

Una formalización del sistema de los números reales

Jorge Ohel Acevedo Acosta
Jose Luis Echeverri Jurado

Tesis
para obtener el título de
Magíster en Matemáticas Aplicadas

Asesor: Andrés Sicard Ramírez

Departamento de Ciencias Matemáticas
Escuela de Ciencias
Universidad EAFIT
Medellín, Colombia
2017

Agradecimientos

A través de estas líneas queremos expresar nuestro más sincero agradecimiento a todas las personas que con su soporte científico y humano han colaborado en la realización de esta tesis de investigación. Muy especialmente a nuestro asesor y director de tesis al profesor Andrés Sicard Ramírez, por la acertada orientación, el soporte y discusión crítica que nos permitió un buen aprovechamiento en el trabajo realizado para que esta tesis llegara a buen término, así mismo agradecer a nuestros evaluadores Juan Carlos Agudelo Agudelo y Orlando García Jaimes, que contribuyeron mediante sus experiencias y criterios a la mejora de esta tesis.

Finalmente, agradecemos a nuestras familias por su comprensión, comunicación constante y apoyo. De manera muy especial a nuestras esposas quienes han estado a nuestro lado compartiendo nuestras alegrías y angustias, por el estímulo para que nos superemos día a día, el apoyo incondicional y la ayuda de siempre.

Resumen

En el campo del razonamiento automático se habla de la formalización de una teoría cuando sus teoremas son demostrados automáticamente o cuando sus demostraciones son verificadas por un asistente de pruebas. Se presenta una formalización de algunos aspectos del sistema de los números reales en el asistente de pruebas **Agda**. Se demuestran formalmente de forma interactiva y automática algunas propiedades del sistema. Las demostraciones automáticas fueron realizadas por demostradores automáticos de teoremas de propósito general para la lógica de predicados de primer orden.

Índice general

1. Introducción	1
1.1. Descripción del trabajo realizado	1
1.2. Estado del arte	3
1.3. Programas y códigos fuentes	4
2. Conceptos básicos de Agda	5
2.1. Términos y tipos	6
2.2. Tipos inductivos	6
2.3. Tipos dependientes	8
2.4. Argumentos implícitos	9
2.5. Tipos de datos y coincidencia de patrones	9
2.6. Funciones recursivas	10
2.7. Clausulas where	11
2.8. Abstracción with	11
2.9. Adición de axiomas	11
2.10. Funciones currificadas	11
2.11. Tipo identidad	12
2.12. Razonamiento ecuacional	13
3. Una formalización de una axiomática para el sistema de los números reales	15
3.1. Formalización de los axiomas de campo	16
3.2. Formalización de los axiomas de orden	18
3.3. Formalización del axioma de completitud	21
3.4. Una axiomática del sistema de los números reales en la matemática intuicionista	23

4. Demostraciones interactivas y automáticas	27
4.1. Demostraciones interactivas	28
4.2. Demostradores automáticos de teoremas y el lenguaje TPTP	37
4.3. El ATP-pragma	38
4.4. EL programa Apia	39
4.5. Demostraciones automáticas	41
5. Límites	45
5.1. Valor absoluto	45
5.2. Distancia entre dos puntos	49
5.3. Límite de una función	51
6. Conclusiones	53
6.1. Resultados	53
6.2. Trabajo futuro	54
Bibliografía	55
A. Propiedades de la igualdad proposicional	58
B. Definiciones	59
C. Demostraciones interactivas auxiliares	61

Capítulo 1

Introducción

En el campo del razonamiento automático se habla de la formalización de una teoría cuando sus teoremas son demostrados automáticamente o cuando sus demostraciones son verificadas por un asistente de pruebas. En esta tesis la palabra 'formalización' es usada con la acepción empleada en el campo del razonamiento automático. La formalización de una teoría matemática requiere de elementos y métodos avalados por esta ciencia, aún más cuando ésta es asistida por un computador. Desde mediados del siglo XX la matemática ha entrado en conexión con la utilización del computador para facilitar y, por qué no, potenciar el desarrollo de diversas teorías matemáticas. En el campo de las ciencias de la computación y la lógica matemática se encuentran, por ejemplo, los demostradores automáticos de teoremas, éstos son programas que permiten, entre otras aplicaciones, demostrar teoremas o propiedades de diferentes teorías. También existen en la actualidad diversos asistentes de prueba los cuales se vienen utilizando, entre otras aplicaciones, como herramientas para la escritura y verificación de demostraciones matemáticas formales [Geuvers 2009]. Estos asistentes son hoy en día una herramienta fundamental para la formalización en diferentes teorías matemáticas, como fue el caso de la demostración de la conjetura de Kepler [Hales 2005] o el teorema de los cuatro colores [Appel y Haken 1989].

1.1. Descripción del trabajo realizado

En la tesis que se presenta a continuación se desarrolla una formalización del sistema de los números reales utilizando el asistente de pruebas Agda [No-

1. Introducción

rell 2007]. Algunos teoremas se demuestran de forma interactiva, otros se demuestran de forma automática empleando el programa *Apia* [Sicard-Ramírez 2015, cap. 6], el cual es una interfaz entre *Agda* y los diferentes demostradores automáticos de teoremas.

La formalización del sistema de números reales presentada en esta tesis es desarrollada de forma axiomática, tal como lo presentan [Apostol 1974; Rosenlicht 1968; Royden y Fitzpatrick 2010]. Este desarrollo se focaliza en la definición de constantes, funciones y los axiomas usuales del sistema de los números reales, para luego demostrar algunos teoremas básicos. En la tesis se demuestran formalmente algunas propiedades de las desigualdades, propiedades que requieren demostraciones por inducción, el valor de algunas sumatorias y algunas propiedades de los límites. Se incluye además una sección desde la matemática intuicionista, donde además de los axiomas usuales se introduce la propiedad Arquimediana como un axioma, debido a que no es posible utilizar la ley del tercero excluido para demostrar esta propiedad a partir de los demás axiomas.

Al realizar la formalización utilizando un asistente de pruebas surgieron en el trasegar algunas limitaciones que no favorecieron el alcance deseado. Se menciona a continuación una de ellas. Si se hace el paralelo entre las demostraciones en papel y las demostraciones utilizando un asistente de pruebas, resulta un tanto más fácil hacerlo en papel dado que algunas propiedades pueden ser asumidas como ciertas. En muchos casos se demuestran propiedades asumiendo la existencia de una previa, o se demuestra una propiedad importante y las demás referidas a ésta son tomadas como consecuencia inmediata de la ya demostrada. Por ejemplo, la definición de la función raíz cuadrada es asumida en la demostración en papel de que el límite de un producto de dos funciones es igual al producto de los límites de cada una de ellas. De otra parte, cuando se formaliza la demostración de esta propiedad, se debe en primer lugar demostrar la existencia de la raíz cuadrada, demostrar algunas de sus propiedades y finalmente demostrar la propiedad deseada. Lo mismo se podría mencionar para la formalización de otras propiedades.

También se menciona que se invirtió un tiempo considerable en seleccionar que enfoque seguir para la formalización del sistema de los números reales. Además comenzar con todo el montaje no fue fácil. De otra parte, cuando se comenzó a formalizar las primeras propiedades implicó un cambio en la forma de razonar, pues escribir demostraciones utilizando un asistente de pruebas requiere tener en cuenta los más mínimos detalles. También el hecho de adentrarse en un mundo nuevo implicó la adquisición y comprensión de

1.2. Estado del arte

nuevos términos, una nueva manera de escribir, emplear nuevas técnicas y tácticas, entre otros aprendizajes.

En el proceso de formalización del sistema de los números reales se estudió su formalización en otros asistentes de pruebas y se observó que en algunos de ellos se emplean axiomas innecesarios. En esta tesis por el contrario cada axioma usado es estrictamente necesario lo cual dificulta el proceso de formalización. La formalización fue ardua y lenta, debido a que formalizar un concepto o una demostración tomó en varias ocasiones más del tiempo que se había planeado, pues aparecieron cuestionamientos, reflexiones, dudas que implicaban analizar, tomar decisiones y seguir el camino.

Esta tesis además de esta introducción está compuesta de cuatro capítulos y conclusiones los cuales son descritos a continuación. En el capítulo 2 se mencionan algunos conceptos básicos del programa **Agda**, el cual es un asistente de pruebas interactivo y un lenguaje de programación funcional donde se abordan temas tales como tipos inductivos, tipos dependientes, funciones recursivas, adición de axiomas y razonamiento ecuacional.

En el capítulo 3 se formalizan en **Agda** las definiciones de las operaciones de adición y multiplicación para luego hacerlo con los axiomas de campo, de orden y de completitud. Adicionalmente se presentan en este capítulo las diferencias que se hacen, desde la matemática intuicionista, al desarrollo axiomático del sistema de los números reales.

En el capítulo 4 se formalizan algunas demostraciones empleando **Agda**. Este tipo de demostraciones es lo que en esta tesis se denomina ‘demostraciones interactivas’. Además, utilizando el programa **Apia**, un traductor de la representación que hace **Agda** de fórmulas de primer orden al lenguaje TPTP (este lenguaje hace parte de la biblioteca de problemas para sistemas que prueban teoremas automáticamente para la lógica clásica) y algunos demostradores automáticos de teoremas, se demuestran de forma automática algunas de las propiedades en el sistema de los números reales que se demostraron de forma interactiva con el programa **Agda**. Finalmente en el capítulo 5 se ilustra la formalización de el límite de una función.

1.2. Estado del arte

Dado la importancia del sistema de los números reales no sorprende que su formalización haya sido realizada empleando diferentes metodologías y diferentes asistentes de prueba. Boldo, Lelay y Melquiond [2016] mencionan que

1. Introducción

los enfoques para la formalización del sistema de los números reales se pueden clasificar en tres grupos: i) formalizaciones que caracterizan los números reales como un conjunto dado con operaciones y propiedades, ii) formalizaciones donde los números reales se construyen como completación de los racionales y iii) formalizaciones basadas en los números hiperreales. Además estos autores mencionan que los asistentes de pruebas empleados incluyen `Coq` [The Coq Development Team 2016], `HOL4` [Slind y Norrish 2008], `HOL Light` [Harrison 2015], `Isabelle` [Nipkow, Paulson y Wenzel 2016] y `Mizar` [Bancerek et al. 2015] entre otros. Cabe mencionar aquí que `Agda` no es un de los asistentes de prueba mencionados.

Geuvers y Niqui [2002] muestran una construcción de los números reales en `Coq`. La construcción se hace de manera clásica: se construyen los números racionales y se muestra que son un campo ordenado (constructivo) y luego los números reales constructivos se introducen como una sucesión de Cauchy sobre los números racionales. Finalmente, se muestra que los axiomas son equivalentes a el conjunto de axiomas para los números reales constructivos introducidos por [Bridges 1999].

Kaliszyk y O'Connor [2009] señalan que en `Coq` existen dos bibliotecas que formalización los números reales. Una de ellas es la biblioteca estándar de `Coq` que da un tratamiento axiomático a los números reales. La otra biblioteca, llamada `CoRN`, define los números reales constructivamente. Estas bibliotecas pueden usarse simultáneamente ya que sus estructuras reales son isomorfas.

1.3. Programas y códigos fuentes

La información sobre cómo obtener el programa `Agda` y más detalles referentes a éste se pueden encontrar en la *Wiki* de `Agda`¹, y todo lo necesario para el programa `Apia` se puede encontrar en un repositorio en GitHub.² La formalización y los ejemplos presentados en esta tesis y otros más se encuentran disponibles también en un repositorio en GitHub.³

¹<http://wiki.portal.chalmers.se/agda/pmwiki.php>.

²<https://github.com/asr/apia>.

³<https://github.com/jechev28/real-numbers-formalisation>.

Capítulo 2

Conceptos básicos de Agda

En los últimos años ha crecido la atención hacia el uso del ordenador por parte de los matemáticos, en donde éstos pueden encontrar datos experimentales, patrones, formular conjeturas, validar teoremas, entre otras actividades. Se mencionan dos casos de demostraciones haciendo uso del computador: una es la demostración del teorema de los cuatro colores [Appel y Haken 1989], verificada en el asistente de pruebas **Coq**, la otra es la demostración presentada por Hales [2005] de la conjetura de Kepler, verificada combinando los asistentes de prueba **Isabelle** y **HOL Light**. En este capítulo se realiza una breve introducción al asistente de pruebas **Agda**, donde se explican sus características utilizadas en esta tesis. Para entrar en más detalles y mirar lo relacionado con este programa se puede consultar [Bove y Dybjer 2009], [Norell 2009], [Sicard-Ramírez 2015, cap. 2] y la página web de **Agda**.¹ El lector que este familiarizado con **Agda** puede omitir este capítulo.

Agda es un lenguaje de programación funcional con soporte para tipos dependientes, es decir tipos de datos que dependen de términos, y también con soporte para definiciones inductivas. **Agda** también es un asistente de pruebas interactivo, donde se puede realizar la escritura y verificación de la demostración de teoremas. **Agda** está basado en la teoría de tipo intuicionista, un sistema fundacional para el desarrollo de la matemática intuicionista [Martin-Löf 1984]. **Agda** es de código abierto y entre sus mayores contribuidores se encuentra la Universidad Tecnológica de Chalmers y la Universidad de Goteburgo. Como ayuda para escribir programas en **Agda** se utiliza la interfaz interactiva para el editor **Emacs** [Stallman et al. 2015], la cual permite y

¹<http://wiki.portal.chalmers.se/agda/pmwiki.php>.

2. Conceptos básicos de Agda

facilita crear objetivos, definir tipos inductivos, definir funciones, refinar los términos en cada prueba, hacer chequeo de tipos, entre otras actividades.

Este capítulo consta de las siguientes secciones: En la sección 2.1 se definen términos y tipos para la teoría intuicionista. En la sección 2.2 se definen los tipos inductivos que se introducen en Agda con la declaración `data`. En la sección 2.3 se ofrece una explicación sobre tipos dependientes, tipos que dependen de otros tipos. En la sección 2.4 se definen argumentos implícitos. En la sección 2.5 se definen funciones recursivas por coincidencia de patrones sobre términos inductivos. En la sección 2.6 se presenta una breve explicación de las funciones recursivas. En la sección 2.7 se introduce la clausura `where` que se puede usar para hacer una definición local. En la sección 2.8 se utiliza la abstracción `with` que permite hacer coincidencia de patrones en el resultado de un cálculo intermedio. En la sección 2.9 se muestra como Agda permite agregar términos con la palabra reservada `postulate`. En la sección 2.10 se introduce la currificación como una técnica donde se reescriben las funciones de más de un argumento a funciones de un argumento. En la sección 2.11 se definen las propiedades de la igualdad. En la sección 2.12 se muestra el empleo del razonamiento ecuacional para verificar pruebas en Agda usando un estilo algebraico.

2.1. Términos y tipos

Desde que Agda está basado en la teoría de tipo intuicionista de Martin-Löf [1984], Agda emplea términos y tipos. El símbolo dos puntos ‘:’ en Agda denota que un término es de un tipo determinado. Por ejemplo `b : B` denota que el término `b` es de tipo `B`. Además, si el tipo `B` representa una proposición entonces el término `b` representa una demostración de esa proposición.

2.2. Tipos inductivos

Los tipos inductivos se introducen en Agda con la declaración `data`. La regla de formación dice como se construye un cierto tipo a partir de otros tipos. Las reglas de introducción dicen como se puede construir términos a partir de otros términos [Martin-Löf 1984]. Las reglas de introducción se implementan nombrando el constructor de datos y su tipo.

2.2. Tipos inductivos

Ejemplo 2.1. Para esta tesis se define el conjunto de los números naturales, nombrado \mathbb{N} , por:

```
data  $\mathbb{N}$  : Set where  
  zero :  $\mathbb{N}$   
  succ :  $\mathbb{N} \rightarrow \mathbb{N}$ 
```

En la definición anterior se introduce un nuevo tipo llamado \mathbb{N} con dos constructores; **zero**, un constructor nulo y **succ**, un constructor unario recursivo.

Otras definiciones que se introducen en **Agda** con la declaración **data**, que serán utilizadas en el capítulo 3; son la conjunción y el cuantificador existencial.

Ejemplo 2.2 (Definición de la conjunción). Una función básica en la lógica es la conjunción, nombrada $_ \wedge _$, la cual es verdadera cuando sus términos también lo son.

```
data  $\_ \wedge \_$  (A B : Set) : Set where  
   $\_ , \_$  : A  $\rightarrow$  B  $\rightarrow$  A  $\wedge$  B
```

El constructor $_ , _$ establece que una prueba de la conjunción consta de dos componentes, el primer componente es un término de tipo **A** que es una prueba de **A** y el segundo componente es un término de tipo **B** que es una prueba de **B**.

Observación. En **Agda**, si el nombre de una declaración contiene un guión bajo ‘ $_$ ’, esta declaración puede ser usada como un operador cuyos argumentos van en los respectivos guiones bajos. De esta manera, el tipo $_ \wedge _$ puede ser utilizado como un operador infijo $a \wedge b$ en vez de $_ \wedge _ a b$. Cuando una declaración está escrita usando la notación infija se le llama un operador [Bird 1998, pág. 13].

Ejemplo 2.3 (Definición del cuantificador existencial). El cuantificador existencial, nombrado \exists_r , indica que existe al menos un término que satisface una determinada función proposicional.

```
data  $\exists_r$  (P :  $\mathbb{R} \rightarrow$  Set) : Set where  
  exist : (x :  $\mathbb{R}$ )  $\rightarrow$  P x  $\rightarrow$   $\exists_r$  P
```

Una prueba del existencial $\exists_r P$, donde **P** es una función proposicional de tipo $\mathbb{R} \rightarrow$ **Set**, está establecida por el constructor **exist** el cual recibe un término x de tipo \mathbb{R} , llamado testigo, y una prueba de que el testigo satisface la función proposicional **P**. En la sección 3.1 se introduce a \mathbb{R} como el tipo del sistema de los números reales.

2.3. Tipos dependientes

Los tipos dependientes son tipos que dependen de elementos de otros tipos [Bove y Dybjer 2009].

Ejemplo 2.4. Se definen los vectores \mathbf{R}^n , donde \mathbf{R} denota los números reales y n denota la longitud del vector.

$$\mathbf{R}^n = \underbrace{\mathbf{R} \times \mathbf{R} \times \dots \times \mathbf{R}}_n.$$

En este caso los vectores \mathbf{R}^n dependen del valor de n .

Observación. Note que se emplea ‘ \mathbf{R} ’ para referirse el conjunto de los números reales y ‘ \mathbb{R} ’ para referirse al tipo de los números reales formalizados.

Ejemplo 2.5. A continuación se define un vector V , donde este vector es una lista de ciertos tipos de elementos de tipo A y de longitud n . El parámetro A es constante y no cambia en este caso, mientras que n puede variar para cada lista. La familia de vectores de tipo dependiente está dada por:

```
data V (A : Set) :  $\mathbb{N}$   $\rightarrow$  Set where
  v0 : V A zero
  vs : (n :  $\mathbb{N}$ )  $\rightarrow$  A  $\rightarrow$  V A n  $\rightarrow$  V A (succ n)
```

En la definición anterior V es de tipo $\mathbf{Set} \rightarrow \mathbb{N} \rightarrow \mathbf{Set}$ y el tipo de $V A$ es $\mathbb{N} \rightarrow \mathbf{Set}$.

Ejemplo 2.6. Una familia de vectores Booleanos en Agda está dada por:

```
data V-Bool :  $\mathbb{N}$   $\rightarrow$  Set where
  v0 : V-Bool zero
  vs : (n :  $\mathbb{N}$ )  $\rightarrow$  Bool  $\rightarrow$  V-Bool n  $\rightarrow$  V-Bool (succ n)
```

Se observa que el parámetro $A : \mathbf{Set}$ de la definición de la familia de vectores V ya no aparece y para este caso particular se usó el tipo \mathbf{Bool} el cual está definido de la forma usual.

2.4. Argumentos implícitos

Agda tiene un mecanismo que permite declarar argumentos implícitos encerrándolos entre llaves `{,}` en lugar de paréntesis `(,)` Norell [2009].

Observación. En Agda los tipos creados por el usuario son de tipo `Set` que es una palabra reservada. Además, el nombre de `'Set'` proviene de la terminología empleada por Martin-Löf.

Ejemplo 2.7. Se define la función identidad, nombrada `id`, para el tipo `A`.

```
id : {A : Set} → A → A
id A x = x
```

Se puede observar que en la función `id` no se tiene que suministrar el argumento al tipo. Este argumento está implícito tanto cuando la función se aplica como cuando se define.

2.5. Tipos de datos y coincidencia de patrones

En Agda, como en otros lenguajes de programación funcional, se puede definir funciones recursivas por coincidencia de patrones sobre los términos que pertenecen a tipos inductivos. Por ejemplo, las funciones sobre `Bool` pueden ser definidas por coincidencia de patrones en su argumento.

Ejemplo 2.8. La función negación, nombrada `not`, está definida para los términos de tipo `Bool` por:

```
not : Bool → Bool
not true  = false
not false = true
```

La definición de funciones en Agda emplea los constructores del tipo de dato, es decir, para la función `not` hay dos casos posibles, i) `not true` y ii) `not false`, los cuales deben ser definidos.

Agda comprueba la definición de las funciones y produce un mensaje de error si falta algún constructor.

2. Conceptos básicos de Agda

Ejemplo 2.9. Se define la función `badNot` por:

```
badNot : Bool → Bool
badNot true = false
```

Agda genera el siguiente mensaje informando que falta un caso:

```
Incomplete pattern matching for badNot. Missing cases:
  badNot false when checking the definition of badNot
```

Ejemplo 2.10. La negación implementada para esta tesis, nombrada `¬_`, está definida por:

```
¬_ : Set → Set
¬ A = A → ⊥
```

La constante `⊥`, llamada *bottom*, está definida por:

```
data ⊥ : Set where
```

Se observa que `⊥` no tiene constructores, es un tipo para el cual es imposible producir un término.

Una definición importante que se plantea a partir del tipo `⊥` es la eliminación por *bottom*, nombrada `⊥-elim`, está definida por:

```
⊥-elim : {A : Set} → ⊥ → A
⊥-elim ()
```

Dado un término de tipo `⊥` se obtiene un término de tipo `A`.

2.6. Funciones recursivas

Una función es recursiva cuando se define en términos de si misma.

Ejemplo 2.11 (Adición de números naturales). La adición de números naturales, nombrada `_+_`, se define por coincidencia de patrones sobre su primer argumento y tiene un llamado recursivo.

```
1 infixl 6 _+_
2 _+_ : ℕ → ℕ → ℕ
3 zero   + n = n
4 (succ m) + n = succ (m + n)
```

2.7. Clausulas **where**

La primera línea denota que el operador binario `_+_` asocia por la izquierda con precedencia 6.

Observación. La asociatividad y precedencia de una función en **Agda** se denota con las palabras **infix**, **infixl** o **infixr**. A estas palabras se les asigna un número que entre mayor sea mayor es la precedencia de la función.

2.7. Clausulas **where**

Una clausula **where** se puede usar para introducir una definición local. Las definiciones locales pueden ser una abreviatura o una definición recursiva.

2.8. Abstracción **with**

La palabra reservada **with** permite hacer coincidencia de patrones en el resultado de un cálculo intermedio, agregando un argumento adicional al lado izquierdo de la igualdad en la definición de una función.

2.9. Adición de axiomas

Agda permite agregar términos de un tipo determinado sin ser necesario definirlos. La palabra reservada para este propósito es **postulate**. A continuación se postula el tipo del sistema de los números reales \mathbb{R} y dos de sus términos r_0 y r_1 , que representan al cero y al uno respectivamente.

Ejemplo 2.12. Se muestra a continuación el tipo del sistema de números reales y dos de sus términos.

```
postulate
ℝ       : Set
r0 r1 : ℝ
```

2.10. Funciones currificadas

La currificación es una técnica donde se reescriben las funciones de más de un argumento a funciones de un sólo argumento. Bird [1998, pág. 11] plantea:

2. Conceptos básicos de Agda

“A useful device for reducing the number of parentheses in an expression is the idea of replacing a structured argument by a sequence of simpler ones.”

La función $*_1 : \mathbf{R} \rightarrow \mathbf{R} \rightarrow \mathbf{R}$ está currificada y es isomorfa con la función $*_2 : (\mathbf{R} \times \mathbf{R}) \rightarrow \mathbf{R}$. La función $*_1$ recibe un número real y retorna otra función, esta otra función recibe un número real como argumento y retorna un número real. Las funciones currificadas son de orden superior, debido a que éstas pueden recibir funciones como parámetros y/o retornar funciones como resultados. Desde la teoría de conjuntos, la currificación es la correspondencia entre el conjunto $A^{B \times C}$ de funciones de $B \times C$ a A y el conjunto $(A^C)^B$ de funciones de B al conjunto de funciones de C a A . En teoría de categorías se demuestra un isomorfismo entre los morfismos $(B \times C) \rightarrow A$ y los morfismos $B \rightarrow A^C$ (véase, por ejemplo, Awodey [2010, cap. 6]).

Ejemplo 2.13. Reducir el número de paréntesis de una expresión en una secuencia de argumentos más simples.

$$\begin{aligned} f_1 &: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ f_1(m, n) &= m + n, \end{aligned}$$

$$\begin{aligned} f_2 &: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\ f_2 \ m \ n &= m + n. \end{aligned}$$

Para cada número natural m la función $f_2 \ m$ adiciona m a un número natural. En particular; $f_2 \ 1$ es la función sucesor que incrementa su argumento en 1; y $f_2 \ 0$ es la función identidad sobre los números naturales. La función f_2 está currificada y es isomorfa con la función f_1 .

2.11. Tipo identidad

Se define la igualdad proposicional (llamada igualdad lógica), además la regla de introducción y de sustitución para esta igualdad. La regla de introducción es la propiedad reflexiva (refl) y la regla de eliminación (subst) es la propiedad de sustitución de términos iguales en proposiciones [Sicard-Ramírez 2015, pág. 28]. Estas reglas están dadas por:

$$\frac{}{(x = x)} \text{ (refl)}$$

2.12. Razonamiento ecuacional

$$\frac{x = y \quad A(x)}{A(y)} \text{ (subst)}$$

La igualdad implementada para el tipo \mathbb{R} , nombrada `_≡_`, usada en esta tesis se define por:

```
data _≡_ : ℝ → ℝ → Set where  
  refl  : (x : ℝ) → x ≡ x
```

La igualdad `_≡_` recibe dos términos en \mathbb{R} y retorna la prueba que los dos términos son iguales vía el constructor `refl` (véase secciones 2.1 y 2.2). Usando la regla de introducción `refl` se puede elaborar una demostración de que un término es igual así mismo.

La propiedad de sustitución para el tipo \mathbb{R} , nombrada `subst`, se define como:

```
subst : (P : ℝ → Set) → {x y : ℝ} → x ≡ y → P x → P y  
subst P (refl x) Px = Px
```

La regla de eliminación `subst` indica que si dos términos x e y de tipo \mathbb{R} son iguales y se tiene una demostración que el término x satisface la función proposicional P , de tipo $\mathbb{R} \rightarrow \mathbf{Set}$, también se tiene una demostración que el término y satisface la función proposicional P .

De otra parte, la negación de la igualdad, nombrada `≠`, está dada por la definición:

```
_≠_ : ℝ → ℝ → Set  
x ≠ y = ¬ x ≡ y
```

2.12. Razonamiento ecuacional

El razonamiento ecuacional [Mu, Ko y Jansson 2009] es empleado en *Agda* para verificar pruebas usando un estilo algebraico, donde cada paso es justificado al frente enunciando las propiedades empleadas.

Se define el siguiente conjunto de combinadores con el objeto de ser utilizados en el razonamiento ecuacional:

2. Conceptos básicos de Agda

```
infixr 2 _≡(_)_  
infix 3 _■
```

```
_≡(_)_ : ∀ x {y z} → x ≡ y → y ≡ z → x ≡ z  
_≡( x≡y ) y≡z = ≡-trans x≡y y≡z
```

```
_■ : ∀ x → x ≡ x  
_■ x = refl x
```

El combinador `_■` recibe un término a y produce una prueba $a \equiv a$ usando reflexividad. El combinador `_≡(_)_` toma tres argumentos: a sobre la izquierda, una prueba que a es igual a a' en los paréntesis angulares $(,)$ y una prueba $a' \equiv a''$ en el lado derecho. Estos combinadores producen una prueba que a es igual a a'' usando la transitividad de la igualdad `_≡_`.

Usando estos combinadores se puede establecer un razonamiento en cadena definido por:

```
a1      ≡(reason1)  
...  
a(n-1) ≡(reason(n-1))  
an ■
```

donde `reasoni : ai ≡ a(i+1)`.

Observación. Los combinadores arriba presentados pueden ser definidos para cualquier relación binaria que sea reflexiva y transitiva (véase apéndice [A](#)).

Capítulo 3

Una formalización de una axiomática para el sistema de los números reales

Se podría hacer una presentación rigurosa del sistema de los números reales a partir de algunos de los principios básicos de la teoría de conjuntos empezando por el sistema de los números naturales, luego el sistema de los números enteros, hasta llegar al sistema de los números racionales e irracionales y finalmente presentar el sistema de los números reales. Una presentación del sistema de los números reales empleando el enfoque anterior se encuentra por ejemplo en Landau [1951, cap. 4] o en Enderton [1977, cap. 5].

Otra forma de presentar el sistema de los números reales es como un modelo de una teoría axiomática [Chang y Keisler 1990]. Este modelo consta de un dominio, constantes, funciones y relaciones. Se puede entonces obtener un cuerpo ordenado, completo y único a partir de una axiomática.¹ La presentación del sistema de los números reales en esta tesis será axiomática.

En los textos de matemáticas que realizan un desarrollo axiomático del sistema de los números reales usualmente se presentan los axiomas clasificados en tres grupos: axiomas de campo, axiomas de orden y un axioma de completitud (o axioma del supremo). Esta tesis empleará la clasificación anterior, en particular, la tesis seguirá la presentación realizada en [Apostol 1974].

En la sección 3.1, se formalizan en Agda las funciones básicas y los axiomas

¹Los números reales son el único cuerpo ordenado y completo.

3. Una formalización de una axiomática para el sistema de los números reales

de campo del sistema de los números reales. En la sección 3.2, se formaliza los axiomas relacionados con la relación de orden. En la sección 3.3, se formaliza el axioma de completitud, además de las definiciones de cota superior, conjunto acotado y supremo de un conjunto. En la sección 3.4 se presentan algunas observaciones a las formalizaciones anteriores desde el punto de vista de la matemática intuicionista.

3.1. Formalización de los axiomas de campo

En el sistema de los números reales, denotado \mathbf{R} , se supone la existencia de dos operaciones binarias, adición (+) y multiplicación (\cdot) que satisfacen los siguientes axiomas:

- Propiedad conmutativa

Para todo $a, b \in \mathbf{R}$: $a + b = b + a$ y $a \cdot b = b \cdot a$.

- Propiedad asociativa

Para todo $a, b, c \in \mathbf{R}$: $(a + b) + c = a + (b + c)$ y $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

- Existencia del neutro aditivo

Existe un elemento 0 de \mathbf{R} tal que para todo $a \in \mathbf{R}$: $a + 0 = a$.

- Existencia del neutro multiplicativo

Existe un elemento 1 de \mathbf{R} tal que para todo $a \in \mathbf{R}$: $a \cdot 1 = a$.

- No trivialidad

$$1 \neq 0.$$

- Existencia del inverso aditivo

Para todo $a \in \mathbf{R}$ existe un elemento de \mathbf{R} denotado $-a$ llamado negativo de a .

- Existencia del inverso multiplicativo

Para todo $a \in \mathbf{R}$, diferente de 0, existe un elemento de \mathbf{R} denotado a^{-1} llamado recíproco de a .

- Inverso aditivo

Para todo $a \in \mathbf{R}$ se cumple: $a + (-a) = 0$.

3.1. Formalización de los axiomas de campo

- Inverso multiplicativo

Para todo $a \in \mathbf{R}$, diferente de 0, se cumple: $a \cdot a^{-1} = 1$.

- Propiedad distributiva

Para todo $a, b, c \in \mathbf{R}$: $a \cdot (b + c) = a \cdot b + a \cdot c$.

Observación. En el texto de Royden y Fitzpatrick [2010] se le llama axioma de la no trivialidad, al hecho de que 0 es diferente de 1. En el texto de Apostol [1974] este axioma está implícito al denotar $a + (-a)$ por 0 y $a \cdot a^{-1}$ por 1, para todo $a \in \mathbf{R}$ ($a \neq 0$ en el segundo caso).

La representación de constantes y funciones del sistema de los números reales se hace a partir de postular el universo o dominio de discurso \mathbf{R} como el tipo de los números reales, habitado por las constantes r_0 y r_1 (véase ejemplo 2.12), las cuales representan el cero y el uno respectivamente. También se postula las funciones adición, nombrada $+_$, y multiplicación, nombrada $*_$. Adicionalmente se postula las funciones $-_$ y $^{-1}$ las cuales son la formalización del negativo y el recíproco respectivamente. Lo anterior es ilustrado en la figura 3.1.

```
postulate
   $\mathbf{R}$       : Set
   $r_0$   $r_1$  :  $\mathbf{R}$ 

   $+_$       :  $\mathbf{R} \rightarrow \mathbf{R} \rightarrow \mathbf{R}$ 
   $*_$       :  $\mathbf{R} \rightarrow \mathbf{R} \rightarrow \mathbf{R}$ 
   $-_$       :  $\mathbf{R} \rightarrow \mathbf{R}$ 
   $^{-1}$     :  $\mathbf{R} \rightarrow \mathbf{R}$ 
```

Figura 3.1: Tipos de las constantes y las funciones.

Observación. En la implementación de la función $^{-1}$ se puede excluir a r_0 en la definición del tipo de la función o en las propiedades de la función. En esta tesis r_0 fue excluido en la propiedad del inverso multiplicativo, nombrada $*-inve$ en la figura 3.2.

3. Una formalización de una axiomática para el sistema de los números reales

La propiedad conmutativa para la adición, nombrada `+-comm`, y la propiedad de la no trivialidad, nombrada `1≠0`, sirven para ilustrar los axiomas de campo formalizados en `Agda`.

postulate

```
+-comm : (x y : ℝ) → x + y ≡ y + x
1≠0    : r₁ ≠ r₀
```

Observación. Empleando la palabra reservada `forall`, la cual puede también escribirse con el símbolo \forall , y la capacidad de inferencia de tipos de `Agda`, el axioma `+-comm` podría reescribirse como

```
+-comm : ∀ (x y) → x + y ≡ y + x
```

Como la primera presentación hace explícito el tipo de las variables, esta será la presentación empleada en esta tesis.

La implementación de los axiomas de campo se encuentra en la figura [3.2](#).

3.2. Formalización de los axiomas de orden

Se introducen los axiomas de orden con el fin de establecer una relación de orden dentro del sistema de los números reales. Para la relación binaria $>$ (mayor que) se introducen las propiedades de orden como un conjunto de axiomas que se describen a continuación:

- Propiedad transitiva

Para todo $a, b, c \in \mathbf{R}$: si $a > b$ y $b > c$ entonces $a > c$.

- Adición por la izquierda

Para todo $a, b, c \in \mathbf{R}$, si $a > b$ entonces $c + a > c + b$.

- Producto mayor que cero

Para todo $a, b \in \mathbf{R}$, si $a > 0$ y $b > 0$ entonces $a \cdot b > 0$.

- Propiedad de tricotomía

Si $a, b \in \mathbf{R}$ entonces una y sólo una de las siguientes afirmaciones es verdadera:

$$a > b,$$

$$a = b \text{ o}$$

$$a < b.$$

3.2. Formalización de los axiomas de orden

```
-- Addition axioms.
postulate
+-comm : (x y : ℝ) → x + y ≡ y + x
+-asso : (x y z : ℝ) → x + y + z ≡ x + (y + z)
+-neut : (x : ℝ) → x + r₀ ≡ x
+-inve : (x : ℝ) → x + (- x) ≡ r₀

-- Multiplication axioms.
postulate
*-comm : (x y : ℝ) → x * y ≡ y * x
*-asso : (x y z : ℝ) → x * y * z ≡ x * (y * z)
*-neut : (x : ℝ) → x * r₁ ≡ x
*-inve : (x : ℝ) → x ≠ r₀ → x * (x-1) ≡ r₁

-- Distributivity axiom.
postulate *-dist : (x y z : ℝ) → x * (y + z) ≡ x * y + x * z

-- The nontriviality assumption.
postulate 1≠0 : r₁ ≠ r₀
```

Figura 3.2: Formalización de los axiomas de campo.

Al formalizar en **Agda** los axiomas de orden se introduce primero el tipo de la relación `_>_`:

```
postulate _>_ : ℝ → ℝ → Set
```

La relación binaria `_>_` crea un nuevo tipo en **Set**. Las relaciones binarias `_<_`, `_≥_` y `_≤_` se definen a partir de la relación `_>_`:

3. Una formalización de una axiomática para el sistema de los números reales

```

-- Less than relation.
_<_ : ℝ → ℝ → Set
y < x = x > y

-- Greater than or equal relation.
_≥_ : ℝ → ℝ → Set
x ≥ y = (x > y) ∨ (x ≡ y)

-- Less than or equal relation.
_≤_ : ℝ → ℝ → Set
x ≤ y = (x < y) ∨ (x ≡ y)

```

Uno de los axiomas de orden formalizados en **Agda** es la propiedad de tricotomía, nombrada **trichotomy**, la cual se define para elementos de tipo \mathbb{R} de la siguiente forma:

```

postulate
  trichotomy : (x y : ℝ) → ((x > y) ∧ ¬ (x ≡ y) ∧ ¬ (x < y)) ∨
    (¬ (x > y) ∧ (x ≡ y) ∧ ¬ (x < y)) ∨
    (¬ (x > y) ∧ ¬ (x ≡ y) ∧ (x < y))

```

La propiedad **trichotomy** representa que para cualquier $x, y \in \mathbf{R}$ se satisface uno y sólo uno de los siguientes tres casos: i) $x > y$, $\neg(x \equiv y)$ y $\neg(x < y)$ son verdaderos; ii) $\neg(x > y)$, $x \equiv y$ y $\neg(x < y)$ son verdaderos; iii) $\neg(x > y)$, $\neg(x \equiv y)$ y $(x < y)$ son verdaderos.

La formalización en **Agda** del axioma de tricotomía fue un proceso donde se aprendió de los errores, debido a que se sortearon varios intentos de formalización hasta llegar a uno definitivo. Inicialmente se formalizó la propiedad de tricotomía implementando la disyunción inclusiva definida en el apéndice **B**. Pero esto no representó lo fundamental de la propiedad de tricotomía, donde sólo se cumple uno de los tres casos para cualquier par de números reales $x, y : x > y, x \equiv y$ o $x < y$. Luego se procedió implementando la disyunción exclusiva (también llamada *xor*), definida mediante tres axiomas: propiedad conmutativa, propiedad asociativa y la propiedad donde cada elemento es inverso de sí mismo [Graham 2005]. La definición por medio de axiomas no fue adecuada para esta tesis, debido a que no tiene constructores y esto imposibilita la demostración de propiedades necesarias que se desprenden del axioma de tricotomía.

Luego se formalizó el axioma utilizando la disyunción exclusiva definida por

3.3. Formalización del axioma de completitud

```
data _⊕_ (A B : Set) : Set where  
  inj : (A ∨ B) → ¬ (A ∧ B) → A ⊕ B
```

Esta definición no fue apropiada para representar el axioma de tricotomía debido a que no es posible definir una disyunción exclusiva ternaria en términos de una disyunción exclusiva binaria [Pelletier y Hartline 2008]; es decir, en el caso binario se produce un valor de verdadero si uno de los argumentos es verdadero pero no ambos, y se puede demostrar que no se puede definir la disyunción ternaria exclusiva por agrupación de disyunciones binarias exclusivas.

Es posible formalizar el axioma de tricotomía a partir de la disyunción ternaria exclusiva, la cual produce un valor de verdadero si un argumento es verdadero pero no los otros dos, definida por:

```
⊕3 : Set → Set → Set → Set  
⊕3 A B C = (A ∨ B ∨ C) ∧ ¬ (A ∧ B ∧ C)
```

No se usó la disyunción ternaria exclusiva porque su definición no está en lógica de primer orden, lo cual es una condición necesaria para utilizar *Apia* (capítulo 4). La explicación de por qué esta definición no es de primer orden se presenta en el ejemplo 5.3.

Finalmente, la versión del axioma de tricotomía formalizada en esta tesis representa la forma estándar de esta propiedad. Se presentan en el ejemplo C.4 una serie de propiedades que garantizan que el axioma de tricotomía formalizado en esta tesis es apropiado.

Observación. Mientras se estaba trabajando con la disyunción exclusiva `_⊕_` se encontró un error en el programa *Apia*. Este error se reportó en el *bug-tracker* de *Apia*.²

La implementación de los axiomas de orden se encuentra en la figura 3.3.

3.3. Formalización del axioma de completitud

Hasta ahora se ha axiomatizado que el sistema de los números reales es un campo ordenado. Esta característica no lo distingue de otros campos ordenados, como los números racionales, lo que lleva a enunciar el axioma

²Véase <https://github.com/asr/apia/issues/98>.

3. Una formalización de una axiomática para el sistema de los números reales

```

postulate
>-trans  : {x y z : ℝ} → x > y → y > z → x > z
>-+-left : {x y z : ℝ} → x > y → z + x > z + y
>-∧-*    : {x y : ℝ} → (x > r₀) ∧ (y > r₀) → x * y > r₀

trichotomy : (x y : ℝ) → ((x > y) ∧ ¬ (x ≡ y) ∧ ¬ (x < y)) ∨
                        (¬ (x > y) ∧ (x ≡ y) ∧ ¬ (x < y)) ∨
                        (¬ (x > y) ∧ ¬ (x ≡ y) ∧ (x < y))

```

Figura 3.3: Formalización de los axiomas de orden.

de completitud. Antes de presentar esta propiedad, se define el concepto de conjunto acotado.

Se dice que un conjunto E de números reales está acotado superiormente cuando existe un número real b tal que $x \leq b$, para todo $x \in E$. El número b es llamado cota superior de E .

- Axioma de completitud: si E es un conjunto no vacío de números reales que está acotado superiormente, entonces existe para E una menor cota superior llamada supremo de E .

Para la formalización del axioma de completitud, inicialmente se formalizan en **Agda** las definiciones de: cota superior, conjunto acotado y supremo de un conjunto, el cual es el elemento mínimo del conjunto formado por las cotas superiores. Estas formalizaciones fueron adaptadas de la librería estándar de **Coq**.

La formalización de cota superior, nombrada **UpperBound**, está dada por:

```

UpperBound : (ℝ → Set) → ℝ → Set
UpperBound E ub = (x : ℝ) → E x → x ≤ ub

```

El predicado E , de tipo $\mathbb{R} \rightarrow \mathbf{Set}$, representa un subconjunto de los números reales. El término ub , de tipo \mathbb{R} , es una cota superior para E si se cumple $x \leq ub$, para cualquier x de tipo \mathbb{R} .

La formalización de conjunto acotado, nombrada **Bound**, está dada por:

```

Bound : (ℝ → Set) → Set
Bound E = ∃ r (λ ub → UpperBound E ub)

```

3.4. Una axiomática del sistema de los números reales en la matemática intuicionista

Si se tiene el predicado E , de tipo $\mathbb{R} \rightarrow \mathbf{Set}$, que representa un subconjunto de \mathbb{R} , entonces éste es acotado si existe un término sup , de tipo \mathbb{R} , que satisface `UpperBound`. (La definición de \exists_r se encuentra en el ejemplo 2.3).

La menor cota superior de un conjunto dado, nombrada sup , debe cumplir la propiedad `Lub`:

```
Lub : (ℝ → Set) → ℝ → Set
Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
    UpperBound E ub → sup ≤ ub)
```

El predicado E , de tipo $\mathbb{R} \rightarrow \mathbf{Set}$, que representa un subconjunto de \mathbb{R} , tiene por supremo al término sup , de tipo \mathbb{R} , si éste satisface `UpperBound` y además, para cualquier término ub , de tipo \mathbb{R} , que cumpla `UpperBound` se debe cumplir $\mathit{sup} \leq \mathit{ub}$. (La definición de `_&_` se encuentra en el ejemplo 2.2).

Igualmente adaptado de la librería estándar de `Coq`, se formaliza el axioma de completitud en `Agda` ilustrado en la figura 3.4.

```
postulate
completeness : (E : ℝ → Set) →
    Bound E → ∃r (λ x → E x) → ∃r (λ sup → Lub E sup)
```

Figura 3.4: Formalización del axioma de completitud.

Si se tiene un predicado E , de tipo $\mathbb{R} \rightarrow \mathbf{Set}$, el cual representa un subconjunto de los números reales, además éste es acotado `Bound E` y E es diferente del vacío $\exists_r (\lambda x \rightarrow E x)$, entonces E tiene supremo $\exists_r (\lambda \mathit{sup} \rightarrow \mathit{Lub} E \mathit{sup})$.

Observación. El axioma de completitud no está expresado en la lógica clásica de primer orden, debido a que el predicado E está precedido por un cuantificador universal, es decir, se está cuantificando sobre un predicado y no sobre una variable.

3.4. Una axiomática del sistema de los números reales en la matemática intuicionista

La axiomática presentada en las secciones anteriores es una axiomática del sistema de los números reales para la matemática clásica. Una de

3. Una formalización de una axiomática para el sistema de los números reales

las características de la matemática intuicionista es no aceptar la ley del tercero excluido. Una presentación axiomática de los números reales en la matemática intuicionista requiere i) adicionar una axioma correspondiente a la propiedad Arquimediana y ii) reemplazar el axioma de tricotomía. Estos planteamientos se desarrollan a continuación.

La propiedad Arquimediana establece que para todo $x \in \mathbf{R}$ existe un número natural n tal que $n > x$. Cuando un campo ordenado, como el sistema de los números reales, cumple la propiedad Arquimediana, se está garantizando que los elementos de dicho campo no sean infinitesimales o infinitos.

Una demostración por contradicción de la propiedad Arquimediana es la siguiente [Rosenlicht 1968, pág. 26]:

Demostración. Sea $x \in \mathbf{R}$ un número que no satisface la propiedad Arquimediana, es decir $n \leq x$ para todo $n \in \mathbb{N}$. Por el axioma de completitud, el conjunto \mathbb{N} tiene supremo, el cual será denotado por a . Sin embargo, para cualquier $n \in \mathbb{N}$, $n + 1$ también es un número natural, por lo que $n + 1 \leq a$ y por lo tanto $n \leq a - 1$, lo que implica que $a - 1$ es una cota superior para el conjunto \mathbb{N} . Por otro lado, puesto que $a - 1 < a$, el número a no es el supremo de \mathbb{N} y se obtiene una contradicción. \square

En la matemática intuicionista para demostrar la existencia de un objeto que satisface cierta propiedad, debe resultar posible construir dicho objeto [Bridges 1999]. Esto contrasta con la idea que establece que la existencia de un objeto puede ser demostrada razonando por contradicción. En el desarrollo axiomático que se hace del sistema de los números reales en la matemática intuicionista, se incluye un axioma adicional correspondiente a la propiedad Arquimediana [Bridges 1999, pág. 103].

Al formalizar el sistema de los números naturales y el sistema de los números reales en **Agda**, resulta útil tener una función que permita trabajar con estos dos conjuntos, que reciba elementos de tipo \mathbb{N} y retorne elementos de tipo \mathbb{R} . La función que hace esto, nombrada $\mathbb{N}2\mathbb{R}$, está definida empleando coincidencia de patrones:

```
 $\mathbb{N}2\mathbb{R} : \mathbb{N} \rightarrow \mathbb{R}$   
 $\mathbb{N}2\mathbb{R} \text{ zero} = r_0$   
 $\mathbb{N}2\mathbb{R} (\text{succ } n) = r_1 + \mathbb{N}2\mathbb{R} n$ 
```

La representación de la propiedad Arquimediana está dada por:

3.4. Una axiomática del sistema de los números reales en la matemática intuicionista

postulate archimedean : (x : ℝ) → ∃ n (λ n → ℕℝ n > x)

Observación. El enunciado de la propiedad **archimedean** corresponde al enunciado presentado por varios autores [Ciaffaglione y Di Gianantonio 2002; Bridges 1999; Rosenlicht 1968], pero este enunciado es diferente al empleado por la librería estándar de **Coq** (véase sección 6.2).

Adicionalmente, en el desarrollo axiomático del sistema de los números reales, que se plantea en la matemática intuicionista, se sustituye el axioma de tricotomía por el axioma [Bridges 1999, pág. 102]:

$$x > y \rightarrow (x > z \vee z > y), \text{ para todo } x, y, z \in \mathbb{R}.$$

Esta sustitución se debe a que el axioma de tricotomía no es válido desde el punto de vista intuicionista [Ciaffaglione y Di Gianantonio 2002].

Para entrar en más detalle sobre el desarrollo axiomático del sistema de los números reales que se hace desde la matemática intuicionista véase [Ciaffaglione y Di Gianantonio 2002; Bridges 1999].

Capítulo 4

Demostraciones interactivas y automáticas

El campo del razonamiento automático se dedica a estudiar como usar un ordenador para ayudar a resolver problemas que requieren un razonamiento. Dentro de este campo se encuentra la demostración automática de teoremas, donde una demostración automática significa que no hay intervención en el proceso deductivo por parte del usuario, éste solo ingresa la propiedad a demostrar y espera que el computador le indique si pudo o no hacer la demostración, suponiendo que los programas y la propiedad se ingresen adecuadamente. Si el computador indica que la prueba fue demostrada no hay problema, pero si indica que no fue capaz por falta de tiempo esto no quiere decir que la propiedad sea falsa, se debe buscar otro programa u otra forma de razonar; como en el caso del ya mencionado razonamiento interactivo, donde el matemático hace parte fundamental del proceso deductivo y la computadora lo asiste.

En la sección 4.1 se presentan algunos ejemplos de demostraciones interactivas con base en la formalización realizada del sistema de los números reales. En la sección 4.2 se presentan los demostradores automáticos de teoremas y un lenguaje de entrada a éstos, este es un lenguaje al que se traslada la representación que hace **Agda** de las fórmulas de primer orden al lenguaje TPTP. En la sección 4.3 se introduce una directriz del compilador empleada para hacer las demostraciones automáticas. En la sección 4.4 se introduce el programa **Apia**, un traductor de la representación que hace **Agda** de fórmulas de primer orden. Finalmente en la sección 4.5 se presentan algunos ejemplos de demostraciones automáticas y me mencionan algunas dificultades y

4. Demostraciones interactivas y automáticas

limitaciones con éstas.

4.1. Demostraciones interactivas

Utilizando las constantes, definiciones, funciones y los axiomas de campo para el sistemas de los números reales se formaliza la demostración de algunas propiedades. La metodología para presentar los ejemplos de esta sección consiste en presentar inicialmente una prueba informal de la propiedad en referencia y después formalizar esta prueba vía una demostración interactiva.

Ejemplo 4.1. Se demuestra que

$$\text{si } x = y \text{ entonces } x + z = y + z, \text{ para todo } x, y, z \in \mathbf{R}. \quad (4.1)$$

Demostración. Se tiene como hipótesis $x = y$, entonces sumando por la derecha a ambos lados de la hipótesis el número real z se tiene $x + z = y + z$, así queda demostrado lo propuesto. \square

Ahora se verifica la anterior demostración en **Agda**. La formalización de la demostración de (4.1), nombrada `≡-+-cong-r`, será realizada usando razonamiento ecuacional (véase sección 2.12) sobre los términos de tipo \mathbb{R} .

```
1 ≡-+-cong-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
2 ≡-+-cong-r {x} {y} {z} h =
3   x + z ≡( subst (λ w → (x + z) ≡ w + z) h (refl (x + z)) )
4   y + z ■
```

En la línea 3 se utiliza la propiedad de sustitución para términos de tipo \mathbb{R} , al considerar la sustitución se plantea la hipótesis $x + z \equiv w + z$ y la hipótesis $x \equiv y$ representada por h , además la propiedad `refl` de la igualdad que permite evidenciar que los términos a sustituir son iguales. En la línea 4 se muestra que el objetivo ha sido alcanzado.

Ejemplo 4.2. Se formaliza la demostración de la ley de simplificación para la suma, es decir, para todo $x, y, z \in \mathbf{R}$:

$$\text{si } x + z = y + z \text{ entonces } x = y.$$

4.1. Demostraciones interactivas

Demostración. Utilizando los axiomas de campo y la hipótesis $x + z = y + z$ se tiene:

- 1 $x = x + 0$ por axioma del elemento neutro de la adición
- 2 $= x + (z + (-z))$ por axioma del inverso aditivo
- 3 $= (x + z) + (-z)$ por propiedad asociativa de la adición
- 4 $= (y + z) + (-z)$ por sustitución con la hipótesis
- 5 $= y + (z + (-z))$ por propiedad asociativa de la adición
- 6 $= y + 0$ por axioma del inverso aditivo
- 7 $= y$ □

La formalización de la demostración anterior en *Agda* se hace empleando los combinadores para el razonamiento ecuacional ilustrados en el ejemplo 4.1.

```

1  ≡+-cancel-r : {x y z : ℝ} → x + z ≡ y + z → x ≡ y
2  ≡+-cancel-r {x} {y} {z} h =
3    x                               ≡( ≡-sym (+-neut x) )
4    x + r₀                          ≡( subst (λ w → (x + r₀) ≡ (x + w))
5                                     (≡-sym (+-inve z))
6                                     (refl (x + r₀)) )
7    x + (z + (- z)) ≡( ≡-sym (+-asso x z (- z)) )
8    (x + z) + (- z) ≡( subst (λ w → (x + z) + (- z) ≡ w + (- z))
9                                     h
10                                    (refl (x + z + - z)) )
11   (y + z) + (- z) ≡( +-asso y z (- z) )
12   y + (z + (- z)) ≡( subst (λ w → y + (z + (- z)) ≡ y + w)
13                                     (+-inve z)
14                                     (refl (y + (z + - z)))) )
15   y + r₀          ≡( +-neut y )
16   y              ■

```

En la línea 3 se hace la prueba $x \equiv x + r_0$ recurriendo a la propiedad simétrica de la igualdad $\equiv\text{-sym}$ (definida en el apéndice A) y el neutro aditivo. En la línea 4 se hace la sustitución de r_0 por $z + (-z)$ utilizando la propiedad simétrica y el inverso aditivo. En la línea 7 se implementa la propiedad asociativa de derecha a izquierda, utilizando primero la propiedad simétrica. En la línea 9 se hace una sustitución utilizando la hipótesis $x + z \equiv y + z$. En

4. Demostraciones interactivas y automáticas

la línea 10 se vuelve a utilizar la propiedad asociativa esta vez de izquierda a derecha. En la línea 11 se sustituye $z + (-z)$ por r_0 utilizando la propiedad del inverso aditivo. Finalmente en la línea 14, implementando la propiedad del neutro aditivo se llega a $y + r_0 \equiv y$.

Ejemplo 4.3. Se demuestra que

$$x < x + 1, \text{ para todo } x \in \mathbf{R}. \quad (4.2)$$

Demostración. Se emplean la propiedad $1 > 0$ y la propiedad cancelativa de la adición, es decir,

$$\text{si } x + z > y + z \text{ entonces } x > y, \text{ para todo } x, y, z \in \mathbf{R}.$$

Utilizando los axiomas de campo y de orden se tiene:

1	$1 > 0$	por propiedad $1 > 0$
2	$1 + 0 > 0$	por axioma del neutro aditivo
3	$1 + (x + (-x)) > x + (-x)$	por axioma del inverso aditivo
4	$(1 + x) + (-x) > x + (-x)$	por propiedad asociativa de la adición
5	$1 + x > x$	por propiedad cancelativa de la adición
6	$x + 1 > x$	por propiedad conmutativa de la adición \square

Antes de la formalización en **Agda** de la demostración de 4.2, nombrada `x<x+1`, se debe demostrar primero las propiedades: $1 > 0$, la cancelativa de la adición y si $x > y$ entonces $y < x$ para todo $x, y \in \mathbf{R}$. Se puede observar una formalización en **Agda** de estas propiedades, nombradas `1>0`, `>-+-cancel-r` y `>-to-<` respectivamente, en el apéndice C, ejemplos C.1, C.2 y C.3.

Razonando igual que en la demostración de (4.2), se formaliza para `x+1 > x`, nombrada `x+1>x`, para lo cual se utiliza una serie de pasos auxiliares, llamados `helper`, donde el paso 1 será parte del último paso de la formalización, el paso 2 será parte del último y el penúltimo paso de la formalización y así sucesivamente hasta lograr el objetivo.

4.1. Demostraciones interactivas

```

1  x+1>x : (x : ℝ) → x + r1 > x
2  x+1>x x =
3    p1-helper (>+-cancel-r
4              (p2-helper (p3-helper (p4-helper 1>0))))
5
6  where
7    p1-helper : r1 + x > x → x + r1 > x
8    p1-helper h = subst2 (λ t1 t2 → t1 > t2
9                          (+-comm r1 x)
10                         (refl x)
11                         h
12
13   p2-helper : r1 + (x + (- x)) > r0 →
14             (r1 + x) + (- x) > x + (- x)
15   p2-helper h =
16     subst2 (λ t1 t2 → t1 > t2) (≡-sym (+-asso r1 x (- x)))
17     (≡-sym (+-inve x)) h
18
19   p3-helper : r1 + r0 > r0 → r1 + (x + (- x)) > r0
20   p3-helper h =
21     subst2 (λ t1 t2 → t1 > t2) (subst (λ w → r1 + r0 ≡ r1 + w)
22     (≡-sym (+-inve x)) (refl (r1 + r0))) (refl r0) h
23
24   p4-helper : r1 > r0 → r1 + r0 > r0
25   p4-helper r1>r0 = subst2 (λ t1 t2 → t1 > t2)
26     (≡-sym (+-neut r1))
27     (refl r0)
28     r1>r0

```

En la línea 8 se formaliza $x + r_1 > x$ utilizando la sustitución de doble parámetro `subst2` (definida en el apéndice A), la propiedad conmutativa de la adición y la hipótesis $r_1 + x > x$. La hipótesis se convierte en el nuevo objetivo que se demuestra implementando la propiedad `>+-cancel-r`, a su vez al aplicar esta propiedad se necesita una prueba de $(r_1 + x) + (- x) > x + (- x)$ la cual se hace en la línea 16 utilizando sustitución de doble parámetro, simetría, asociatividad de la adición e inverso aditivo. Similarmente en la línea 16 tiene como hipótesis $r_1 + (x + (- x)) > r_0$, la cual se demuestra en la línea 21 implementando sustitución de doble parámetro, simetría, inverso

4. Demostraciones interactivas y automáticas

aditivo y reflexiva, donde en la línea 21 se tiene como hipótesis $r_1 + r_0 > r_0$, la cual se demuestra en la línea 25 utilizando sustitución de doble parámetro, simetría, neutro aditivo y reflexiva. Finalmente, la hipótesis $r_1 > r_0$ en la línea 28 es la propiedad $1 > 0$ formalizada previamente. Por último, se aplica la propiedad $>-to-<$:

```
x<x+1 : (x : ℝ) → x < x + r1
x<x+1 x = >-to-< (x+1>x x)
```

Se presenta a continuación algunos ejemplos relacionados con el axioma de completitud.

Ejemplo 4.4. El conjunto vacío es un conjunto acotado debido a que todo número real b es cota superior de este conjunto. Se presenta una formalización de esta afirmación.

El conjunto vacío se representa por:

```
∅ : ℝ → Set
∅ x = x ≠ x
```

donde \emptyset es un predicado que representa al conjunto vacío, éste se refiere a la imposibilidad de construir los términos x que son diferentes a sí mismo.

La formulación de que el conjunto vacío es un conjunto acotado está dada por:

```
Bes : Bound ∅
Bes = exist r1 (λ x ∅x → ⊥-elim (∅x (refl x)))
```

Para la demostración en **Agda** se utiliza la propiedad de conjunto acotado **Bound**, la cual recibe el predicado \emptyset para el cual existe un término r_1 , de tipo \mathbb{R} , que satisface la propiedad **UpperBound**, además para cualquier término x , de tipo \mathbb{R} , que cumple $\emptyset x$ se cumple $x \leq r_1$. En el desarrollo de la demostración el constructor **exist** recibe un término de tipo \mathbb{R} , en este caso r_1 , luego para cualquier x , de tipo \mathbb{R} , que cumpla $\emptyset x$ se debe demostrar $x \leq r_1$, esto se hace utilizando \perp -**elim** donde se logra la contradicción mediante $x \equiv x$, por propiedad **refl**, y $x \neq x$ por definición de conjunto vacío.

No se puede aplicar el axioma de completitud al conjunto vacío para demostrar que éste tiene supremo, debido a que este axioma exige que el conjunto sea diferente de vacío.

4.1. Demostraciones interactivas

Ejemplo 4.5. El axioma de completitud garantiza que todo conjunto de números reales, diferente de vacío y acotado superiormente, tiene supremo.

Sea A un subconjunto de los números reales definido por:

$$A = \{x \in \mathbb{R} \mid x \leq 2\}.$$

El conjunto A es diferente de vacío y acotado superiormente por 2. Por axioma de completitud el conjunto A tiene supremo.

Se formaliza en **Agda** empleando **UpperBound**, **Bound** y el **completeness** la afirmación anterior. El conjunto A es definido por:

```
A : ℝ → Set
A x = x ≤ ℕℝ 2
```

La formalización que el número 2 es cota superior del conjunto A está dada por:

```
UBA : UpperBound A (ℕℝ 2)
UBA x (inj₁ x<2) = inj₁ x<2
UBA x (inj₂ x≡2) = inj₂ x≡2
```

La formulación que el conjunto A es acotado está dada por:

```
BA : Bound A
BA = exist (ℕℝ 2) UBA
```

Y finalmente la formalización que el conjunto A tiene supremo está dada por:

```
LA : ∃r (λ sup → Lub A sup)
LA = completeness A BA (exist (ℕℝ 2) (inj₂ (refl (ℕℝ 2))))
```

El axioma de completitud garantiza la existencia del supremo, para conjuntos no vacíos y acotados, pero no indica quien es el supremo del conjunto. Sin embargo, en algunos casos se puede conocer el supremo de un conjunto.

Ejemplo 4.6. Para el conjunto A del ejemplo 4.5 se demuestra en **Agda** que 2 es el supremo de este conjunto.

```
UBAx : (x : ℝ) → UpperBound A x → ℕℝ 2 ≤ x
UBAx x A2 = A2 (ℕℝ 2) (inj₂ (refl (ℕℝ 2)))
```

```
LbA : ∃r (λ x → Lub A (ℕℝ 2))
LbA = exist (ℕℝ 2) (UBA , UBAx)
```

4. Demostraciones interactivas y automáticas

Se define en `Agda` la potenciación como una función recursiva, nombrada `_^_` de la siguiente forma:

$$\begin{aligned} _ \wedge _ &: \mathbb{R} \rightarrow \mathbb{N} \rightarrow \mathbb{R} \\ x \wedge \text{zero} &= r_1 \\ x \wedge (\text{succ } n) &= x * (x \wedge n) \end{aligned}$$

Observación. En la definición de la función `_^_` : $\mathbb{R} \rightarrow \mathbb{N} \rightarrow \mathbb{R}$ se sigue la convención que $0^0 = 1$, véase por ejemplo, Knuth [1992] o la definición en algunos asistentes de prueba tales como `Coq`.

Se presenta a continuación algunas demostraciones empleando el principio de inducción matemática.

Ejemplo 4.7. En matemáticas se realizan demostraciones de propiedades por el principio de inducción matemática cuando se tienen conjuntos inductivos. Un conjunto A se dice inductivo si y sólo si i) $0 \in A$ y ii) si $a \in A$ entonces $a' \in A$, donde a' es el sucesor de a . Los números naturales son un conjunto inductivo que está contenido en cualquier conjunto inductivo [Enderton 1977, pág. 68].

Se demuestra por el principio de inducción matemática la propiedad $1^n = 1$, para cualquier número natural n . Luego se formaliza la demostración razonando de forma interactiva en `Agda`. Se asumen las propiedades $1^1 = 1$ y $x^{m+n} = x^m \cdot x^n$, para todo número real x y números naturales m y n . Se puede observar una demostración de estas propiedades en el repositorio.

Demostración. Primero, base de la inducción, se procede a mostrar la validez de la propiedad para cuando $n = 0$. Si se observa la definición de potenciación se confirma tal afirmación.

Segundo, paso inductivo para $n = k + 1$, por lo cual la hipótesis inductiva está dada por $1^k = 1$.

$$\begin{aligned} 1^{k+1} &= 1^k \cdot 1 && \text{por propiedad de la potenciación} \\ &= 1 \cdot 1 && \text{por hipótesis inductiva} \\ &= 1 && \text{por axioma del neutro multiplicativo} \quad \square \end{aligned}$$

Se presenta a continuación la formalización de la demostración de la propiedad $1^n = 1$ de manera interactiva en `Agda`.

4.1. Demostraciones interactivas

```

1  1^n : ( n : ℕ ) → r₁ ^ n ≡ r₁
2  1^n zero      = refl r₁
3  1^n (succ n) = p-helper
4
5  where
6  p-helper : r₁ * r₁ ^ n ≡ r₁
7  p-helper =
8    r₁ * r₁ ^ n ≡( *-comm r₁ (r₁ ^ n) )
9    r₁ ^ n * r₁ ≡( *-neut (r₁ ^ n) )
10   r₁ ^ n      ≡( 1^n n )
11   r₁          ■

```

En la línea 1 se muestra el objetivo que se quiere demostrar, $r_1 \wedge n \equiv r_1$, para todo $n : \mathbb{N}$. En la línea 2 se demuestra para el caso cuando $n = \text{zero}$ utilizando la propiedad de la igualdad `refl`. En la línea 3 se demuestra para el caso sucesor $n = \text{succ } n$ (paso inductivo) la propiedad $r_1 * r_1 \wedge n \equiv r_1$, nombrada `p-helper`, utilizando razonamiento ecuacional (véase sección 2.12).

Ejemplo 4.8. Se demuestra el valor de una sumatoria por el principio de inducción matemática.

Se demostrará que $S_n = 1 + 3 + 5 + \dots + 2n - 1 = n^2$, para todo número natural n mayor que cero.

Demostración. Primero, base de la inducción. Si $n = 1$, se tiene $1 = 1$, por lo que queda demostrado el paso base.

Segundo, paso inductivo para $n = k + 1$, para todo k natural mayor que cero se debe demostrar:

$$1 + 3 + 5 + \dots + 2k - 1 + (2(k + 1) - 1) = (k + 1)^2.$$

Inicialmente se supone verdadera la propiedad para el caso $n = k$ (hipótesis inductiva):

$$1 + 3 + 5 + \dots + 2k - 1 = k^2.$$

Luego se procede a sumar $2(k+1) - 1$ a ambos lados de la hipótesis inductiva:

$$1 + 3 + 5 + \dots + 2k - 1 + (2(k + 1) - 1) = k^2 + (2(k + 1) - 1).$$

Utilizando la propiedad distributiva por la izquierda al lado derecho de la igualdad se tiene:

$$1 + 3 + 5 + \dots + 2k - 1 + (2(k + 1) - 1) = k^2 + 2k + 2 - 1.$$

4. Demostraciones interactivas y automáticas

Finalmente, se aplica la propiedad del trinomio cuadrado perfecto al lado derecho de la igualdad:

$$1 + 3 + 5 + \dots + 2k - 1 + (2(k + 1) - 1) = (k + 1)^2. \quad \square$$

Para formalizar la demostración, inicialmente se define la sumatoria. Como es necesario incluir el **zero** en la definición por coincidencia de patrones sobre los números naturales, la sumatoria S_n se define recursivamente por:

$$\begin{aligned} S &: \mathbb{N} \rightarrow \mathbb{R} \\ S \text{ zero} &= r_0 \\ S (\text{succ } n) &= S n + (\mathbb{N}2\mathbb{R} 2) * (\mathbb{N}2\mathbb{R} n) + r_1 \end{aligned}$$

La formalización del valor de la sumatoria S se presenta a continuación.

$$\begin{aligned} S\text{-con} &: (n : \mathbb{N}) \rightarrow S n \equiv \text{sqr } (\mathbb{N}2\mathbb{R} n) \\ S\text{-con zero} &= \text{p-helper} \\ \mathbf{where} & \\ \text{p-helper} &: S \text{ zero} \equiv \text{sqr } (\mathbb{N}2\mathbb{R} \text{ zero}) \\ \text{p-helper} &= \\ &S \text{ zero} \quad \equiv \langle \text{refl } r_0 \rangle \\ &r_0 \quad \equiv \langle \equiv\text{-sym } r_0\text{-sqr} \rangle \\ &\text{sqr } (\mathbb{N}2\mathbb{R} \text{ zero}) \blacksquare \end{aligned}$$

4.2. Demostradores automáticos de teoremas y el lenguaje TPTP

```

S-con (succ n) = p-helper
where
p-helper : S (succ n) ≡ sqr (N2R (succ n))
p-helper =
  S (succ n)
    ≡( refl (S n + N2R 2 * N2R n + r1) )
  S n + N2R 2 * N2R n + r1
    ≡( subst (λ w → S n + N2R 2 * N2R n + r1 ≡
              w + N2R 2 * N2R n + r1)
            (S-con n)
            (refl (S n + N2R 2 * N2R n + r1)))
    )
  sqr (N2R n) + N2R 2 * N2R n + r1
    ≡( ≡-sym (PST1 (N2R n)) )
  sqr (N2R n + r1)
    ≡( subst (λ w → sqr (N2R n + r1) ≡ sqr w)
            (+-comm (N2R n) r1)
            (refl (sqr (N2R n + r1))))
    )
  sqr (r1 + N2R n) ■

```

Las definiciones y propiedades no mencionadas en este ejemplo se encuentran en el repositorio.

4.2. Demostradores automáticos de teoremas y el lenguaje TPTP

Los demostradores automáticos de teoremas son programas que implementan métodos y técnicas para demostrar propiedades y teoremas de diferentes teorías formales. Los demostradores automáticos de teoremas son empleados por ejemplo en la industria para el diseño y la verificación de circuitos integrados. De ahora en adelante estos programas serán llamados ‘ATPs’. Este nombre proviene de las siglas de su nombre en inglés (*Automated Theorem Provers*).

La TPTP (Thousands of Problems for Theorem Provers) es una biblioteca de problemas para sistemas que prueban teoremas automáticamente para la lógica clásica [Sutcliffe 2010]. El lenguaje TPTP hace parte de esta biblioteca.

4. Demostraciones interactivas y automáticas

Adicionalmente este lenguaje es soportado por varios de los ATPs, por lo tanto a éste se traslada la representación que hace Agda de las fórmulas de primer orden.

La sintaxis de las fórmulas escritas en lenguaje TPTP es de la forma:

```
fof(name, role, formula)
```

donde `name` identifica la fórmula dentro del conjunto de fórmulas, `formula` es una fórmula de la lógica clásica de primer orden con identidad y `role` puede ser uno de los siguientes: `axiom`, `definition`, `hypothesis` o `conjecture`. Para el lenguaje TPTP la conjetura es la fórmula a ser demostrada. Los axiomas son las propiedades que se asumen como verdaderas y son la base para demostrar las conjeturas. Las hipótesis son propiedades que se han demostrado previamente y se comportan como un axioma para los ATPs. Las definiciones se utilizan para introducir nuevos símbolos.

4.3. El ATP-pragma

En Agda, al igual que en otros lenguajes de programación, es posible escribir un tipo especial de comentario, en particular unos llamados directivas del compilador o *pragmas*. Los *pragmas* son leídos e interpretados por el compilador de Agda, de esta manera el *pragma* modifica el comportamiento de Agda.

Ejemplo 4.9. En Agda se pueden introducir los números naturales empleando la declaración `data` como fue ilustrado en el ejemplo 2.1.

```
data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ
```

Al adicionar el *pragma*

```
{-# BUILTIN NATURAL ℕ #-}
```

el compilador de Agda sabe que los números literales, es decir ‘0’, ‘1’, ‘2’, ..., deben tener su correspondencia con los constructores del tipo de datos ℕ. Por ejemplo, el número literal 0 será interpretado como `zero` y el número literal 1 será interpretado como `succ zero`.

4.4. EL programa Apia

En esta tesis se emplea un *pragma* para *Agda* llamado el ATP-pragma. Este *pragma* adiciona información relacionada con el lenguaje TPTP a los archivos generados por *Agda* (archivos con extensión `.agdai`). Estos archivos son interpretados por el programa *Apia*. El ATP-pragma es la manera de informarle a los ATPs, a través del programa *Apia*, que una determinada fórmula es un axioma, una definición, una hipótesis o una conjetura.

La sintaxis del ATP-pragma es la siguiente:

```
{-# ATP rol formula #-}
```

donde `rol` corresponde a los diferentes roles del lenguaje TPTP y `formula` es una fórmula de la lógica de predicados de primer orden definida en *Agda*.

La forma de indicar que las fórmulas *A* y *B* deberán ser consideradas como axiomas por los ATPs es:

```
{-# ATP axiom A #-}  
{-# ATP axiom B #-}
```

Se puede utilizar la sintaxis alternativa:

```
{-# ATP axioms A B #-}
```

La forma de indicar que se emplearán los ATPs para intentar realizar una demostración automática de una fórmula postulada *A* es la siguiente:

```
{-# ATP prove A #-}
```

También se le puede indicar a los ATPs que empleen como hipótesis adicionales las fórmulas h_1 y h_2 cuando intenten demostrar la fórmula *A* incluyendo estas hipótesis en el ATP-pragma:

```
{-# ATP prove A h1 h2 #-}
```

4.4. EL programa Apia

Apia es un programa que traslada la representación que realiza *Agda* de las fórmulas de primer orden al lenguaje TPTP. Además este programa es una interfaz entre *Agda* y los ATPs. *Apia* interactúa con los ATPs preguntándoles por pruebas, entregándoles hipótesis e informándoles sobre definiciones y axiomas. Como resultado de esta interacción el programa *Apia* llama a los

4. Demostraciones interactivas y automáticas

ATPs elegidos por el usuario para que demuestren propiedades automáticamente y le informa al usuario el resultado de esta llamada. Al momento de escribir esta tesis, sólo es posible demostrar automáticamente teoremas escritos en la lógica de primer orden empleando **Apia**.

El programa **Apia** soporta varios ATPs los cuales soportan el lenguaje TPTP y pueden emplearse para realizar demostraciones automáticas. En esta tesis se trabajó con dos ATPs, **Eprover** (versión E 1.9.1-001 Sungma) [Schulz 2013] y **Vampire** (versión 0.6 (revisión 903)) [Kováč y Voronkov 2013], ya que éstos fueron suficientes para demostrar las propiedades propuestas.

Para intentar demostrar automáticamente un formula se realizan los siguientes pasos: i) adicionar los ATP-pragmas necesarios al archivo con la formalización en **Agda**, ii) hacer chequeo de tipos en el archivo empleando **Agda**, iii) ejecutar **Apia** en el archivo con la formalización en **Agda** seleccionando los ATPs de preferencia y iv) verificar si los ATPs seleccionados demostraron o no la formula.

Durante el proceso de demostración automática **Apia** llama indistintamente a los ATPs seleccionados por el usuario. Si uno de los ATPs demuestra la propiedad, entonces el proceso se detiene y **Apia** informa cual ATP realizó la demostración. Si al finalizar el tiempo asignado a los ATPs ninguno de ellos pudo demostrar la propiedad, entonces **Apia** le informa al usuario esta situación. Una situación que se puede presentar es que si ninguno de los ATPs seleccionados demostró la propiedad, puede ser que otro ATP soportado por **Apia** la pueda demostrar. En algunos casos resulta necesario agregar propiedades demostradas previamente como hipótesis auxiliares a la propiedad que se desea demostrar.

Por ejemplo, para demostrar automáticamente la propiedad conmutativa de la disyunción, inicialmente se escribe

```
postulate
  A B      : Set
  v-comm  : A v B → B v A
  {-# ATP prove v-comm #-}
```

en un archivo **Agda** llamado **CommDisjunction.agda**. Este archivo se compila escribiendo lo siguiente:

```
$ agda CommDisjunction.agda
```

En la pantalla se muestra un mensaje que indica que se ha generado un

4.5. Demostraciones automáticas

archivo con extensión `.agdai` que será interpretado por `Apia`. A continuación se ejecuta `Apia` indicándole que use los ATPs `E` y `Vampire`.

```
$ apia --atp=e --atp=vampire CommDisjunction.agda
```

En la pantalla se muestra un mensaje que indica que se ha generado un archivo en lenguaje TPTP y su localización en un directorio temporal. Además brinda información acerca de la propiedad que fue demostrada y también aparece una descripción sobre el primero de los ATPs en realizar la demostración.

```
Proving the conjecture in /tmp/CommDisjunction/10-8744-comm.tptp
Vampire 0.6 (revision 903) proved the conjecture
```

El archivo TPTP es creado en el directorio `/tmp/CommDisjunction`. La propiedad `v-comm` es renombrada como `10-8744-comm`, el número `10` indica la línea donde la propiedad es postulada, `8744` es el código decimal del símbolo Unicode ‘v’ y `Vampire` fue el primero de los ATPs en demostrar la conjetura.

4.5. Demostraciones automáticas

Se presenta la forma como se realizan demostraciones automáticas de propiedades del sistema de los números reales. Además se mencionan algunas limitaciones en el uso de este tipo de demostraciones.

Puesto que la presentación del sistema de números reales fue axiomática, el primer paso para emplear los ATPs por medio del programa `Apia` es emplear el ATP-pragma para indicarle a los ATPs cuales axiomas deben usar.

```
{-# ATP axiom +-comm +-asso +-neut +-inve l≠0 #-}
{-# ATP axiom *-comm *-asso *-neut *-inve *-dist #-}
{-# ATP axiom >-trans >-+-left >-∧-* trichotomy #-}
```

También se debe emplear el ATP-pragma para señalar las definiciones empleadas en la formalización.

```
{-# ATP definition _≠_ #-}
{-# ATP definition _<_ #-}
{-# ATP definition _≥_ #-}
{-# ATP definition _≤_ #-}
```

4. Demostraciones interactivas y automáticas

Ejemplo 4.10. La propiedad `≡+-cong-r` fue demostrada de forma interactiva en el ejemplo 4.1. Para demostrar automáticamente esta propiedad, se postula ésta y se emplea el ATP-pragma para indicarle a los ATPs que la intenten demostrar.

```
postulate ≡+-cong-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
{-# ATP prove ≡+-cong-r #-}
```

De forma similar se demuestra automáticamente la ley de simplificación para la suma presentada en el ejemplo 4.2.

```
postulate ≡+-cancel-r : {x y z : ℝ} → x + z ≡ y + z → x ≡ y
{-# ATP prove ≡+-cancel-r #-}
```

Se puede observar en el ejemplo anterior la simplificación considerable en la formalización de las propiedades cuando en lugar de realizar una demostración interactiva, se realiza una demostración automática.

Ejemplo 4.11. Una posible demostración automática del valor de la sumatoria demostrado interactivamente en el ejemplo 4.8 tendría la siguiente forma:

Primero, base de la inducción `S zero`:

```
S-con : (n : ℕ) → S n ≡ sqr (ℕ2ℝ n)
S-con zero = p-helper
where
postulate p-helper : S zero ≡ sqr (ℕ2ℝ zero)
{-# ATP prove p-helper #-}
```

Segundo, paso inductivo `S-con (succ n)`:

```
S-con (succ n) = p-helper
where
postulate p-helper : S (succ n) ≡ sqr (ℕ2ℝ (succ n))
{-# ATP prove p-helper #-}
```

Se observa que la demostración es por inducción, pero se emplean los ATPs en los pasos base e inductivo. Desafortunadamente no es posible emplear este tipo de demostraciones automáticas por las razones que se exponen a continuación.

4.5. Demostraciones automáticas

La demostración automática ilustrada en el ejemplo 4.11 es diferente a las demostraciones automáticas presentadas en los ejemplos anteriores desde el punto de vista lógico. En esta demostración se emplean dos universos de discurso diferentes (el tipo \mathbb{R} de los números reales y el tipo \mathbb{N} de los números naturales). En la actualidad el programa **Apia** no soporta lógicas donde se utilizan universos de discurso diferentes.

Capítulo 5

Límites

En este capítulo se emplean los elementos presentados hasta el momento en esta tesis para ilustrar la formalización de un concepto de uso cotidiano en las matemáticas: límite de una función.

La noción intuitiva de límite indica que los valores que toma una función dentro de un intervalo se van aproximando arbitrariamente a un punto fijo, independientemente de que éste pertenezca o no al rango de la función. Para formalizar el concepto de límite de una función se procede primero a formalizar el valor absoluto de un número real, la distancia entre dos puntos y demostrar algunas de sus propiedades.

5.1. Valor absoluto

Si x es un número real entonces el valor absoluto de x es un número real no negativo, el cual se designa por $|x|$ y se define de la siguiente forma:

$$|x| = \begin{cases} x & \text{si } x \geq 0, \\ -x & \text{si } x < 0. \end{cases}$$

En **Agda** se formaliza la definición de valor absoluto de la siguiente forma:

```
abs : ℝ → ℝ
abs x with x < 0 ∨ x ≥ 0
... | inj₁ _ = - x
... | inj₂ _ = x
```

donde

5. Límites

$$x < 0 \vee x \geq 0 : (x : \mathbb{R}) \rightarrow (x < r_0) \vee (x \geq r_0)$$

La formalización de $|x|$ se hace implementando una función, nombrada **abs** de tipo $\mathbb{R} \rightarrow \mathbb{R}$, la cual si recibe un argumento x que cumple $x < 0$ entonces devuelve $-x$, de lo contrario devuelve x .

Algunas de las propiedades que cumple el valor absoluto, para todo número real x e y , son las siguientes:

$$\begin{aligned} |0| &= 0, \\ \text{si } x > 0 \text{ entonces } |x| &= x, \\ \text{si } x < 0 \text{ entonces } |x| &= -x, \\ |-x| &= |x|, \\ |x| &\geq x, \end{aligned} \tag{5.1}$$

$$-|x| \leq x, \tag{5.2}$$

$$\text{(desigualdad triangular) } |x + y| \leq |x| + |y|, \tag{5.3}$$

$$|x - y| \leq |x| + |y|.$$

Ejemplo 5.1. Se demuestra la desigualdad triangular (propiedad 5.3) para el valor absoluto.

Demostración. A partir de las propiedades (5.1) y (5.2), para $x, y \in \mathbf{R}$, se tiene

$$\begin{aligned} -|x| &\leq x \leq |x|, \\ -|y| &\leq y \leq |y|. \end{aligned}$$

Luego, sumando miembro a miembro estas desigualdades se obtiene:

$$-|x| - |y| \leq x + y \leq |x| + |y|.$$

lo que es igual a:

$$-(|x| + |y|) \leq x + y \leq |x| + |y|.$$

Finalmente por la definición de valor absoluto se tiene:

$$|x + y| \leq |x| + |y|. \quad \square$$

La formalización de la demostración en **Agda** de la desigualdad triangular, nombrada **abs-tri**, está dada por:

5.1. Valor absoluto

```

abs-tri : (x y : ℝ) → abs (x + y) ≤ abs x + abs y
abs-tri x y with x<0vx≥0 (x + y)
... | inj₁ p = helper

where
helper : - (x + y) ≤ abs x + abs y
helper = p₁-helper (≤-to-≥ (p₂-helper (-absx+-absy≤x+y x y)))

where
p₁-helper : x + y ≥ - (abs x + abs y) →
             - (x + y) ≤ abs x + abs y
p₁-helper h₁ = x≥-y→-x≤y (x + y) (abs x + abs y) h₁

p₂-helper : - (abs x) + ( - abs y) ≤ x + y →
             - (abs x + abs y) ≤ x + y
p₂-helper h₂ = subst₂ (λ t₁ t₂ → t₁ ≤ t₂)
                (≡-sym mul-x+y)
                (refl (x + y))
                h₂
... | inj₂ _ = x+y≤absx+absy x y

```

Como la formalización de valor absoluto emplea `with` (ver 2.8) sobre la propiedad $x < 0 \vee x \geq 0$, entonces se da un argumento adicional al lado izquierdo de la igualdad en la definición de valor absoluto. Por esta razón se generan dos objetivos:

```

... | inj₁ p = helper
... | inj₂ _ = x+y≤absx+absy x y

```

El primer objetivo es demostrar la propiedad nombrada `helper`, la cual está definida por:

```

helper : - (x + y) ≤ abs x + abs y

```

Esta propiedad a su vez requiere la demostración de las propiedades nombradas `p₁-helper` y `p₂-helper`. El segundo objetivo consiste en demostrar la propiedad

```

x+y≤absx+absy : (x y : ℝ) → x + y ≤ abs x + abs y

```

Una prueba de esta propiedad se encuentra en el repositorio.

5. Límites

La formalización de la demostración en **Agda** de las demás propiedades de valor absoluto mencionadas al comienzo de esta sección se encuentran en el repositorio.

En la sección 4.5 se señaló una limitación para realizar pruebas automáticas cuando se emplean varios universos en la formalización. Las pruebas relacionadas con valor absoluto no pudieron ser automatizadas debido a una razón diferente.

Apia puede trasladar al lenguaje TPTP las definiciones, desde el punto de vista lógico, de relaciones o funciones. Por ejemplo, empleando el ATP-pragma se le indica a **Apia** que traslade la relación binaria `_<_` y la función unaria `sqr`.

```
_<_ : ℝ → ℝ → Set
y < x = x > y
{-# ATP definition _<_ #-}

sqr : ℝ → ℝ
sqr x = x * x
{-# ATP definition sqr #-}
```

Agda además de soportar definiciones desde el punto de vista lógico, también soporta definiciones de funciones y relaciones con base en sus características como lenguaje de programación. Estas definiciones no pueden ser trasladadas por **Apia**.

Ejemplo 5.2. En el caso de la definición de la función valor absoluto, **Apia** no sabe como trasladar al lenguaje TPTP la funcionalidad de la palabra reservada **with**.

Ejemplo 5.3. La disyunción ternaria exclusiva fue definida por (véase sección 3.2):

```
⊕3 : Set → Set → Set → Set
⊕3 A B C = (A ∨ B ∨ C) ∧ ¬ (A ∧ B ∧ C)
```

Desde que la definición de \oplus^3 no corresponde desde el punto de vista lógico ni a la definición de una función ni a la definición de una relación, **Apia** no puede trasladar esta definición. En este caso, la definición de \oplus^3 es una definición realizada “por fuera” de la lógica de primer orden.

5.2. Distancia entre dos puntos

5.2. Distancia entre dos puntos

Si x es un número real, la distancia de x a 0 es el número $|x|$. De igual forma, si x e y son números reales, la distancia entre x e y es el número $|x - y|$.

Para todo número real x, y y z , la función distancia, nombrada d , satisface las siguientes propiedades:

$$\begin{aligned}d(x, y) &\geq 0, \\d(x, y) &= d(y, x), \\d(x, y) = 0 &\leftrightarrow x = y, \\d(x, y) &\leq d(x, z) + d(z, y).\end{aligned}$$

Se formaliza en Agda la definición de distancia entre dos puntos, nombrada `dist`, a partir de la definición de valor absoluto:

```
dist : ℝ → ℝ → ℝ
dist x y = abs (x - y)
```

Ejemplo 5.4. Demostrar que para cualesquier números reales x, y, z y w se cumple:

$$d(x + z, y + w) \leq d(x, y) + d(z, w). \quad (5.4)$$

Demostración. Se parte de la desigualdad triangular donde el primer número está dado por $x - y$ y el segundo número por $z - w$, es decir:

$$|(x - y) + (z - w)| \leq |x - y| + |z - w|.$$

Por definición de distancia entre dos puntos, el lado derecho de la desigualdad anterior es equivalente al mismo lado de la propiedad a demostrar, por lo que basta con probar la igualdad:

$$|(x - y) + (z - w)| = d(x + z, y + w).$$

Reagrupando los términos del lado izquierdo de la igualdad se obtiene:

$$|(x - y) + (z - w)| = |(x + z) - (y + w)|.$$

Finalmente por definición de distancia:

$$|(x + z) - (y + w)| = d(x + z, y + w).$$

Queda así demostrada la propiedad. □

5. Límites

La formalización de la demostración de (5.4), nombrada `dis-des`, está dada por:

```
dis-des : (x y z w : ℝ) →
  dist (x + z) (y + w) ≤ dist x y + dist z w
dis-des x y z w = p1-helper (abs-tri (x - y) (z - w))
```

A partir de la desigualdad triangular evaluada en los números $(x - y)$ y $(z - w)$ se demuestra la propiedad `p1-helper`, la cual indica:

```
p1-helper : abs ((x - y) + (z - w)) ≤ abs (x - y) + abs (z - w) →
  dist (x + z) (y + w) ≤ dist x y + dist z w
p1-helper h =
  subst₂ (λ t₁ t₂ → t₁ ≤ t₂)
    p11-helper
    (refl (abs (x - y) + abs (z - w)))
  h
```

Se observa que el lado derecho de la desigualdad `dist x y + dist z w` es equivalente a `abs (x - y) + abs (z - w)` por definición de distancia entre dos puntos, por lo que se implementa la propiedad `refl`. Luego se demuestra la propiedad nombrada `p11-helper` reagrupando términos, lo que se logra implementando razonamiento ecuacional.

```
p11-helper : abs (x - y + (z - w)) ≡ dist (x + z) (y + w)
```

La formalización completa de esta propiedad se encuentra en el repositorio.

Ejemplo 5.5. A diferencia de las propiedades del valor absoluto, las propiedades de la función distancia pudieron ser demostradas automáticamente.

```
postulate d-refl : (x y : ℝ) → (dist x y ≡ r₀) ↔ (x ≡ y)
{-# ATP prove d-refl abs-0 x>0→absx=x x<0→absx=-x #-}
```

```
postulate d-pos : (x y : ℝ) → dist x y ≥ r₀
{-# ATP prove d-pos abs-0 x>0→absx=x x<0→absx=-x #-}
```

```
postulate d-sym : (x y : ℝ) → dist x y ≡ dist y x
{-# ATP prove d-sym abs-0 x>0→absx=x x<0→absx=-x #-}
```

```
postulate d-tri : (x y z : ℝ) → dist x y ≤ dist x z + dist z y
{-# ATP prove d-tri abs-tri #-}
```

5.3. Límite de una función

postulate

```
dis-des : (x y z w : ℝ) →
  dist (x + z) (y + w) ≤ dist x y + dist z w
{-# ATP prove dis-des abs-tri #-}
```

Las hipótesis auxiliares empleadas en las demostraciones anteriores se encuentran formalizadas en el repositorio.

5.3. Límite de una función

Sea f una función definida sobre algún intervalo abierto que contiene al número a , excepto posiblemente en a misma. Se dice entonces que el límite de $f(x)$ cuando x tiende a a es L , si para todo número $\varepsilon > 0$ existe un número $\delta > 0$ tal que si la distancia de x a a es menor que δ , entonces la distancia de $f(x)$ a L es menor que ε .

La formalización de la definición del límite, nombrado `lim`, de una función `f`, de tipo $\mathbb{R} \rightarrow \mathbb{R}$, fue adaptada de la librería estándar de `Coq`.

```
lim : (ℝ → ℝ) → ℝ → ℝ → Set
lim f a L = (ε : ℝ) → ε > r_0 →
  ∃ r (λ δ → δ > r_0 → (x : ℝ) →
    dist x a < δ → dist (f x) L < ε)
```

Ejemplo 5.6. Demostración de la propiedad que indica que el límite de una función constante es la constante.

Demostración. El límite cuando x tiende a a de la función $f(x) = k$ es igual a k . Se tiene entonces que para todo $\varepsilon > 0$ existe un $\delta > 0$ tal que la distancia de x a a es menor que δ , entonces la distancia de $f(x)$ a k es menor que ε . Como la distancia de k ($f(x) = k$) a k es cero se cumple que $\varepsilon > 0$, entonces se concluye que para todo $\delta > 0$ el límite existe y es igual a k . \square

Se formaliza en `Agda` la demostración de la propiedad, nombrada `cte-lim`, que indica que el límite de una función `f` constante, de tipo $\mathbb{R} \rightarrow \mathbb{R}$, es igual a una constante `k` para todo `x` cuando este tiende a `a`, donde `a`, `L` y `k` son de tipo \mathbb{R} .

```
cte-lim : (a L k : ℝ) → (f : ℝ → ℝ) → lim (λ _ → k) a k
cte-lim a L k f ε ε>0 = exist ε p-helper
```

5. Límites

La demostración emplea el constructor `exist` el cual requiere un testigo ε y la prueba de la propiedad `p-helper`:

```
1  p-helper :  $\varepsilon > r_0 \rightarrow (x : \mathbb{R}) \rightarrow \text{dist } x \ a < \varepsilon \rightarrow \text{dist } k \ k < \varepsilon$ 
2  p-helper  $\varepsilon > 0$  x h = subst2 ( $\lambda \ t_1 \ t_2 \rightarrow t_1 < t_2$ )
3                               ( $\equiv$ -sym ( $\wedge$ -proj2 (d-refl k k))
4                               (refl k)
5                               (refl  $\varepsilon$ )
6                               (>-to-<  $\varepsilon > 0$ ))
```

En la línea 1 se muestra el objetivo que se quiere lograr, `dist k k < ε` , donde se tienen las hipótesis: $\varepsilon > 0$, $(x : \mathbb{R})$ y `dist x a < ε` . En la línea 2, se realiza la demostración de la propiedad `p-helper` donde se utiliza: la sustitución de doble parámetro `subst2`, la propiedad de la igualdad `\equiv -sym`, la proyección `\wedge -proj2`, la propiedad de la igualdad `refl` y la propiedad `>-to-<`.

Como fue mencionado con anterioridad, **Apia** sólo soporta la lógica de primer orden, es decir, **Apia** no soporta lógicas de orden superior. La definición del límite de una función no está expresada en la lógica de primer orden, debido a que ésta es una función de orden superior (su primer argumento es una función). Por esta razón, la demostraciones acerca de las propiedades de los límites no se pueden realizar de forma automática empleando la metodología presentada en esta tesis.

Otras demostraciones formalizadas en **Agda** de las propiedades de los límites se encuentran en el repositorio.

Capítulo 6

Conclusiones

En esta tesis se desarrolló una formalización de las demostraciones de propiedades matemáticas en el sistema de los números reales, demostrando de forma interactiva con el programa **Agda** y demostrando automáticamente empleando el programa **Apia** y algunos demostradores automáticos de teoremas. A partir del trabajo realizado se obtuvieron los siguientes resultados:

6.1. Resultados

- Se presentó de forma axiomática el sistema de los números reales.
- Se utilizó el asistente de pruebas **Agda** para realizar demostraciones interactivas de propiedades matemáticas, por medio de la interfaz para el editor **Emacs**.
- Se emplearon los demostradores automáticos de teoremas **E** y **Vampire**, así como el programa **Apia**, que es un traductor de la representación que hace **Agda** de formulas de primer orden al lenguaje **TPTP**, para demostrar automáticamente diferentes propiedades. Si una propiedad no es demostrada empleando la metodología presentada en esta tesis esto se puede asociar a uno de los siguientes casos: a) el tiempo de espera máximo que se ha programado en el computador para que los ATPs intenten realizar la demostración es corto y en consecuencia no alcanzaron a realizar la demostración, b) de los ATPs elegidos ninguno logra hacer la demostración, en este caso se puede usar un ATP diferente, c) no sabe si la propiedad se puede demostrar automáticamente.

6. Conclusiones

- Se presentaron ejemplos sobre la manera de realizar demostraciones interactivas y automáticas, ambas maneras de demostrar pueden ser el inicio para otros proyectos de investigación, pero también una motivación para profundizar más al respecto sobre este tipo de demostraciones.
- Como experiencia en la formalización de algunas propiedades del sistema de los números reales en **Agda** se destaca que el proceso fue arduo y extenso, esto debido a que cada vez que se formalizaba una definición o una demostración se necesitaban una serie de propiedades y definiciones previas. Por ejemplo, cuando se formalizó el concepto del límite de una función se tuvo que definir: la función valor absoluto, la función de distancia entre dos puntos y la función que retorna el mínimo entre dos valores. Además, fue necesario demostrar una serie de propiedades que dichas funciones cumplen y otras tantas más.
- Se reconocieron y experimentaron algunas de las limitaciones para realizar demostraciones automáticas. En el contexto de esta tesis las principales limitaciones están relacionadas con el programa **Apia**.
- Terminar esta tesis no fue fácil, pero valió la pena trasegar por este camino ya que se obtuvieron grandes aprendizajes.

6.2. Trabajo futuro

Todas las preguntas e ideas que puedan surgir y conceptos que puedan ir más allá de esta tesis pueden ser tomados como elementos para trabajo futuro. Se propone lo siguiente:

- **Coq** formaliza una versión más fuerte de la propiedad Arquimediana presentada en esta tesis. Sería interesante conocer si se puede debilitar la versión de esta propiedad empleada por **Coq**.
- Una experiencia enriquecedora consistiría en ampliar la formalización realizada en esta tesis, por ejemplo los conceptos de continuidad de funciones, derivadas e integrales, tal como se hace por ejemplo en **Coq**.

Bibliografía

- Apostol, T. M. (1974). *Mathematical Analysis*. 2.^a ed. Addison-Wesley (vid. págs. 2, 15, 17).
- Appel, K. I. y Haken, W. (1989). Every Planar Map is Four Colorable. Vol. 98. American Mathematical Society (vid. págs. 1, 5).
- Awodey, S. (2010). *Category Theory*. 2.^a ed. Oxford Logic Guides. Oxford University Press (vid. pág. 12).
- Bancerek, G., Byliński, C., Grabowski, A., Kornilowicz, A., Matuszewski, R., Naumowicz, A., Pak, K. y Urban, J. (2015). Mizar: State-of-the-art and Beyond. En: *Intelligent Computer Mathematics: International Conference (CICM 2015)*. Ed. por Kerber, M., Carette, J., Kaliszyk, C., Rabe, F. y Sorge, V. Vol. 9150. *Lecture Notes in Computer Science*. Springer, págs. 261-279 (vid. pág. 4).
- Bird, R. (1998). *Introduction to Functional Programming using Haskell*. 2.^a ed. Prentice Hall (vid. págs. 7, 11).
- Boldo, S., Lelay, C. y Melquiond, G. (2016). Formalization of Real Analysis: A Survey of Proof Assistants and Libraries. *Mathematical Structures in Computer Science* 26.7 (vid. pág. 3).
- Bove, A. y Dybjer, P. (2009). Dependent Types at Work. En: *LerNet ALFA Summer School 2008*. Ed. por Bove, A., Soares Barbosa, L., Pardo, A. y Sousa Pinto, J. Vol. 5520. *Lecture Notes in Computer Science*. Springer, págs. 57-99 (vid. págs. 5, 8).
- Bridges, D. S. (1999). *Constructive Mathematics: A Foundation for Computable Analysis*. *Theoretical Computer Science* 219.1, págs. 95-109 (vid. págs. 4, 24, 25).
- Callaghan, P., Luo, Z., McKinna, J. y Pollack, R., eds. (2002). *Types for Proofs and Programs (TYPES 2000)*. Vol. 2277. *Lecture Notes in Computer Science*. Springer.

Bibliografía

- Chang, C. C. y Keisler, J. H. (1990). Model Theory. 3.^a ed. Vol. 73. Studies in Logic and the Foundations of Mathematics. North Holland (vid. pág. 15).
- Ciaffaglione, A. y Di Gianantonio, P. (2002). A Tour with Constructive Real Numbers. En: Types for Proofs and Programs (TYPES 2000). Ed. por Callaghan, P., Luo, Z., McKinna, J. y Pollack, R. Vol. 2277. Lecture Notes in Computer Science. Springer, págs. 41-52 (vid. pág. 25).
- Enderton, H. B. (1977). Elements of Set Theory. Academic Press (vid. págs. 15, 34).
- Geuvers, H. (2009). Proof Assistants: History, Ideas and Future. *Sadhana* 34.1, págs. 3-25 (vid. pág. 1).
- Geuvers, H. y Niqui, M. (2002). Constructive reals in Coq: Axioms and Categoricity. En: Types for Proofs and Programs (TYPES 2000). Ed. por Callaghan, P., Luo, Z., McKinna, J. y Pollack, R. Vol. 2277. Lecture Notes in Computer Science. Springer, págs. 79-95 (vid. pág. 4).
- Graham, S. (2005). Deduction with XOR Constraints in Security API Modelling. En: Automated Deduction (CADE-20). Ed. por Nieuwenhuis, R. Vol. 3362. Lecture Notes in Computer Science. Springer, págs. 328-333 (vid. pág. 20).
- Hales, T. C. (2005). A Proof of the Kepler Conjecture. *Annals of Mathematics* 162.3, págs. 1065-1185 (vid. págs. 1, 5).
- Harrison, J. (2015). HOL Light Tutorial. Intel Corporation (vid. pág. 4).
- Kaliszyk, C. y O'Connor, R. (2009). Computing with Classical Real Numbers. *Journal of Formalized Reasoning* 2.1, págs. 27-39 (vid. pág. 4).
- Knuth, D. E. (1992). Two Notes on Notation. *American Mathematical Monthly* 99.5, págs. 403-422 (vid. pág. 34).
- Kováč, L. y Voronkov, A. (2013). First-Order Theorem Proving and VAMPIRE. En: Computer Aided Verification (CAV 2013). Ed. por Sharygina, N. y Veith, H. Vol. 8044. Lecture Notes in Computer Science. Springer, págs. 1-35 (vid. pág. 40).
- Landau, E. (1951). Foundations of Analysis. Chelsea Publishing Co (vid. pág. 15).
- Martin-Löf, P. (1984). Intuitionistic Type Theory. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980. Bibliopolis (vid. págs. 5, 6).
- Mu, S.-C., Ko, H.-S. y Jansson, P. (2009). Algebra of Programming in Agda: Dependent Types for Relational Program Derivation. *Journal of Functional Programming* 19.5, págs. 545-579 (vid. pág. 13).

Bibliografía

- Nipkow, T., Paulson, L. C. y Wenzel, M. (2016). A Proof Assistant for Higher-Order Logic. Springer-Verlag (vid. pág. 4).
- Norell, U. (2007). Towards a Practical Programming Language Based on Dependent Type Theory. Tesis doct. Department of Computer Science and Engineering. Chalmers University of Technology y University of Gothenburg (vid. pág. 1).
- (2009). Dependently Typed Programming in Agda. En: Advanced Functional Programming (AFP 2008). Ed. por Koopman, P., Plasmeijer, R. y Swierstra, D. Vol. 5832. Lecture Notes in Computer Science. Springer, págs. 230-266 (vid. págs. 5, 9).
- Pelletier, F. J. y Hartline, A. (2008). Ternary Exclusive *Or*. Logic Journal of the IGPL 16.1, págs. 75-83 (vid. pág. 21).
- Rosenlicht, M. (1968). Introduction To Analysis. Dover Publications, Inc. (vid. págs. 2, 24, 25).
- Royden, H. L. y Fitzpatrick, P. M. (2010). Real Analysis. Prentice Hall (vid. págs. 2, 17).
- Schulz, S. (2013). System Description: E 1.8. En: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19). Ed. por McMillan, K., Middeldorp, A. y Voronkov, A. Vol. 8312. Lecture Notes in Computer Science. Springer, págs. 735-743 (vid. pág. 40).
- Sicard-Ramírez, A. (2015). Reasoning about Functional Programs by Combining Interactive and Automatic Proofs. Tesis doct. PEDECIBA Informática. Universidad de la República. Uruguay (vid. págs. 2, 5, 12).
- Slind, K. y Norrish, M. (2008). A Brief Overview of HOL4. En: Theorem Proving in Higher Order Logics (TPHOLs 2008). Ed. por Mohamed, O. A., Muñoz, C. y Tahar, S. Vol. 5170. Lecture Notes in Computer Science. Springer, págs. 28-32 (vid. pág. 4).
- Stallman, R. et al. (2015). GNU Emacs Manual. Free Software Foundation (vid. pág. 5).
- Sutcliffe, G. (2010). The TPTP World—Infrastructure for Automated Reasoning. En: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16). Ed. por Clarke, E. y Voronkov, A. Vol. 6355. Lecture Notes in Computer Science. Springer, págs. 1-12 (vid. pág. 37).
- The Coq Development Team (2016). Reference Manual: Version 8.5 (vid. pág. 4).

Apéndice A

Propiedades de la igualdad proposicional

En este apéndice se presentan algunas propiedades de la igualdad proposicional definida en la sección 2.11 empleadas en esta tesis.

Simetría de la igualdad.

```
≡-sym : {x y : ℝ} → x ≡ y → y ≡ x
≡-sym (refl x) = refl x
```

Transitividad de la igualdad.

```
≡-trans : {x y z : ℝ} → x ≡ y → y ≡ z → x ≡ z
≡-trans (refl z) (refl z) = refl z
```

Doble sustitución para la igualdad.

```
subst₂ : (P : ℝ → ℝ → Set) → {x₁ x₂ y₁ y₂ : ℝ} →
  x₁ ≡ x₂ → y₁ ≡ y₂ → P x₁ y₁ → P x₂ y₂
subst₂ P (refl x) (refl y) h = h
```

Apéndice B

Definiciones

Se presentan algunas definiciones empleadas en la formalización del sistema de números reales.

```
-- The inclusive disjunction data type.
```

```
data _v_ (A B : Set) : Set where
```

```
  inj1 : A → A v B
```

```
  inj2 : B → A v B
```

```
-- Substraction.
```

```
_ - _ : ℝ → ℝ → ℝ
```

```
x - y = x + (- y)
```


Apéndice C

Demostraciones interactivas auxiliares

En este apéndice se presentan algunas demostraciones usadas en esta tesis pero no presentadas en el cuerpo principal del trabajo.

Ejemplo C.1. Se demuestra la propiedad `l>0`:

```
l>0 : r1 > r0
l>0 = p1-helper (>-sqr (l≠0))
  where
    p1-helper : sqr r1 > r0 → r1 > r0
    p1-helper h = subst2 (λ t1 t2 → t1 > t2) r1-sqr (refl r0) h
```

Ejemplo C.2. Se demuestra la propiedad `>-+-cancel-r`:

```
>-+-cancel-r : {x y z : ℝ} → x + z > y + z → x > y
>-+-cancel-r h = p1-helper (p2-helper (p3-helper (p4-helper h)))

  where
    p1-helper : {x y : ℝ} → x + r0 > y + r0 → x > y
    p1-helper {x} {y} h =
      subst2 (λ t1 t2 → t1 > t2) (+-neut x) (+-neut y) h

    p2-helper : {x y z : ℝ} → x + (z + (- z)) > y + (z + (- z)) →
      x + r0 > y + r0
    p2-helper {x} {y} {z} h = subst2 (λ t1 t2 → x + t1 > y + t2)
      (+-inve z) (+-inve z) h
```

C. Demostraciones interactivas auxiliares

```
p3-helper : {x y z : ℝ} → (x + z) + (- z) > (y + z) + (- z) →
              x + (z + (- z)) > y + (z + (- z))
p3-helper {x} {y} {z} h = p-helper h
```

where

```
p-helper : {x y x1 x2 : ℝ} → (x + x1) + x2 > (y + x1) + x2 →
              x + (x1 + x2) > y + (x1 + x2)
```

```
p-helper {x} {y} {x1} {x2} h =
  subst2 (λ t1 t2 → t1 > t2)
    (+-asso x x1 x2)
    (+-asso y x1 x2) h
```

```
p4-helper : {x y z : ℝ} → x + z > y + z →
              x + z + (- z) > y + z + (- z)
p4-helper {x} {y} {z} h = >-+-right h
```

Observación. Las propiedades `>-sqr`, `r1-sqr` y `>-+-right` se encuentran formalizadas en el repositorio archivo `Properties.agda`.

Ejemplo C.3. Propiedad `>-to-<`:

```
>-to-< : {x y : ℝ} → x > y → y < x
>-to-< {x} {y} h = h
```

Ejemplo C.4. A partir del axioma de tricotomía definido en la sección 3.2, se prueba la propiedad asimétrica, nombrada `>-asym`, de la siguiente forma:

```
>-asym : {x y : ℝ} → x > y → x ≠ y
>-asym {x} {y} x>y x<y = case prf1 prf2 (trichotomy x y)
where
prf1 : (x > y) ∧ (¬ x ≡ y) ∧ (¬ x < y) → ⊥
prf1 (_ , _ , p) = p x<y

prf2 : (¬ x > y ∧ x ≡ y ∧ ¬ x < y) ∨
        (¬ x > y ∧ ¬ x ≡ y ∧ x < y) → ⊥
prf2 (inj1 (p , _)) = p x>y
prf2 (inj2 (p , _)) = p x>y
```

La propiedad `>-asym` indica que si existe una relación de orden $>$ entre dos elementos distintos de tipo \mathbb{R} , es decir $x > y$, y si $y > x$ entonces se produce una contradicción.

Además, a partir del axioma de tricotomía se demuestra que para todo $x, y \in \mathbb{R}$ si $x \not\equiv y$ y $x \not< y$ entonces se tiene $x > y$, es decir:

```
≠∧<→ : {x y : ℝ} → (x ≠ y) ∧ (x < y) → x > y
≠∧<→ {x} {y} (x≠y , x<y) = case prf1 prf2 (trichotomy x y)
```

where

```
prf1 : x > y ∧ ¬ x ≡ y ∧ ¬ x < y → x > y
prf1 h = ∧-proj1 h
```

```
prf2 : (¬ x > y ∧ x ≡ y ∧ ¬ x < y)
      ∨ (¬ x > y ∧ ¬ x ≡ y ∧ x < y) → x > y
prf2 h = case prf21 prf22 h
```

where

```
prf21 : ¬ x > y ∧ x ≡ y ∧ ¬ x < y → x > y
prf21 h1 = ⊥-elim (x≠y (∧-proj1 (p-helper h1)))
```

where

```
p-helper : ¬ x > y ∧ (x ≡ y ∧ ¬ x < y) →
           x ≡ y ∧ (¬ x < y ∧ ¬ x > y)
p-helper h2 = ∧-assoc2 (∧-comm h2)
```

```
prf22 : (¬ x > y) ∧ (¬ x ≡ y) ∧ (x < y) → x > y
prf22 h3 = ⊥-elim (x<y (∧-proj2 (∧-assoc1 h3)))
```

Igualmente se prueba que cada uno de los disyuntos del axioma de tricotomía implica la negación de los otros dos, es decir:

```
>→<∧≠ : (x y : ℝ) → x > y → x < y ∧ x ≠ y
>→<∧≠ x y x>y = (λ x<y → >-asym x>y x<y) , (λ x=y → ≡→< x=y x>y)
```

```
≡→<∧> : (x y : ℝ) → x ≡ y → x < y ∧ x > y
≡→<∧> x y x=y = (λ x<y → <-≠ x<y x=y) , (λ x>y → >→≠ x>y x=y)
```

```
<->∧≠ : (x y : ℝ) → x < y → x > y ∧ x ≠ y
<->∧≠ x y x<y = (λ x>y → <-asym x<y x>y) , (λ x=y → <-≠ x<y x=y)
```

La demostración de las propiedades $<-asym$, $\equiv \rightarrow <$, $> \rightarrow \neq$ y $<- \neq$ se encuentran en el repositorio en el archivo `Properties.agda`.

