Apunte de Matemática y Lógica para computación

Docente: Araceli Acosta [1]

2016

[1]Extracto del material elaborado por Araceli Acosta, Renato Cherini, Leticia Losano, Miguel Pagano

.

Capítulo 1

Revisando la aritmética

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.

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 $(x + y)$, la resta $(z - 6)$, la multiplicación $(a*$ b$)$, la división $(6/3)$ y la potenciación (2^x). A su vez, cada una de estas expresiones puede volver a combinarse con otras, por ejemplo: $\frac{(x+y)}{2^x} + 6/3$.

Hasta aquí hemos hablado de la **sintaxis** de nuestro lenguaje, es decir, de los símbolos que utilizaremos y de la forma correcta en que deben estar dispuestos para armar expresiones. Pero no hemos dicho nada acerca de su significado, ni de qué representan esos símbolos, o sea, no hemos establecido la **semántica** de nuestro lenguaje. 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. Estas distinciones no suelen 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.

Actividad

Usando las tres reglas anteriores construí algunas expresiones con al menos dos operadores. ¿Cómo podrías justificar que tus expresiones están bien construidas?

1.1.1. Relaciones

Contando sólo 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 las **relaciones**. Cuando combinamos dos expresiones usando un símbolo de relación decimos que construimos una **fórmula**. Una fórmula es un tipo particular de expresiones.

Los símbolos de relaciones que consideraremos por el momento serán: $=$, $\neq$, $<$, $\leqslant$, $>$ y $\geqslant$ cuya semántica 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 dado la semántica de cada una de las relaciones que utilizaremos usando variables. Pero, al igual que los operadores $+$, $-$, etc, las relaciones también pueden utilizarse para relacionar dos números, por ejemplo la fórmula: $3 \leqslant 45$.

Actividad

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

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

Actividad

¿Qué valor tiene la expresión (formula) $2 + 3 = 5$?

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 $5 > 1$. Así, diremos que las fórmulas representan valores de **tipo booleano**: son verdaderas o son falsas.

Actividad

¿Son $2^3 + 2 = 8 + 2 = 10$ y $4 = 3 + 1 = 5$ expresiones? ¿Por qué?

1.1.2. Reglas de precedencia

Actividad

¿Te parece que las siguientes expresiones representan lo mismo?

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

Actividad

1. Indicá cómo se fue construyendo la expresión marcando cada etapa de la construcción 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 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

¿Qué relación podés establecer entre estas dos fórmulas?
$(a^2 - b^2) = (a + b) * (a - b) \longrightarrow (16 - y^2) = (4 + y) * (4 - y)$
$(a * b) * c = a * (b * c) \longrightarrow (\frac{2*x}{y^3} * 350) * 23 = \frac{2*x}{y^3} * (350 * 23)$

1.2. Simplificación de expresiones

Actividad

Simplificá las siguientes expresiones aritméticas.

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

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 Equ las siguientes expresiones

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

Hasta este momento tenemos una noción de forma canónica para expresiones; 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 $2^3 + 2 = 2^4$.

1.2.1. Simplificando fórmulas

Hasta ahora hemos trabajado mayormente simplificando expresiones numéricas. En esta sección vamos a comenzar a manipular fórmulas. La pregunta que intentaremos responder será: ¿como simplificarlas?

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:

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

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 tiene el 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

¿Es $True + 3$ una expresión?

1.3. ¿Cómo asegurarnos que una expresión tiene sentido?

Actividad

Considerá las siguientes expresiones:

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

¿Se pueden simplificar? ¿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. 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**. Como ya vimos anteriormente, los tipos son categorías que se les asigna 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 reducir una expresión que en algún momento no se podría reducir.

¿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 la expresión. Ese tipo no puede ser cualquiera; para averiguarlo construimos *árboles de tipado*. 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; y que los árboles de $2 - 4 * 5$ y de $3 + 5$ son respectivamente:

$$\begin{array}{c} 2 \;\; - \;\; 4 \;\; * \;\; 5 \\ \hline Nat * Nat \end{array}$$

$$\begin{array}{c} Nat - \quad Nat \\ \hline Nat \end{array}$$

$$\begin{array}{c} 3 \;\; + \;\; 5 \\ \hline Nat + Nat \\ \hline Nat \end{array}$$

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

Actividad

Tal vez al construir el árbol de la actividad anterior te encontraste con situaciones parecidas en diferentes ramas. Ahora te proponemos que observando los siguientes árboles de tipos, sistematices 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

Utilizando EQU sacá todos los paréntesis que sean *superfluos*, es decir, manteniendo el árbol de tipado calculado.

1. $(-(5 + x) + ((3 * 6)/(4 * 5))) * (8 * 5)$
2. $(((2)^2 + 5) - (4 * 2)/(4)) + 1 = ((5 * x)/8) + (3 * 5)^3$

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. Ecuaciones: 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

En estos ejercicios trabajamos sobre ecuaciones aritméticas centrándonos en diferentes aspectos: algunas veces buscando valores que satisfacen las ecuaciones, otras demostrando que las ecuaciones son siempre verdaderas independientemente de los valores que tomen las variables.

En este curso nos interesa particularmente trabajar sobre fórmulas que expresan verdades generales. Cuando el valor de una fórmula es verdadero, independientemente de los valores que tomen las variables que ocurren en ella, decimos que es *válida*. Los teoremas, lemas y proposiciones que podes haber visto en materias como el Algebra son ejemplos de fórmulas válidas.

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: $x = 1$

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

¿qué podes decir acerca de la resolución de Pedro?

1.4.2. Satisfactibilidad

La fórmulas que tiene al menos una solución las llamamos **satisfactibles**; y como ya dijimos, cuando cualquier valor de las variables es una solución, decimos que es **válida**.

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. Cada una de estas categorías determina un conjunto de fórmulas.

Actividad

¿Que relaciones podes establecer entre los conjuntos de formulas determinados por estas categorías?

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. $x + 100 = x$

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 normalmente, cuando se habla de demostración en matemática, se esta 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 antes mencionada?

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).

Capítulo 2

Lógica Proposicional

En este capítulo centramos nuestra atención en los *razonamientos*. Los siguientes ejemplos 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. 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 si un razonamiento es correcto o no pero no si la conclusión de éste es válida o no. 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:

- ¿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?

2.1. 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, el hecho de desconocer 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.

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". Esto se escribe de la siguiente manera:

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

Para realzar 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"

$$p \wedge q$$

A las letras p y q las llamamos *variables proposicionales*, porque muchas veces cuando trabajemos con estos símbolos nos vamos a olvidar del significado específico que le dimos (*semántica*) y sólo nos vamos a concentrar en la relación que hay entre las expresiones. En otras palabras, su significado puede variar.

En los capítulos anteriores trabajamos con fórmulas *sencillas*. Estas estaban formadas por relaciones entre expresiones númericas, como por ejemplo $6 + 7 * 2 > 21$. A partir de ahora, vamos a extender la idea de fórmula, incorporando 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*:

- 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 $\wedge, \vee, \neg, \Rightarrow, \Leftarrow, \equiv, \neq$.

Describimos ahora los operadores booleanos considerados y su significado. Más adelante los usaremos para interpretar oraciones del lenguaje natural y consideraremos los aspectos más sutiles o controversiales de cómo usar 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.

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.

Negación. El operador $\neg$ es unario, es decir toma una sola fórmula y construye otra cuyo valor de verdad es el opuesto.

Implicación. El operador binario $\Rightarrow$ captura la idea de consecuencia lógica. Si F_1 y F_2 son fórmulas, 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. Por lo tanto podía usarse también el operador usual de la igualdad para este caso. Sin embargo, ambos operadores tienen propiedades distintas y por lo tanto preferimos distinguirlos.

2.1.1. Reglas de precedencia

Cuando tenemos expresiones numéricas, por ejemplo, 3*2+5, sabemos que primero tenemos que multiplicar y luego sumar. Estos acuerdos generales en qué operación se hace primero en cada caso, se llaman *reglas de precedencia*. 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
$\equiv \neq$	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

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

2.2. 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).

2.2.1. 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.

2.2.2. 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.

2.3. Validez y Satisfactibilidad

Los conceptos de Validez y satisfactibilidad que vimos en el capítulo anterior se aplican sobre cualquier fórmula, en particular las fórmulas de la lógica proposicional.

Conceptos teóricos

Decimos que una fórmula es **satisfactible** cuando es verdadera para algunos valores posibles de las variables. Decimos que es **válida** cuando es verdadera para todos los valores posibles de las variables.

Nuestro principal interés es distinguir si una fórmula es válida y cuándo no. Como la validez es independiente de los valores de las variables involucradas, si queremos demostrar la validez de una fórmula no podemos hacer ninguna suposición sobre los valores de las variables. Por el contrario, cuando queremos demostrar que una fórmula es **no valida** es suficiente con encontrar al menos un valor que haga que la fórmula sea falsa: esto es un contraejemplo. Algunas veces además es posible demostrar directamente que la fórmula es falsa para todos los valores de las variables. En este caso decimos que la fórmula es **no satisfactible**, porque no existe ningún valor posible que haga que sea verdadera.

Actividad

Decidí si cada una de las siguientes fórmulas proposicionales es válida o no. En caso que una fórmula no sea válida, decidí si es satisfactible o no. En todos los casos justificá con una tabla de verdad, un ejemplo o un contraejemplo, según corresponda.

1. p
2. $p \equiv p$
3. $p \equiv p \equiv p$
4. $p \Rightarrow q \equiv q \Rightarrow p$
5. $p \vee q \Rightarrow p$
6. $p \wedge q \Rightarrow p$
7. $p \Rightarrow q \wedge p$
8. $p \Rightarrow q \vee p$
9. $p \Rightarrow q$
10. $p \Rightarrow (q \Rightarrow p)$

1. $p \equiv p \equiv \mathit{True}$
2. $\mathit{True} \vee p$
3. $\mathit{True} \wedge p$
4. $\mathit{False} \vee p$
5. $\mathit{False} \wedge p$
6. $\mathit{False} \vee p \equiv p$
7. $\mathit{False} \wedge p \equiv false$
8. $\mathit{True} \vee p \equiv \mathit{True}$
9. $\mathit{True} \wedge p \equiv p$

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).

Capítulo 3

Cálculo proposicional

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.

3.1. 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.

3.1.1. 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 ($=, \neq, <, >, \leqslant, \geqslant$), 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 2.1 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. $p \vee q$
   $\equiv \{ \quad \}$
   $(p \equiv \mathit{True}) \vee q$

2. $p \Rightarrow q$
   $\equiv \{ \quad \}$
   $(p \equiv \mathit{True}) \Rightarrow q$

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. $q \Rightarrow (p \vee q)$
   $\equiv \{ \quad \}$
   $q \Rightarrow (q \vee p)$

3.1.2. 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:

$$
\begin{array}{ll}
& F_1 \\
\equiv & \{ \texttt{Razón 1} \} \\
& F_2 \\
\equiv & \{ \texttt{Razón 2} \} \\
& F_3 \\
\equiv & \{ \texttt{Razón 3} \} \\
& True
\end{array}
$$

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 .

3.2. 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. $True$

3.3. 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: $p \equiv \neg p \equiv False$

3.4. 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. $((p \neq q) \equiv r) \equiv (p \neq (q \equiv r))$
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 $\equiv$, $\neq$ y $\neg$ una válida (teorema) y una no válida.

3.5. 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.

3.6. 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. $\big(a \vee b \vee c \equiv a \vee b \equiv a \vee c \equiv b \vee c \equiv \mathit{False}\big) \equiv \big((a \equiv b \equiv c) \wedge \neg(a \wedge b \wedge c)\big)$

3.7. 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*: $(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$.

3.8. 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. $s \vee r \equiv s$
   $\equiv \{$ $\}$
   $s \Leftarrow r$
2. $(s \vee s) \equiv ((p \Rightarrow q) \Rightarrow (s \vee s))$
   $\equiv \{$ $\}$
   $(p \Rightarrow q) \vee (s \vee s)$

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.

3.9. ¿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 4

Conjuntos

[1]

La teoría de conjuntos es una de la ramas fundamentales de la matemática. Muchos de los avances profundos de la matemática en el siglo pasado tuvieron lugar en la teoría de conjuntos, y hay una profunda conexión entre teoría de conjuntos y lógica. Más importante, para la ciencia de la computación ha sido de mucha utilidad para describir algoritmos y casi cualquier rama de la computación utiliza conjuntos de vez en cuando.

4.1. Notación para describir conjuntos

No definiremos formalmente que es un conjunto. La razón de ello es que la teoría de conjuntos tenía el objetivo de servir como fundamento para toda la matemática: todo en matemática se debe definir, al menos en principio, en términos de conjuntos. Como el conjunto es el concepto de nivel más básico, no hay nada más primitivo que nos sirva para definirlo formalmente.

Informalmente, un conjunto es sólo una colección de objetos llamados *elementos*. Podés pensar a un conjunto como un grupo de miembros, una colección, una clase, etc. Sin embargo estas palabras son sólo sinónimos y no constituyen una definición precisa de un conjunto. Una forma de describir un conjunto es escribir todos sus elementos entre llaves {}. Aquí hay algunos ejemplos

$A = \{1, 2, 3\}$

$B = \{'a', 'b', 'd', 'q', 'k'\}$

$D = \{0, 1, ..., 100\}$

$E = \{\ \}$

$N = \{0, 1, 2, 3, 4, 5, 6, ...\}$

$P = \{0, 2, 4, 6, 8, 10, ...\}$

$S = \{$"hola", "buen día", "qué tal"$\}$

Cualquier elemento particular puede aparecer sólo una vez en un conjunto; esto significa que tiene sentido que preguntemos si x es un elemento del conjunto S, para lo cual la respuesta será "si" o "no", pero no tiene sentido preguntar cuantas veces ocurre x en S. No está bien visto escribir un conjunto con elementos repetidos, ya que las ocurrencias extras del elemento no tienen utilidad y pueden ser confusas. Es decir, siempre se puede eliminar las copias redundantes sin cambiar el conjunto.

Un conjunto puede tener cualquier número de elementos. Por ejemplo, A tiene 3 elementos, E tiene cero y N tienen infinitos. Es común utilizar letras minúsculas $(x, y, z, a, b, ...)$ para referirse a elementos de un conjunto y letras mayúsculas $(X, Y, Z, A, B, C, ...)$ para referirse a nombres de

[1] La primera parte es una traducción libre de Discrete Mathematics using a computer - O'Donell, Hall, Page

conjuntos. Esto es una recomendación sólo una convención (es decir, un acuerdo entre la comunidad matemática), y no una regla estricta, y por su puesto se viola cuando un conjunto es un elemento de otro conjunto.

Un caso especial de conjuntos es el conjunto vacío { }. El símbolo $\emptyset$ se utiliza para denotar este conjunto. También utilizaremos los símbolos $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ y $\mathbb{R}$ para denotar a los siguientes conjuntos:

- $\mathbb{N}$: el conjunto de los números naturales.
- $\mathbb{Z}$: el conjunto de los números enteros.
- $\mathbb{Q}$: el conjunto de los números racionales.
- $\mathbb{R}$: el conjunto de los números reales.

Supongamos que tenemos un valor x y un conjunto S y queremos saber si x es un elemento de S. Existe una notación para esa pregunta: la expresión $x \in S$ es verdadera si x es un elemento de S y falsa en caso contrario. Esta expresión se lee "x pertenece a S", "x es un elemento de S" o simplemente "x está en S". Por ejemplo,

$$3 \in A \equiv \mathit{True}$$

$$4 \in A \equiv \mathit{False}$$

Un operador que podemos definir en término del $\in$ es el símbolo de "no pertenece" $\notin$. $x \notin A$ se puede interpretar como que no es cierto que $x \in A$. En términos lógicos lo podemos expresar de la siguiente manera

$$x \notin A \equiv \neg(x \in A)$$

Cuando un conjunto tiene pocos elementos se pueden escribir entre llaves; como los conjuntos A y B que definimos arriba. Sin embargo cuando un conjunto tiene muchos elementos esta posibilidad se hace más difícil y, si los conjuntos tienen un número infinito de elementos se hace imposible. El conjunto D tiene 101 elementos, pero es más legible si utilizamos la notación con puntos suspensivos que si escribimos cada uno de los elementos. El conjunto $\mathbb{N}$ tiene un número infinito de elementos y no podemos escribirlos todos entre llaves.

Una de las razones por las que utilizamos la matemática en la computación es para ser precisos y formales, para estar absolutamente seguros de que no hay ambigüedades en lo que estamos definiendo. La notación con puntos suspensivos es informal y se basa en la intuición del lector para entender y llenar los puntos. Es fácil construir casos en los que gente diferente puede interpretar de manera diferente la misma expresión. Por ejemplo $\{2, 3, ..., 7\}$, ¿significa el conjunto de los números naturales de 2 asta 7? ¿o el conjunto de los números primos desde 2 hasta 7? Se estás consiente de este problema cuando describís un conjunto podrías prevenirlo, pero ¿cómo podés estar seguro que tu conjunto está realmente bien definido, es decir, que hay sólo una forma de interpretarlo? Es problema no es tan serio para conjuntos finitos porque, en principio, podrías escribir todos sus elementos. El problema es más fundamental para conjuntos infinitos.

Otra manera estándar de definir conjuntos es por *comprensión*. Es su forma más simple, un conjunto por comprensión se escribe como

$$\{x \mid p.x\}$$

donde p es simplemente una expresión que contiene x y es verdadera o falsa para diferentes valores de x. Esta expresión se llama **predicado**. La expresión $\{x \mid p.x\}$ se lee, "el conjunto de los x tal que $p.x$". Por ejemplo, podemos definir el conjunto de los números pares como

$$\{x \mid x \in \mathbb{N} \wedge par.x\}$$

En este caso el predicado $p.x$ sería la expresión $x \in \mathbb{N} \wedge par.x$. Esta expresión es verdadera cuando x es un número natural y es par. En castellano podemos leer extra expresión como "el conjunto de los x tal que x es natural y es par" o simplemente "el conjunto de los números naturales pares".

Una forma más general de las definiciones por comprensión es

$$\{f.x \mid p.x\}$$

En este caso, el conjunto no son los valores de x que satisfacen el predicado, si no el resultado de aplicar la función f a esos valores. Esta forma de conjuntos por comprensión muchas veces nos permite definir los conjuntos de manera más simple. Por ejemplo

$$\{2 * x \mid x \in \mathbb{N}\}$$

define el conjunto de números naturales pares, es decir, el mismo conjunto que en la definición anterior, pero en este caso no necesitamos definir el predicado $par.x$.

4.2. Operaciones sobre conjuntos

Hasta ahora vimos dos operaciones sobre conjunto. La operación de pertenece ($\in$) y la de no pertenece ($\notin$). En esta sección veremos otras operaciones que nos permitirán comparar conjuntos y construir nuevos conjuntos a partir de otros.

4.2.1. Igualdad e inclusión

Una relación entre conjuntos, más elemental que pertenece, es la *igualad*. La igualdad de dos conjuntos A y B se denota

$$A = B$$

y cuando no son iguales lo podemos denotar

$$A \neq B$$

Conceptos teóricos

Dos conjuntos son *iguales* si y sólo sí tienen los mismos elementos.
en símbolos esta definición se puede escribir de la siguiente manera:

$$A = B \equiv x \in A \equiv x \in B$$

[a]

[a] Notar que esta fórmula contiene un cuantificador universal implícito para la x que no escribiremos para simplificar el cálculo.

Otra relación muy utilizada es la de inclusión. Esta relación la denotamos $\subseteq$. La expresión $A \subseteq B$ se lee com "A está incluido en B" o "A es un subconjunto de B" y es verdadera cuando cada elemento que pertenece a A también pertenece a B.

Conceptos teóricos

Decimos que un conjunto A está incluido en B, $(A \subseteq B)$ si todo elemento que pertenece a A también pertenece a B.
en símbolos esta definición se puede escribir de la siguiente manera:

$$A \subseteq B \equiv x \in A \Rightarrow x \in B$$

[a]

[a] Notar que esta fórmula contiene un cuantificador universal implícito para la x que no escribiremos para simplificar el cálculo.

Una técnica muy utilizada para demostrar igualdad de conjuntos es demostrar la doble inclusión, es decir $A = B$ si y sólo sí $A \subseteq B$ y además $B \subseteq A$.

Si A es un subconjunto de B pero no son iguales, entonces todos los elementos de A están en B pero debe haber algún elemento de B que no está en A. En este caso se dice que A es un *subconjunto propio* de B y se denota $A \subset B$ o $A \subsetneq B$.

Conceptos teóricos

Decimos que un conjunto A es un subconjunto propio de B, $(A \subset B)$ si todo elemento que pertenece a A también pertenece a B y $B \neq B$.
en símbolos esta definición se puede escribir de la siguiente manera:

$$A \subset B \equiv A \subseteq B \wedge A \neq B$$

4.3. Unión, Intersección y Diferencia

Conceptos teóricos

La unión de dos conjuntos A y B da como resultado un nuevo conjunto que contiene todos los elementos de ambos conjuntos y se denota $A \cup B$. Es decir
$A \cup B = \{x \mid x \in A \vee x \in B\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$x \in A \cup B \equiv x \in A \vee x \in B$$

Conceptos teóricos

La intersección de dos conjuntos A y B da como resultado un nuevo conjunto que contiene sólo los elementos que comparten ambos conjuntos y se denota $A \cup B$. Es decir
$A \cap B = \{x \mid x \in A \wedge x \in B\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$x \in A \cap B \equiv x \in A \wedge x \in B$$

Conceptos teóricos

La diferencia de dos conjuntos A y B da como resultado un nuevo conjunto que contiene todos los elementos de A que no están en B y se denota $A - B$.Es decir
$A - B = \{x \mid x \in A \wedge x \notin B\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$x \in A - B \equiv x \in A \wedge x \notin B$$

4.4. Partes, producto cartesiano y cardinal

Conceptos teóricos

Partes de un conjunto A es el conjunto de todos los subconjuntos de A. Es decir,
$\mathcal{P}(A) = \{s \mid s \subseteq A\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$s \in \mathcal{P}(A) \equiv s \subseteq A$$

Por ejemplo, si $A = \{1, 2, 3\}$ entonces

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$$

Conceptos teóricos

El producto cartesiando de dos conjuntos A y B es el conjunto de pares ordenados, con el primer elemento pertenece a A y el segundo elemento pertence a B, y se denota $A \times B$. Es decir,
$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$(a, b) \in A \times B \equiv a \in A \wedge b \in B$$

Por ejemplo, si $A = \{1, 2, 3\}$ y $B = \{5, 7\}$ entonces

$$A \times B == \{(1, 5), (1, 7), (2, 5), (2, 7), (3, 5), (3, 7)\}$$

Conceptos teóricos

El cardinal de un conjunto A, para conjuntos finitos, es la cantidad de elementos que tiene y se denota $|A|$ o $\#A$

Por ejemplo, si $A = \{1, 2, 3\}$ entonces $|A| = 3$

Actividad

Si $|A| = n$ y $|B| = m$ calcular $|A \times B|$

Actividad

Si $|A| = n$ calcular $|\mathcal{P}(A)|$

4.4.1. Complemento y conjunto universal

La operación de complemento es una operación muy delicada porque asume la existencia del conjunto universal. Es decir, un conjunto que contenga todos los elementos posibles. Este conjunto, en términos generales NO EXISTE. De todas maneras, para algunos casos, es posible identificar un dominio que nos permita definir un conjunto de referencia sobre el dominio de trabajo. A este conjunto lo denotaremos $\mathcal{U}$.

Conceptos teóricos

El complemento de un conjunto A es el conjunto de todos los elementos del dominio, es decir, del conjunto universal de referencia, que no pertencen al conjunto A. A este conjunto lo denotamos A' o A^c. Es decir,
$A' = \{x \mid x \notin A\}$
en símbolos esta definición se puede escribir de la siguiente manera:

$$x \in A' \equiv a \notin A$$

4.5. Propiedades sobre conjuntos

T1 *Idempotencia de la unión:*

$A \cup A \equiv A$

T2 *Elemento Neutro de la union:*

$A \cup \emptyset \equiv A$

T3 *Elemento Absorbente de la union:*

$A \cup \mathcal{U} \equiv \mathcal{U}$

T4 *Idempotencia de la intersección:*

$A \cap A \equiv A$

T5 *Elemento Neutro de la intersección:*

$A \cap \mathcal{U} \equiv A$

T6 *Elemento Absorbente de la intersección:*

$A \cap \emptyset \equiv \emptyset$

T7 *De Morgan:*

$(A \cup B)' \equiv A' \cap B'$

$(A \cap B)' \equiv A' \cup B'$

Actividad

Enunciar las propiedades de asociatividad y conmutatividad de la suma y la intersección.

Actividad

¿Te animás a enunciar la propiedad de transitividad de la inclusión?

Actividad

Demostrá los siguientes teoremas

1. $A - B \equiv A - (A \cap B)$
2. $(A - B) \cup (B - A) \equiv (A \cup B) - (A \cap B)$

Capítulo 5

Relaciones y funciones

En este capítulo definiremos los que son las relaciones y algunas propiedades importantes que sirven para caracterizar diferentes relaciones y, además, trabajaremos con un tipo particular de relaciones que son las funciones.

5.1. Relaciones

Hay muchos tipos de relaciones en la vida diaria. Algunas pueden describir cómo los miembros de una familia se relacionan entre ellos: padres, hijos, hermanos. También podríamos definir la función *estaEn* para ciudades y países: por ejemplo Londres está en Gran Bretaña, París está en Francia, Córdoba está en Argentina. O pedemos tener una relación que describe qué auto es producido cuál fabricante. Las relaciones son utilizadas en matemática para describir como dos números se relacionan entre sí; por ejemplo, expresiones como $x < y$ y $p \geqslant q$ utilizan las relaciones $<$ y $\geqslant$.

Una *relación binaria* se utiliza para describir una relación entre dos objetos. Otras relaciones que relacionan n objetos se llaman *relaciones n-arias*. De todas maneras, las relaciones binarias son las más importantes por lo que nos centraremos en éstas.

Conceptos teóricos

Una relación binaria R con tipo $R : A \times B$ es un subconjunto de $A \times B$ (el producto cartesiano de A por B), donde A es el dominio de la relación y B es el codominio.

Notación: Si el dominio de R es A y el codominio es B, lo denotamos como $R : A \times B$ y se lee "R es de tipo A por B".

Para $x \in A$ y $y \in B$, la notación $x\ R\ y$ significa que $(x, y) \in R$. Cuando no haya dudas de cuál es la relación involucrada escribiremos $x \sim y$, y se lee "x se relaciona con y".

Actividad

De 10 ejemplos de elementos que pertenecen a las siguientes relaciones.

1. $R_1 = \{(x, y) \mid (y - x) \in \mathbb{N} \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}\}$
2. $R_2 = \{(x, y) \mid (x - y) \in \mathbb{N} \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}\}$
3. $R_3 = \{(x, y) \mid x \mod 10 = y \mod 10\}$
4. $R_4 = \{(x, y) \mid x \mod 5 = y \mod 5\}$

Conceptos teóricos

La relación identidad con dominio A y codominio A, y denotada id_A se define como $id_A = \{(x, x) \mid x \in A\}$

Actividad

Definir por extensión la relación identidad sobre el conjunto $A = \{1, 2, 3\}$

Propiedades de las relaciones

Pensemos en un conjunto de personas P que vive en una cuadra determinada. Estas personas viven en 4 casas diferentes. Sofía, Patricia y Fernando viven en la primera casa. Agustín, Hernan, Victor y Elena viven en la segunda. José vive solo en la tercera y Karina y Tania viven juntas en la cuarta casa.

Queremos definir por extensión la relación *viveCon* que dadas dos personas, la primera se relaciona con la segunda si viven en la misma casa. Para simplificar la descripción utilizaremos sólo la inicial del nombre para referirnos a cada persona.

Esta relación se puede expresar por extensión de la siguiente manera

$$\begin{aligned} viveCon = \{&(s,s),(s,p),(s,f),(p,s),(p,p),(p,f),(f,s),(f,p),(f,f),\\ &(a,a),(a,h),(a,v),(a,e),(h,a),(h,h),(h,v),(h,e),(v,a),(v,h),(v,v),(v,e),\\ &(e,a),(e,h),(e,v),(e,e),(j,j),(k,k),(k,t),(t,k),(t,t)\} \end{aligned}$$

En este caso, interpretamos la definición de la relación "viven en la misma casa" como una relación *reflexiva*, es decir, que una determinada persona vive en la misma casa que ella misma. Por ejemplo, Sofía vive en la misma casa que Sofía ($s \sim s$). También se puede observar que a pesar de que agregamos que Patricia vive en la misma casa que Fernando ($p \sim f$), es necesario agregar expresamente que Fernando vive en la misma casa que Patricia ($f \sim p$), porque si bien nosotros lo podemos interpretar cuando pensamos qué significa vivir en la misma casa, cuando lo expresamos matemáticamente es necesario ser riguroso con todos los detalles. Esta propiedad que cumplen algunas relaciones se llama *simetría*. Otra propiedad que aparece frecuentemente en las relaciones es la *transitividad*. En términos de nuestro ejemplo sería, Si Hernán vive en la misma casa que Víctor ($h \sim v$) y Víctor vive en la misma casa que Elena ($v \sim e$), entonces Hernán y Elena viven en la misma casa ($h \sim e$).

A aquellas relaciones que cumplen estas tres propiedades, se les llama relaciones de equivalencia. Observar que las relaciones R_3 y R_4 que ya definimos, son relaciones de equivalencia.

Conceptos teóricos

- Una relación es *reflexiva* si para todo x del dominio, $x \sim x$
- Una relación es *simétrica* si para todo x, y si $x \sim y$ entonces $y \sim x$.
- Una relación es *transitiva* si para todo x, y, z si $x \sim y \wedge y \sim z$ entonces $x \sim z$.

Se llama *relación de equivalencia* a las relaciones que son reflexivas, transitivas y simétricas.

No todas las relaciones son relaciones de equivalencia. Supongamos que tenemos un conjunto de palabras $\{hola, casa, azul, perro\}$ definamos la relación *antesQue* que describe si una palabra se escribe antes que otra en el diccionario. Esta relación quedaría definida por extensión de la siguiente manera

$$\{(azul, casa), (azul, hola), (azul, perro), (casa, hola), (casa, perro), (hola, perro)\}$$

Esta relación no es reflexiva, porque, por ejemplo, "azul" no se escribe antes que "azul".

Actividad

Explicar por qué la relación *antesQue* es transitiva.

Observar que esta propiedad no es simétrica, ya que no puede pasar que si una palabra se escribe antes que otra, entonces la otra se escribe antes que la primera. En algunas relaciones que no son simétricas puede pasar que cuando $x \sim y$ a veces pasa que $y \sim x$ y otras veces no. Hay otras relaciones, como es el caso de *antesQue* en qué nunca sucede que si $x \sim y$ ocurra que $y \sim x$, salvo que x e y sean iguales. Esta propiedad se llama *antisimetría*.

Conceptos teóricos

- Una relación es *antisimétrica* si $x \sim y \wedge y \sim x$ sólo puede ocurrir cuando $x = y$.

Se llama *relación de orden* a las relaciones que son reflexivas, transitivas y antisimétricas.

Actividad

Explicar por qué la relación *antesQue* no es de orden y la relación menor o igual ($\leqslant$) sí lo es.

5.1.1. Composición e inversa

Conceptos teóricos

Sean $R : A \times B$ y $S : B \times C$ dos relaciones, entonces $R \circ S$ es una nueva relación de tipo $A \times C$ definida de la siguiente manera

$$R \circ S = \{(x, z) \mid \exists y \in B \text{ tal que } x \sim y \wedge y \sim z\}$$

la operación $\circ$ se llama *composición* de relaciones.

Supongamos que $R = \{(1, a), (1, b), (2, c)\}$ y $S = \{(a, 25), (b, 33), (b, 50), (c, 50)\}$; entonces

$$R \circ S = \{(1, 25), (1, 33), (1, 50), (2, 50)\}$$

Supongamos que tenemos un conjunto de personas y definimos la definición *esPadreDe* que relaciona dos personas si la primera es el padre de la segunda, es decir $(x, y) \in$ *esPadreDe* significa que x es el padre de y. Ahora podemos definir la función *esHijoDe* tomando la *inversa* de la relación ya definida; es decir, si $(x, y) \in$ *esPadreDe* entonces y es hijo de x y por lo tanto $(y, x) \in$ *esHijoDe*. Esta operación de invertir una relación R se llama *inversa* y se denota con R^{-1}. En caso podemos definir la relación *esHijoDe* como *esPadreDe*$^{-1}$.

Conceptos teóricos

La inversa de una relación $R : A \times B$ es otra relación de tipo $B \times A$ y denotada R^{-1}, definida de la siguiente manera:

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}$$

5.2. Funciones entendidas como relaciones

Conceptos teóricos

Una función de tipo A en B ($A \to B$), es una relación de tipo $A \times B$ tal que para todo x que pertenece a A ($\forall x \in A$), existe un único y en B ($\exists! y \in B$) tal que $x \sim y$.

Notación: Dada $f : A \to B$, si $(x, y) \in f$ lo denotamos como $f(x) = y$ o como $f.x = y$

Actividad

- Dar un ejemplo de una función con dominio y codominio $A = \{1, 2, 3\}$
- Dar dos ejemplos de relaciones que no sean función de tipo $A \times A$. Explicar por qué no son funciones.

Observar que la relación identidad (id) es una función.

Conceptos teóricos

Una función $f : A \to B$ es *suryectiva* si $\forall y \in B \exists x \in A \text{tq} f.x = y$.

En palabras: Una función es suryectiva, si para cada elemento y que pertenzca al codominio B podemos encontrar un elemento x en el dominio A tal que $f.x = y$.

Las funciones suyectivas también suelen ser llamadas *subyectivas* o *sobreyectivas*.

Conceptos teóricos

Una función $f : A \to B$ es *inyectiva* si $\forall x_1, x_2 \in A$ si $x_1 \neq x_2$ entonces $f.x \neq f.y$

En palabras: una función es inyectiva si cada vez que elegimos dos elementos diferentes del dominio ($x_1 \neq x_2$) también sus imágenes son diferentes ($f.x_1 \neq f.x_2$).

Las funciones inyectivas también suelen ser llamadas *uno a uno*.

Conceptos teóricos

Una función es *biyectiva* si es inyectiva y suryectiva.

Actividad

Sean $A = \{1, 2, 3\}$, $B = \{1, 2\}$ y $C = \{1, 2, 3, 4\}$. Definir

- Definir una función inyectiva, una función suryectiva y una función biyectiva de tipo $A \to A$.
- Definir una función inyectiva, una función suryectiva y una función biyectiva de tipo $A \to B$.
- Definir una función inyectiva, una función suryectiva y una función biyectiva de tipo $A \to C$.

Para reflexionar: ¿Es posible definir todas las funciones solicitadas?

Capítulo 6

Funciones y recursión

Este capítulo está enteramente dedicado a dos conceptos claves en programación: las funciones y la recursión.

6.1. Un breve repaso

Para comenzar a realizar una revisión de las formas estándar de definir una 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, la duración de un viaje T entre entre dos ciudades depende de la distancia d en la que están separadas y de la velocidad v que lleve el tren. 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 ($T = d/v$) pueden ser de proporcionalidad
>
> 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:

$$\begin{array}{ccccccccc} \ldots & & \text{estación} & \text{museo} & \text{arrollo} & \text{rosa} & \text{avión} & \ldots \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & & \text{E} & \text{M} & \text{A} & \text{R} & \text{A} & \end{array}$$

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

Actividad

- ¿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:

$$f : A \to B$$

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

Utilizaremos la siguiente notación:

$$f.a = b$$

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 esta al valor x que se escribe como $f.x$. El símbolo f representa al conjunto de todas las aplicaciones, en cambio una aplicación de la función, como $f.5$ representa sólo el valor que la función le asigna a ese argumento.

6.2. ¿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 la 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 $\doteq$. Este será necesario para no confundirlo con el símbolo $=$.

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

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

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 fórmulas, como en los siguientes ejemplos:

$$f.x \doteq 2x^2 + 4$$

$$g.y \doteq 3$$

$$h.z \doteq 7z$$

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 estos casos anteriores las variables x, y, z representan números naturales. Todas ellas se denominan *variables libres* porque pueden reemplazarse por otra sin alterar la definición de la función. Así, el función h podría definirse en términos de la variable w de la siguiente forma y seguiría siendo la misma función:

$$h.w \doteq 7w$$

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

$$f.x.y \doteq x + y$$

toma dos números naturales y devuelve el resultado de sumarlos.

Actividad

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

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

6.3. 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.

Las funciones toman uno o más argumentos, todos ellos con un tipo asociado.

Así, la función f definida por la siguiente fórmula:

$$f.x \doteq x + 4$$

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:

$$f :: Num \rightarrow Num$$

Veamos otro ejemplo. Tomemos la función g definida así:

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

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

$$g :: Num \rightarrow Num \rightarrow Bool$$

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

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 tienen que tener sus argumentos y declará el tipo de cada una de las funciones:

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

6.4. Evaluación de funciones

Una vez que tenemos definida una función esta puede evaluarse para argumentos particulares. Es importante resaltar que las variables con las que está definida una función pueden tomar cualquier valor siempre y cuando el tipo sea el correcto. Así, es 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:

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

Evalualas en los siguientes argumentos:

1. $g.7$
2. $multiplicar.3.(g.2)$
3. $por5.(duplica.8)$

6.5. Distintas formas de definir funciones

En esta sección nos ocuparemos de introducir la notación necesaria para definir funciones de diferentes formas.

6.5.1. Definiciones locales

Cuando definimos una función es posible también incluir en ella una o varias *definiciones locales*. 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 raices de la ecuación $ax^2 * bx * c$. Una posible definición de esta función, utilizando definiciones locales, es la siguiente:

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

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

En esta definición se incluye a la variable *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:

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

6.5.2. 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 depende de que ciertas expresiones booleanas sean ciertas o no.

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

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

donde las B_i son expresiones de tipo booleano, llamadas *guardas* y las f_i son expresiones del mismo tipo que f. El valor que tomará la función corresponde a aquel para la cual la expresión booleana es cierta.

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:

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

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 y el valor de la función es entonces y, es decir, 5.

Por último, definamos la función *anterior* que toma un número y devuelve otro número que será 0 si el número es 0 o el número anterior si éste es distinto de cero:

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

donde $x \; mod \; 2$ representa el resto de la división entera de x por 2

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 menoro 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.

6.5.3. 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 a un conjunto de números.

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 de la función en los 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:

$$f.0 \doteq 0$$

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:

$$f.(n + 1) \doteq n$$

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 si no lo es entonces puede escribirse con el patrón $(n + 1)$.

La definición completa de la función será:

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

Es importante notar que el patrón sirve no sólo para distinguir los casos sino también para tener un nombre en el cuerpo de la definición —$n + 1$ aparece también del lado derecho de la definición.

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 la función *anterior* quede mucho más compacta sin tener que recurrir a la resta, como lo hacíamos cuando la definíamos por casos.

Otros patrones posibles para los naturales podrían ser por ejemplo 0, 1 y $n + 2$, lo cual estaría asociado a tres casos, dos que consideran los casos en que el argumento es 0 ó 1 y uno para el caso en que el número considerado es 2 o más. Una definición que utilice este patrón tendrá la siguiente forma:

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

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

Un último patrón para los naturales podría ser separar los números 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:

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

Actividad

Descubrí que hace la función definida de la siguiente forma:

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

6.5.4. 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. Recordar 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*.

6.5.5. Resumiendo

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

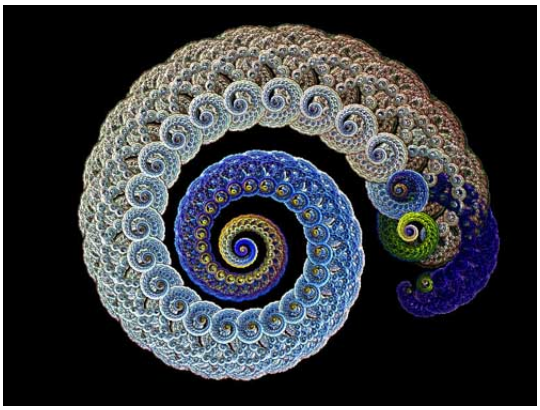
- Una o más definiciones cada una con la siguiente forma: lo que está a la izquierda del símbolo $\doteq$ es el nombre de la función y el o los argumentos que toma y a la derecha hay
  - o una expresión simple
  - o una expresión por casos.

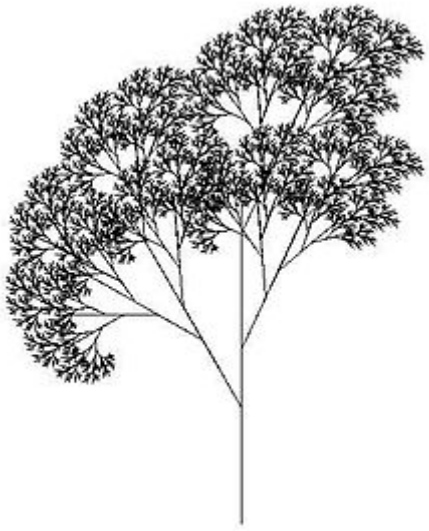
Notá que esta enumeración de definiciones puede ser o una enumeración caso por caso donde se establece el valor de la función para cada uno de los argumentos posibles, lo que significaría definir por extensión una función o puede ser una enumeración que utilice una serie de patrones, lo que involucraría definir la función usando pattern matching.

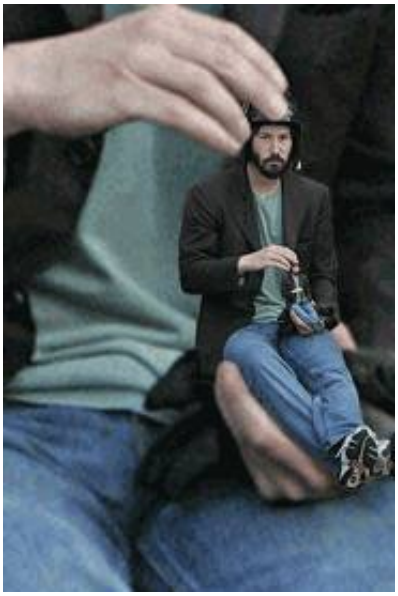
6.6. Recursión

Actividad

¿Qué tienen en común estas fotografías y el siguiente texto?







> 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

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.*

6.6.1. Funciones recursivas

Definir una función recursiva 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: $factorial.0 = 1$, $factorial.3 = 3 * 2 * 1$. Esta función se puede definir de manera recursiva de la siguiente forma:

$$factorial.0 = 1$$

$$factorial.(n+1) = (n+1) * factorial.n$$

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 definiendo algo en función de sí mismo. Esto implicaría una circularidad, y conduciría a una regresión infinita o a una paradoja. En realidad, una definición recursiva de una función —si está correctamente formulada— jamás conduce a una regresión infinita ni a una paradoja 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:

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

En el penúltimo paso se aplica el caso base, que es el que nos asegura que el cálculo en algún momento acaba. Al intentar calcular $factorial.n$ la función llamará a $factorial.(n-1)$ lo que requerirá, a su vez, que se calcule $factorial.(n-2)$ y así sucesivamente hasta que se llegue a $factorial.0$. 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:
>
> $$factorial.n = 1 * 2 * 3 * \ldots * n$$
>
> 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 algeraicas 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 entonces encuentra una forma de resolver las ecuaciones.
>
> *Adaptación libre de O'Donnell, Hall & Page (2006) Discrete Mathematics using a computer. Págs: 47-48.*

- ¿Qué vínculos hay entre definciones recursivas y la programación?
- ¿Qué es un programa 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.

Capítulo 7

Listas

7.1. Definiendo las listas

En este curso, consideraremos que una lista es una colección de valores ordenados, los cuales deben ser todos del mismo tipo. Consideraremos sólo 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*.

Un aspecto que diferencia a las listas de los conjuntos es que éstas 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 como podremos determinar 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.

7.1.1. ¿Cómo construir una lista?

Hasta aquí hemos introducido la notación y el tipo de nuestro nuevo tipo de datos: las listas. 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?

Este tipo de datos tendrá dos *constructores*:

- 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 más cómoda de describir a 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. $[[[5]]]$

7.2. 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.

7.2.1. 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 bien 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 respuestas anteriores determiná qué patrón puede ser útil para definir funciones sobre listas.

Notemos ahora que los constructores de las 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 $\triangleright$. Esta característica de la forma en que se construyen las listas hace que sea 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.

En el caso de la función $\#$, el **caso base** trabajará con la lista vacía. Como esta lista no contiene ningún elemento entonces para este caso el cardinal debe valer 0.

El **caso inductivo** trabajará con una lista que tenga al menos un elemento, denotándola por $x \triangleright xs$. 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.

7.2.2. 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.

7.2.3. 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.

7.2.4. 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.

7.3. Expresiones que contengan 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 es preciso resolver 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:

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

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. Interpretar los resultados a partir de tu conocimiento de las funciones:

1. $[23, 45, 6] \triangleleft (cabeza.[1, 2, 3, 10, 34])$
2. $[0, 11, 2, 5] \triangleright [\,]$
3. $cabeza.(0 \triangleright [1, 2, 3])$
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 reglas 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 reglas 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. $x \triangleright ([[True]] \triangleright y)$

7.4. 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?

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

4. ¿Qué hace la siguiente función?

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

   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 compartiría 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 las listas de la lista 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?

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

4. La función f definida a continuación utiliza a la función *maximo* que desarrollamos en la sección 6.5.2, en la página 49. Revisá la definición de *maximo* y luego evaluá y analizá la definición de f para averiguar qué hace:

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

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 *quita*0 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 cual será el tipo de la función.

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

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

   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 te es útil, podés evaluarla en FUN para descubrir qué hace.

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

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 la función toma 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 es una 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:

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

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.

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

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 #?

   $$f.xss = \#(concat.xss)$$

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

7.5. 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

entonces 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 el caso base 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.

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

Conceptos teóricos

Sea que $P(xs)$ es una propiedad de una lista xs de tipo $[A]$. Si

1. $P([\,])$ es verdadera.
2. **Si** $P(xs)$ vale para una lista arbitraria xs de tipo $[A]$ **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:

$$\begin{array}{ll} & \underline{[\,] +\!\!+ [\,]} = [\,] \\ \equiv & \{\ \text{Definición de } +\!\!+ \text{ caso base}\ \} \\ & [\,] = [\,] \\ \equiv & \{\ \text{Reflexividad}\ \} \\ & True \end{array}$$

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:

$$\begin{array}{ll} & \underline{(x \triangleright xs) +\!\!+ [\,]} = (x \triangleright xs) \\ \equiv & \{ \text{ Definición de } +\!\!+ \ \} \\ & x \triangleright \underline{(xs +\!\!+ [\,])} = (x \triangleright xs) \\ \equiv & \{ \text{ Hipótesis inductiva } \} \\ & x \triangleright xs = (x \triangleright xs) \\ \equiv & \{ \text{ Reflexividad } \} \\ & True \end{array}$$

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

Este formato también es el que FUN te propondrá cada vez que quieras construir una demostración por inducción usando esta herramienta.

Actividad

Utilicemos el principio de inducción para demostrar un teorema útil acerca de la relacón 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: $\#(duplica.xs) = \#xs$
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 62? ¿Por qué?
5. Demostrá el siguiente teorema: $sum(duplica.xs) = 2 * sum.xs$
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: $sum(sumauno.xs) = \#xs + sum.xs$
   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 $+\!\!+$. La propiedad que demostraremos vincula la función *concat* con la función $+\!\!+$ de manera tal que nos permitirá hacer más efectivos algunos de nuestros programas:

$$concat(xss +\!\!+ yss) = concat.xss +\!\!+ concat.yss$$

Capítulo 8

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. 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 matemática cuyas ideas claves están mencionadas en la cita de Pascal de más arriba. 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 he implica un esfuerzo de formalización que será recompensado ganando estrategias de prueba muy potentes.

8.1. 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,este es un sistema que 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*? 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.

8.2. Definición 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:

$$+ :: Nat \to Nat \to Nat$$

$$a + b \doteq \text{ definición de la función}$$

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 $\mathbb{N}$. Visto que dicha definición dado un número nos permite construir el número siguiente a partir del constructor *suc*, parece 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 uno solo de ellos. Sólo cuando, después de intentarlo, concluyamos que no es posible hacerlo en términos recursivos solo para uno de los argumentos, consideraremos definirla recursivamente para los dos argumentos.

Así, definamos a 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 o 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 8.2)

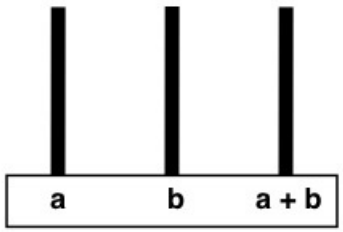


Figura 8.1: Abaco

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 ahora qué significaría realizar la operación 3+4 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 8.2)

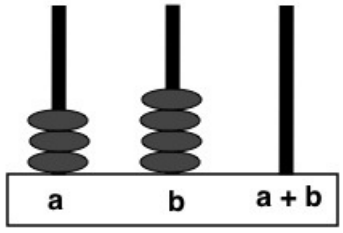


Figura 8.2: Situación inicial de la operación $3 + 4$

Sumar utilizando un ábaco implica mover todas las fichas de la primera y la segunda vara a la tercer vara que representaba 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 tercer columna (ver figura 8.2):

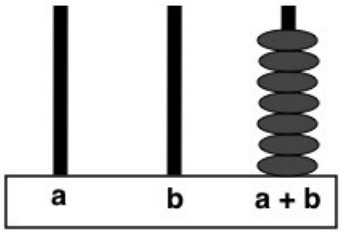


Figura 8.3: Situación final de la operación $3 + 4$

En la última columna 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 esto 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 8.2)

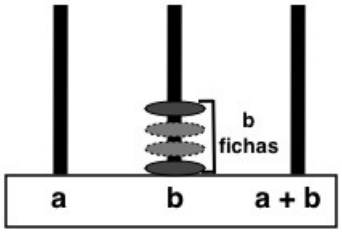


Figura 8.4: Situación inicial del caso base

Realizar la suma $0 + b$ implicaba 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 + visto que 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).

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 8.2)

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.

Al contar cuantas fichas quedaron en la tercera vara tendremos algo del estilo (ver figura 8.2):

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

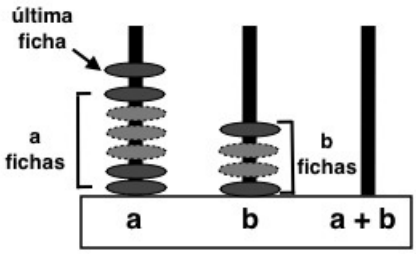


Figura 8.5: Situación inicial del caso inductivo

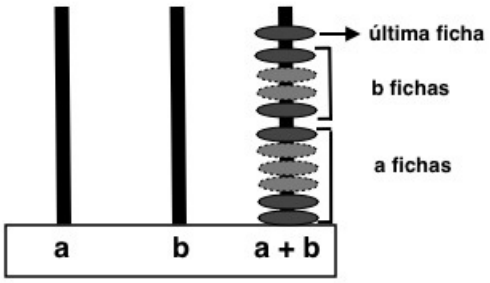


Figura 8.6: Situación final del caso inductivo

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. Esto nos permite transformar la segunda parte de la expresión anterior así:

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

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

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 algunos casos particulares:

$$\begin{array}{rcl} 2 + 5 & = & suc(suc(0)) + suc(suc(suc(suc(suc(0))))) \\ & & \text{Definición de suma, caso inductivo} \\ & = & suc(\ suc(0)) + suc(suc(suc(suc(suc(0))))\ ) \\ & & \text{Definición de suma, caso inductivo} \\ & = & suc(\ suc\ (0 + suc(suc(suc(suc(suc(0))))\ )\ ) \\ & & \text{Definición de suma, caso base} \\ & = & suc(\ suc\ (suc(suc(suc(suc(suc(0))))\ )\ ) \end{array}$$

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 recordaremos cómo aprendimos a multiplicar en la escuela primaria: la multiplicación se define en términos de la suma, así 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:

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

Actividad

La siguiente función es una versión en los naturales de la resta. 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. Escribila en FUN , completá las definiciones para los otros dos casos y evaluala para chequear que esté bien definida:

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

¿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:

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

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 básicas sobre los naturales. 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 cosas sobre estas funciones.

8.3. Principio de Inducción

Para introducir el principio de inducción utilizaremos la siguiente 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 de la inducción matemática, uno de los métodos de demostración más importantes en matemática y en programación.

Este principio de utiliza cuando, dado un conjunto infinito de proposiciones:

$$P(0), P(1), P(2), \ldots, P(n), P(n+1)$$

se quiere demostrar que son todas verdaderas. Utilizando la definición inductiva de los números naturales, este conjunto de proposiciones puede escribirse como sigue:

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

Cada proposición 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 la validez de una proposición, por ejemplo, $P(n)$ implica la validez de la proposición siguiente, $P(n+1)$, entonces probaríamos que cada ficha está lo suficientemente cerca como para voltear la siguiente. Si también podemos demostrar que la proposición inicial $P(0)$ es verdadera —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. 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. 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. Visto que la inducción nos servirá para demostrar proposiciones no sólo sobre los números naturales sino también sobre otros tipos de datos que iremos introduciendo, a lo largo del desarrollo del material iremos ocupándonos de cada uno de estos casos.

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 proposiciones:

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

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

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

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 proposición que vamos a probar —usualmente la llamaremos $P(n)$. 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 una proposición que 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 cualquiera 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, que se suele llamar *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 +:

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

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 llamado *caso inductivo*. Siempre 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 lo que se hace generalmente es usar la propiedad $P(n)$ como hipótesis en la demostración de que $P(suc(n))$ es equivalente a $True$. A esta hipótesis 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:

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

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: Sea P(n) una proposición. Sí

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 proposiciones anteriores para demostrar la validez de

la proposición 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 el siguiente:

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 proposición 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** $P(0), P(suc(0)), \dots P(n) \Rightarrow P(suc(n))$. 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.

Este formato también es el que FUN te propondrá cada vez que quieras construir una demostración por inducción usando esta herramienta.

8.3.1. 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:**

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

   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:

$a + suc(0) = suc(a)$

8.3.2. 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:

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

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:

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

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.

8.3.3. 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 que 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 restanets son rubias. Esta es sólo una manera retorcida de decir "una rubia es rubia". Por lo tanto $P(suc0)$ es verdadera.
Caso inductivo: Ahora debemos demostrar que, dado un conjunto arbitrario S de $suc(n)$ mujeres 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, dentro de las $suc(n)$ mujeres de S. Entonces, ordenemos a las $suc(n)$ mujeres con la rubia en alguna posición:

$$r_1, r_2, \ldots, r, \ldots, r_n, r_{suc(n)}$$

A partir de este conjunto podemos definir dos subconjuntos:

$$A = r_1, r_2, \ldots, r, \ldots, r_n$$

y

$$B = r_2, \ldots, r, \ldots, r_n, r_{suc(n)}$$

Estos dos subconjuntos contienen n elementos y contienen a una rubia. Entonces, por hipótesis inductiva todos 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* Stephen, Maurer & Anthony Ralston (1998).

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

8.3.4. 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 ve que una ley matemática general ha sido descubierta por inducción, debemos demostrarla rigurosamente aplicando el enfoque deductivo.
>
> Traducción libre de *Fundamental of Algorithmics* (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

8.4. Funciones que «construyan» listas

En esta sección vamos a trabajar con funciones recursivas que, de distintas maneras, construyan una lista. Así, comenzamos a combinar en la recursión tanto a los naturales como a las listas, haciendo nuestro lenguaje cada vez más expresivo.

Para hacer más fácil la notación y visto que ya lo hemos demostrado utilizaremos en las definiciones el teorema que establece que $suc(n) = n + 1$. Esto nos permitirá escribir patrones en términos de n, $n + 1$, $n - 1$, etc., que es una notación un poco más intuitiva.

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 par tomar esta decisión intentando dar 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.

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 último 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.

Índice general