificación } P_{n-1} \equiv P_n \ \} \\
 & P_n \equiv Q_n \\
\equiv & \{ \text{ Justificación } P_n \equiv Q \ \} \\
 & Q \equiv Q \\
\equiv & \{ \text{ Reflexividad } \} \\
 & True
\end{array}
$$

Notá que la regla de Leibnitz es la que nos permite, en cada paso, transformar parte de la fórmula (por ejemplo, P en P_1 en el primer paso) dentro de un contexto que no se modifica (en el caso del primer paso el contexto sería $\equiv Q$).

A continuación presentaremos los Axiomas que corresponden a cada operador lo que nos permitirá comenzar a realizar demostraciones en nuestro sistema formal utilizando EQU .

7.2. Equivalencia

La equivalencia se define en términos formales como el operador binario que satisface los siguientes axiomas:

Asociatividad: $((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$

Conmutatividad: $(P \equiv Q) \equiv (Q \equiv P)$

Neutro de la equivalencia: $(P \equiv True) \equiv P$

Es importante notar que la asociatividad de la equivalencia permite la omisión de los paréntesis en los dos axiomas siguientes, aquí se han incluido sólo con la finalidad de hacer más fácil la comprensión intuitiva de cada axioma. Como además la equivalencia es conmutativa será irrelevante el orden de los términos en una equivalencia. Así, los axiomas de la conmutatividad y del neutro de la equivalencia pueden escribirse directamente como:

Conmutatividad

$$(P \equiv Q) \equiv (Q \equiv P)$$

Neutro de la equivalencia

$$(P \equiv True) \equiv P$$

Es importante notar que la asociatividad de la equivalencia permite la omisión de los paréntesis en los dos axiomas siguientes, aquí se han incluido sólo con la finalidad de hacer más fácil la comprensión intuitiva de cada axioma. Como además la equivalencia es conmutativa será irrelevante el orden de los términos en una equivalencia. Así, los axiomas de la conmutatividad y del neutro de la equivalencia pueden escribirse directamente como:

$$P \equiv Q \equiv Q \equiv P$$

y

$$P \equiv True \equiv P$$

y este último axioma puede utilizarse, por ejemplo, para reemplazar $p \equiv p$ por $True$.

Actividad

Descubrí qué axioma se ha utilizado para realizar cada uno de las siguientes transformaciones. Anotá también por qué fórmulas se ha reemplazado a cada variable del axioma:

1. $q \equiv True$
   $\equiv \{ \quad \}$
   q
2. $((q \equiv \neg r) \equiv p \vee q)$
   $\equiv \{ \quad \}$
   $(q \equiv (\neg r \equiv p \vee q))$
3. $False \equiv p \wedge \neg s \equiv p \wedge \neg s$
   $\equiv \{ \quad \}$
   $False$

Actividad

Demostrá usando EQU los siguientes teoremas:

1. Reflexividad: $p \equiv p$. Notá que hemos utilizado numerosas veces este teorema y ahora será la primera vez que lo demostraremos formalmente.
2. $True$

7.3. Negación

La negación es el operador unario que cumple los siguientes axiomas:

Negación y equivalencia: $\neg(P \equiv Q) \equiv \neg P \equiv Q$

Definición de $False$: $False \equiv \neg True$

El primer axioma nos permite manipular fórmulas que contengan equivalencias y negaciones. Notá que la negación no es "distributiva" respecto a la equivalencia, la negación sólo se aplica a una de las subfórmulas involucradas.

El segundo axioma define a la constante $False$ que veníamos utilizando, en términos de la constante $True$.

Actividad

Demostrá utilizando los axiomas introducidos hasta ahora los siguientes teoremas:

1. Doble negación: $\neg\neg p \equiv p$
2. Contrapositiva: $p \equiv \neg p \equiv False$

7.4. Discrepancia

La discrepancia se define como el operador binario que satisface el siguiente axioma:

Definición: $P \not\equiv Q \equiv \neg(P \equiv Q)$

La precedencia de la discrepancia es la misma que la de la equivalencia. Notá que este único axioma con el que definimos a la discrepancia nos permite reescribirla en términos de los operaciones negación y equivalencia, para los cuales ya tenemos un conjunto de axiomas. Para demostrar propiedades de un operador nuevo definido en términos de otros (como en este caso la discrepancia en términos de la negación y la equivalencia) un método frecuentemente exitoso es reemplazar al operador por su definición y manipular los operadores "viejos", volviendo a obtener el operador nuevo si es necesario.

Actividad

Descubrí qué axioma de los que se introdujeron hasta ahora se ha utilizado para realizar cada uno de las siguientes transformaciones. Anotá también por qué fórmulas se ha reemplazado a cada variable del axioma:

1. $\neg q \equiv p$
   $\equiv \{ \quad \}$
   $\neg(q \equiv p)$

2. $\neg(r \vee p \equiv s \vee True))$
   $\equiv \{ \quad \}$
   $r \vee p \not\equiv s \vee True$

3. $\neg r$
   $\equiv \{ \quad \}$
   $\neg q \equiv \neg(r \equiv \neg q)$

Actividad

Utilizando el axioma de definición de la discrepancia y los que corresponden a la equivalencia y la negación demostrá los siguientes teoremas:

1. Asociatividad de la discrepancia: $((p \not\equiv q) \not\equiv r) \equiv (p \not\equiv (q \not\equiv r))$
2. Conmutatividad de la discrepancia: $(p \not\equiv q) \equiv (q \not\equiv p)$
3. Asociatividad mutua entre la discrepancia y la equivalencia: $p \equiv (q \not\equiv r) \equiv (p \equiv q) \not\equiv r$
4. ¿Cómo nos permite reescribir el último teorema la siguiente fórmula: $((r \equiv s) \not\equiv t) \equiv (s \not\equiv r)$?

Actividad

Demostrá los siguientes teoremas:

1. Neutro de la discrepancia: $p \not\equiv false \equiv p$
2. Intercambiabilidad: $p \equiv q \not\equiv r \equiv p \not\equiv q \equiv r$. Para contrastar, construí la tabla de verdad para demostrar que esta forma es un teorema.

Actividad

Decidí si son válidas o no las siguientes fórmulas. Justificá apropiadamente. Podés usar EQU tanto para evaluar como para demostrar las fórmulas:

1. $p \equiv p \equiv p \equiv True$
2. $((p \not\equiv q) \equiv r) \equiv (p \not\equiv (q \equiv r))$
3. $(p \equiv q) \equiv (\neg p \equiv \neg q)$
4. $\neg p \equiv False$
5. $\neg(p \equiv q) \equiv (\neg p \equiv \neg q)$

Actividad

Inventá dos fórmulas que utilicen sólo $\equiv$, $\not\equiv$ y $\neg$ una válida (teorema) y una no válida.

7.5. Disyunción

La disyunción es el operador binario definido por los siguientes axiomas:

Asociatividad: $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$

Conmutatividad: $P \vee Q \equiv Q \vee P$

Idempotencia: $P \vee P \equiv P$

Distributividad con la equivalencia: $P \vee (Q \equiv R) \equiv (P \vee Q) \equiv (P \vee R)$

Tercero excluido: $P \vee \neg p \equiv True$

Actividad

Descubrí qué axioma se ha utilizado para realizar cada una de las siguientes transformaciones. Anotá también por qué fórmulas se ha reemplazado a cada variable del axioma:

1. $r \vee (p \vee (p \Rightarrow r))$
   $\equiv \{ \quad \}$
   $(r \vee p) \vee (p \Rightarrow s)$
2. $3x \leqslant 2y$
   $\equiv \{ \quad \}$
   $(3x \leqslant 2y) \vee (3x \leqslant 2y)$
3. $True$
   $\equiv \{ \quad \}$
   $True \vee \neg True$

Actividad

Demostrá los siguientes teoremas sobre la disyunción:

1. $p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$
2. Elemento absorbente de la disyunción: $p \vee True \equiv True$
3. Elemento neutro de la disyunción: $p \vee False \equiv p$

Actividad

En el siguiente paso de demostración se han aplicado varios axiomas simultáneamente:

$$\begin{array}{ll} & (p \vee q \equiv p \equiv q) \vee p \equiv q \\ \equiv & \{ \text{ Distributividad de } \vee \text{ con } \equiv, \text{ idempotencia de } \vee, \text{ neutro de } \equiv \} \\ & True \end{array}$$

Teniendo en cuenta que esta sucesión de aplicaciones de axiomas nos permitirá simplificar muchos términos en las demostraciones que realizaremos a continuación, desarrolla en detalle este paso.

7.6. Conjunción

La conjunción es un operador binario con la misma precedencia que la disyunción, lo cual hace necesario el uso de paréntesis en las expresiones que involucran a ambos operadores. Por ejemplo: no es lo mismo $(p \vee q) \wedge r$ que $p \vee (q \wedge r)$.

Introduciremos un único axioma para definir conjunción:

Regla dorada: $P \wedge Q \equiv P \equiv Q \equiv P \vee Q$

La regla dorada aprovecha fuertemente la asociatividad de la equivalencia. En principio, la interpretaríamos como $P \wedge Q \equiv (P \equiv Q \equiv P \vee Q)$, pero nada nos impide hacer otras interpretaciones, por ejemplo:

$$(P \wedge Q \equiv P) \equiv (Q \equiv P \vee Q)$$

, o bien

$$(P \wedge Q \equiv P \equiv Q) \equiv (P \vee Q)$$

, o bien, usando conmutatividad de la equivalencia,

$$(P \equiv Q) \equiv (P \wedge Q \equiv P \vee Q)$$

, etcétera.

La regla dorada nos será de gran utilidad para demostrar propiedades de la conjunción y la disyunción, dado que provee una relación entre ambas.

Actividad

En las siguientes transformaciones se ha utilizado la **Regla Dorada** para llegar de la primer fórmula a la segunda. Encontrá por qué fórmulas han reemplazado a las variables del axioma en cada caso:

1. $(r \Rightarrow z \vee q) \wedge (r \equiv s) \equiv (r \Rightarrow z \vee q) \equiv (r \equiv s)$
   $\equiv \{$ Regla Dorada $\}$
   $(r \Rightarrow z \vee q) \vee (r \equiv s)$

2. $(z \wedge r) \vee (m \Rightarrow n \vee r) \equiv (z \wedge r) \equiv (m \Rightarrow n \vee r)$
   $\equiv \{$ Regla Dorada $\}$
   $(z \wedge r) \wedge (m \Rightarrow n \vee r)$

3. $(p \vee (q \equiv r)) \wedge (m \Rightarrow n)$
   $\equiv \{$ Regla Dorada $\}$
   $(p \vee (q \equiv r)) \vee (m \Rightarrow n) \equiv (p \vee (q \equiv r)) \equiv (m \Rightarrow n)$

4. $(p \vee q \vee r) \vee (s \vee t \vee z)$
   $\equiv \{$ Regla Dorada $\}$
   $(p \vee q \vee r) \wedge (s \vee t \vee z) \equiv (p \vee q \vee r) \equiv (s \vee t \vee z)$

5. $((p \vee q) \Rightarrow r) \equiv z \wedge r$
   $\equiv \{$ Regla Dorada $\}$
   $((p \vee q) \Rightarrow r) \vee z \wedge r \equiv ((p \vee q) \Rightarrow r) \wedge z \wedge r$

6. $p \Rightarrow q$
   $\equiv \{$ Regla Dorada $\}$
   $(p \Rightarrow q) \vee z \equiv z \equiv p \Rightarrow q \wedge z$

Actividad

Demostrá que las siguientes fórmulas son teoremas, justificando cada paso con el axioma aplicado.

1. *Asociatividad de la conjunción*: $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
2. *Conmutatividad de la conjunción*: $p \wedge q \equiv q \wedge p$
3. *Idempotencia de la conjunción*: $p \wedge p \equiv p$
4. *Elemento absorbente de la conjunción*: $p \wedge \mathit{False} \equiv \mathit{False}$
5. *Elemento neutro de la conjunción*: $p \wedge \mathit{True} \equiv p$
6. *Distributividad de la disyunción con la conjunción*: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
7. *Teorema Estrella* : $p \vee q \equiv p \vee \neg q \equiv p$

Actividad

Demostrá que las siguientes fórmulas son teoremas justificando cada paso con el axioma o teorema aplicado. **Aclaración:** Desde ahora en adelante, en cada ejercicio se pueden utilizar los teoremas ya demostrados en los ejercicios anteriores.

1. *De Morgan para la disyunción*: $\neg(p \vee q) \equiv \neg p \wedge \neg q$
2. *De Morgan para la conjunción*: $\neg(p \wedge q) \equiv \neg p \vee \neg p$
3. *Ley de absorción*: $p \wedge (p \vee q) \equiv p$
4. *Ley de absorción (bis)*: $p \vee (p \wedge q) \equiv p$

Actividad

Decidí si cada una de las siguientes fórmulas proposicionales son válidas o no. En todos los casos justificá con una demostración o un contraejemplo, según corresponda.

1. $p \wedge (q \equiv r) \equiv (p \wedge q) \equiv (p \wedge r)$
2. $p \wedge (q \equiv r) \equiv (p \wedge q) \equiv (p \wedge r) \equiv p$
3. $p \wedge (q \equiv r \equiv s) \equiv (p \wedge q) \equiv (p \wedge r) \equiv (p \wedge s)$
4. $\big(a \vee b \vee c \equiv a \vee b \equiv a \vee c \equiv b \vee c \equiv \mathit{False}\big) \equiv \big((a \equiv b \equiv c) \wedge \neg(a \wedge b \wedge c)\big)$

7.7. Implicación

La implicación es uno de los mecanismos más intuitivos para razonar, ya que de esta forma representamos la **causalidad**. De la misma forma que la causalidad no es simétrica, tampoco lo es la implicación. Es decir, los dos elementos que componen una implicación no participan de igual forma en la relación. Por ejemplo, cuando como demasiado (causa) me duele la panza (efecto), pero no a la inversa (cuando me duele la panza no necesariamente es porque he comido demasiado, puede ser porque estoy nervioso). El elemento a la izquierda de la implicación se llama **antecedente**, y el de la derecha, **consecuente**. Esta asimetría también se pone de manifiesto en la tabla de verdad de la implicación:

p	q	p ⇒ q
True	True	True
True	False	False
False	True	True
False	False	True

También las diferentes formas de reescribir la implicación nos muestran como sus dos elementos participan de forma bien distinta en la relación. Las dos definiciones más comunes de la implicación son las siguientes:

Definición de implicación: $P \Rightarrow Q \equiv P \vee Q \equiv Q$

Caracterización de la implicación: $P \Rightarrow Q \equiv \neg P \vee Q$

En estas dos fórmulas puede notarse cómo el antecedente y el consecuente tienen reescrituras bien distintas. Consideraremos a la primera de las dos fórmulas anteriores como el axioma de la implicación y demostraremos la segunda de ellas como teorema.

Actividad

Demostrá que las siguientes fórmulas son teoremas, justificando cada paso con el axioma o teorema aplicado.

1. *Caracterización de implicación*: $p \Rightarrow q \equiv \neg p \vee q$
2. *Definición dual de implicación*: $p \Rightarrow q \equiv p \wedge q \equiv p$
3. *Negación de una implicación*: $\neg(p \Rightarrow q) \equiv p \wedge \neg q$
4. *Absurdo*: $p \Rightarrow \mathit{False} \equiv \neg p$.
5. *Debilitamiento para* $\wedge$: $p \wedge q \Rightarrow p$.
6. *Debilitamiento para* $\vee$: $p \Rightarrow p \vee q$.
7. *Modus Ponens* $(p \Rightarrow q) \wedge p \equiv p \wedge q$.
8. *Modus Tollens* $(p \Rightarrow q) \wedge \neg q \equiv \neg p \wedge \neg q$.

Actividad

Decidí si cada una de las siguientes fórmulas proposicionales son válidas o no. Justificá apropiadamente.

1. $(p \Rightarrow q) \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$
2. $p \vee (p \Rightarrow q) \equiv \mathit{True}$.
3. $p \wedge (q \Rightarrow p) \equiv p$.
4. $p \vee q \Rightarrow q$.
5. $p \wedge q \Rightarrow q$.
6. $p \Rightarrow p \vee q$.
7. $p \Rightarrow p \wedge q$.
8. $\mathit{False} \Rightarrow p \equiv p$.
9. $\mathit{False} \Rightarrow p \equiv \mathit{True}$.
10. $\mathit{True} \Rightarrow p \equiv p$.
11. $\mathit{True} \Rightarrow p \equiv \mathit{True}$.
12. $p \vee (q \Rightarrow p) \equiv q \Rightarrow p$.
13. $(p \Rightarrow p') \wedge (p \equiv q) \Rightarrow (p' \equiv q)$.
14. $(p \wedge q \Rightarrow r) \equiv (p \Rightarrow \neg q \vee r)$.

Actividad

Ejercicios Extra: Demostrá que las siguientes fórmulas son teoremas, justificando cada paso con el axioma o teorema aplicado.

1. *Currificación*: $p \Rightarrow (q \Rightarrow r) \equiv p \wedge q \Rightarrow r$.
2. *Contrarrecíproca*: $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$.
3. *Distributividad a izquierda de* $\Rightarrow$ *con* $\equiv$: $p \Rightarrow (q \equiv r) \equiv p \Rightarrow q \equiv p \Rightarrow r$.
4. *Doble implicación*: $(p \Rightarrow q) \wedge (q \Rightarrow p) \equiv p \equiv q$.
5. *Transitividad*: $(p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$.
6. *Monotonía conjunción*: $(p \Rightarrow q) \Rightarrow (p \wedge r \Rightarrow q \wedge r)$.
7. *Monotonía disjunción*: $(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$.

7.8. Consecuencia

La consecuencia es un operador binario que tiene la misma precedencia que la implicación y que satisface el siguiente axioma:

Definición de consecuencia: $P \Leftarrow Q \equiv P \vee Q \equiv P$

Actividad

Descubrí qué axioma se ha utilizado para realizar cada una de las siguientes transformaciones. Anotá también por qué fórmulas se ha reemplazado a cada variable del axioma:

1. $s \vee r \equiv s$
   $\equiv \{ \quad \}$
   $s \Leftarrow r$

2. $(s \vee s) \equiv ((p \Rightarrow q) \Rightarrow (s \vee s))$
   $\equiv \{ \quad \}$
   $(p \Rightarrow q) \vee (s \vee s)$

Actividad

Demostrá el siguiente teorema:

1. $p \Leftarrow q \equiv q \Rightarrow p$
2. Este teorema establece que la consecuencia es el operador *dual* de la implicación. En base a este teorema y a los teoremas de la implicación escribí tres teoremas para la consecuencia.

7.9. ¿Por qué trabajar de esta manera?

Esta sección está dedicada a reflexionar acerca de las características de nuestro sistema formal y de las ventajas que nos trae trabajar en él. Para ello te proponemos las siguientes actividades:

Actividad

El siguiente es un fragmento de un texto que escribió E.W. Dijktra, un científico que fue muy influyente en la fundación de las ciencias de la computación:

> la sentencia de un teorema es en esencia una expresión booleana, y su prueba es en esencia el cálculo al evaluar esa expresión booleana al valor $True$. La forma más directa de evaluar una expresión booleana es someterla a una o más transformaciones que preserven su valor hasta que se alcance $True$. La razón de que un número de transformaciones deban ser necesarias radica en que deseamos confinarlas a manipulaciones a partir de un *repertorio restringido*.
>
> Denotemos con $[A]$ una expresión booleana a ser evaluada, y digamos que la evaluación toma, para decir algo, 3 pasos, que involucran la ocurrencia de 2 resultados intermedios. Más precisamente, para algunas expresiones $[B]$ y $[C]$:
>
> el primer paso establecerá $[A] \equiv [B]$;
>
> el segundo paso establecerá $[B] \equiv [C]$;
>
> el tercer paso establecerá $[C] \equiv True$;
>
> entonces todo el cálculo establecerá que $[A] \equiv True$.
>
> *Traducción libre de Our proof format de Edger W. Dijktra - EWD999-0*

- ¿A qué te parece que hace referencia Dijktra cuando habla de *repertorio restringido*?
- ¿Qué opinás de la siguiente demostración?

$$\begin{array}{ll} & (x = 3 \Rightarrow y = 5) \wedge (x + y = z \equiv x = 3 \equiv x + y = z \vee x = 3) \\ \equiv \{ & \text{propiedades de la lógica y la aritmética } \} \\ & x = 3 \wedge y = 5 \wedge z = 8 \end{array}$$

Al ver la demostración de la actividad anterior todos nos preguntamos: ¿qué querrá decir el que la escribió con "propiedades de la lógica y la aritmética"?.

Tampoco se podría decir que se entiende cuál es el argumento principal de la demostración, ni los supuestos utilizados para tal fin. Tal vez quién escribió esa demostración, consideró que era *trivial* esa propiedad, pero no hay dudas de que falló en la instancia de comunicar y convencernos de que realmente la expresión $(x = 3 \Rightarrow y = 5) \wedge (x + y = z \equiv x = 3 \equiv x + y = z \vee x = 3) \equiv x = 3 \wedge y = 5 \wedge z = 8$ es válida, es decir, es verdadera para todo valor posible de x ,y y z.

A partir de esta reflexiones podemos comenzar a pensar a una demostración como un argumento ordenado para convencer a los otros de que la sentencia que afirmamos es válida. Para ello, esta comunicación debería en primera instancia utilizar un lenguaje compartido por todos; y por otro lado, debería desdoblar todos los pasos necesarios para una correcta comprensión.

Actividad

Demostrá con tus compañeros la propiedad **diferencia de cuadrados**: $a^2 - b^2 = (a + b)(a - b)$. Identifiquen cuáles serán los elementos que conforman el *lenguaje compartido* al que se hace referencia en el párrafo anterior que utilizarán para construir la demostración.

Actividad

En la actividad anterior detectamos que para poder demostrar diferencia de cuadrados debemos utilizar como justificación la distributividad del producto con la suma. Entonces discutí con tus compañeros cómo se podría demostrar esta propiedad (i.e. $a * (b + c) = a * b + a * c$)? ¿y cómo se demuestran las propiedades que sean necesarias para demostrar la misma?¿dónde termina?

Estas actividades nos plantean la necesidad de contar, para construir demostraciones, con un conjunto de reglas y propiedades que podamos utilizar pro que no requieran demostración. Estas propiedades son, como ya lo dijimos, *axiomas*. Para el caso de nuestro sistema formal en las secciones anteriores hemos presentado los axiomas correspondientes a cada operador proposicional.

Actividad

Ahora bien, hemos acordado sobre la necesidad de establecer axiomas; pero ahora nos surgen las siguientes preguntas: ¿cuáles de todas las propiedades que conocemos serán elegidas como axiomas? ¿con qué criterios los elegiremos? y, pensando en el sistema formal que hemos venido presentando: ¿por qué escoger esos axiomas para los operadores y no otros? leé el siguiente texto y discutí con tus compañeros y tu docente estas respuestas para las preguntas anteriores

> *Los criterios generales para elegir los axiomas son:* consistencia*,* independencia *y* completitud*.*
>
> *Un conjunto de axiomas se llama* inconsistente *o contradictorio si una sentencia válida y su contradicción son ambas demostrables a partir de los axiomas. El criterio de la consistencia es el que posee la mayor importancia práctica: un sistema será escasamente útil si posee como teoremas sentencias contradictorias.*
>
> *Un conjunto de axiomas se llama* independiente *si ningún axioma se puede demostrar a partir del resto de los axiomas. La forma usual de establecer que un axioma es dependiente es construir una derivación de dicho axioma. Cuando esto sucede el axioma redundante, o posiblemente otro axioma, simplemente cambia de estatus y se vuelve un teorema.*
>
> *Un conjunto de axiomas se dice* completo *si cualquier sentencia válida (es decir, cualquier sentencia formulada de acuerdo con las reglas de formación del sistema) puede ser o demostrada o refutada a partir de los axiomas. Determinar si un sistema axiomático es completo es de gran interés para los matemáticos y lógicos. El propósito de asignarle a una teoría matemática o lógica la forma de un sistema axiomático es, de hecho, darle a la teoría la estructura en la cual todas las sentencias verdaderas sean derivables de un sólo conjunto de axiomas. Si el sistema axiomático no puede ser mostrado completo, el valor de la axiomatización es cuestionable.*
>
> *Traducción libre de Hanna, G. (1983) Rigorous proof in mathematics education. págs: 36-42*

Capítulo 8

Lógica de predicados

En el capítulo 6 analizamos la corrección de cierto tipo de razonamientos a partir de la construcción de las tablas de verdad de las proposiciones que formalizaban dichos razonamientos. En el capítulo 7 dimos una definición formal de nuestro sistema, agregando los axiomas que corresponden a cada operador proposicional y estudiando diversas estrategias para demostrar teoremas sobre determinadas expresiones proposicionales. Todo este trayecto nos permitió explar la potencia que posee la lógica proposicional para resolver ciertos tipos de problemas.

Ahora bien, la lógica proposicional muestra sus limitaciones cuando es necesario razonar sobre enunciados del tipo *'todo tiene la propiedad p'* o *'algo tiene la propiedad p'*. Un ejemplo clásico de esta situación, que ya se conocía en la antigua Grecia, es el siguiente razonamiento lógico que también discutimos al comenzar el capítulo 6:

> *Todos los hombres son mortales. Sócrates es un hombre. Por lo tanto, Sócrates es mortal.*

La lógica proposicional no es lo suficientemente expresiva como para trabajar con este tipo de enunciados. Veamos por qué. Podríamos definir una variable proposicional p que signifique *'todos los hombres son mortales'* y otra, q que signifique *'Sócrates es mortal'*. Ahora bien, el razonamiento al que nos enfrentamos requiere que podamos establecer una relación entre que todos los hombres son mortales y que Sócrates es mortal, es decir, una relación entre la variable p y la variable q. Ninguno de los operadores proposicionales es capaz de capturar esta relación.

La **lógica de predicados** es una extensión de la lógica proposicional que agrega dos **cuantificadores** que permiten que ejemplos como el razonamiento de Sócrates sean expresados. Es importante resaltar que todo lo que es válido en lógica proposicional continúa siendo válido en la lógica de predicados: las definiciones, las reglas de inferencia, los teoremas, las leyes algebraicas, etc. continúan valiendo. Por lo tanto lo que haremos en este capítulo es continuar extendiendo nuestro lenguaje para poder hacerlo más expresivo pero de una manera constructiva.

Antes de introducir los conceptos propios de la lógica de predicados es relevante reflexionar, al menos a un nivel intuitivo, sobre los rasgos del tipo de razonamientos que trataremos. Para ello te proponemos la siguiente actividad cuyos ejemplos han sido extraídos del libro de Barwise & Etchemendy (1999).

Actividad

Las propiedades lógicas de los razonamientos dentro de la lógica de predicados son sumamente dependientes de los adjetivos cuantificadores que forman cada sentencia. Dados los siguientes razonamientos muy similares entre sí:

Todo actor rico es un buen actor.
Brad Pitt es un actor rico.
Por lo tanto, Brad Pitt es un buen actor.

Muchos actores ricos son buenos actores.
Brad Pitt es un actor rico.
Por lo tanto, Brad Pitt es un buen actor.

Ningún actor rico es un buen actor.
Brad Pitt es un actor rico.
Por lo tanto, Brad Pitt es un buen actor.

1. ¿Qué opinás de la validez de estos razonamientos? Justificá tu respuesta.
2. ¿Qué podrías decirle a una persona que te presentara argumentos como estos? ¿Cómo podrías rebatir algunos de ellos?

8.1. Extendiendo el lenguaje

Hemos visto que con un conjunto de variables proposicionales no es posible expresar el razonamiento sobre Sócrates. Dividamos el problema en partes y concentrémonos, en primer lugar, en formalizar sólo el fragmento de enunciado *'los hombres son mortales'*. Luego nos ocuparemos del resto. Llevar a cabo esta tarea involucra tener una «estructura» que me permita expresar que los hombres son mortales con la propiedad de poder evaluarse en casos concretos de hombres, como Sócrates. Pero nuestro lenguaje ya cuenta con estructuras de este tipo: las funciones.

Así, podemos definir una función que se llame `mortal`, que tenga un argumento —la función toma un valor posible del conjunto de los hombres— y devuelva verdadero o falso según el hombre sea mortal o no. A este tipo de funciones, que tenían argumentos de distinto tipo pero que devolvían un valor booleano, los llamamos **predicados**. Tal como su nombre lo indica, en la lógica de predicados este tipo de funciones son la piedra fundamental. Esta estructura permite ser evaluada en distintos argumentos, en particular, `mortal`.S denotará al predicado evaluado en el hombre Sócrates.

Un predicado puede verse como una afirmación de que un determinado objeto x posee una cierta propiedad o no. Los mismos se pueden definir sobre distintos conjuntos de datos: sobre el conjunto de los hombres como lo hicimos anteriormente, sobre los números naturales, sobre las listas, etc. Veamos un ejemplo numérico. La afirmación $x < 5$ es un predicado y será verdadero dependiendo el valor que tome el argumento. Formalmente el predicado se podría expresar así:

`menor5` :: $Nun \to Bool$

`menor5`.$x \doteq x < 5$

Por supuesto, un predicado puede tener mas de un argumento, es mas, puede tener todos cuantos sean necesarios. Así, el predicado $x < y$ es un predicado con dos argumentos x e y y será verdadero o falso dependiendo de los valores que esos argumentos tomen en un estado dado.

En capítulos anteriores ya has definido predicados. Las **guardas** de las definiciones por casos, la función `esta` que desarrollaste en la actividad de la página 40, la funcion `ordenada` de la página 41, la función `pip` de la página 25 y la función `bisiesto` 24 son todos ejemplos de predicados.

Ahora bien, los predicados siempre se definen en relación a un conjunto, a un **universo**. El universo es el conjunto de posibles valores que pueden tomar las variables de los predicados. De

esta manera, los universos estarán dados por los tipos de los argumentos de un predicado dado. Como vimos en los ejemplos anteriores los universos pueden ser muy variados, un predicado se puede referir al conjunto de los hombres, al conjunto de todas las listas, al conjunto de los números naturales, al conjunto de las figuras geométricas, el conjunto de todas las cosas que existen, las personas en una habitación, un conjunto particular de objetos físicos, etc.

Haciendo uso solamente de los predicados, si quisiéramos formalizar una sentencia como *'todos los hombres son mortales'* deberíamos listar todas las evaluaciones posibles del predicado `mortal` en una expresión de la forma:

$$\texttt{mortal}.h_1 \wedge \texttt{mortal}.h_2 \wedge \texttt{mortal}.h_3 \wedge \texttt{mortal}.h_4 \wedge \dots$$

Notá que es preciso vincular cada evaluación con una conjunción porque es necesario que todos satisfagan el predicado.

Claramente esta notación no es práctica y ni siquiera es posible para universos infinitos como el de los números naturales. Esto hace necesario la introducción de una nueva estructura: un **cuantificador**. En este caso queremos poder expresar que *todo en nuestro universo tiene cierta propiedad*: 'para todo x en el universo un dado predicado p vale' o 'todo x tiene la propiedad p'. Esta es una **cuantificación universal** y se denota por el símbolo $\forall$.

Las cuantificaciones universales se utilizan para establecer ciertas propiedades. Por ejemplo, si necesitás decir formalmente que un programa dará el resultado correcto para cualquier valor de entrada, deberás utilizar $\forall$.

El símbolo de la letra A invertida se utiliza porque recuerda a la palabra *All*, en inglés, todos.

Pero esta no es la única situación posible. Imaginá que quisiéramos decir *'existe un hombre que es mortal'*. En este caso también deberiamos listar todas las evaluaciones posibles del predicado `mortal` vinculando cada una de ellas con una disyunción, ya que con que uno solo de los hombres satisfaga el predicado la expresión completa será verdadera. Deberíamos escribir algo de la forma:

$$\texttt{mortal}.h_1 \vee \texttt{mortal}.h_2 \vee \texttt{mortal}.h_3 \vee \texttt{mortal}.h_4 \vee \dots$$

Para superar las dificultades que acarrea esta notación se introduce un nuevo cuantificador: el **cuantificador existencial**. Una expresión que contenga a este cuantificador establecerá que algún elemento del universo posee cierta propiedad: 'existe un x en el universo que satisface un dado predicado p' o 'algún x posee la propiedad p'.

El cuantificador universal se denota por el símbolo $\exists$ y la letra E invertida se utiliza para recordar a la palabra existe.

Las cuantificaciones existenciales se utilizan para establecer propiedades que se satisfacen al menos una vez. Por ejemplo, si quisieras establecer que una base de datos posee un registro e una persona dada deberías usar un $\exists$.

8.1.1. Nuestra notación

Toda cuantificación está asociada a una o más variables. Siempre decimos 'para todo x vale el predicado $p.x$' o 'existen y y z que satisfacen el predicado $p.y.z$'. Estas variables se llamarán **variables ligadas** a un determinado cuantificador.

Para escribir una expresión cuantificada necesitamos establecer:

- Cuál es el cuantificador que utilizaremos
- Cuáles son las variables ligadas
- Cuáles son los predicados involucrados en la cuantificación

Esto podría hacerse a través de la siguiente notación, bastante extendida en los libros de texto:

$\forall x\ p(x)$

$\exists y\ p(y)$

En ella los paréntesis se utilizan para demarcar la aplicación de una función. Esta notación introduce dificultades ya que admite diferentes interpretaciones. Las analizaremos en la siguiente actividad:

Actividad

¿Cómo interpretarías las siguientes expresiones cuantificadas? Introducí paréntesis para determinar cada interpretación.

- $\forall x\ P(x) \wedge (\neg q)$
- $\forall y\ S(y) \Rightarrow \exists z\ T(z)$
- $\forall w\ R(w) \vee w$

La actividad anterior nos permitó ver la necesidad de marcar claramente el alcance de cada cuantificación y las complicaciones que trae aparejada la utilización de los paréntesis para referirse a la aplicación de funciones. La notación que proponemos supera estas dificultades:

- Delimitaremos el alcance de una cuantificación y, por lo tanto, el alcance de las variables ligadas a la misma a través de paréntesis quebrados: '$\langle$' y '$\rangle$'. Utilizar estos paréntesis nos permitirá continuar utilizando los paréntesis comunes para modificar las reglas de precedencia.
- Cuando las variables cuantificadas sean más de una éstas se separarán por comas.
- Para separar claramente el listado de variables ligadas de los predicados involucrados utilizaremos el símbolo ::. Esta notación se explicará más adelante.

Por ejemplo, trabajemos en el universo de los números naturales. Si quisiéramos formalizar la sentencia *'todos los x son pares'* definimos primero el predicado $\texttt{par}.x$ y luego podemos escribir la expresión cuantificada:

$$\langle \forall x\ ::\ \texttt{par}.x \rangle$$

Trabajando en este universo si tenemos el predicado $2 * x = y$ podemos escribir una expresión cuantificada que establezca que existe un x y un y que satisfacen el predicado de la siguiente forma:

$$\langle \exists x, y\ ::\ 2 * x = y \rangle$$

En este casos las variables x e y están ligadas a la cuantificación. También podría escribirse una expresión como:

$$\langle \exists x\ ::\ 2 * x = y \rangle$$

en donde sólo la variable x está ligada a la cuantificación y la variable y es libre.

Algunas veces las expresiones que contienen cuantificadores son verdaderas o falsas dependiendo al universo dentro del cual estamos trabajando. Esto hace necesario que en ciertas ocasiones se deba explicitar el universo al que nos estamos refiriendo. Para reflexionar acerca de esto te proponemos la siguiente actividad:

Actividad

Considerá la siguiente sentencia "para todo x existe un y tal que $x = 2 * y$".

- Si el universo es el universo de los enteros pares, ¿esta sentencia es válida?
- Si el universo es el universo de los números naturales, ¿esta sentencia es válida?
- ¿Qué ocurre si el universo con el cual trabajamos es el universo de las listas de números naturales?

8.2. SAT: Nuestro universo y nuestros predicados

En este capítulo nos vamos a focalizar en un universo particular: el de las figuras geométricas. El sofware que utilizaremos, llamado SAT , te permitirá construir determinados mundos en este universo agregando o quitando figuras geométricas en el universo. Al mismo tiempo, SAT te permite escribir expresiones cuantificadas y el programa se encarga de verificar si cada una de estas expresiones es satisfecha por el mundo que construiste.

Las figuras geométricas que SAT pone a nuestra disposición son tres: círculos, cuadrados y triángulos. Cada uno de ellos puede tener distintas propiedades: ser pequeño, mediano o grande y rojo, azul o amarillo. A su vez existen predicados que relacionan a más de una figura: podemos establecer que una figura está arriba de, abajo de, a la derecha de o a la izquierda de otra figura o que está entre dos figuras dadas.

En la siguiente tabla se muestran todos estos predicados con su semántica intuitiva:

Predicado	**Semántica**
`esCuadrado`.x	x es un cuadrado
`esCirculo`.x	x es un círculo
`esTriangulo`.x	x es un triángulo
`esPequeño`.x	x es una figura pequeña
`esMediano`.x	x es una figura mediana
`esGrande`.x	x es una figura grande
`esRojo`.x	x es rojo
`esAzul`.x	x es azul
`esAmarillo`.x	x es amarillo
`estaArribaDe`.$x.y$	x está arriba de y
`estaAbajoDe`.$x.y$	x está abajo de y
`estaALaDerechaDe`.$x.y$	x está a la derecha de y
`estaALaIzquierdaDe`.$x.y$	x está a la izquierda de y
`estaEntre`.$x.y.z$	x está entre y y z

Actividad

Para familiarizarnos con estos predicados te proponemos que explores qué propiedades satisfacen. Por ejemplo,

`estaAbajoDe`.$x.y \equiv$ `estaArribaDe`.$y.x$

- ¿Qué otras relaciones como estas podés encontrar?
- ¿Cómo se podrían expresar las negaciones de cada uno de los predicados en términos de los otros? Por ejemplo, ¿cómo podrías expresar que una figura *no es grande* en términos de `esMediano` y `esPequeño`?

Una vez que hayas creado un elemento es posible asignarle un "nombre" a la misma. Para estos nombres utilizaremos las letras: a, b, c, ...

Actividad

En el siguiente mundo de SAT , aparecen un conjunto de objetos geométricos. Las siguientes son sentencias que dicho mundo satisface:

- el elemento a tiene la misma forma que el elemento b
- existe un elemento entre d y e
- Todas las pirámides son pequeñas
- Existen cubos pequeños
- Nada está a la derecha de b.

1. Explicá con tus palabras por qué cada una de las expresiones es satisfecha por el mundo.
2. ¿Qué cambios habría que hacer para que la segunda sentencia no se satisfaga? ¿Y para falsear la tercera? ¿y para falsear la cuarta? ¿y la quinta?

Actividad

1. Escribí expresiones, algunas de ellas requerirán de cuantificadores otras no, para formalizar cada una de las siguientes sentencias:
   a) Existe un elemento que es un círculo
   b) Existe un elemento que es rojo
   c) b es un cuadrado
   d) Todos los elementos son grandes
   e) Existe un elemento que está a la derecha de b
2. Construí un mundo en el que se satisfagan todas las sentencias anteriores.
3. ¿Cuál es la menor cantidad de figuras necesarias para que se satisfagan todas las sentencias?

8.3. Las cuatro formas aristotélicas

Aristóteles fue un filósofo griego que, al estudiar las leyes del razonamiento, logró formalizar un sistema lógico de forma tal que sus propuestas han trascendido hasta nuestros días. Aristóteles planteó sus ideas en varias obras, reunidas posteriormente bajo el nombre de Organon (órgano, herramienta), para difundir sus conocimientos sobre las leyes del razonamiento, argumentando que estas eran vitales para adentrarse en el mundo de la filosofía.

Aristóteles estudió formas de razonamiento asociadas con expresiones cuantificadas del tipo *'todo hombre'*, *'algún hombre'* o *'ningún hombre'*. Las cuatro formas aristotélicas se pueden enunciar de la siguiente manera:

'Todo A es B'
'Algunos A son B'
'Ningún A es B'
'Algunos A no son B'

En las siguientes actividades analizaremos cada una de ellas a través de ejemplos en nuestro universo de figuras geométricas:

Actividad

1. Una versión de la primer forma aristotélica para nuestro universo es la siguiente sentencia: *'Todo cuadrado es grande'*. Al enfrentarse a la tarea de formalizar dicha sentencia dos estudiantes dieron respuestas diferentes. Ellas eran:
   - Estudiante 1: $\langle \forall x \,:: \texttt{esCuadrado}.x \wedge \texttt{esGrande}.x \rangle$
   - Estudiante 2: $\langle \forall x \,:: \texttt{esCuadrado}.x \Rightarrow \texttt{esGrande}.x \rangle$

   ¿Qué opinás de estas formalizaciones? ¿Cuál es la correcta?
2. Una versión de la segunda forma aristotélica para nuestro universo es la siguiente sentencia: *'Algunos triángulos son amarillos'*. Los dos estudiantes anteriores volvieron a formalizar la sentencia de manera diferente:
   - Estudiante 1: $\langle \exists x \,:: \texttt{esTriangulo}.x \wedge \texttt{esAmarillo}.x \rangle$
   - Estudiante 2: $\langle \exists x \,:: \texttt{esTriangulo}.x \Rightarrow \texttt{esAmarillo}.x \rangle$

   ¿Qué opinás de estas formalizaciones? ¿Cuál es la correcta?
3. ¿Cómo formalizarías la siguiente versión de la tercer forma aristotélica: *'Ningún elemento amarillo es grande'*? Elaborá una expresión que contenga un cuantificador universal y otra que contenga un cuantificador existencial.
4. ¿Cómo formalizarías la siguiente versión de la cuarta forma aristotélica: *'Algunos elementos grandes no son cuadrados'*?
5. Construí un mundo en el que estas cuatro sentencias se satisfagan.

8.4. Rango y Término

Es frecuente que necesitemos expresar sentencias complejas, donde más de un predicado es aplicado a una variable ligada a un cuantificador. Este era el caso de las expresiones que formalizaban las dos primeras formas aristotélicas:

$$\langle \forall x \,:: \texttt{EsCuadrado}.x \Rightarrow \texttt{EsGrande}.x \rangle$$

$$\langle \exists x \,:: \texttt{EsTriangulo}.x \wedge \texttt{EsAmarillo}.x \rangle$$

Esta situación es tan frecuente que es común realizar una distinción entre el predicado que determina el rango de valores en donde puede moverse la o las variables cuantificadas y el predicado que debe o deben satisfacer dichas variables. Al primero de ellos se lo denomina **rango** y al segundo **término**. Nuestra notación nos permite realizar esta distinción de manera sencilla. El rango se coloca luego de los primeros dos puntos y el término luego de los segundos dos puntos, de la siguiente manera:

$$\langle \forall x \,:\, \texttt{EsCuadrado}.x \,:\, \texttt{EsGrande}.x \rangle$$

$$\langle \exists x \,:\, \texttt{EsTriangulo}.x \,:\, \texttt{EsAmarillo}.x \rangle$$

Es importante resaltar que esto es sólo una notación, no cambia el sentido de la expresión.

También hay que notar que para el caso del cuantificador existencial si tenemos una expresión del tipo:

$$\langle \exists x \,::\, p_1.x \wedge p_2.x \wedge \ldots \wedge p_n.x \rangle$$

como la conjunción es conmutativa cualquiera de los predicados $p_i.x$ puede tomar el rol de rango, generando las siguientes expresiones todas equivalentes entre sí:

$$\langle \exists x \,:\, p_1.x \,:\, p_2.x \wedge \ldots \wedge p_n.x \rangle$$

$\langle \exists x : p_2.x : p_1.x \wedge \ldots \wedge p_n.x \rangle$

$\langle \exists x : p_n.x : p_1.x \wedge p_2.x \wedge \ldots \wedge p_{n-1}.x \rangle$

No ocurre lo mismo en el caso del cuantificador universal. Si tenemos una expresión del tipo:

$\langle \forall x :: p_1 \Rightarrow p_2 \rangle$

Podemos escribirla utilizando rango y término sencillamente como:

$\langle \forall x : p_1 : p_2 \rangle$

Ahora, si quisiéramos que p_2 estuviera en el rango deberíamos hacer uso del teorema de la contrapositiva que establece que: $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$ Así, la expresión con rango y término quedaría expresada como:

$\langle \forall x : \neg p_2 : \neg p_1 \rangle$

Los cuantificadores también pueden mezclarse, dando lugar a expresiones más complejas. Esto es así porque una expresión cuantificada es también un predicado y por lo tanto puede jugar el rol del rango o el término.

Por ejemplo podríamos querer expresar la siguiente propiedad: *'Para todo ciculo existe un cuadrado que está a la derecha'*. En fórmulas, esto de expresaría así:

$\langle \forall x : \texttt{EsCirculo}.x : \langle \exists y : \texttt{EsCuadrado}.y : \texttt{estaALaDerecha.y.x} \rangle \rangle$

Otro ejemplo es el siguiente: Formalizar la sentencia *'Existe un elemento que está arriba de todos los círculos'*. En este caso la expresión sería:

$\langle \exists x : \langle \forall y :: \texttt{esCirculo}.x \rangle : \texttt{estaArribaDe}.x.y \rangle$

En las siguientes actividades iremos construyendo mundos y formalizando sentencias cada vez más complejas de manera que puedas familiarizarte con la semántica de los cuantificadores universal y existencial.

Actividad

Describiendo un mundo: Abrí el mundo correspondiente a esta actividad.

1. Escribí expresiones que formalicen cada una de las siguientes expresiones que pueden utilizarse para describirlo. Para ello podés utilizar tus conocimientos de las cuatro formas aristotélicas. Corroborá que cada expresión esté bien escrits y que es verdadera en este mundo.
   a) Escribí una sentencia para describir el tamaño de todos los cuadrados.
   b) Escribí una sentencia para describir el tamaño de todos los círculos.
   c) Escribí una sentencia para expresar que todo triángulo es o bien pequeño, o bien mediano o bien grande.
   d) Notá que algún triángulo es grande. Expresá este hecho.
   e) Notá que algún triángulo no es grande. Expresá este hecho.
   f) Observá que algún triángulo es pequeño. Expresá este hecho.
   g) Observá que algún triángulo no es pequeño. Expresá este hecho.
   h) Notá que algún triángulo no es ni grande ni pequeño. Expresá este hecho.
   i) Expresá la observación de que ningún cuadrado es grande.
   j) Expresá el hecho de que ningún cubo es grande.
2. Ahora cambiá los tamaños de los objetos de la siguiente forma: hacé grande a uno de los círculos, hacé mediano a un cuadrado y a todos los triángulos pequeños. ¿Cuáles de las sentencias se siguen satisfaciendo y cuáles no?
3. Realizá otros cambios haciendo predicciones acerca de lo que sucederá con cada una de las sentencias. Luego analizá si tus predicciones se cumplieron.

Actividad

Abrí el mundo correspondiente a esta actividad.

1. Interpretá con tus palabras las siguientes sentencias:
   a) $\langle \forall x : : \texttt{esTriangulo}.x \Rightarrow \texttt{esGrande}.x \rangle$
   b) $\langle \forall x : : \texttt{esTriangulo}.x \Rightarrow \texttt{esCuadrado}.x \rangle$
2. ¿Ambas sentencias se satisfacen en este mundo? ¿Qué podés decir al respecto?
3. Formalizá y verificá si la siguiente afirmación es válida en este mundo: *Existe una esfera grande*. ¿Qué conclusiones podés sacar?

Actividad

1. Construí un mundo en el que se satisfagan progresivamente cada una de las siguientes sentencias. Luego formalizá cada una de ellas y verificá que efectivamente se satisfagan:
   a) Algo es grande
   b) Hay un cuadrado
   c) Hay un cuadrado grande
   d) Un cuadrado grande está a la izquierda de b
   e) b está a la derecha de un cuadrado grande
   f) Algo que está a la izquierda de b está detrás de c
   g) Un cuadrado grande que está a la izquierda de b está detrás de c
   h) Algún círculo no es grande
   i) Algo no es un círculo grande
   j) b está a la izquierda de un cuadrado
2. Analizá el mundo que te proponemos. ¿Las sentencias anteriores se satisfacen? Para hacerlo deberás introducir las formalizaciones nuevamente en este mundo.
3. ¿Qué sucede si movés el cuadrado grande a la esquina superior derecha? ¿Cuáles sentencias se satisfacen y cuáles no?
4. A partir del mundo original, ¿qué sucede si movés c hacia arriba por la misma columna y hacés grande a b? ¿Cuáles sentencias se satisfacen y cuáles no?

Actividad

A partir del siguiente mundo:

1. Las siguientes sentencias se satisfacen para este mundo. Formalizá cada una de ellas y verificalas con SAT

   a) Todos los cuadrados son pequeños

   b) Cada cuadrado pequeño está a la derecha de a

   c) Todos los círculos son grandes

   d) a está a la izquierda de todo círculo

   e) Cada cuadrado está o bien delante de b o detrás de a

   f) Todo cuadrado está a la derecha de a y a la izquierda de b

   g) Todo lo que es más pequeño que a es un cuadrado

   h) Ningún círculo es pequeño

   i) Si algo es un cuadrado, entonces está a la derecha de a y a la izquierda de b

2. ¿Qué sucede si movés a hacia la esquina superior derecha del tablero?

Actividad

El siguiente listado contiene expresiones cuantificadas un poco más complejas sobre nuestro universo:

1. $\langle \forall x : \texttt{esCirculo}.x : \texttt{esRojo}.x \equiv \texttt{esGrande}.x \rangle$
2. $\langle \exists x : : \langle \exists y : \neg(x = y) : \texttt{esGrande}.x \equiv \neg(\texttt{esRojo}.y) \rangle \rangle$
3. $\langle \forall x : \neg\texttt{esCirculo}.x : \langle \exists y : : \texttt{esCirculo}.x \wedge \texttt{esRojo}.x \rangle \rangle$
4. $\langle \forall x : \texttt{esGrande}.x : \langle \forall y : : \neg\texttt{esRojo}.y \equiv \texttt{esTriangulo}.y \rangle \rangle$

1. Para comprenderlas mejor traducí cada una de ellas a una sentencia en el lenguaje natural.
2. Construí un mundo en SAT en el que se satisfagan, simultáneamente, cada una de ellas.

Actividad

Construí un mundo en SAT en el que se satisfagan, simultáneamente, las siguientes propiedades:

1. $\langle \forall x : \neg esRojo.x \vee esTriangulo.x : \langle \exists y : x = y : esGrande.y \rangle \rangle$
2. $\langle \exists x : esRojo.x : \langle \exists y : : esRojo \equiv \neg esRojo.y \rangle \rangle$
3. $\langle \forall x : esCuadrado.x : \langle \exists y : \neg(x = y) : esTriangulo.y \rangle \rangle$
4. $\langle \forall x : esRojo.x : esCuadrado.x \wedge \langle \exists y : : esGrande.x \rangle \rangle$

8.5. Formalizando sentencias en otros universos

Las actividades anteriores, centradas en el universo de las figuras geométricas, te han permitido familiarizarte con las características y sutilezas de los cuantificadores exitencial y universal. A continuación, los utilizaremos para formalizar sentencias o interpretar expresiones en otros universos, por ejemplo, el de los números naturales, el de las listas, etc.

Para traducir expresiones cuantificadas del español lo primero que hay que saber es cómo tratar sentencias complejas, como, por ejemplo, 'un perro que vive en el patio' o 'toda chica que vive en Buenos Aires'. Nos concentremos, en primer lugar, en el primer tipo de sentencia cuya traducción más natural involucra un cuantificador existencial. Tipicamente las sentencias que usen este cuantificador comenzarán con adjetivos cuantiicadores como *algún*, *un*, *una*, *algo*.

Veamos un ejemplo simple adaptado de Barwise & Etchemendy (1999). Traduzcamos la frase: *'un perro pequeño y feliz está en casa'*. Esta sentencia afirma que hay un objeto que es simultáneamente pequeño, feliz, perro y está en casa. Introduciendo un predicado para cada una de estas condiciones esta frase podría escribirse de la siguiente forma, sin utilizar rango:

$$\langle \exists x :: \texttt{Pequeño}.x \wedge \texttt{Feliz}.x \wedge \texttt{Perro}.x \wedge \texttt{estaEnCasa}.x \rangle$$

Como ya vimos, cualquiera de estos predicados podría tomar el lugar del rango, generado entonces expresiones como:

$$\langle \exists x : \texttt{Pequeño}.x : \texttt{Feliz}.x \wedge \texttt{Perro}.x \wedge \texttt{estaEnCasa}.x \rangle$$

ó

$$\langle \exists x : \texttt{Feliz}.x : \texttt{Pequeño}.x \wedge \texttt{Perro}.x \wedge \texttt{estaEnCasa}.x \rangle$$

Las sentencias que comienzan con adjetivos cuantificadores como *Todos*, *cada*, *todo* usualmente se traducen con un cuantificador universal. Veamos un ejemplo, también extraído de Barwise & Etchemendy (1999). Consideremos la sentencia: *'Todo pequeño perro que está en casa es feliz'*. Esta oración afirma que todo lo que posea un propiedad compleja —la de ser un pequeño perro y estar en casa— también posee otra propiedad —la de ser feliz. Esto sugiere que la sentencia podría formalizarse como el primer tipo de forma aristotélica: Todos los A son también B. Pero en este caso la propiedad A es un poco más compleja. Introduciendo predicados similares a los de la oración anterior, esta sentencia puede formalizarse de la siguiente manera:

$$\langle \forall x :: \texttt{Pequeño}.x \wedge \texttt{Perro}.x \wedge \texttt{estaEnCasa}.x \Rightarrow \texttt{esFeliz}.x \rangle$$

Utilizando la notación con rango y término la expresión queda:

$$\langle \forall x : \texttt{Pequeño}.x \wedge \texttt{Perro}.x \wedge \texttt{estaEnCasa}.x : \texttt{esFeliz}.x \rangle$$

En los dos ejemplos anteriores los adjetivos cuantificadores aparecían al comienzo de la oración, facilitando en parte nuestro trabajo. Pero ese puede no ser siempre el caso. Para verlo considerá los siguientes ejemplos:

Miguel es dueño de un perro pequeño y feliz: $\langle \exists x : \texttt{Pequeño}.x \wedge \texttt{Perro}.x \wedge \texttt{esFeliz}.x : \texttt{MigelesDueño}.x \rangle$

Actividad

Introduciendo los predicados que creas necesarios formalizá las siguientes sentencias en lenguaje natural:

1. Los caballos de carrera son todos de pura raza.
2. Nada que valga la pena tener puede obtenerse fácilmente.
3. Sólo se admiten hombres.
4. No se admiten mujeres. ¿Las dos sentencias anteriores dicen lo mismo?
5. Sólo el valiente sabe como perdonar.
6. Ningún hombre es una isla.
7. Toda nación tiene el gobierno que se merece.
8. No todo lo que brilla es oro.

8.5.1. El universo de los números

Actividad

Formalizá las siguientes sentencias escritas en el lenguaje natural sobre los números. Para ello utilizá los cuantificadores apropiados y los predicados que te sean necesarios como por ejemplo, `esPar`.x y `esImpar`.x que establecen si un número dado es par o impar respectivamente.

1. Todo entero es par o impar. Un estudiante propuso la siguiente respuesta:

   $$\langle \forall x : x \in \mathbb{Z} : \texttt{esPar}.x \rangle \vee \langle \forall x : x \in \mathbb{Z} : \texttt{esImpar}.x \rangle$$

   ¿Qué opinás al respecto?

2. El producto de dos impares es impar.

3. El producto de un par y un impar es par.

4. Un entero positivo es primo si y sólo si ningún número distinto de él y de 1 lo divide. [Utilizá el predicado `esDivisible.x`]

5. Todo entero es divisible por un primo.

6. Todo entero positivo es la suma de dos cuadrados.

7. Existe un entero no negativo que es más chico que todos los demás enteros no negativos. Un estudiante propuso la siguiente respuesta:

   $$\langle \forall x : x \in \mathbb{Z} \wedge x > 0 : \langle \exists y : y \in \mathbb{Z} : x < y \rangle \rangle$$

   ¿Qué opinás al respecto?

8. Dados dos números enteros positivos, existe un tercer entero tal que el primer entero multiplicado por el tercer entero es mayor que el segundo entero. Un estudiante propuso la siguiente respuesta:

   $$\langle \forall x, y : x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge x > 0 \wedge y > 0 : \langle \exists z : z \in \mathbb{Z} \wedge z \neq x \wedge z \neq y : x * z > y \rangle \rangle$$

   ¿Qué opinás de la respuesta del estudiante?

Actividad

Centrémonos ahora en interpretar, en lenguaje natural, expresiones cuantificadas que involucren al universo de los números. Para ello escribí una oración que describa a cada una de las expresiones siguientes:

1. $\langle \forall x : x \in Num : \langle \exists y : y \in Int : x < y \rangle \rangle$
2. $\langle \exists x : x \in Num : \langle \forall y : y \in Int : x < y \rangle \rangle$ ¿Cuál es la diferencia con la expresión anterior?
3. $\langle \forall x, z : x, z \in Num \wedge x \neq z : \langle \exists y : y \in Num : x < y < z \rangle \rangle$

8.5.2. El universo de las listas

A continuación formalizaremos sentencias e interpretaremos expresiones referidas al universo de las listas. A diferencia de lo que veníamos haciendo, no vamos a introducir predicados arbitrarios sino que trabajaremos con las funciones sobre listas que ya conocés, principalmente la función cardinal y la función indexar.

Actividad

Formalizá las siguientes sentencias en lenguaje natural sobre listas:

1. x está en la lista xs. Para construir la expresión pensá en qué función sobre listas nos permitía acceder a un elemento determinado de una dada lista.
2. La lists xs consiste de ceros y unos.
3. Si el 1 está en xs entonces tamién el 0 está.
4. La lista xs contiene al menos un $true$.
5. x ocurre exactamente dos veces en xs.
6. Todos los elementos de xs son iguales. Un estudiante encontró una forma de formalizar esta sentencia utilizando sólo un cuantificador y la propiedad de transitividad de la igualdad. ¿Podés encontrarla vos también?
7. Las listas xs e ys tienen los mismos elementos. Un estudiante propuso la siguiente respuesta:

   $$\langle \forall i \,:\, 0 \leqslant i < \#xsmin\#ys \,:\, xs.i = ys.i \rangle$$

   ¿Qué opinás de esta resuesta?
8. n es el menor entero par en xs
9. x es el segundo valor más grande de xs
10. La lista xs está ordenada de manera decreciente. Un estudiante encontró una manera de formalizar esta sentencia usando sólo un cuantificador. ¿Podés encontrarla vos también?

Actividad

Centrémonos ahora en interpretar, en lenguaje natural, expresiones cuantificadas que involucren al universo de los números. Para ello escribí una oración que describa a cada una de las expresiones siguientes:

1. $\langle \forall i \,:\, 0 \leqslant i < N \wedge N \leqslant \#xs \,:\, xs.i \geqslant 0 \rangle$
2. $\langle \exists i \,:\, 0 \leqslant i < N \wedge N \leqslant \#xs \,:\, xs.i = 0 \rangle$
3. $\langle \forall p, q \,:\, 0 \leqslant p \wedge 0 \leqslant q \wedge p + q = \#xs - 1 \,:\, xs.p = xs.q \rangle$

A lo largo de todo este capítulo nos hemos introducido en la lógica de predicados prestando especial atención a cómo formalizar sentencias escritas en el lenguaje natural dentro de esta lógica. Si bien este ha sido un paso importante que enriquece nuestro lenguaje todavía quedan problemas por resolver. Por ejemplo, todavía no hemos estudiado cómo determinar si un razonamiento como el de Sócrates que presentamos al comienzo del capítulo es válido o no. De ello nos ocuparemos en le próximo capítulo.

Capítulo 9

Cálculo de predicados

En el capítulo anterior hemos introducido en nuestro lenguaje dos nuevos cuantificadores —el universal y el existencial—, estudiamos su semántica y su notación y los utilizamos para formalizar sentencias del lenguaje natural en distintos universos.

Ahora bien, ya vimos que cuando las expresiones dentro de la lógica de predicados se vuelven complejas se hace bastante difícil interpretarlas. Este es un primer motivo para introducir una cálculo, es decir, un conjunto de reglas que nos permitan manipular expresiones cuantificadas, con la idea de simplificarlas para hacer su interpretación más sencilla.

Pero hay otra razón que motiva la introducción de un cálculo de predicados. Si quisiéramos demostrar formalmente que un razonamiento como el de Sócrates es correcto podríamos, en un primer momento, intentar recurrir a las tablas de verdad, tal como lo hicimos con los razonamientos que formalizamos con lógica proposicional en el capítulo 6. Pero visto que cuando trabajamos con predicados los universos a los que están asociados son potencialmente infinitos nunca terminaríamos de completar la tabla de verdad. Así, si tuviéramos una expresión cuantificada sobre el universo de los números naturales como la siguiente:

$$\langle \exists x \, : \, x \in \mathbb{N} \, : \, \texttt{esPar}.x \rangle$$

deberíamos incluir una fila para cada uno de los valores posibles del argumento del predicado $\texttt{esPar}.x$, determinar el valor de verdad del predicado en dicho caso y luego el valor de verdad de la expresión cuantificada. Esta sería una tabla infinita que nunca podríamos terminar de llenar. En lógica proposicional esta dificultad estaba salvada porque las variables proposicionales solo podían tomar dos valores: verdadero o falso. Esta importante limitación hace imperativa la introducción de un cálculo que nos permita transformar las expresiones cuantificadas hasta que sea evidente cuál es su valor de verdad. A esta tarea nos enfrentaremos en este capítulo.

Para ello vamos a extender el sistema formal que venimos desarrollando para incluir a los cuantificadores universal y existencial. Realizar esta tarea implicará introducir los axiomas asociados a cada uno de ellos y desarrollar estrategias de demostración formales para demostrar teoremas que involucren a estos cuantificadores.

9.1. Axiomas del cuantificador universal

El primer axioma que daremos de este cuantificador se refiere a un rango particular del mismo. Supongamos que el rango es constantemente $True$. Ejemplos de predicados de este tipo son $0 = 0$, $5 > 7$, etc. Cuando esta es la situación el rango directamente se omitirá. Este es el hecho que captura el siguiente axioma:

Conceptos teóricos

Rango true: $\langle \forall x \, : \, True \, : \, f.x \rangle \equiv \langle \forall x \, : : \, f.x \rangle$

El segundo axioma que presentaremos es uno de los más importantes para el cuantificador universal. El mismo establece como reescribir una cuantificación que posee rango en una cuantificación

que no lo posee. Hacer esto permite darle significado al rol que juega el rango en una cuantificación universal.

Introduzcamos este axioma a través de un ejemplo particular.

Actividad

1. Escribí utilizando una expresión cuantificada la siguiente oración: 'para todos los n enteros, $2 * n$ es par'.
2. Ahora escribí utilizando una expresión cuantificada la oración: 'Para todos los n, si n es entero entonces $2 * n$ es par'.
3. ¿Crees que estas dos oraciones especifican conjuntos diferentes? Justificá tu respuesta.

El siguiente axioma captura las ideas discutidas en la actividad anterior:

Conceptos teóricos

Intercambio entre rango y término: $\langle \forall x : r.x : t.x \rangle \equiv \langle \forall x : : r.x \Rightarrow t.x \rangle$

Actividad

El cuantificador universal puede pensarse como una generalización de la conjunción. Esto es así porque afirmar que una cierta propiedad vale para todos los elementos de un conjunto dado es lo mismo que afirmar que vale para el primero **y** para el segundo **y** para el tercero, y así sucesivamente.

- Dadas estas ideas podría pensarse que algunas propiedades de la conjunción pueden extenderse al cuantificador universal. ¿Qué pensás de esta afirmación? ¿cuales serían las propiedades de la conjunción que podrían ser trasladadas al cuantificador universal?

Los siguientes axiomas dan cuenta de las propiedades que se trasladan desde la conjunción al cuantificador universal:

Conceptos teóricos

Regla del término: $\langle \forall x : : t_1.x \rangle \wedge \langle \forall x : : t_2.x \rangle \equiv \langle \forall x : : t_1.x \wedge t_2.x \rangle$

Conceptos teóricos

Distributividad del $\wedge$ con el $\forall$: $X \vee \langle \forall x : : t.x \rangle \equiv \langle \forall x : : F \vee t.x \rangle$ siempre que x no aparezca libremente en F.

Actividad

Analiza los dos axiomas anteriores para dar respuesta a las siguientes preguntas:

- ¿Cuáles son las propiedades de la conjunción que «se trasladan» y están involucradas en los dos axiomas para el cuantificador universal?
- El axioma denominado *Regla del término* está formulado para el rango $True$. Este axioma puede generalizarse para cualquier rango demostrando el siguiente teorema:

  $\langle \forall x : r.x : t_1.x \rangle \wedge \langle \forall x : r.x : t_2.x \rangle \equiv \langle \forall x : r.x : t_1.x \wedge t_2.x \rangle$

  Demostrá en FUN este teorema.
- En el segundo axioma aparece la expresión F que puede ser una fórmula cualquiera posiblemente con variables libres. ¿Por qué está incluida en la formulación de ese axioma la cláusula "siempre que x no aparezca libremente en F"?
- Lee atentamente la siguiente demostración:

$$
\begin{array}{ll}
 & X \vee \langle \forall x : r.x : t.x \rangle \\
\equiv & \{ \text{ Intercambio entre rango y término } \} \\
 & X \vee \langle \forall x : : \underline{r.x \Rightarrow t.x} \rangle \\
\equiv & \{ \text{ Caracterización de la implicación: } p \Rightarrow q \equiv \not\equiv q \vee q \ \} \\
 & X \vee \langle \forall x : : \not\equiv r.x \vee t.x \rangle \\
\equiv & \{ \text{ Distributividad del } \forall \text{ con } \vee, \ x \text{ no ocurre en } X \ \} \\
 & \langle \forall x : : \underline{X \vee \not\equiv r.x \vee t.x} \rangle \\
\equiv & \{ \text{ Conmutatividad del } \vee \ \} \\
 & \langle \forall x : : \underline{\not\equiv r.x \vee X \vee t.x} \rangle \\
\equiv & \{ \text{ Caracterizacion de la implicación } \} \\
 & \langle \forall x : : r.x \Rightarrow X \vee t.x \rangle \\
\equiv & \{ \text{ Intercambio entre rango y término } \} \\
 & \langle \forall x : r.x : X \vee t.x \rangle
\end{array}
$$

  1. ¿Qué propiedad demuestra esta prueba?

La actividad anterior introdujo el problema de que, cuando escribimos expresiones que involucren al cuantificador universal existen variables *libres*, es decir que puede ser reemplazadas por otras, y variables *ligadas*. La siguiente es una definición formal de cuándo una variable está libre:

Conceptos teóricos

Variable libre: Se define inductivamente cuándo una variable está libre en una expreción:

- Una variable, x está libre en la expresión x.
- Si la variable x está libre en E entonces lo está también en (E)
- Si la variable x está libre en E y f es una operación válida en el tipo E, entonces x está libre en $f.(\ldots, E, \ldots)$.
- Si la variable i está libre en E y no aparece en la secuencia de variables y, entonces lo está también en $\langle \forall y : F : E \rangle$ y en $\langle \forall y : E : F \rangle$.

Notación: Dada una expresión E, el conjunto de las variables libres de E se deonominan $VL.E$

Actividad

Ahora definamos el concepto de variable ligada. Una primer definición podría ser: todas las variables que no son libres son ligadas a un cuantificador.

1. Considerá la siguiente expresión:

$$i = 0 \vee \langle \forall i \,:\, 0 \leqslant i \leqslant n \,:\, i^2 + 1 = 2 \rangle$$

¿La variable i es libre o está ligada a un cuantificador?

La siguiente definición formaliza lo discutido en la actividad anterior:

Conceptos teóricos

Variable ligada: Sea i una variable libre en la expresión E y $i \in V.x$. Luego, la variable i está ligada a la variable de cuantificación correspondiente en la expresión $\langle \forall x \,:\, E \,:\, T \rangle$ y en la expresión $\langle \forall x \,:\, R \,:\, E \rangle$.
Se extiende la definición inductivamente: Si i está ligada en E, entonces también lo estará en (E), $f(\ldots, E, \ldots)$, $\langle \forall x \,:\, E \,:\, T \rangle$, $\langle \forall x \,:\, R \,:\, E \rangle$.
Notación: Dada una expresión E, el conjunto de las variables ligadas de E se denotará por $VG.E$

Actividad

Dadas las siguientes expresiones determiná, para cada una de ellas, los conjuntos $VL.E$ y $VG.E$:

1. $\langle \forall x \,:\, s < x \,:\, \frac{x}{2*s} \neq 0 \rangle$
2. $z \wedge m \Rightarrow \langle \forall z \,::\, z \leqslant m \rangle$
3. $\langle \forall x, y \,:\, 0 \leqslant x \leqslant y \,:\, x + y < z \rangle \equiv p$

Actividad

El siguiente teorema se denomina comunmente **partición de rango**:

$$\langle \forall x \,:\, r_1.x \vee r_2.x \,:\, f.x \rangle \equiv \langle \forall x \,:\, r_1.x \,:\, f.x \rangle \wedge \langle \forall x \,:\, r_2.x \,:\, f.x \rangle$$

1. Para ayudarte a darle sentido a este teorema escribí con tus palabras cuál es la propiedad que este teorema establece.
2. Teniendo en cuenta que el predicado $par.x$ indica si la variable x es par y que el predicado $impar.x$ indica si la variable es impar, ¿podríamos utilizar este teorema para realizar la siguiente transformación?:

$$\begin{array}{l} \quad \langle \forall x \,:\, x \in \mathbb{N} \,:\, x^2 \geqslant 0 \rangle \\ \equiv \{ \ldots \} \\ \quad \langle \forall x \,:\, par.x \vee impar.x \,:\, x^2 \geqslant 0 \rangle \\ \equiv \{ \ldots \} \\ \quad \langle \forall x \,:\, par.x \,:\, x^2 \geqslant 0 \rangle \wedge \langle \forall x \,:\, impar.x \,:\, x^2 \geqslant 0 \rangle \end{array}$$

¿Qué otras formas de dividir el rango se te ocurren?

3. Utilizando FUN , los axiomas del cuantificador universal que ya enunciamos y algunas propiedades del cálculo proposicional construí una demostración para este teorema.

El siguiente axioma, al igual que el primero que presentamos, se refiere a un rango particular de la cuantificación universal. De qué ocurre cuando hay sólo un valor que satisface el rango de la cuatificación se ocupa este axioma:

Conceptos teóricos

Rango unitario: $\langle \forall x \,:\, x = x_1 \,:\, t.x \rangle \equiv t.x_1$

Actividad

Usando tus palabras dá un ejemplo concreto en donde este axioma se ponga en funcionamiento

Actividad

Cuando afirmamos que una cierta propiedad vale para todos los x de un rango dado, entonces, esa propiedad debería valer en particular para uno de esos x arbitrario, llamémoslo x_i. Esta es la idea de un teorema muy importante denominado **Instanciación**.

1. ¿Qué fórmula propondrías para enunciar el teorema de instanciación?
2. Completá la siguiente demostración del teorema:

$$
\begin{array}{ll}
 & \langle \forall x \,::\, t.x \rangle \Rightarrow t.x_1 \\
\equiv & \{ \text{ Definición dual } \Rightarrow\text{: } p \Rightarrow q \equiv p \wedge q \equiv p \ \} \\
 & \langle \forall x \,::\, t.x \rangle \wedge \underline{t.x_1} \equiv \langle \forall x \,::\, t.x \rangle \\
\equiv & \{ \text{ Rango unitario } \} \\
 & \langle \forall x \,::\, t.x \rangle \wedge \langle \forall x \,:\, x = x_1 \,:\, t.x \rangle \equiv \langle \forall x \,::\, t.x \rangle \\
\equiv & \{ \text{ Rango } true \ \} \\
 & \langle \forall x \,:\, True \,:\, t.x \rangle \wedge \langle \forall x \,:\, x = x_1 \,:\, t.x \rangle \equiv \langle \forall x \,::\, t.x \rangle \\
\equiv & \{ \ \ldots \ \} \\
 & \vdots \\
\equiv & \{ \ \ldots \ \} \\
 & True
\end{array}
$$

En algunas ocasiones es necesario escribir expresiones que involucren más de una cuantificación. Es el caso, por ejemplo, del enunciado de la conmutatividad de la suma que demostramos unos capítulos atrás:

$$\langle \forall x \,::\, \langle \forall y \,::\, x + y = y + x \rangle \rangle$$

Nada hubiese cambiado si la cuantificación se realizaba primero sobre y y luego sobre x. El siguiente axioma captura esta idea:

Conceptos teóricos

Intercambio de cuantificadores: $\langle \forall x \,::\, \langle \forall y \,::\, t.x.y \rangle \rangle \equiv \langle \forall y \,::\, \langle \forall x \,::\, t.x.y \rangle \rangle$

En castellano este axioma se enunciaría así en el caso de la conmutatividad de la suma:
"Decir que para todo x tal que para todo y, $x + y$, es equivalente a decir que para todo y tal que para todo x, $y + x$"

Ahora bien, cuando una expresión involucra a más de un cuantificador en castellano es más fácil y natural decir "para todo x e y vale tal propiedad". Esta es la idea involucrada en el último axioma que presentaremos para el cuantificador universal:

Conceptos teóricos

Anidado: $\langle \forall x, y \,::\, t.x.y \rangle \equiv \langle \forall x \,::\, \langle \forall y \,::\, t.x.y \rangle \rangle$

Actividad

El siguiente teorema generaliza el axioma de *Anidado* para un rangos que no sean constantemente equivalentes a $True$:
$\langle \forall x, y : r_1 \wedge r_2 : t \rangle \equiv \langle \forall x : r_1 : \langle \forall y : r_2 : t \rangle \rangle$ cuando y no ocurre libremente en r_1.

1. ¿Por qué es necesaria la condición 'cuando y no ocurre libremente en r_1'?
2. Utilizando el teorema del cálculo proposicional que establece que: $a \Rightarrow (b \Rightarrow c) \equiv a \wedge b \Rightarrow c$ construí una demostración para el teorema de anidado generalizado.

Actividad

Describí con tus palabras qué propiedad expresa el siguiente teorema:
$\langle \forall x : r.x : t.x \rangle \equiv \langle \forall y : r.y : t.y \rangle$ siempre que y no ocurra en $r.x$ o en $t.x$ y que x no ocurra en $r.y$ o en $t.y$

1. Dada la siguiente expresión cuantificada:

   $\langle \forall x : 0 \leqslant x \leqslant y^2 + 4 : x^3 \leqslant 0 \rangle$

   ¿Podría utilizarse el teorema anterior para reemplazar la variable de cuantificación x por otra llamada y? ¿Por qué? Escribí dos expresiones cuantificadas, una en la cual el teorema pueda aplicarse y otra en la que no.

2. Completá la siguiente demostración del teorema:

$$
\begin{array}{ll}
 & \langle \forall y : r.y : \underline{t.y} \rangle \\
\equiv & \{ \text{ Rango unitario } \} \\
 & \langle \forall y : r.y : \langle \forall x : x = y : t.x \rangle \rangle \\
\equiv & \{ \text{ Intercambio entre rango y término } \} \\
 & \langle \forall y : : r.y \Rightarrow \langle \forall x : x = y : t.x \rangle \rangle \\
\equiv & \{ \ldots \} \\
 & \vdots \\
\equiv & \{ \ldots \} \\
 & \langle \forall x : r.x : t.x \rangle
\end{array}
$$

El teorema anterior de cambio de variables nos permitía cambiar los nombres de las variables ligadas a un cuantificador universal. Pero existe otro teorema más general aún que establece que una variable puede reemplazarse por cualquier función f biyectiva, es decir, que posea inversa:

Conceptos teóricos

Cambio de variable generalizado: Si f es una función biyectiva en el rango de especificación y j es una variable que no aparece en r ni en t, las variables ligadas a la cuantificación pueden renombrarse como sigue:

$$\langle \forall x : r.x : t.x \rangle \equiv \langle \forall j : r.(f.j) : t.(f.j) \rangle$$

Actividad

¿Son las siguientes expresiones instancias del teorema de cambio de variable generalizado? En caso afirmativo reconocé cuál es la función involucrada en el cambio de variable:

1. $\langle \forall i : 0 \leqslant i \leqslant n : i + 1 = n \rangle \equiv \langle \forall j : 0 < i < n : i + 2 = n \rangle$
2. $\langle \forall z : 0 \leqslant z \leqslant q : 2 * z + 1 \leqslant q \rangle \equiv \langle \forall z : 1 \leqslant z \leqslant q + 1 : 2 * z \leqslant m \rangle$

Actividad

1. El siguiente teorema se denomina *Rango vacío*:

   $\langle \forall x : false : t.x \rangle \equiv True$

   a) ¿Por qué crees que recibe este nombre?

   b) Decidí si las siguientes fórmulas son instancias del teorema:

   1) $\langle \forall x : x \equiv \neg x : x \Rightarrow z \rangle \equiv True$
   2) $\langle \forall z : z \vee true : z \vee z \rangle \equiv True$
   3) $\langle \forall y : false : false \rangle \equiv True$

   c) Da dos instancias de este teorema.

   d) Construí en FUN una demostración del teorema.

2. El siguiente teorema se denomina *Término constante*:

   $\langle \forall x : : True \rangle \equiv True$

   a) ¿Por qué crees que recibe este nombre?

   b) Decidí si las siguientes fórmulas son instancias de este teorema:

   1) $\langle \forall p : : \neg(p \vee q) \equiv \neg p \wedge \neg q \rangle \equiv \neg(p \vee q) \equiv \neg p \wedge \neg q$
   2) $\langle \forall x : x \vee True : True \rangle \equiv True$
   3) $\langle \forall z : z \wedge z : z \vee \neg z \rangle \equiv True$

   c) Dá dos instancias de este teorema.

   d) Construí en FUN una demostración del teorema.

El siguiente teorema generaliza el teorema anterior a expresiones que tengan un término constante C cualquiera y que posean además un rango r:

Conceptos teóricos

Término constante generalizado: Si $r.x$ es no vacío, entonces

$$\langle \forall x : r.x : C \rangle \equiv C$$

Para analizar la necesidad de la condición "si $r.x$ es no vacío" en el teorema del término constante generalizado, te proponemos la siguiente actividad

Actividad

Supongamos que el teorema del término constante generalizado pudiera enunciarse sin la condición respecto al rango, es decir de la forma:

$$\langle \forall x : r.x : C \rangle \equiv C$$

para cualquier $r.x$ pudiendo ser éste vacío.
Ahora bien, podemos construir usando este "teorema" la siguiente demostración:

$True$
$\equiv$ { Rango vacío del $\forall$ }
$\langle \forall x : False : C \rangle$
$\equiv$ { "teorema término constante sin condición sobre el rango" }
C

¿Qué establece esta demostración? ¿Qué dice sobre C? ¿Cómo fue elegida C? ¿Cual es la contradicción que se establece?

Actividad

Para sistematizar lo visto hasta ahora, construí un listado con dos columnas. En una de ellas enumerá los axiomas que establecimos para el $\forall$ y en la otra los teoremas que demostramos para este cuantificador

Actividad

A partir de tus conocimientos del cuantificador universal escribí una expresión cuantificada para describir cada una de las siguientes situaciones:

1. Dadas dos listas de números xs e ys, ys es la lista cuyos elementos son el resultado de multiplicar por dos a cada uno de los elementos de la lista xs.
2. Dadas dos constantes numéricas n y x y una lista ys, ys es una lista con n elementos todos iguales a x.
3. Dadas dos listas de números xs e ys, xs es igual a ys.

9.2. El cuantificador existencial

Para definir a este cuantificador utilizaremos sólo un axioma que se encarga de «traducir» una expresión que contenga un cuantificador existencial en otra expresión que posea un cuantificador universal. Esta «traducción» nos permitirá deducir todas las propiedades que satisfaga el cuantificador existencial. De esta manera, las propiedades del existencial serán teoremas y no axiomas.

Definir un nuevo objeto matemático en función de otro definido con anterioridad y a partir de ello derivar las propiedades del nuevo objeto es una práctica bastante común en matemática y lógica que permite minimizar el número de axiomas de un cálculo.

Actividad

Dada una expresión cuantificada existencialmente del tipo:

$$\langle \exists : r.x : t.x \rangle$$

1. ¿De qué formas podrías reescribirla utilizando una cuantificación universal?
2. Dadas las siguientes oraciones en lenguaje natural:

   "Existe un natural x tal que elevado al cuadrado es igual a cuatro".

   "No es cierto que todos los naturales x elevado al cuadrado es distinto de cuatro".

   a) ¿Pensás que ambas representan a conjuntos distintos o iguales? ¿Por qué? Para justificar tu respuesta podés hacer gráficos, bosquejos de conjuntos, etc.

   b) Escribí expresiones cuantificadas que representen a ambas oraciones.

3. Ayudándote de este ejemplo, ¿modificarías o mejorarías tu respuesta a la primer pregunta?

El siguiente axioma captura las ideas discutidas en la actividad anterior:

Conceptos teóricos

Definición cuantificador existencial: $\langle \exists x : r.x : t.x \rangle \equiv \neg \langle \forall x : r.x : \neg t.x \rangle$

Actividad

Los siguientes teoremas son los análogos de los axiomas y teoremas del cuantificador universal para el cuantificador existencial:

1. $\langle \exists x : r.x : t.x \rangle \equiv \langle \exists x :: r.x \wedge t.x \rangle$
2. $\langle \exists x :: t_1.x \rangle \vee \langle \exists x :: t_2.x \rangle \equiv \langle \exists x :: t_1.x \vee t_2.x \rangle$
3. $X \wedge \langle \exists x :: t.x \rangle \equiv \langle \exists x :: X \wedge t.x \rangle$ siempre que x no ocurra en $VL.X$
4. $\langle \exists x : r_1.x : t.x \rangle \vee \langle \exists x : r_2.x : t.x \rangle \equiv \langle \exists x : r_1.x \vee r_2.x : t.x \rangle$
5. $\langle \exists x : False : t.x \rangle \equiv False$
6. $\langle \exists x : x = N : t.x \rangle \equiv T.N$
7. $\langle \exists x : r_1.x : \langle \exists y : r_2.x.y : t.x.y \rangle \rangle \equiv \langle \exists x, y : r_1.x \wedge r_2.x.y : t.x.y \rangle$
8. $\langle \exists x : r.x : t.x \rangle \equiv \langle \exists y : r.y : t.y \rangle$ siempre que y no ocurra en $t.x$ o en $r.x$ y que x no ocurra en $t.y$ o en $r.y$.
9. Si f es una función biyectiva y j es una variable que no ocurre en r ni en t, entonces $\langle \exists x : r.x : t.x \rangle \equiv \langle \exists j : r.(f.j) : t.(f.j) \rangle$
10. $t.Y \Rightarrow \langle \exists x :: t.x \rangle$

1. Compará uno a uno ellos con los axiomas y teoremas que dimos para el cuantificador universal con el objetivo de darle un nombre a cada uno.
2. Para el teorema 1, armá un ejemplo concreto que te ayude a comprender por qué este teorema es válido.
3. Vimos que el cuantificador universal podía pensarse como una generalización de la conjunción, ¿Qué operador lógico generaliza el operador existencial? ¿Cuáles de los teoremas que presentamos en esta actividad tiene relación con alguna propiedad del operador generalizado??
4. Por qué son necesarias las cláusulas "siempre que x no ocurra en $FV.X$" en el teorema 3 y "para cualquier y que no aparezca en r ni en t" en el teorema 8?
5. Demostrá los 8 primeros teoremas utilizando FUN . ¿Cuál es la estrategia general que utilizarás para realizar estas demostraciones?
6. Completá la siguiente demostración del último teorema: La misma parte de una instancia particular del teorema de instanciación para el $\forall$:

$$
\begin{array}{ll}
 & True \\
\equiv & \{ \text{ Teorema de instanciación } \forall \ \} \\
 & \neg t.Y \Leftarrow \langle \forall x :: \neg t.x \rangle \\
\equiv & \{ \text{ Contrapositiva: } p \Rightarrow q \equiv \neg q \Rightarrow \neg p \ \} \\
 & \neg(\neg t.Y) \Rightarrow \neg \langle \forall x :: \neg t.x \rangle \\
\equiv & \{ \ \ldots \ \} \\
 & \vdots \\
\equiv & \{ \ \ldots \ \} \\
 & t.Y \Rightarrow \langle \exists x :: t.x \rangle
\end{array}
$$

Cuando estudiamos cálculo proposicional demostramos los siguientes teoremas llamados de De Morgan:

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

Ahora bien, visto que el cuantificador universal puede considerarse como una generalización de la conjunción y el cuantificador existencial como una generalización de la disyunción, pueden formularse teoremas análogos a los anteriores pero que involucren a estos dos cuantificadores.

Actividad

Visto que el cuantificador universal puede asociarse con expresiones del tipo:

$$\langle \forall x :: t.x \rangle \rightsquigarrow t.x_1 \wedge t.x_2 \wedge t.x_3 \wedge \ldots$$

Y que el cuantificador existencial puede asociarse con expresiones del tipo:

$$\langle \exists x :: t.x \rangle \rightsquigarrow t.x_1 \vee t.x_2 \vee t.x_3 \vee \ldots$$

¿Qué fórmulas que involucren un cuantificador existencial y uno universal propondrías para formular las leyes de De Morgan para estos cuantificadores? ¿Qué relaciones hay entre estas dos propiedades y la definición del existe? Demuestren estos teoremas usando FUN .

Actividad

El siguiente teorema se denomina *término constante* y es el análogo al que demostramos para el cuantificador universal:
Si el rango es no vacío entonces:

$$\langle \exists x : r.x : C \rangle \equiv C$$

Construí, usando FUN una demostración para dicho teorema.

Vamos a ocuparnos ahora de dos teoremas sumamente potentes que luego serán útiles para demostrar muchos otros teoremas. Estos teoremas se denominan comúnmente como *debilitamiento* del $\forall$ y del $\exists$.

Conceptos teóricos

Debilitamiento para el cuantificador universal:

$$(t \Rightarrow s) \wedge \langle \forall x : r : t \rangle \Rightarrow \langle \forall x : r : s \rangle$$

Actividad

Escribí con tus palabras cuál es la propiedad que establece este teorema. ¿Por qué creés que se denomina debilitamiento?

La siguiente es una demostración de este teorema:

$$\begin{array}{ll} & (t \Rightarrow s) \wedge \langle \forall x : r : t \rangle \Rightarrow \langle \forall x : r : s \rangle \\ \equiv & \{ \text{ Distributividad del } \forall \ \} \\ & \langle \forall x : r : t \wedge (t \Rightarrow s) \rangle \Rightarrow \langle \forall x : r : s \rangle \\ \equiv & \{ \text{ Definición dual de } \Rightarrow \text{: } p \Rightarrow q \equiv p \wedge q \equiv p \ \} \\ & \langle \forall x : r : t \wedge (t \Rightarrow s) \rangle \wedge \langle \forall x : r : s \rangle \equiv \langle \forall x : r : t \wedge (t \Rightarrow s) \rangle \\ \equiv & \{ \text{ Regla del término } \} \\ & \langle \forall x : r : t \wedge (t \Rightarrow s) \wedge s \rangle \equiv \langle \forall x : r : t \wedge (t \Rightarrow s) \rangle \\ \equiv & \{ \ \ \} \end{array}$$

Conceptos teóricos

Debilitamiento para el cuantificador existencial:

$$(t \Rightarrow s) \wedge \langle \exists x : r : t \rangle \Rightarrow \langle \exists x : r : s \rangle$$

.

Actividad

Dado el siguiente teorema:

$$\langle \exists x : r.x : t.x \rangle \Rightarrow \langle \exists : : r.x \rangle$$

1. Describí con tus palabras cuál es la propiedad que establece y colocale un nombre que te parezca apropiado.
2. Da dos intancias de este teorema.
3. Utilizando el teorema de debilitamiento del cuantificador existencial demostrá el teorema.

Actividad

Decidí cuáles de las siguientes fórmulas son teoremas y cuáles no, dando un contraejemplo o demostrando que la fórmula es válida según corresponda:

- $\langle \forall x : r.x : t.x \rangle \equiv \langle \forall x : \neg t.x : \neg r.x \rangle$
- $\langle \forall x : r.x : t.x \rangle \equiv \langle \forall x : \neg r.x : \neg t.x \rangle$
- $\langle \exists x : : t_1.x \rangle \wedge \langle \forall y : : t_2.y \rangle \Rightarrow \langle \exists x : : t_1.x \wedge t_2.x \rangle$
- $\langle \forall x : : T.x \rangle \wedge X \equiv \langle \forall x : : T.x \wedge X \rangle$, siempre que x no ocurra en X.
- $\langle \forall x : \langle \exists y : : R.x.y \rangle : T.x \rangle \equiv \langle \forall x, y : R.x.y : T.x \rangle$
- $\langle \forall x : : T.x \rangle \Rightarrow \langle \exists x : : T.x \rangle$
- $\langle \exists x : : T.x \rangle \Rightarrow \langle \forall x : : T.x \rangle$

9.3. Análisis de razonamientos utilizando cálculo de predicados

Una de las aplicaciones tradicionales del cálculo de predicados es el análisis de razonamientos formulados en lenguaje natural. En esta sección extenderemos, entonces, el análisis que realizamos en el capítulo dedicado a lógica proposicional.

Utilizar el cálculo de predicados que hemos ido introduciendo en este capítulo nos permitirá formalizar razonamientos que no podían ser formalizados utilizando sólo lógica proposicional. Así, no sólo utilizaremos variables proposicionales para representar proposiciones atómicas, sino que analizaremos la estructura de estas proposiciones utilizando predicados y cuantificadores.

Veamos un ejemplo de razonamiento bastante conocido:

Todos los hombres son mortales
Sócrates es hombre

Sócrates es mortal

Para modelar este razonamiento es necesario introducir predicados que describan algunas de las categorías que lo conforman. Por ejemplo, para formalizar la idea de 'ser hombre' es necesario un predicado, que llamaremos $H.x$ que establezca si la variable x satisface o no la condición de 'ser

hombre'. Una situación similar ocurre con la idea de 'ser mortal'. Ahora, visto que Sócrates es un hombre particular, para representarlo necesitaremos una constante, que llamaremos s. Resumiendo hasta aquí tenemos:

$H.x$: x es hombre
$M.x$: x es mortal
s: Sócrates
Utilizando el cuantificador universal el razonamiento puede escribirse de la siguiente manera:

$$\frac{\langle \forall x : H.x : M.x \rangle \quad H.s}{M.s}$$

Utilicemos nuestros conocimientos de cálculo proposicional y de predicados para demostrar que el razonamiento es válido. Para ello partimos de la conjunción de las premisas para arribar a que éstas implican a la conclusión:

$$\begin{array}{ll} & \langle \forall x : H.x : M.x \rangle \wedge H.s \\ \Rightarrow & \{ \text{ Instanciación } \forall \text{ y monotonía del } \wedge \ \} \\ & (H.s \Rightarrow M.s) \wedge H.s \\ \Rightarrow & \{ \text{ Modus ponens } \} \\ & M.x \end{array}$$

Un tipo de razonamientos que tuvo gran popularidad y fue considerado el paradigma de los razonamientos válidos es el de los silogismos. Estos son ejemplos de razonamientos correctos a los cuales Aristóteles pensaba que se podían reducir todos los razonamientos. Si bien hoy se sabe que no es así, estos constituyen un ejemplo interesante para trabajar con nuestros métodos.

El primer silogismo que consideraremos será que posee dos premisas cuantificadas universalmente y una conclusión también universal. Es siguiente razonamiento es un ejemplo:

$$\frac{\text{Las arañas son mamíferos} \quad \text{Los mamíferos tienen seis patas}}{\text{Las arañas tienen seis patas}}$$

Actividad

1. Establecé los tres predicados que son necesarios para formular el razonamiento.
2. Formalizá el razonamiento utilizando expresiones cuantificadas.
3. Considerá las siguientes transformaciones:

$$\begin{array}{ll} & \langle \forall x :: a.x \Rightarrow m.x \rangle \wedge \langle \forall x :: m.x \Rightarrow s.x \rangle \\ \equiv & \{ \text{ Regla del término } \} \\ & \langle \forall x :: (a.x \Rightarrow m.x) \wedge (m.x \Rightarrow s.x) \rangle \\ \Rightarrow & \{ \text{ Transitividad, usando la monotonía de } \wedge \text{ y del } \forall \ \} \\ & \langle \forall x :: a.x \Rightarrow s.x \rangle \end{array}$$

   a) Escribilos en EQU para averiguar si son válidos o no.
   b) Son estas transformaciones una demostración válida del razonamiento? ¿por que?

El siguiente tipo de silogismos que consideraremos será aquel que posee una premisa cuantificada universalmente, otra cuantificada existencialmente y una conclusión con un cuantificador existencial. El siguiente es un ejemplo:

$$\frac{\text{Las vacas son mamíferos} \quad \text{Hay animales que no son mamíferos}}{\text{Hay animales que no son vacas}}$$

Actividad

1. Introduciendo los predicados necesarios, formalizá el razonamiento.
2. Utilizando modus ponens, contrapositiva y el siguiente teorema ya demostrado: $\langle \exists x :: t_1.x \rangle \wedge \langle \forall y :: t_2.y \rangle \Rightarrow \langle \exists x :: t_1.x \wedge t_2.x \rangle$, construí en FUN una demostración para este razonamiento.

Actividad

Utilizando tus conocimientos de cálculo proposicional y de predicados formalizá y demostrá en FUN los siguientes razonamientos:

1. Algún mamífero es ovíparo, por lo tanto algún ovípero es mamífero.
2. No es cierto que existan reyes pobres, por lo tanto no es cierto que existan pobres que sean reyes.
3. Ningún hombre es un ángel. Por lo tanto ningún ángel es un hombre.
4. Si algunos unicornios no están en mi casa, entonces existe algún unicornio.
5. Todo elefante es un animal, luego todo elefante gris es un animal gris. Para demostrarlo podés usar el siguiente teorema de cálculo proposicional: $(p \Rightarrow q) \Rightarrow (p \wedge r \Rightarrow q \wedge r)$

Actividad

Analizá los siguientes razonamientos, traduciendo cada paso a notación lógica, y descubrí en qué paso fallan. Destacá también aquellos pasos que sean correctos.

1. Ningún matemático ha logrado cuadrar el círculo. Luego, nadie que haya cuadrado el círculo es matemático. Por lo tanto, todos los que han cuadrado el círculo son no-matemáticos. Entonces, algunos de los que han cuadrado el círculo son no-matemáticos. Luego, algún no-matemático ha cuadrado el círculo.
2. Es verdad que ningún unicornio está en mi casa. Luego, es falso que todos los unicornios estén en mi casa. Por lo tanto, es verdad que algunos unicornios no están en mi casa. Con lo que concluimos que existe algún unicornio.
3. Estas demostraciones no son correctas, pero esto no significa que de la premisa no se pueda llegar a la conclusión final ¿Cómo construirías una demostración que pruebe que es válido partir de la primer premisa del primer razonamiento y llegar a su conclusión?

Capítulo 10

Expresiones cuantificadas

10.1. Sumatoria

Una notación muy útil usada en matemática es la que nos permite aplicar operaciones a una secuencia de expresiones las cuales dependen de alguna variable.

Ejemplos de esta notación es la que desarrollamos en el capítulo anterior para el cuantificador existencial y universal. Otro ejemplo paradigmático es la sumatoria. Por ejemplo, es usual escribir expresiones como:

$$\sum_{i=0}^{n-1} 2 * i + 1$$

La cual se puede interpretar como la suma de los primeros n números impares.
Analicemos en detalle esta notación. En ella se distinguen varias componentes:

- Aparece el símbolo de la sumatoria, que representa a un **operador** binario —toma dos elementos—, en este caso la suma.
- En la notación también hay **variables**, en este caso i, las cuales tienen un rango de variación: i puede tomar valores entre 0 y $n - 1$.
- Por último, hay una expresión que indica cuáles serán los términos a sumar. En este caso, $2 * i + 1$.

Así, para $n = 3$ la sumatoria da como resultado:

$$\sum_{i=0}^{3-1} 2 * i + 1 = \sum_{i=0}^{2} 2 * i + 1 = (2 * 0 + 1) + (2 * 1 + 1) + (2 * 2 + 1) = 1 + 3 + 5 = 9$$

Y, en general, para un n dado la sumatoria puede reducirse a una expresión aritmética de la forma:

$$\sum_{i=0}^{n-1} 2 * i + 1 = 1 + 3 + 5 + \ldots + 2 * (n - 1)$$

Si bien la notación con puntos suspensivos puede ayudarnos a interpretar mejor cuál el el significado de una sumatoria, utilizar sólo la notación con puntos suspensivos puede dar lugar a confusiones. Así, si la suma de los primeros n impares sólo se indica con la expresión:

$$1 + 3 + 5 + \ldots + 2 * (n - 1)$$

se asume que el lector puede discernir que esta expresión no está bien definida cuando n es 0 o 1. Sin embargo, estas ambigüedades se vuelve más perniciosas cuando se desarrollan formalmente los

programas dado que las expresiones que se manipulan suelen ser más complejas haciendo necesario tener más cuidado con los casos límite.

Hasta aquí hemos visto que la notación con la sumatoria es más ventajosa que la notación con puntos suspensivos. Ahora analicemos con más detalle la primer notación.

Actividad

1. Utilizando la notación con sumatoria, escribí expresiones que modelicen las siguientes oraciones:

   a) La suma de las potencias cuadradas de los primeros n números naturales tales que son divisibles por 2 y no son divisibles por 5

   b) La suma de $\frac{4*j}{4-i}$ donde j sólo puede tomar valores pares entre 5 y n e i sólo puede tomar valores entre 0 y $n+1$

2. ¿El resultado de la siguiente sumatoria:

   $$\sum_{i=1}^{3} 1 + i^2$$

   es 17 o $3 + n^2$?

La actividad anterior nos permitió explorar las limitaciones de la notación con sumatoria. Dentro de este curso utilizaremos una notación similar a la introducida para los cuantificadores existencial y universal que nos permitirá superar estas limitaciones. La sumatoria de los primeros n impares se escribirá como sigue:

$$\langle \sum i \ : \ 0 \leqslant i < n \ : \ 2 * i + 1 \rangle$$

Describamos en detalle esta notación:

- Aparece un símbolo que representa al operador cuantificado, en este caso $\sum$.
- A continuación se presenta la variable que está cuantificada, i, que sólo tiene sentido dentro de los símbolos $\langle \ \rangle$. Cuando hayan más de una variable cuantificada estas se enumerarán separadas por comas.
- Los paréntesis $\langle \ \rangle$ determinan exactamente el alcance de las variables cuantificadas.
- El **rango** de la variable cuantificada se establece a través de un **predicado**, en este caso $0 \leqslant i < n$.
- Por último, aparece la expresión que indica qué términos se sumarán: $2 * i + 1$. Esta parte de la cuantificación se denomina **término**. Notá que como el operador que se está cuantificando es la suma, el término es de tipo Num porque la suma toma parámetros de ese tipo.
- Los símbolos ":" nos permiten delimitar claramente el rango y el término.

Actividad

Traducí a esta notación las oraciones y la sumatoria de la actividad anterior.

Otra de las ventajas que nos ofrece esta notación es que se pueden enunciar reglas explícitas para el manejo de expresiones del estilo de la sumatoria, las cuales nos asegurarán la corrección de las operaciones realizadas con ellas. De esta tarea nos ocuparemos en este capítulo.

10.1.1. Axiomas para la sumatoria

En esta sección nos ocuparemos de presentar un conjunto de axiomas para la sumatoria lo que nos permitirá manipular con facilidad expresiones cuantificadas que involucren sumatorias.

Los dos primeros axiomax que presentaremos se refieren al caso en que el rango de la cuantificación es vacío, es decir, es un predicado constantemente equivalente a $false$ y al caso en que el rango es unitario, respectivamente:

Conceptos teóricos

Rango vacío:

$$\langle \sum i \ : \ False \ : \ t.i \rangle = 0$$

Conceptos teóricos

Rango unitario:

$$\langle \sum i \ : \ i = N \ : \ t.i \rangle = t.N$$

Notá que los axiomas se formulan en términos de igualdades y no de equivalencias porque el resultado de la sumatoria es un número.

Actividad

Para pensar en cómo podría formularse el axioma de partición de rango para la sumatoria te proponemos que analices la siguiente situación:
Visto que el predicado $0 \leqslant i < 10$ puede reescribirse como $(0 \leqslant i < 7) \vee (5 \leqslant i < 10)$ Decidí si la expresión cuantificada:

$$\langle \sum i \ : \ 0 \leqslant i < 10 \ : \ i^2 \rangle$$

podría reemplazarse por la siguiente expresión:

$$\langle \sum i \ : \ 0 \leqslant i < 7 \ : \ i^2 \rangle + \langle \sum i \ : \ 5 \leqslant i < 10 \ : \ i^2 \rangle$$

Si te ayuda podés evaluar las expresiones para calcular sus resultados.

A partir de la actividad anterior podemos concluir que para poder formular el axioma de partición de rango es preciso que los rangos involucrados sean disjuntos, es decir, que ninguna de las variables cuantificadas satisfaga ambos a la vez.

Cuando formulamos los axiomas de partición de rango para los cuantificadores existencial y universal no era precisa esta condición. Esto se debe a que ambos generalizan a la disyunción y la conjunción respectivamente y a que estos operadores son ambos idempotentes ($p \wedge p \equiv p$ y $p \vee p \equiv p$). Entonces, si considerábamos «casos repetidos» el resultado no se alteraba. La suma no es un operador idempotente, ciertamente $p + p \neq p$, por lo tanto es preciso agregar la cláusula que requiere que los rangos sean disjuntos. El axioma queda formulado entonces de la siguiente manera:

Conceptos teóricos

Partición de rango: Si $r_1.i$ y $r_2.i$ son disjuntos,

$$\langle \sum i \ : \ r_1.i \vee r_2.i \ : \ t.i \rangle = \langle \sum i \ : \ r_1.i \ : \ t.i \rangle + \langle \sum i \ : \ r_2.i \ : \ t.i \rangle$$

Actividad

Utilizando los axiomas introducidos hasta aquí, completá las justificaciones que permiten las siguientes transformaciones con expresiones que involucran sumatorias:

1. $\langle \sum i : 2 \leqslant i \leqslant 8 : i^3 - 1 \rangle$
   $\equiv \{ \dots \}$
   $\langle \sum i : i = 2 \vee 2 < i \leqslant 8 : i^3 - 1 \rangle$
   $\equiv \{ \dots \}$
   $\langle \sum i : i = 2 : i^3 - 1 \rangle + \langle \sum i : 2 < i \leqslant 8 : i^3 - 1 \rangle$
   $\equiv \{ \dots \}$
   $7 + \langle \sum i : 2 < i \leqslant 8 : i^3 - 1 \rangle$
2. $\langle \sum j : 0 < j \leqslant 7 : \frac{1}{j} \rangle + \langle \sum j : 8 < j \leqslant 13 : \frac{1}{j} \rangle$
   $\equiv \{ \dots \}$
   $\langle \sum j : 0 < j \leqslant 13 : \frac{1}{j} \rangle$

Actividad

Pensemos ahora en cómo formularíamos el axioma comúnmente llamado **regla del término**. Pensando en las propiedades de la suma y tomando como ejemplo las reglas del término para los cuantificadores universal y existencial, pensá por qué expresión podría reemplazarse la siguiente cuantificación:

$$\langle \sum i : r.i : t_1.i + t_2.i \rangle = \dots$$

El siguiente axioma hace uso de la propiedad distributiva de la suma con respecto al producto y de la conmutatividad de ambos operadores:

Conceptos teóricos

Distributividad: Si $r.x$ es no vacío:

$$\langle \sum i : r.i : x * t.i \rangle = x * \langle \sum i : r.i : t.i \rangle$$

y

$$\langle \sum i : r.i : t.i * x \rangle = \langle \sum i : r.i : t.i \rangle * x$$

Actividad

¿Por qué es necesaria la condición de que el rango sea no vacío en el axioma de distributividad? Pensá en algún ejemplo concreto que ponga en evidencia esta necesidad.

Los dos últimos axiomas que presentaremos para la sumatoria son los que tratan de la posibilidad de cambiar las variables cuantificadas y del caso en que hay más de una variable cuantificada y el rango es una conjunción de predicados, uno de los cuales es independiente de alguna de las variables cuantificadas:

Conceptos teóricos

Anidado:

$$\langle \sum i, j : r_1.i \wedge r_2.i.j : t.i.j \rangle = \langle \sum i : r_1.i : \langle \sum j : r_2.i.j : t.i.j \rangle \rangle$$

Conceptos teóricos

Cambio de variable: Para toda f biyectiva y para cualquier j que no aparezca en r ni en t

$$\langle \sum i \ : \ r.i \ : \ t.i \rangle = \langle \sum j \ : \ r.f(.j) \ : \ t.f(j) \rangle$$

Actividad

Utilizando tus conocimientos sobre la sumatoria escribí una expresión cuantificada que sirva para describir la siguiente situación:

- Dada una constante c y una lista de números xs, c es la suma de los elementos de xs

10.2. Generalizando las expresiones cuantificadas

Hasta aquí hemos presentado expresiones cuantificadas que generalizan a los operadores conjunción —cuantificador universal—, disyunción —cuantificador existencial— y suma —sumatoria. También dimos los axiomas asociados que permiten manipular expresiones que involucren a estos cuantificadores.

En esta sección vamos a generalizar este mecanismo para definir expresiones cuantificadas para cualquier operador binario asociativo y conmutativo, por ejemplo $*$, max, min, la unión de conjuntos, etc. Esto nos permitirá que las reglas provistas se apliquen aun conjunto grande de expresiones, las cuales serán suficientes para especificar casi todos los problemas con los que nos enfrentaremos en este curso. Así, nuestro lenguaje se habrá vuelto lo suficientemente rico como para tener la potencia expresiva que necesitamos.

En diversos contextos —aritmética, lógica, teoría de conjuntos, etc.— aparece cierta noción de cuantificación, entendiéndose por esto al uso de variables formales con un alcance delimitado explícitamente las cuales pueden usarse para construir expresiones dependientes de estas pero sólo dentro de ese alcance.

Propondremos una notación unificada para estas expresiones. Esta notación debe tener en cuenta al operador con el cual se cuantifica (hasta aquí tenemos a la suma, la conjunción y la disyunción), las variables que van a usarse para crear las expresiones (denotadas generalmente con las letras i, j, k, ...), el rango de variación de estas variables y la expresión dependiente de las variables que define los términos de la cuantificación.

Una **expresión cuantificada** será entonces de la siguiente forma:

$$\langle \oplus i \ : \ r.i \ : \ t.i \rangle$$

donde:

- $\oplus$ designa a un operador asociativo y conmutativo.
- $r.i$ es un predicado que se denomina **rango de especificación**.
- $t.i$ es una función denominada **término** de la cuantificación.
- Se utiliza i para denotar a una secuencia de variables en la cual no importa el orden, usando la expresión $V.i$ para denotar al conjunto de todas las variables de i.
- La ocurrencia de i junto al operador suele llamarse **variable de cuantificación**. La variable cuantificada sólo tiene sentido dentro de los símbolos $\langle \ \rangle$

El tipo de la variable cuantificada puede inferirse del contexto, en caso contrario se lo definirá explícitamente. Para los fines de la cuantificación la propiedad de que una variable sea de un tipo dado es equivalente a la de pertenecer al conjunto de valores de ese tipo. En este sentido, vamos

a considerar a los tipos como un predicado más. Así, si se quisiera hacer explícito el tipo de las variables cuantificadas en la siguiente sumatoria, usaremos el siguiente rango:

$$\langle \sum i \ : \ i \in \mathbb{N} \wedge 0 \leqslant i < n \ : \ i + 9 \rangle$$

En otras ocasiones no se desea restringir el rango de especificación al conjunto de variables que satisfacen un predicado r, como hemos escrito más arriba. En estos casos escribiremos $\langle \oplus i \ : : \ t.i \rangle$, entendiéndose que el rango abarca a todos los elementos del tipo de i.

Existe una serie de reglas que sirven para manipular expresiones cuantificadas. Las enunciaremos para un operador general $\oplus$ y luego las ejemplificaremos aplicándolas a una serie de operadores usuales. En lo que sigue, vamos a utilizar $True$ y $False$ para indicar los predicados constantemente iguales a "verdadero" y "falso" respectivamente. Por lo dicho anteriormente, $\langle \oplus i \ : : \ t.i \rangle \equiv \langle \oplus i \ : \ True \ : \ t.i \rangle$

10.2.1. Variables libres, ligadas y alcance

Uno de los conceptos esenciales para comprender y manejar a las expresiones cuantificadas es el de **variable ligada**. Asociados a este concepto están el complementario de **variable libre** y el de **alcance**.

Comencemos analizando el ejemplo dado en la sección referida a la sumatoria:

$$\langle \sum i \ : \ 0 \leqslant i < n \ : \ 2 * i + 1 \rangle$$

El valor de esta sumatoria depende de n. Por ejemplo, cuando n sea 1 su resultado será 1, cuando n sea 2 su resultado será 4 y así sucesivamente. Sin embargo, el valor de la sumatoria no depende de i. Esta variable podría cambiarse por otra sin que el valor de la sumatoria cambie, por ejemplo $\langle \sum i \ : \ 0 \leqslant i < n \ : \ 2 * i + 1 \rangle$. Diremos que las ocurrencias de la variable i (o j en la segunda versión) en el rango y en el término están **ligadas** a la variable que aparece junto a la sumatoria.

Ahora analicemos una expresión un poco más complicada:

$$i + \langle \sum i \ : \ 0 \leqslant i < n \ : \ 2 * i + 1 \rangle$$

La ocurrencia de i antes de la suma no tiene nada que ver con las ocurrencias dentro del alcance de la expresión cuantificada. Las ocurrencias internas de la sumatoria tienen ya su rango especificado. Por ejemplo, en un estado en el que la i de fuera de la sumatoria tome el valor 6 y que n tome el valor 3 el valor de la sumatoria será:

$$6 + \langle \sum i \ : \ 0 \leqslant i < 3 \ : \ 2 * i + 1 \rangle$$

o sea

$$6 + 1 + 3 + 5$$

A continuación generalizaremos una definición de variable libre similar a la que dimos en el capítulo referido a los cuantificadores universal y existencial:

Conceptos teóricos

Variable libre: Se define inductivamente cuándo una variable está libre en una expresión.

- Una variable i está libre en la expresión i.
- Si la variable i está libre en E, entonces también lo está en (E).
- Si la variable i está libre en E y f es una operación válida en el tipo de E, entonces i también está libre en $f(\ldots, E, \ldots)$.
- Si la variable i está libre en E y no aparece en la secuencia de varaiables x, es decir, $x \notin V.x$, entonces lo está también en $\langle \oplus x \ : \ F \ : \ E \rangle$ y en $\langle \oplus x \ : \ E \ : \ F \rangle$

Notación: Dada una expresión E, el conjunto de las variables libres de E se denotará por $VL.E$.

Conceptos teóricos

Sea i una variable libre en la expresión E, tal que $i \in V.x$, luego la variable i está ligada a la variable de cuantificación correspondiente en las expresiones $\langle \oplus x : F : E\rangle$ y $\langle \oplus x : E : F\rangle$.
Se extiende la definición inductivamente. Si i está ligada en E, también estará ligada a la misma variable de cuantificación en (E), $f(\ldots, E, \ldots)$, $\langle \oplus x : E : F\rangle$ y en $\langle \oplus x : F : E\rangle$.

Notación: Dada una expresión E, el conjunto de variables ligadas de E se denotará por $VG.E$.

Actividad

Dadas las definiciones anteriores determina, para las siguientes expresiones, cual es el conjunto de variables libres y de variables ligadas a cada una:

1. $\langle \forall i, j : m \leqslant i < j \leqslant n : \frac{i+m}{j+n} < s\rangle$
2. $\langle \exists z : par.z : z + 1\rangle lorz \Rightarrow x$
3. $\langle \sum h : s < h : h\rangle * \langle \sum h : s < h : h\rangle$

10.2.2. Cambio de variables y colisión de variables

Hemos visto que las variables ligadas a una cuantificación tienen su alcance delimitado de manera explícita por los símbolos $\langle$ y $\rangle$. Si se cambian por un nombre "fresco" (que no aparezca en el alcance), el significado de la expresión no cambiará, como lo ejemplifica la siguiente igualdad:

$$\langle \sum i : 0 \leqslant i \leqslant n : 2 * i + 1\rangle = \langle \sum j : 0 \leqslant j \leqslant n : 2 * j + 1\rangle$$

Sin embargo, debe tenerse cuidado con las colisiones de nombres. Tomemos sólo el lado izquierdo de la igualdad anterior:

$$\langle \sum i : 0 \leqslant i \leqslant n : 2 * i + 1\rangle$$

Si, por ejemplo, se reemplaza i por n se obtiene una expresión obviamente diferente (que, en particular, será siempre igual a 0, sin importar el valor de n):

$$\langle \sum n : 0 \leqslant n \leqslant n : 2 * n + 1\rangle$$

Esta situación hace preciso que la regla de sustitución sea reformulada de la siguiente forma:

Conceptos teóricos

Sustitución para expresiones cuantificadas

$$V.y \cap (V.x \cup VL.E) = \emptyset \Rightarrow$$

$$\langle \oplus y : r : t\rangle(x := E) = \langle \oplus y : r(x := E) : t(x := E)\rangle$$

Si la condición no se cumple, es preciso que en primer lugar se renombre a la variable de cuantificación y luego se realice la sustitución.

Actividad

Utilizando la nueva regla de sustitución para expresiones cuantificadas decidí si en la siguiente expresión $\langle \sum i : n \leqslant i \leqslant 8 : i^3 + 1\rangle$ es posible realizar las siguientes sustituciones:

1. $i := j$
2. $i := j + 1$
3. $i := j + n$

Revisemos ahora qué ocurre con la regla de Leibniz. Recordemos rápidamente que la actual regla de Leibniz establecía que:

$$\frac{X = Y}{E(x := X) = E(x := Y)}$$

Ahora bien, sabemos que la siguiente igualdad es válida:

$$2 + i + 1 = 2 * (i + 1) - 1$$

Usando la regla de Leibniz tal como está formulada hasta ahora podríamos afirmar la siguiente transformación:

$$\frac{2 + i + 1 = 2 * (i + 1) - 1}{\langle \sum i \ : \ 0 \leqslant i < n \ : \ y \rangle (y := 2 * i + 1) = \langle \sum i \ : \ 0 \leqslant i < n \ : \ y \rangle (y := 2 * (i + 1) - 1}$$

Pero, dado que la variable i aparece en la lista de variables de cuantificación es necesario primero renombrar las variables ligadas y luego hacer la sustitución. Esto implica que el resultado de

$$\langle \sum i \ : \ 0 \leqslant i < n \ : \ y \rangle (y := 2 * i + 1)$$

será

$$\langle \sum j \ : \ 0 \leqslant j < n \ : \ 2 * i + 1 \rangle$$

y no el esperado. Es preciso, entonces, generalizar la regla de Leibniz para las expresiones cuantificadas, agregando una regla para el rango y para el término:

$$\frac{X = Y}{\langle \oplus i \ : \ r(i := X) \ : \ t \rangle = \langle \oplus i \ : \ r(i := Y) \ : \ t \rangle}$$

$$\frac{X = Y}{\langle \oplus i \ : \ r \ : \ t(i := X) \rangle = \langle \oplus i \ : \ r \ : \ t(i := Y) \rangle}$$

10.3. Reglas generales para las expresiones cuantificadas

En esta sección enunciaremos de manera general los axiomas y algunas propiedades para expresiones cuantificadas. Les daremos nombres a los axiomas y propiedades para poder mencionarlos luego en las demostraciones o derivaciones en los que sean pertinentes.

Conceptos teóricos

Rango vacío: Cuando el rango de especificación es vacío, la expresión cuantificada es igual al neutro e del operador $\oplus$:

$$\langle \oplus i \ : \ False \ : \ t \rangle = e$$

Si $\oplus$ no posee elemento neutro, la expresión no está bien definida.

Conceptos teóricos

Rango unitario: Si el rango de especificación consiste en un solo elemento, la expresión cuantificada es igual al término evaluado en dicho elemento:

$$\langle \oplus i \ : \ i = N \ : \ t \rangle = t.N$$

Conceptos teóricos

Partición de rango: Cuando el rango de especificación es de la forma $r_1 \vee r_2$ y además se cumple al menos una de las siguientes condiciones:

- el operador $\oplus$ es idempotente,
- los predicados r_1 y r_2 son disjuntos,

la expresión cuantificada puede reescribirse como sigue:

$$\langle \oplus i \,:\, r_1 \vee r_2 \,:\, t \rangle = \langle \oplus i \,:\, r_1 \,:\, t \rangle \oplus \langle \oplus i \,:\, r_2 \,:\, t \rangle$$

El hecho que la igualdad es una relación simétrica, permite indistintamente reemplazar cualquiera de los dos miembros del axioma anterior por el otro, vale decir que la regla de partición de rango puede leerse también de derecha a izquierda. Lo mismo ocurre con todas las reglas que siguen:

Conceptos teóricos

Partición de rango generalizada: Si el operador $\oplus$ es idempotente y el rango de especificación es una cuantificación existencial, entonces:

$$\langle \oplus i \,:\, \langle \exists j \,:\, r.i.j \,:\, s.i.j \rangle \,:\, t \rangle = \langle \oplus i, j \,:\, r.i.j \wedge s.i.j \,:\, t \rangle$$

Conceptos teóricos

Regla del término: Cuando el operador $\oplus$ aparece en el término de la cuantifiación, la expresión cuantificada puede reescribirse de la siguiente manera:

$$\langle \oplus i \,:\, r \,:\, t_1 \oplus t_2 \rangle = \langle \oplus i \,:\, r \,:\, t_1 \rangle \oplus \langle \oplus i \,:\, r \,:\, t_2 \rangle$$

Conceptos teóricos

Regla del término constante: Si el término de la cuantificación es constantemente igual a C, el operador $\oplus$ es idempotente y el rango de especificación es no vacío, entonces:

$$\langle \oplus i \,:\, r \,:\, C \rangle = C$$

Conceptos teóricos

Distributividad: Si el operador $\otimes$ es distributivo a izquierda con respecto al operador $\oplus$ y se cumple al menos una de las siguentes condiciones:

- el rango de especificación es no vacío,
- el elemento neutro del operador $\oplus$ existe y es absorvente para $\otimes$,

entonces:

$$\langle \oplus i \,:\, r \,:\, t \otimes x \rangle = \langle \oplus i \,:\, r \,:\, t \rangle \otimes x$$

Conceptos teóricos

Anidado: Cuando hay más de una variable cuantificada y el rango de especificación es una conjunción de predicados, uno de los cuales es independiente de alguna de las variables de cuantificación, la expresión cuantificada puede reescribirse de la siguiente manera:

$$\langle \oplus i, j \,:\, r_1.i \wedge r_2.i.j \,:\, t.i.j \rangle = \langle \oplus i \,:\, r_1.i \,:\, \langle \oplus j \,:\, r_2.i.j \,:\, t.i.j \rangle \rangle$$

Conceptos teóricos

Cambio de variable: Si $V.j \cap (VL.r \cup VL.t) = \emptyset$ pueden renombrarse las variables:

$$\langle \oplus i \,:\, r.i \,:\, t.i \rangle = \langle \oplus j \,:\, r.j \,:\, t.j \rangle$$

Todos estos axiomas pueden particularizarse para referirse a operadores concretos. Es lo que haremos más adelante, con los operadores más usuales.

Pasemos ahora a enunciar de manera general algunos teoremas fundamentales.

Conceptos teóricos

Teorema cambio de variable generalizado: Si f es una función que tiene inversa en el rango de especificación y j es una variable que no aparece en r ni en t, las variables cuantificadas pueden renombrarse como sigue:

$$\langle \oplus i \,:\, r.i \,:\, t.i \rangle = \langle \oplus j \,:\, r.(f.j) \,:\, t.(f.j) \rangle$$

Conceptos teóricos

Teorema separación primer término:

$$\langle \oplus i \,:\, 0 \leqslant i < n+1 \,:\, t.i \rangle = t.0 \oplus \langle \oplus i \,:\, 0 \leqslant i < n \,:\, t.(i+1) \rangle$$

Conceptos teóricos

Teorema separación último término:

$$\langle \oplus i \,:\, 0 \leqslant i < n+1 \,:\, t.i \rangle = \langle \oplus i \,:\, 0 \leqslant i < n \,:\, t.i \rangle \oplus t.n$$

Actividad

Utilizando los axiomas presentados demostrá los teoremas de separación del primer y último término presentados anteriormente.

En las secciones siguientes nos dedicaremos a instanciar los axiomas y teoremas presentados de manera general en esta sección a operadores particulares, tales como la unión de conjuntos, el máximo y el mínimo, la productoria y otros operadores construidos a partir de cuantificadores ya presentamos.

10.4. Productoria

Así como la sumatoria es una cuantificación que generaliza a la suma, la productoria generaliza al producto. Así, si quisiéramos expresar el producto de los primeros n números naturales, escribiríamos una expresión cuantificada como la siguiente:

$$\langle \prod x \,:\, 0 \leqslant x \leqslant n \,:\, x \rangle$$

A través de la siguiente actividad te proponemos que, a partir de lo presentado en la sección anterior, definas los axiomas y teoremas correspondientes a la productoria.

Actividad

A partir de los axiomas y teoremas para expresiones cuantificadas en general presentados en la sección anterior y teniendo en cuenta que el producto no es un operador idempotente y que su elemento neutro es el 1:

- Definí los axiomas que correspondan a este cuantificador.
- Explicá por qué algunos axiomas generales no se pueden instanciar para este cuantificador.
- Enunciá y demostrá los teoremas correspondientes.

Actividad

A partir de tus conocimientos sobre la productoria, escribí una expresión cuantificada para describir la siguiente situación:

- Dadas una constante numérica c y una lista de números xs, c es el producto de todos los elementos de xs.

10.5. El operador de conteo

Hasta aquí hemos mencionado únicamente cuantificadores que provienen de un operador conmutativo y asociativo. Pero también es posible definir nuevos cuantificadores a partir de otros ya definidos. Éste es el caso del cuantificador de conteo, designado comúnmente por el símbolo N, el cual cuenta la cantidad de elementos en el rango de especificación que satisfacen el término de la cuantificación:

$$\langle N\ i\ :\ r.i\ :\ t.i\rangle = \langle \sum i\ :\ r.i \wedge t.i\ :\ 1\rangle$$

Actividad

1. ¿De qué tipo deben ser el rango y el término del cuantificador de conteo?
2. Utilizarse este cuantificador expresá formalmente la siguiente oración: Cuantos números naturales impares son menores que 20.
3. Calculá el valor de las siguientes expresiones:

   a) $\langle N\ i\ :\ 0 \leqslant i \leqslant 9\ :\ par.i\rangle$
   b) $\langle N\ j\ :\ -4 \leqslant j \leqslant 4\ :\ j \geqslant 0\rangle$
   c) $\langle N\ k\ :\ par.k\ :\ 3 * k \leqslant 15\rangle$

Actividad

A partir de la definición del operador de conteo y de los axiomas y teoremas que enunciamos para la sumatoria, enunciá los axiomas que correspondan a este operador y demostrá los teoremas más importantes para el cuantificador N

Actividad

Dado el siguiente teorema que vincula a la sumatoria y al operador de conteo:

$$\langle \sum i\ :\ r.i \wedge t.i\ :\ k\rangle = k * \langle N\ i\ :\ r.i\ :\ t.i\rangle$$

1. Escribí con tus palabras cuál es la propiedad que captura este teorema.
2. Construí en FUN una demostración para el mismo.

10.6. Máximo y Mínimo

Trabajemos ahora con los cuantificadores que generalicen los operadores máximo y mínimo.

Actividad

En primer lugar, pensemos sobre los neutros de los operadores máximo y mínimo. El elemento neutro del máximo, e_{max}, deberá cumplir que:

$$max.a.e_{max} = a$$

y el elemento neutro del mínimo, e_{min}:

$$min.a.e_{min} = a$$

1. Si restringimos a los operadores máximo y mínimo a los naturales, ¿cuáles serían sus elementos neutros?
2. ¿Qué sucede si queremos definir estos operadores para el conjunto de los números enteros?

Para referirnos a los cuantificadores que generalizan al máximo y al mínimo utilizaremos los símbolos Max y Min respectivamente. Así, si queremos especificar formalmente la frase: 'el mínimo elemento de una lista xs', utilizaremos una expresión cuantificada como la siguiente:

$$\langle Min\ i\ :\ 0 \leqslant i < \#xs\ :\ xs.i \rangle$$

y si queremos especificar la oración: 'la raíz mayor de la ecuación de segundo grado $x^2 + 2 * x - 1$', utilizaremos la siguiente expresión cuantificada:

$$\langle Max\ i\ ::\ x^2 + 2 * x - 1 = 0 \rangle$$

Actividad

Teniendo en cuenta la actividad anterior y que ambos operadores son idempotentes, es decir que $max.a.a = a$ y que $min.a.a = a$, definí los axiomas y teoremas correspondientes a las expresiones cuantificadas que involucren a los operadores Max y Min

Actividad

A partir de tus conocimientos sobre el operador máximo cuantificado, escribí una expresión para describir la siguiente situación:

- Dada una constante numérica c y una lista xs, c es igual al máximo elemento de la lista xs.

Capítulo 11

El proceso de construcción de programas

En este capítulo muchos de los conceptos y estrategias que tratamos hasta aquí comienzan a vincularse estrechamente entre sí. Prácticamente todas las herramientas que venimos presentando se pondrán al servicio de la construcción de programas.

En primer lugar, presentaremos los diferentes procesos que están implicados en la construcción de un programa. Luego presentaremos distintas técnicas comúnmente utilizadas en la programación que nos permitirán construir programas complejos sobre listas y números.

11.1. El problema de construir programas

Una de las dificultades esenciales del proceso de construir programas radica en el hecho de que las descripciones de los problemas no suelen ser precisas ni completas.

Cuando surgieron las computadoras éstas eran utilizadas principalmente para resolver problemas científicos, los cuales estaban expresados en un lenguaje bastante preciso. Entre ellos estaban el cálculo de trayectorias de proyectiles (balística) y el cifrado y descifrado de códigos —que jugó un rol protagónico durante la Segunda Guerra Mundial—.

Pero al final de la década de 1960 los costos empezaron a bajar y las computadoras proliferaron en distintos ámbitos. Esto hizo que comenzaran a utilizarse para resolver problemas originados en los más diversos ámbitos. Estos nuevos problemas estaban expresados en un lenguaje mucho más vago y resultaron bastante más difíciles de resolver.

El método que se utilizaba en ese entonces era simplemente partir del problema, el cual estaba expresado de manera informal y poco detallada, y construir un programa que, por definición, está escrito en una notación formal. Se realizaba, entonces, un salto muy grande entre el problema y el programa y este salto era el causante de muchos de los errores de los softwares de la época.

Pero ahí no terminaban los inconvenientes. Para comprobar que el programa era correcto se realizaban ensayos con conjuntos de datos para los cuales se conocía el resultado y, si éstos daban los resultados esperados, se terminaba la tarea. Pero frecuentemente los resultados no eran los esperados. En este caso se procedía a modificar el programa para corregirlo. Pero esta tarea era sumamente difícil, porque no se sabía si el resultado inesperado se debía a errores de programación o a una concepción inadecuada del programa.

Mucho trabajo se ha llevado a cabo desde aquella época hasta nuestros días. Tanto la academia como la industria han invertido innumerables recursos y mucho dinero para desarrollar técnicas y herramientas que faciliten la construcción y la verificación de los programas.

Actividad

En los capítulos anteriores hemos definido una gran cantidad de funciones. Es decir, hemos construido numerosos programas.

- A partir de la lectura de los párrafos anteriores relacioná la forma en la que hemos construido los programas hasta ahora con la manera en la que se elaboraban los programas en aquella época.

En la actualidad es ampliamente aceptado que el proceso de construcción de programas debe dividirse en al menos dos etapas: la etapa de **especificación** del problema y la etapa de desarrollo del programa o de **programación**.

El resultado de la primera etapa es una **especificación formal** del problema, la cual seguirá siendo abstracta (poco detallada) pero estará escrita con precisión en algún lenguaje cuya semántica esté definida rigurosamente. La segunda etapa dará como resultado un programa y una demostración de que el programa es *correcto respecto de la especificación dada*. A esta demostración se la llama **verifiación.** Notá que no tiene sentido hablar de un programa correcto *per se*; la noción de corrección de un programa siempre está referida a una especificación dada.

La separación en dos pasos permite discernir ahora si un programa cuyos resultados no son los esperados es incorrecto o si, en cambio, es la especificación la que no describe al problema de manera adecuada. Es importante resaltar que la propiedad de *adecuación* entre problema y especificación no puede demostrarse formalmente, debido a que los problemas son de naturaleza informal. De todas maneras, dado que una especificación es más abstracta que un programa, es mucho más fácil convencerse de que una especificación expresa nuestra idea informal de un problema que determinar esto a partir del código de un programa.

Actividad

Construí un esquema que vincule las siguientes palabras claves en la discusión anterior: problema, formal, adecuación, especificación formal, programa, informal, verificación.

11.2. Especificaciones

En un sentido general podemos definir una especificación como *una descripción formal de la tarea que un programa tiene que realizar*. Así, las especificaciones responden a la pregunta *¿qué* hace el programa? mientras que el programa mismo responde a la pregunta *¿cómo* se realiza la tarea?

A veces se entiende a la especificación como un contrato entre el programador y el potencial usuario. Este contrato establece de manera precisa cuál es el comportamiento del producto que el programador debe proveer al usuario. Usualmente las especificaciones dicen que si el programa se ejecuta para un valor de un conjunto dado, el resultado satisfará una cierta propiedad. Una de las consecuencias de dicho contrato es que el usuario se compromete a usar el programa solo para valores en ese conjunto predeterminado y el programador a asegurar que el programa producirá un resultado satisfactorio. En principio si el usuario ingresa al programa datos que no están en el conjunto de datos aceptables, el comportamiento del programa puede ser cualquiera sin con ello violar el contrato establecido por la especificación.

En el contexto del lenguaje que estamos construyendo, podemos ser más precisos al definir qué es una especificación. Diremos que una **especificación** es un predicado que caracteriza los valores de entrada-salida esperados para nuestro programa.

La tarea de programación de un problema especificado por un predicado dado puede describirse ahora como encontrar una función que satisfaga dicho predicado.

Tomemos un ejemplo. Supongamos que queremos construir un programa que administre el desempeño de los alumnos de una materia. A primera vista parece un problema fácil pero si se lo examina con un poco más de detalle puede notarse que su formulación es demasiado imprecisa. Si se da ese enunciado a varios programadores es probable que cada uno obtenga ocho programas diferentes.

Consideremos un problema más especifico. Dado un conjunto de alumnos de un curso se pide obtener el alumno con el segundo promedio más alto. Este problema parece más simple, pero aún quedan varias cuestiones por resolver. Por ejemplo, puede ser que dicho alumno no exista (en caso de que todos los alumnos del curso tengan el mismo promedio) o que haya más de uno. Si nos quedamos solo con el enunciado informal, la solución para estos casos vuelve a quedar librada al criterio del programador.

En esta sección utilizaremos las herramientas que hemos desarrollado hasta ahora para escribir especificaciones precisas. Principalmente haremos uso del cálculo de predicados y de las expresiones cuantificadas.

Esta tarea nos permitirá resolver las ambigüedades en la formulación de los problemas. Por supuesto, siempre queda la pregunta de si la especificación formal construida efectivamente expresa el enunciado del problema a resolver. Esta pregunta es inevitable y es bueno responderla lo más temprano posible en el proceso de construcción de programas.

Por otro lado, no siempre una especificación puede ser satisfecha por un programa o la construcción de este puede ser muy costosa o su ejecución muy ineficiente. En estos casos es necesario cambiar la especificación.

Comenzamos esta tarea con una actividad adaptada de un ejemplo presente en Borda, Eisenbach, Khoshneisan & Vickers (1994).

Actividad

Un usuario le propone a un programador el problema de construir un programa que calcule la raíz cuadrada de un número real dado. El programador, después de pensar un poco y de recurrir a sus conocimientos en álgebra, le propone al usuario construir la función `raiz` especificada como sigue:

$$\texttt{raiz} : Num \to Num$$

$$\langle \forall x \,:\, 0 \leqslant x \,:\, (\texttt{raiz}.x)^2 = x \rangle$$

1. ¿Qué restricciones agregó el programador sobre la formulación informal del programa? ¿Por qué creés que las agregó?
2. ¿En qué parte del predicado aparecen estas restricciones?

Notá que una especificación sobre una función establece el comportamiento de la misma para todos los valores posibles en los cuales pueda aplicarse. Es por esta razón que la variable x estaba cuantificada universalmente en la especificación de la funcion `raiz`.

Actividad

Continuemos con la especificación de la función `raiz`. Nuestro programador se encuentra ahora con una segunda dificultad: el sistema de cómputo en el que implementará su programa solo maneja aproximaciones finitas a los números irracionales, por lo que le será imposible calcular, por ejemplo, la solución exacta de `raiz`.2. El problema es que la especificación que se presentó en la actividad anterior requiere que la solución sea exacta. Dada esta situación el programador necesita cambiar la especificación, es decir, cambiar el contrato con el usuario.
Al programador se le ocurre solucionar este problema de la siguiente forma: visto que no puede calcular una solución exacta puede hacer que la diferencia entre $(\texttt{raiz}.x)^2$ y x sea menor que una determinada constante, ϵ cuyo valor negociará con el usuario.
La especificación queda entonces de la forma:

$$\texttt{raiz} : Num \to Num$$

$$\langle \forall x \,:\, 0 \leqslant x \,:\, |(\texttt{raiz}.x)^2 - x| < \epsilon \rangle$$

1. ¿Creés que esta nueva especificación es más débil que la primera? Justificá tu respuesta.
2. ¿Qué otras formas hay de debilitar una especificación de este estilo?

Una especificación también puede *fortalecerse*. Esto ocurre cuando el usuario puede darse cuenta de que necesita un programa con mejores propiedades que el anterior, o poder usarlo para un conjunto de datos de entrada más amplio que el que se había pensado.

Actividad

Siguiendo con la especificación de la función `raiz` supongamos que el usuario le pide al programador fortalecer la especificación, requiriendo que el resultado de aplicar la función sea siempre positivo.

1. Proponé una nueva especificación de la función `raiz` que también satisfaga esta condición.
2. ¿Qué parte del cuantificador univesal se debe modificar si se quiere construir una especificación más fuerte?

A continuación vamos a dedicarnos a construir especificaciones formales para varias de las funciones sobre listas que hemos construido en los capítulos 3 y 5. También van a ser de utilidad los conocimientos que aprendiste en el capítulo sobre especificación (PONER REFERENCIA CUANDO ESTÉ)

Actividad

Un tipo de funciones que ya definimos eran los mapeos. Este conjunto de funciones tomaba una lista y devolvía otra cuyos elementos eran el resultado de aplicarle una función a cada uno de los elementos.

1. Construí una especificación formal para la función `duplicar` que toma una lista de números y devuelve otra lista de números. El resultado de aplicar esta función es una lista que contiene duplicados cada uno de los elementos de la lista original. Para construir la especificación puede serte útil pensar que la misma debe establecer una propiedad que deben satisfacer todos los elementos de la lista resultado de aplicar la función.
2. ¿Qué función es la que se especifica formalmente de la siguiente manera? Describila con tus palabras.

   $$g : [[Num]] \to [[Num]]$$

   $$\langle \forall i \ : \ 0 \leqslant i < \#xss \ : \ (g.xss)!i = 0 \triangleright (xss!i) \rangle$$

Actividad

Especifiquemos ahora algunas funciones que pertenecen al conjunto de los acumuladores que ya has definido en el capítulo 3. Recordá que todas las funciones de esta clase operan sobre todos los elementos de una lista, acumulando este resultado a medida que se realiza el cómputo.

1. Especificá formalmente la función `sum` que toma una lista de números y devuelve un número que es el resultado de sumar todos los elementos de la lista.
2. Especificá formalmente la función `prod` que toma una lista de números y devuelve un número que es el resultado de multiplicar todos los elementos de la lista.

Para construir las especificaciones de estas funciones puede serte útil recordar las distintas expresiones cuantificadas que estudiamos en el capítulo 10.

1. ¿Qué función se especifica formalmente de la siguiente manera? Describila con tus palabras.

   $$f : [Num] \to Num$$

   $$f.xs = \langle Max \ i \ : \ 0 \leqslant i < \#xs \ : \ xs!i \rangle$$

Actividad

Trabajemos con funciones un poco más complejas que has definido en el capítulo 5

1. La función `repetir` que toma dos naturales x y n y devuelve una lista con x repetido n veces.
2. La función `desdehasta` que toma dos naturales, n y m, y devuelve una lista cuyos elementos son los naturales desde n hasta m. Para que la función pueda ser definida es preciso que $n \leqslant m$.
3. La función `iguales` que dadas dos listas devuelve $True$ si ambas son iguales y $False$ en caso contrario.

11.3. Verificación

En la sección anterior construimos especificaciones para algunas de las funciones que ya hemos definido. A continuación nos concentraremos en verificar que dichas definiciones satisfacen la especificación. Llevar a cabo esta tarea involucra construir una demostración.

Tomemos como ejemplo la función `sum` especificada como sigue:

$$\texttt{sum} : [Num] \to Num$$

$$\langle \forall xs : : sum.xs = \langle \sum i : 0 \leqslant i < \#xs : xs!i \rangle \rangle$$

y definida así:

$$\begin{array}{rcl} \texttt{sum} & :: & [Num] \to Num \\ \texttt{sum}.[\,] & \doteq & 0 \\ \texttt{sum}.(x \triangleright xs) & \doteq & x + \texttt{sum}.xs \end{array}$$

Debemos construir una demostración que pruebe que, para cualquier lista xs, la función `sum` satisface la especificación. Notá que en este caso la variable xs es una variable libre cuantificada universalmente. Al mismo tiempo, xs es el argumento de la función `sum`. En general, para que la notación no sea engorrosa dejaremos implícita la cuantificación universal que pesa sobre los argumentos de una función para realizar las demostraciones. Esto implica que la especificación de la función `sum` puede escribirse directamente como:

$$\texttt{sum} : [Num] \to Num$$

$$sum.xs = \langle \sum i : 0 \leqslant i < \#xs : xs!i \rangle$$

Por supuesto, cuando llevemos a cabo nuestra demostración no podremos hacer ninguna suposición sobre la variable xs porque la prueba que debemos construir debe establecer que esa especificación es válida para cualquier valor que la función `sum` tome como argumento.

Para realizar la verificación de que nuestra función efectivamente satisface la especificación dada echaremos mano a nuestro conocimiento del principio de inducción. Haciendo inducción sobre el largo de la lista, si podemos demostrar que la especificación vale para la lista vacía y que si vale para una lista de largo arbitrario entonces vale para una lista con un elemento mas habremos cumplido con nuestra tarea.

El *caso base* supondrá que la lista es vacía. En esta situación la especificación se convierte en:

$$\sum.[\,] = \langle \sum i : 0 \leqslant i < \#[\,] : [\,]!i \rangle$$

Actividad

A partir de la definición de la función, de la definición de funciones sobre listas y de los axiomas para el cuantificador sumatoria, construí la demostración del caso base.

Tal como sucedió con el caso base, en el caso inductivo también nos serán útiles los axiomas y teoremas de las expresiones cuantificadas que estudiamos en el capítulo 10. Dos de ellos, de los cuales haremos uso muy frecuentemente en este tipo de demostraciones, son los teoremas se separación del primer y del último término.

Enunciamos estos teoremas de manera general para un cuantificador $\oplus$:

Teorema separación primer término:

$$\langle \oplus i \ : \ 0 \leqslant i < n+1 \ : \ t.i \rangle = t.0 \oplus \langle \oplus i \ : \ 0 \leqslant i < n \ : \ t.(i+1) \rangle$$

Teorema separación último término:

$$\langle \oplus i \ : \ 0 \leqslant i < n+1 \ : \ t.i \rangle = \langle \oplus i \ : \ 0 \leqslant i < n \ : \ t.i \rangle \oplus t.n$$

Actividad

Instanciá ambos teoremas para el caso del cuantificador sumatoria.

Para el *caso inductivo* la especificación toma la forma:

$$sum.(x \triangleright xs) = \langle \sum i \ : \ 0 \leqslant i < \#(x \triangleright xs) \ : \ (x \triangleright xs)!i \rangle$$

Para construir esta demostración partamos del lado derecho de la expresión a demostrar y lleguemos a que es igual al lado izquierdo:

$\langle \sum i \ : \ 0 \leqslant i < \#(x \triangleright xs) \ : \ (x \triangleright xs)!i \rangle$
= { Definición de # }
$\langle \sum i \ : \ 0 \leqslant i < 1 + \#xs) \ : \ (x \triangleright xs)!i \rangle$
= { Teorema separación del primer término }
$(x \triangleright xs)!0 + \langle \sum i \ : \ 0 \leqslant i < \#xs \ : \ (x \triangleright xs)!(i+1) \rangle$
= { Definición de ! }
$x + \langle \sum i \ : \ 0 \leqslant i < \#xs \ : \ xs!i \rangle$
= { Hipótesis inductiva }
$x + \texttt{sum}.xs$
= { Definición de `sum` }
$\texttt{sum}.(x \triangleright xs)$

Actividad

Construí una verificación de que la función `duplicar` definida como sigue:

$$\begin{array}{rcl} \texttt{duplicar} & :: & [Num] \to [Num] \\ \texttt{duplicar}.[\,] & \doteq & [\,] \\ \texttt{duplicar}.(x \triangleright xs) & \doteq & 2 * x \triangleright \texttt{duplicar}.xs \end{array}$$

satisface la siguiente especificación:

$$\langle \forall xs \ : : \ \langle \forall i \ : \ 0 \leqslant i < \#xs \ : \ (duplicar.xs)!i = 2 * (xs!i) \rangle \rangle$$

Para hacerlo deberás utilizar tus conocimientos en cálculo de predicados.

Actividad

Construí una verificación de que la función `maxlist` definida como sigue:

$$\begin{array}{rcl} \texttt{maxlist} & :: & [Num] \to Num \\ \texttt{maxlist}.[\,] & \doteq & 0 \\ \texttt{maxlist}.(x \triangleright xs) & \doteq & \texttt{max}.x.(\texttt{maxlist}.xs) \end{array}$$

satisface la siguiente especificación:

$$\langle \forall xs \ : : \ maxlist.xs = \langle Max \ i \ : \ 0 \leqslant i < \#xs \ : \ xs!i \rangle \rangle$$

Tené en cuenta que el elemento neutro del máximo entre dos números es el 0.

Actividad

Construí una verificación de que la función `iguales` definida como sigue:

$$\begin{array}{rcl} \texttt{iguales} & :: & [A] \to [A] \to Bool \\ \texttt{iguales}.[\,].[\,] & \doteq & True \\ \texttt{iguales}.[\,].(y \triangleright ys) & \doteq & False \\ \texttt{iguales}.(x \triangleright xs).[\,] & \doteq & False \\ \texttt{iguales}.(x \triangleright xs).(y \triangleright ys) & \doteq & (x = y) \wedge \texttt{iguales.xs.ys} \end{array}$$

satisface la siguiente especificación:

$$\#xs = \#ys \wedge \langle \forall i \,:\, 0 \leqslant i < \#xs \,:\, xs!i = ys!i \rangle$$

En primer lugar pensá qué tipo de inducción vas a utilizar teniendo en cuenta que la función está definida a partir de cuatro casos.

11.4. Derivación

Hasta aquí hemos desarrollado tres etapas para la construcción de un programa. En primer lugar, elaborar una especificación formal para el mismo. En segundo lugar, construir el programa basándonos en las diversas intuiciones y técnicas que hayamos aprendido con la experiencia. En tercer lugar, dar una demostración de que dicho programa satisface la especificación.

Nos ahorraríamos mucho trabajo si pudiéramos realizar algunas de estas etapas simultáneamente. Esto es precisamente lo que haremos a continuación. En esta sección presentaremos un método que nos permitirá construir al mismo tiempo un programa funcional y la demostración de su corrección respecto a la especificación. Esta metodología fue desarrollada a partir de los trabajos de E. W. Dijkstra uno de los pioneros de las ciencias de la computación.

A la construcción en simultáneo del programa y de su verificación se la llama **derivación**. En este proceso partiremos de una especificación formal de una función —es decir, partimos de lo que nosotros queremos que haga la función— y nuestra tarea será encontrar es la definición de la función. Así, la especificación de una función f puede pensarse como una ecuación a resolver siendo la incógnita f a definir y verificar. Luego, el proceso de derivación consistirá en "despejar" la función f para saber qué valor toma (obviamente puede tener varias soluciones posibles). El proceso de despejar nos asegura que si ahora reemplazamos el valor obtenido en la ecuación original tendremos un resultado verdadero.

Como siempre, partamos de un ejemplo. Derivaremos la función `cuadrado` que dado un número natural n devuelve su cuadrado. Esta función puede especificarse formalmente como sigue:

$$\texttt{cuadrado}.n = n^2$$

Obviamente existe una forma trivial de encontrar una función que satisfaga la especificación:

$$\texttt{cuadrado}.n = n * n$$

Si bien es inmediato que esta definición satisface la especificación y que además es una definición extremadamente eficiente, a modo de ejemplo, imaginemos que debemos encontrar una definición recursiva para la función que será implementada en un artefacto que no está preparado para realizar multiplicaciones. Nuestra función deberá darse, entonces, sólo en términos de sumas.

Visto que estamos trabajando con números naturales parece apropiado utilizar la inducción matemática. Notá que como no conocemos `cuadrado` lo único que podemos hacer es manipular el lado derecho de la especificación.

Para el caso base consideraremos a n igual a 0. Para este caso nuestra especificación toma la siguiente forma:

$$\texttt{cuadrado}.0 = 0^2$$

manipulemos el lado derecho de la especificación para llegar a una expresión más simple:

$$\begin{array}{l} \quad \texttt{cuadrado}.0 = 0^2 \\ \equiv \{ \text{ Álgebra: } 0^2 = 0 \ \} \\ \quad \texttt{cuadrado}.0 = 0 \end{array}$$

Esta expresión no contiene el operador potencia cuadrada y por lo tanto puede ser una buena expresión para definir el caso base de la función.

En el caso inductivo supondremos que la función cuadrado satisface la especificación para un n cualquiera y derivemos la definición para el número siguiente $n+1$. En este caso la especificación toma la forma:

$$\texttt{cuadrado}.(n+1) = (n+1)^2$$

Manipulemos ahora el lado derecho de la expresión:

$$\begin{array}{l} \quad \texttt{cuadrado}.(n+1) = (n+1)^2 \\ \equiv \{ \text{ Álgebra: Binomio cuadrado } \} \\ \quad \texttt{cuadrado}.(n+1) = n^2 + 2 * n + 1 \ \texttt{cuadrado}.(n+1) = \texttt{cuadrado} + 2 * n + 1 \\ \equiv \{ \text{ Aritmética } \} \\ \quad \texttt{cuadrado}.(n+1) = \texttt{cuadrado}.n + n + n + 1 \end{array}$$

Por lo tanto si definimos recursivamente a `cuadrado` como sigue:

$$\begin{array}{rcl} \texttt{cuadrado} & :: & Num \to Num \\ \texttt{cuadrado}.0 & \doteq & 0 \\ \texttt{cuadrado}.(n+1) & \doteq & \texttt{cuadrado}.n + n + n + 1 \end{array}$$

Tendremos una definición de la función que satisface la especificación dada.

Actividad

Especificá formalmente y derivá la función `factorial` que, dado un natural, devuelve el factorial del mismo. Para la derivación necesitarás hacer uso de tus conocimientos en expresiones cuantificadas.

Trabajemos ahora sobre otro ejemplo. En este caso estudiaremos cómo calcular la constante matemática e, un número real sumamente importante. Este número, conocido a veces como número de Euler fue reconocido y utilizado por primera vez por el matemático escocés John Napier, quien introdujo el concepto de logaritmo en el cálculo matemático. e es considerado el número por excelencia del cálculo, así como lo es π de la geometría y el número i en el análisis complejo. Se utiliza para describir, entre otras cosas el comportamiento de acontecimientos físicos regidos por leyes sencillas, como pueden ser la velocidad de vaciado de un depósito de agua, el giro de una veleta frente a una ráfaga de viento, el movimiento del sistema de amortiguación de un automóvil o el cimbreo de un edificio metálico en caso de terremoto.

Esta constante, al igual que el número π, es un irracional, lo que significa que no puede expresarse a través de la razón de dos enteros; o bien, que no puede ser expresado con un número finito de cifras decimales o con decimales periódicos.

Su valor aproximado (truncado) es:

$$e \approx 2,71828182845904523536028747135266249775724709369995...$$

La forma más común de definir el valor de e es a través de la siguiente serie infinita:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Nuestra tarea será construir un programa —una función que llamaremos `e`— que calcule en forma aproximada este número. Por supuesto no podemos sumar infinitos términos por lo que nuestra especificación considerará sólo los primeros n términos de la serie. Por supuesto, mientras más grande sea n, al evaluar la función `e` mejor será la aproximación que tengamos.

Utilizando la notación adoptada en el libro para los cuantificadores, la `e` queda especificada de la siguiente manera:

$$\texttt{e} : Num \to Num$$

$$\texttt{e}.n = \langle \sum i \ : \ 0 \leqslant i < n \ : \ \frac{1}{n!} \rangle$$

Actividad

Construí una derivación para la función `exponencial` en la cual la inducción se realice sobre la cantidad de términos de la serie que especifica la función, es decir, sobre n.
Para el caso base y el caso inductivo necesitarás utilizar tus conocimientos relacionados con el cuantificador sumatoria.
Para poder construir una definición del caso inductivo que puede ser efectivamente computable será necesario que utilices una función que ya hemos definido.

En las secciones siguientes exploraremos algunas técnicas de programación que nos permitirán construir programas todavía más complejos.

11.5. Modularización

La actividad anterior ya nos enfrentó a la necesidad de ir combinando funciones previamente definidas. De esto se ocupa la técnica de modularización.

En la sección anterior derivamos programas que involucraban números, utilizando como herramienta principal la inducción sobre $\mathbb{N}$. A continuación vamos a derivar una función sobre listas. Como la inducción también es una herramienta poderosa para este tipo de datos volveremos a utilizarla.

Actividad

Trabajemos con la función `promedio` que, dada una lista no vacía de números devuelve el promedio de sus elementos:

1. Especificá formalmente la función `promedio`.
2. Intentá derivar esta función.

El problema que emergió en nuestro primer intento de derivar la función `promedio` está relacionado con que estábamos intentando computar dos recursiones simultáneamente: la recursión implicada en el numerador y la recursión implicada en el denominador.

Una técnica que puede ayudarnos a resolver este problema y que es una de las metodologías básicas en la programación se llama **modularización**. Esta técnica se basa en la idea de dividir el problema al que nos enfrentamos en partes, haciendo que la solución se obtenga componiendo esas partes. Esto suele involucrar definir nuevas funciones que se encarguen de computar cada una de las partes del problema.

Actividad

A partir de la especificación de la funcion `promedio`:

$$\texttt{promedio}.xs = \frac{\langle \sum i \ : \ 0 \leqslant i < \#xs \ : \ xs!i \rangle}{\#xs}$$

¿En qué partes dividirías el problema de definir la función?

Para esta función construiremos dos funciones auxiliares. La primera de ella, que llamaremos `sum` satisfará la siguiente especificación:

$$sum.xs = \langle \sum i \ : \ 0 \leqslant i < \#xs \ : \ xs!i \rangle$$

y la segunda, será la función cardinal (#).

Ahora bien, estas dos funciones ya son bastante conocidas por nosotros y las hemos definido en los capítulos anteriores. Ellas eran:

$$\begin{array}{rcl} \texttt{sum} & :: & [Num] \to Num \\ \texttt{sum}.[\,] & \doteq & 0 \\ \texttt{sum}.(x \triangleright xs) & \doteq & x + \texttt{sum}.xs \end{array}$$

$$\begin{array}{rcl} \texttt{\#} & :: & [Num] \to Num \\ \texttt{\#}.[\,] & \doteq & 0 \\ \texttt{\#}.(x \triangleright xs) & \doteq & 1 + \texttt{\#}.xs \end{array}$$

De esta forma, la definición de la función `promedio` estará dada en términos de estas dos funciones. Luego de haberlas definido la función `promedio` puede definirse de la siguiente manera:

$$\begin{array}{rcl} \texttt{promedio} & :: & [Num] \to Num \\ \texttt{promedio}.xs & \doteq & \frac{\texttt{sum}.xs}{\texttt{\#}xs} \end{array}$$

Notá que la solución del problema original —calcular el promedio de los elementos de una lista— requirió la solución de dos "sub-problemas". Aplicar la técnica de modularización no se ataca a todos los problemas simuláneamente sino "por módulos", cada uno de los cuales debe ser independiente de los demás. En nuestro caso tenemos dos módulos, uno para cada una de las definiciones de `sum` y `#`. La independencia de los módulos implica que una vez que hayamos encontrado una función para solucionar cada uno de ellos, el problema original también estará resuelto.

Construyamos ahora la función `unoEsSuma` que, dada una lista de números devuelve *true* si hay un elemento en la lista que es igual a la suma de los elementos que le siguen y false en caso contrario. Por ejemplo: $\texttt{unoEsSuma}.[4,2,2] = true$; $\texttt{unoEsSuma}.[7,5,5] = true$; $\texttt{unoEsSuma}.[1] = False$

Actividad

Haciendo uso la técnica de modularización, aplicada ahora a la especificación de una función, utilizá la funcion `sum` con la que ya hemos trabajado para especificar formalmente la función `unoEsSuma`.

El caso base considerará lo que sucede cuando la lista es vacía. En esta situación la especificación toma la forma:

$$\texttt{unoEsSuma}.[\,] \equiv \langle \exists i : 0 \leqslant i < \#[\,] : [\,].i = sum.([\,] \downarrow (i+1)) \rangle$$

Actividad

A partir de:

- Las definiciones de las funciones `sum` y $\downarrow$,
- Tus conocimientos en el cuantificador existencial.

Derivá la función `unoEsSuma` para el caso base.

Para el caso inductivo vamos a suponer que la función que queremos definir satisface la especificación para una lista de largo arbitrario y derivaremos una definición para una lista con un elemento más.

Actividad

Como forma de comenzar la derivación:

1. Escribí cuál será la hipótesis inductiva.
2. Escribí la especificación para el caso $x \triangleright xs$.

Actividad

Completá la siguiente demostración para el caso inductivo:

$$
\begin{array}{ll}
 & \texttt{unoEsSuma}.(x \triangleright xs) \equiv \langle \exists i \,:\, 0 \leqslant i < \#(x \triangleright xs) \,:\, (x \triangleright xs).i = sum.((x \triangleright xs) \downarrow (i+1)) \rangle \\
\equiv & \{ \quad \ldots \quad \} \\
 & \texttt{unoEsSuma}.(x \triangleright xs) \equiv \langle \exists i \,:\, 0 \leqslant i < 1 + \#xs) \,:\, (x \triangleright xs).i = sum.(xs \downarrow i) \rangle \\
\equiv & \{ \text{ Teorema separación del primer término } \} \\
 & \ldots \\
\equiv & \{ \quad \ldots \quad \} \\
 & \texttt{unoEsSuma}.(x \triangleright xs) \equiv x = sum.xs \vee \langle \exists i \,:\, 0 \leqslant i < \#xs) \,:\, (x \triangleright xs).(i+1) = sum.(xs \downarrow (i+1)) \rangle \\
\equiv & \{ \quad \ldots \quad \} \\
 & \texttt{unoEsSuma}.(x \triangleright xs) \equiv x = sum.xs \vee \langle \exists i \,:\, 0 \leqslant i < \#xs) \,:\, xs.i = sum.(xs \downarrow (i+1)) \rangle \\
\equiv & \{ \text{ Hipótesis inductiva, la función } \texttt{unoEsSuma} \text{ satisface la especificación para } xs \} \\
 & \ldots
\end{array}
$$

La función quedará entonces definida como sigue:

$$
\begin{array}{rcl}
\texttt{unoEsSuma} & :: & [Num] \to Bool \\
\texttt{unoEsSuma}.[\,] & \doteq & False \\
\texttt{unoEsSuma}.(x \triangleright xs) & \doteq & x = sum.xs \vee \texttt{unoEsSuma}.xs
\end{array}
$$

Y ya tenemos una demostración de que esta función satisface la especificación de la que partimos.

Actividad

Al derivar las funciones `cuadrado` y `unoEsSuma` pudimos ver que la derivación es una herramienta bastante poderosa para construir programas. A partir de las definiciones de las dos funciones y de la especificáción, ¿cómo podrías transformar la derivación para que se transforme en una verificación?

Un problema que se presenta frecuentemente en matemática es el de evaluar un polinomio de la forma:

$$p(x) = a_0x^0 + a_1x^1 + \ldots + a_{n-1}x^{n-1} + a_nx^n$$

en un valor dado de la variable independiente x. Enfrentémonos al problema de derivar una función recursiva que permita hacer efectiva la evaluación. Llamaremos `evaluar` a esta función.

Claramente, los coeficientes del polinomio —$a_n; a_{n-1}; \ldots; a_0$ jugarán un rol fundamental en la función `evaluar`. Una manera conveniente de trabajar con ellos es construir una lista con ellos de forma tal que su posición coincida con el exponente al que están asociados. Esta lista tendrá la forma:

$$xs = [a_0, a_1, ..., a_{n-1}, a_n]$$

Nuestra función `evaluar` tiene, así, dos argumentos: una lista, que representa a los coeficientes del polinomio, y un número, que denota la constante en la que evaluamos el polinomio.

Actividad

Especificá formalmente y derivá la función `evaluar`.

Actividad

1. ¿Qué hace la función especificada formalmente de la siguiente manera?

   $\texttt{f} : [Num] \to Bool$

   $\texttt{f}.xs \equiv \langle \forall i : 0 < i < \#xs : xs.i = xs.(i-1) \rangle$

2. Al descubrir qué hace esta función un estudiante afirmó que para derivarla es necesario considerar tres casos.¿Estas de acuerdo con el estudiante? ¿por qué? ¿Cuáles serían los casos a considerar?

3. Derivá la función para los casos base.

4. Completá la derivación del caso inductivo en la cual se demuestra que la especificación vale para una lista con al menos dos elementos considerando que la especificación es válida para una lista con un elemento menos. Así la hipótesis inductiva se aplica a una lista de la forma $x \triangleright xs$

   $\texttt{f}.(x \triangleright y \triangleright xs) \equiv \langle \forall i : 0 < i < \#(x \triangleright y \triangleright xs) : (x \triangleright y \triangleright xs).i = (x \triangleright y \triangleright xs).(i-1) \rangle$
   $\equiv$ { Definición de # dos veces }
   ...
   $\equiv$ { ... }
   $\texttt{f}.(x \triangleright y \triangleright xs) \equiv \langle \forall i : i = 1 \vee 1 < i < 2 + \#xs : (x \triangleright y \triangleright xs).i = (x \triangleright y \triangleright xs).(i-1) \rangle$
   $\equiv$ { Partición de rango }
   ...
   $\equiv$ { ... }
   $\texttt{f}.(x \triangleright y \triangleright xs) \equiv (x \triangleright y \triangleright xs).1 = (x \triangleright y \triangleright xs).0 \wedge \langle \forall i : 1 < i < 2 + \#xs : (x \triangleright y \triangleright xs).i = (x \triangleright y \triangleright xs).(i-1)$
   $\equiv$ { Cambio de variables, $i \to i+1$ }
   ...
   $\equiv$ { ... }
   ⋮

5. Escribí la definición completa de la función.

Actividad

Derivá la funcion `creciente` que determina si los elementos de una lista están ordenados en forma creciente. La misma está especificada de la siguiente manera:

$\texttt{creciente} : [Int] \to Bool$

$\texttt{creciente}.xs = \langle \forall i : 0 < i < \#xs : xs.i \geqslant xs.(i-1) \rangle$

- Para determinar cuántos casos considerar tené en cuenta que la especificación relaciona un elemento de la lista con el anterior.

11.6. Reemplazo de constantes por variables

Comencemos a trabajar con otro ejemplo. Consideremos el siguiente problema: queremos derivar una función que calcule la suma de las primeras n potencias de dos. Esta función, que llamaremos `sumaPotencias` tomará un número natural y devolverá otro número natural.

Por ejemplo:

$\texttt{sumaPotencias}.3 = 2^0 + 2^1 + 2^2$

$\texttt{sumaPotencias}.5 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4$

$\texttt{sumaPotencias}.0 = 0$

Actividad

A partir de la especificación en lenguaje natural de la función `sumaPotencias`:

1. Construí una especificación formal de la función.
2. Derivá la función para el caso base.
3. Derivá la función para el caso inductivo en la cual apliques en algún paso el teorema de separación del último término.

Recordá que en la derivación necesitarás hacer uso de tus conocimientos en el manejo de expresiones cuantificadas.

Cuando derivamos esta función utilizando para el caso inductivo el teorema de separación del último término obtenemos la siguiente definición para `sumaPotencias`:

$$\begin{array}{rcl} \texttt{sumaPotencias} & :: & Num \to Num \\ \texttt{sumaPotencias}.0 & \doteq & 0 \\ \texttt{sumaPotencias}.(n+1) & \doteq & \texttt{sumaPotencias}.n + 2^n \end{array}$$

Si el lenguaje de programación en el que estamos trabajando permite calcular 2^n, hemos terminado nuestra tarea. En caso contrario, es necesario desarrollar un programa para el cálculo de 2^n.

Supongamos que este es el caso. Aplicando nuestra técnica de modularización podemos definir a `sumaPotencias` de la siguiente manera:

$$\begin{array}{rcl} \texttt{sumaPotencias} & :: & Nat \to Nat \\ \texttt{sumaPotencias}.0 & \doteq & 0 \\ \texttt{sumaPotencias}.(n+1) & \doteq & \texttt{sumaPotencias}.n + g.n \end{array}$$

Donde la funcion `g` deberá ser derivada a partir de la especificación:

$\texttt{g} :: Nat \to Nat$

$\texttt{g}.i = 2^i$

Tenemos entonces el desafío de derivar la función `g`. Ahora bien, encontrar una función que satisfaga la especificación de `g` parece ser un problema bastante puntual. En este sentido, si luego tuviéramos que derivar otra función que satisfaga una especificación muy similar como:

$\texttt{h} :: Nat \to Nat$

$\texttt{h}.i \doteq 3^i$

tendríamos que hacer todo el trabajo de nuevo. Parece tentador intentar construir una función que sea más general.

La técnica de programación llamada **reemplazo de constantes por variables** captura estas ideas. El objetivo de la técnica es cambiar el problema original por uno más general, lo que se logra reemplazando las constantes en la especificación por variables. Así, en nuestra especificación de `g`:

$\texttt{g} :: Nat \to Nat$

$\texttt{g}.i \doteq 2^i$

tenemos una constante, el 2. Utilizando esta técnica derivemos una función más general que dados dos números naturales i y j devuelve j^i. Esta nueva función, que tendrá ahora dos parámetros y que llamaremos `g'`, se especifica de la siguiente manera:

$\texttt{g'} \to Nat \to Nat$

$$\texttt{g'}.j.i = j^i$$

El problema original que nosotros queremos resolver se obtiene al evaluar esta nueva función `g` cuando j es 2:

$$\texttt{g'}.2.i = \texttt{g}.i$$

Derivemos ahora esta nueva función. Intentaremos realizar la derivación haciendo inducción sólo en uno de los parámentros, si esto no es posible recién ahí analizaremos la posibilidad de hacer inducción en ambos parámetros. Ahora bien, ¿en cuál de los dos? Si hiciéramos inducción en la base —es decir, en el parámetro j—, al derivar el caso inductivo tendríamos que encontrar una relación entre $(j+1)$ elevado a la i y j elevado a la i. En fórmulas:

$$(j+1)^i \approx j^i \oplus \dots$$

Esta relación no parece tan fácil de encontrar. Si hiciéramos inducción en el exponente —es decir, en el parámetro i—, al derivar el caso inductivo tendríamos que encontrar una relación entre j elevado a la $i+1$ y j elevado a la i. En fórmulas:

$$j^{(i+1)} \approx j^i \oplus \dots$$

Esta relación es bastante más fácil de hallar porque ya conocemos las propiedades de la pontenciación. Particularmente:

$$j^{(i+1)} = j^i * j$$

Esta es una muy buena razón para intentar hacer inducción sobre el parámetro i. Realicemos la derivación:

Caso base

$$\begin{array}{l} \quad \texttt{g'}.j.0 = j^0 \\ \equiv \{ \text{ Álgebra } \} \\ \quad \texttt{g'}.j.0 = 1 \end{array}$$

Caso inductivo

Como siempre consideraremos como hipótesis que la especificación se satisface para un número i y, con esto, demostraremos que la especificación se satisface para el número siguiente, $i+1$.

$$\begin{array}{l} \quad \texttt{g'}.j.(i+1) = j^{(i+1)} \\ \equiv \{ \text{ Álgebra: propiedades de la potencia } \} \\ \quad \texttt{g'}.j.(i+1) = j^i * j^1 \\ \equiv \{ \text{ Hipótesis inductiva } \} \\ \quad \texttt{g'}.j.(i+1) = \texttt{g'}.j.i * j^1 \\ \equiv \{ \text{ Álgebra: propiedades de la potencia } \} \\ \quad \texttt{g'}.j.(i+1) = \texttt{g'}.j.i * j \end{array}$$

Por lo tanto la definición de la función `sumaPotencias` quedará de la siguiente manera:

$$\begin{array}{rcl} \texttt{g'} & :: & Nat \to Nat \\ \texttt{g'}.j.0 & \doteq & 0 \\ \texttt{g'}.j.(i+1) & \doteq & \texttt{g'}.j.i * j \\ \texttt{sumaPotencias} & :: & Nat \to Nat \\ \texttt{sumaPotencias}.0 & \doteq & 0 \\ \texttt{sumaPotencias}.(n+1) & \doteq & \texttt{sumaPotencias}.n + \texttt{g'}.2.n \end{array}$$

Actividad

Como hemos mencionado antes, no existe una única forma de derivar una función; distintos pasos en la derivación pueden dar origen a diferentes definiciones de una misma función.

- Derivá nuevamente la función `sumaPotencias` pero ahora utilizando el teorema de separación del primer término en alguno de los pasos de la derivación del caso inductivo.
- Compará las definiciones que obtuviste siguiendo los dos caminos. ¿Podrías establecer criterios para afirmar que una es mejor que otra?

11.6.1. Generalización por abstracción

Consideremos el siguiente problema: Se quiere derivar una función que determine si todas las sumas iniciales de una lista son mayores o iguales a cero. Con sumas iniciales nos referimos a las sumas de dos o más elementos consecutivos de una lista. Si denominamos `sumasIniciales` a la función estos serían los resultados que debería devolver en algunos casos particulares:

$$\begin{array}{rl} \texttt{sumasIniciales}.[1,2,3] \equiv True & \text{Porque } 1 \leqslant 0,\ 1+2 \leqslant 0,\ 1+2+3 \leqslant 0 \\ \texttt{sumasIniciales}.[9,-4] \equiv True & \text{Porque } 9 \leqslant 0,\ 9+(-4) \leqslant 0 \\ \texttt{sumasIniciales}.[3,-7] \equiv False & \text{Porque } 3+(-7) < 0 \end{array}$$

Utilizando las funciones `sum` y $\uparrow$ `sumasParciales` se puede especificar de la siguiente forma:

$\texttt{sumasIniciales} : [Num] \to Bool$

$\texttt{sumasIniciales}.xs \equiv \langle \forall i : 0 \leqslant i < \#xs : \texttt{sum}.(xs \uparrow i) \geqslant 0 \rangle$

Actividad

Intentá derivar esta función: ¿Qué ocurre en el caso inductivo?

Al intentar derivar la función `sumasIniciales` para el caso inductivo arribamos a la siguiente fórmula:

$$\texttt{sumasIniciales}.xs \equiv 0 \geqslant 0 \wedge \langle \forall i : 0 \leqslant i < \#xs : x + \texttt{sum}.(xs \uparrow i) \geqslant 0 \rangle$$

En esta expresión no es posible aplicar la hipótesis inductiva porque hay una x que queda sumando y no parece haber ninguna forma de eliminarla.

Este tipo de problemas es frecuente y para solucionarlos existe una técnica llamada **Generalización por abstracción**. La idea principal de dicha técnica consiste en buscar una especificación más general, uno de cuyos casos particulares sea la función original que queríamos derivar. Para encontrar la generalización adecuada se introducen parámetros nuevos los cuales servirán para dar cuenta de las subexpresiones que no permiten aplicar la hipótesis inductiva en la derivación original.

Para el problema de la función `sumasParciales` lo que haremos será derivar la definición de una nueva función generalizada con un parámetro más, n. Esta nueva función, `sumasParcialesGen`, se especifica como sigue:

$\texttt{sumasInicialesGen} : Num \to [Num] \to Bool$

$\texttt{sumasInicialesGen}.n.xs \equiv \langle \forall i : 0 \leqslant i < \#xs : n + \texttt{sum}.(xs \uparrow i) \geqslant 0 \rangle$

Notá que la especificación de `sumasParcialesGen` está inspirada en la última expresión a la que arribamos cuando intentamos derivar `sumasParciales`.

Actividad

1. ¿Por qué `sumasInicialesGen` generaliza a `sumasIniciales`? Es decir, ¿para qué caso particular `sumasInicialesGen` se transforma en `sumasIniciales`?
2. A partir de tu respuesta a las preguntas anteriores definí a `sumasIniciales` en términos de `sumasInicialesGen`.

En este punto es importante hacer una distinción entre esta técnica y la técnica de reemplazo de constantes por variables. Si bien ambas comparten la característica de introducir generalizaciones no son iguales a un nivel metodológico. Al usar la técnica de reemplazo de constantes por variables lo que se hace es, justamente, reemplazar una constante que figura en la especificación original por una variable —el 2 en el caso de la función `g` de la sección anterior—. En la técnica de generalización por abstracción lo que se hace es modificar la especificación original de forma que se pueda aplicar la hipótesis inductiva. Cómo debe modificarse la especificación original es algo difícil de saber a priori y generalmente se descubre cuando nos encontramos con dificultades en la derivación.

Luego de haber corroborado que nuestra función efectivamente generaliza a la original, lo cual constituye el primer paso de la técnica de generalización por abstracción, se lleva adelante la derivación de la nueva función. Que esta nueva derivación pueda llevarse adelante dependerá de las propiedades del dominio en cuestión (en este caso los números, en particular la asociatividad de la suma) y puede no llegar a buen puerto o tener que volver a generalizarse a su vez. La programación es una actividad creativa y esto se manifiesta en este caso en la elección de las posibles generalizaciones.

Actividad

Derivá la función `sumasInicialesGen` realizando inducción en el largo de la lista, para ello:

1. Derivá el caso base.
2. En el caso inductivo suponé que la especificación vale para una lista de largo arbitrario xs y para un número cualquiera. Tené en cuenta que este número puede tener varias formas a la hora de aplicar la hipótesis inductiva.

Es importante resaltar que en esta derivación existe una novedad. La función que intentamos construir posee dos argumentos, un número y una lista de números. Al derivarla hacemos inducción sólo en uno de ellos, la lista. Para aplicar la hipótesis inductiva sólo tenemos que asegurarnos de que el parámetro sobre el cual estamos haciendo inducción se "achique". El otro parámetro no debe cumplir esta restricción, es más puede ser cualquier expresión. Esto es lo que ocurre en el caso de la función `sumasParcialesGen` y ocurrirá repetidas veces cuando apliquemos la técnica de generalización por abstracción.

El resultado completo de esta derivación es, entonces:

$$\texttt{sumasIniciales} : [Num] \to Bool$$

$$\texttt{sumasInicialesGen} : Num \to [Num] \to Bool$$

$$\begin{array}{rcl} \texttt{sumasIniciales}.xs & \doteq & \texttt{sumasInicialesGen}.0.xs \\ \texttt{sumasInicialesGen}.n.[\,] & \doteq & True \\ \texttt{sumasInicialesGen}.n.(x \triangleright xs) & \doteq & n \geqslant 0 \wedge \texttt{sumasInicialesGen}.(n+x).xs \end{array}$$

La utilización de la técnica de generalización por abstracción no es algo que se pueda determinar de antemano. Lo que se hace habitualmente es intentar derivar una función y cuando de llega a una situación en la que no se puede aplicar la hipótesis inductiva se propone generalizar.

Trabajemos ahora con el siguiente programa. Se quiere derivar una función que determine si, dada una lista, hay algún elemento que sea igual a la suma de los elementos anteriores. Llamaremos `existeSuma` a esta función.

Actividad

1. ¿Cuál será el tipo de esta función?
2. Pensá qué resultado debería devolver la función para algunas listas particulares.
3. Un estudiante propuso la siguiente especificación formal para la función:

   $$\texttt{existeSuma}.xs \equiv \langle \exists i : 0 \leqslant i < \#xs : xs.i = \langle \sum j : 0 \leqslant j < i : xs.j \rangle \rangle$$

   ¿Qué opinás de esta especificación? ¿se te ocurre alguna más simple? Podés usar funciones que ya hemos definido.
4. Intentá derivar esta función.
5. A partir de tu experiencia en la derivación proponé una nueva función, `existeSumaGen` que generalice a `existeSuma`. Corroborá que esto realmente suceda definiendo a `existeSuma` en términos de `existeSumaGen`.
6. Derivá `existeSumaGen`.
7. Dá la definición de ambas funciones.

Ocupémonos ahora de un problema un poco más complejo que requerirá que pongamos en marcha los conocimientos que aprendimos en este capítulo.

La lista balanceada. Supongamos que tenemos una lista de booleanos y queremos determinar si en ella hay igual cantidad de elementos *true* como de elementos $false$. Llamemos a esta función `bal`. Dicha función deberá devolver los siguientes resultados para estos casos particulares:

$\texttt{bal}.[True, True, False, True] \equiv False$

$\texttt{bal}.[True, True] \equiv False$

$\texttt{bal}.[False, True, False, True] \equiv True$

$\texttt{bal}.[\,] \equiv True$

Si imaginamos que nuestra función contará la cantidad de elementos $True$, luego contará la cantidad de elementos $false$ y finalmente, comparará los resultados podemos pensar que la especificación de `bal` podría construirse de manera sencilla si tuviéramos dos funciones ya definidas que contaran cada una de ellas, la cantidad de elementos *true* y $false$ de la lista. Llamemos a estas funciones `cantTrue` y `cantFalse`, respectivamente.

Una vez que tengamos definidas ambas funciones la función `bal` quedará definida como sigue:

$\texttt{bal}.xs \doteq \texttt{cantTrue} = \texttt{cantFalse}$

Actividad

1. Especificá formalmente las funciones `cantTrue` y `cantFalse`. Para ello puede serte útil repasar los cuantificadores que introdujimos en el capítulo anterior.
2. Derivá ambas funciones para el caso base.
3. Un estudiante realizó la siguiente derivación del caso inductivo de la función `cantTrue`:

$$
\begin{array}{ll}
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, 0 \leqslant i < \#(x \triangleright xs) \,:\, (x \triangleright xs).i \rangle \\
\equiv & \{ \text{ Definición de } \# \ \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, 0 \leqslant i < 1 + \#xs \,:\, (x \triangleright xs).i \rangle \\
\equiv & \{ \text{ Partición de rango } \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle + \langle Ni \,:\, 1 \leqslant i < 1 + \#xs \,:\, (x \triangleright xs).i \rangle \\
\equiv & \{ \text{ Cambio de variables } i \to i + 1 \ \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle + \langle Ni \,:\, 1 \leqslant i + 1 < 1 + \#xs \,:\, (x \triangleright xs).(i+1) \rangle \\
\equiv & \{ \text{ Álgebra } \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle + \langle Ni \,:\, 0 \leqslant i < \#xs \,:\, (x \triangleright xs).(i+1) \rangle \\
\equiv & \{ \text{ Definición de } \texttt{indexar} \ \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle + \langle Ni \,:\, 0 \leqslant i < \#xs \,:\, xs.i \rangle \\
\equiv & \{ \text{ Hipótesis inductiva } \} \\
 & \texttt{cantTrue}.(x \triangleright xs) \equiv \langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle + \texttt{cantTrue}.xs
\end{array}
$$

   a) ¿Por qué creés que el estudiante no aplicó directamente el teorema de separación del primer término?
   b) ¿Qué valores puede tomar la cuantificación $\langle Ni \,:\, i = 0 \,:\, (x \triangleright xs).i \rangle$? ¿Para qué parámetros esa cuantificación toma valores diferentes? Recordá que estamos trabajando con una lista de booleanos.
   c) A partir de tu respuesta a las preguntas anteriores da una definición de la función para los dos casos inductivos que quedan establecidos.
4. Escribí la definición completa de la función `cantTrue`.
5. Derivá el caso inductivo de la función `cantFalse`. Como resultado da su definición.
6. Da la definición de la función `bal`

Ahora bien, la definición que construimos para la función `bal` necesita recorrer dos veces la lista, una vez para contar la cantidad de elementos *true* y otra vez para la cantidad de elementos $false$. Esto puede ser bastante costoso cuando la lista es realmente larga. Como forma de superar esta dificultad intentemos derivar una definición recursiva para `bal` especificada como sigue:

$$\texttt{bal}.xs \doteq \texttt{cantTrue} = \texttt{cantFalse}$$

Para hacerlo seguramente utilizaremos las definiciones de las funciones `cantTrue` y `cantFalse`.

Actividad

Derivá una definición recursiva para `bal`:

1. Derivá el caso base para `bal`
2. Las definiciones de `cantTrue` y `cantFalse` nos inducen a realizar análisis por casos según el valor del primer elemento de la lista. Intentá derivar los dos casos inductivos para la función `bal`. Necesitarás utilizar la técnica de generalización por abstracción.

Resumiendo

Actividad

Construí una tabla en donde se mencionen:

- En una primer columna las técnicas de programación que estudiamos en este capítulo.
- En una segunda columna una descripción de la técnica elaborada con tus palabras.
- En una tercer columna enumerá algunos ejemplos que te parezcan relevantes de las derivaciones realizadas utilizando dicha técnica.