Una formalización del sistema de los números reales

Jorge O. Acevedo Acosta Jose L. Echeverri Jurado

Asesor: Andrés Sicard Ramírez
Departamento de Ciencias Matemáticas
Escuela de Ciencias
Universidad EAFIT
8 de Junio 2017

Formalización de una teoría

En el campo del razonamiento automático se habla de formalización de una teoría cuando sus teoremas son demostrados automáticamente o cuando sus demostraciones son verificadas por un asistente de pruebas.

Resumen

Se presenta la formalización de una axiomática del sistema de los números reales, tal como lo presenta Apostol [1974], en el asistente de pruebas Agda. Así mismo, 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.

Algunas formalizaciones

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 los enfoques para la formalización del sistema de los números reales se pueden clasificar en tres grupos:

- formalizaciones que caracterizan los números reales como un conjunto dado con operaciones y propiedades,
- formalizaciones donde los números reales se construyen como completación de los racionales y
- formalizaciones basadas en los números hiperreales.

Estado del Arte

Asistentes de pruebas

Los asistentes de pruebas empleados incluyen Coq [The Coq Development Team 2016], HOL4 [Slind y Norrish 2008], HOL Light [Harrison 2015] e Isabelle [Nipkow, Paulson y Wenzel 2016], entre otros.

Agda: Un lenguaje de programación funcional y un asistente de pruebas

Preliminares

- Para realizar las demostraciones de propiedades se emplearon algunos conceptos de Agda [Norell 2007], tales como: tipos inductivos, tipos dependientes, funciones recursivas, adición de axiomas y razonamiento ecuacional.
- Se implementó las conectivas lógicas incluyendo la constante $\bot$ (llamada *bottom*).

Agda: Un lenguaje de programación funcional y un asistente de pruebas

Preliminares

- Para realizar las demostraciones de propiedades se emplearon algunos conceptos de Agda [Norell 2007], tales como: tipos inductivos, tipos dependientes, funciones recursivas, adición de axiomas y razonamiento ecuacional.
- Se implementó las conectivas lógicas incluyendo la constante $\bot$ (llamada *bottom*).

Razonamiento ecuacional

Para demostrar propiedades siguiendo un estilo algebraico, se define la igualdad implementada para el tipo $\mathbb{R}$.

Tipo del sistema de los números reales

```
postulate
  ℝ : Set
```

Igualdad

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

Combinadores

```
_≡⟨_⟩_ : ∀ x {y z} → x ≡ y → y ≡ z → x ≡ z
_ ≡⟨ x≡y ⟩ y≡z = ≡-trans x≡y y≡z


_∎ : ∀ x → x ≡ x
_∎ x = refl x
```

Razonamiento ecuacional

Para demostrar propiedades siguiendo un estilo algebraico, se define la igualdad implementada para el tipo $\mathbb{R}$.

Tipo del sistema de los números reales

```
postulate
  ℝ : Set
```

Igualdad

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

Combinadores

```
_≡⟨_⟩_ : ∀ x {y z} → x ≡ y → y ≡ z → x ≡ z
_ ≡⟨ x≡y ⟩ y≡z = ≡-trans x≡y y≡z


_∎ : ∀ x → x ≡ x
_∎ x = refl x
```

Razonamiento ecuacional

Para demostrar propiedades siguiendo un estilo algebraico, se define la igualdad implementada para el tipo $\mathbb{R}$.

Tipo del sistema de los números reales

```
postulate
  ℝ : Set
```

Igualdad

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

Combinadores

```
_≡⟨_⟩_ : ∀ x {y z} → x ≡ y → y ≡ z → x ≡ z
_ ≡⟨ x≡y ⟩ y≡z = ≡-trans x≡y y≡z


_∎ : ∀ x → x ≡ x
_∎ x = refl x
```

Razonamiento ecuacional

Para demostrar propiedades siguiendo un estilo algebraico, se define la igualdad implementada para el tipo $\mathbb{R}$.

Tipo del sistema de los números reales

```
postulate
  ℝ : Set
```

Igualdad

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

Combinadores

```
_≡⟨_⟩_ : ∀ x {y z} → x ≡ y → y ≡ z → x ≡ z
_ ≡⟨ x≡y ⟩ y≡z = ≡-trans x≡y y≡z

_∎ : ∀ x → x ≡ x
_∎ x = refl x
```

Formalización de los números reales

Axiomas

Se realiza la formalización de los axiomas clasificados en tres grupos:

- Axiomas de campo. Se supone la existencia de dos operaciones binarias, adición (_+_) y multiplicación (_*_) que satisfacen ciertas propiedades.
- Axiomas de orden. Para la relación binaria > (mayor que) se introducen las propiedades de orden como un conjunto de axiomas.
- Axioma de completitud. Para la formalización del axioma de completitud, se formaliza en Agda las definiciones de: cota superior, conjunto acotado y supremo de un conjunto (adaptadas de la librería estándar de Coq).

Formalización de los números reales

Axiomas

Se realiza la formalización de los axiomas clasificados en tres grupos:

- Axiomas de campo. Se supone la existencia de dos operaciones binarias, adición (_+_) y multiplicación (_*_) que satisfacen ciertas propiedades.
- Axiomas de orden. Para la relación binaria > (mayor que) se introducen las propiedades de orden como un conjunto de axiomas.
- Axioma de completitud. Para la formalización del axioma de completitud, se formaliza en Agda las definiciones de: cota superior, conjunto acotado y supremo de un conjunto (adaptadas de la librería estándar de Coq).

Formalización de los números reales

Axiomas

Se realiza la formalización de los axiomas clasificados en tres grupos:

- Axiomas de campo. Se supone la existencia de dos operaciones binarias, adición (_+_) y multiplicación (_*_) que satisfacen ciertas propiedades.
- Axiomas de orden. Para la relación binaria > (mayor que) se introducen las propiedades de orden como un conjunto de axiomas.
- Axioma de completitud. Para la formalización del axioma de completitud, se formaliza en Agda las definiciones de: cota superior, conjunto acotado y supremo de un conjunto (adaptadas de la librería estándar de Coq).

Formalización de los números reales

Axiomas

Se realiza la formalización de los axiomas clasificados en tres grupos:

- Axiomas de campo. Se supone la existencia de dos operaciones binarias, adición (_+_) y multiplicación (_*_) que satisfacen ciertas propiedades.
- Axiomas de orden. Para la relación binaria > (mayor que) se introducen las propiedades de orden como un conjunto de axiomas.
- Axioma de completitud. Para la formalización del axioma de completitud, se formaliza en Agda las definiciones de: cota superior, conjunto acotado y supremo de un conjunto (adaptadas de la librería estándar de Coq).

Axioma de completitud

Definiciones

- ```
  UpperBound : (ℝ → Set) → ℝ → Set
  UpperBound E ub = (x : ℝ) → E x → x ≤ ub
  ```
- ```
  Bound : (ℝ → Set) → Set
  Bound E = ∃ᵣ (λ ub → UpperBound E ub)
  ```
- ```
  Lub : (ℝ → Set) → ℝ → Set
  Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
                                    UpperBound E ub → sup ≤ ub)
  ```

Axioma

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

Axioma de completitud

Definiciones

- ```
  UpperBound : (ℝ → Set) → ℝ → Set
  UpperBound E ub = (x : ℝ) → E x → x ≤ ub
  ```
- ```
  Bound : (ℝ → Set) → Set
  Bound E = ∃ᵣ (λ ub → UpperBound E ub)
  ```
- ```
  Lub : (ℝ → Set) → ℝ → Set
  Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
                                    UpperBound E ub → sup ≤ ub)
  ```

Axioma

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

Axioma de completitud

Definiciones

- ```
  UpperBound : (ℝ → Set) → ℝ → Set
  UpperBound E ub = (x : ℝ) → E x → x ≤ ub
  ```
- ```
  Bound : (ℝ → Set) → Set
  Bound E = ∃ᵣ (λ ub → UpperBound E ub)
  ```
- ```
  Lub : (ℝ → Set) → ℝ → Set
  Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
                                    UpperBound E ub → sup ≤ ub)
  ```

Axioma

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

Axioma de completitud

Definiciones

- ```
  UpperBound : (ℝ → Set) → ℝ → Set
  UpperBound E ub = (x : ℝ) → E x → x ≤ ub
  ```
- ```
  Bound : (ℝ → Set) → Set
  Bound E = ∃ᵣ (λ ub → UpperBound E ub)
  ```
- ```
  Lub : (ℝ → Set) → ℝ → Set
  Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
                                    UpperBound E ub → sup ≤ ub)
  ```

Axioma

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

Axioma de completitud

Definiciones

- ```
  UpperBound : (ℝ → Set) → ℝ → Set
  UpperBound E ub = (x : ℝ) → E x → x ≤ ub
  ```
- ```
  Bound : (ℝ → Set) → Set
  Bound E = ∃ᵣ (λ ub → UpperBound E ub)
  ```
- ```
  Lub : (ℝ → Set) → ℝ → Set
  Lub E sup = (UpperBound E sup) ∧ ((ub : ℝ) →
                                    UpperBound E ub → sup ≤ ub)
  ```

Axioma

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

Demostración interactiva

Realizar pruebas interactivas consiste en utilizar un asistente de pruebas en la que el usuario interactúa con la computadora indicándole el paso a paso.

Preliminares

- Agda es entre otras, un asistente de pruebas interactivo, donde se puede realizar la escritura y verificación de la demostración de teoremas.
- 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 formaliza a continuación una propiedad

Demostración interactiva

Realizar pruebas interactivas consiste en utilizar un asistente de pruebas en la que el usuario interactúa con la computadora indicándole el paso a paso.

Preliminares

- Agda es entre otras, un asistente de pruebas interactivo, donde se puede realizar la escritura y verificación de la demostración de teoremas.
- 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 formaliza a continuación una propiedad

Demostración interactiva

Realizar pruebas interactivas consiste en utilizar un asistente de pruebas en la que el usuario interactúa con la computadora indicándole el paso a paso.

Preliminares

- Agda es entre otras, un asistente de pruebas interactivo, donde se puede realizar la escritura y verificación de la demostración de teoremas.
- 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 formaliza a continuación una propiedad

Demostración interactiva

Realizar pruebas interactivas consiste en utilizar un asistente de pruebas en la que el usuario interactúa con la computadora indicándole el paso a paso.

Preliminares

- Agda es entre otras, un asistente de pruebas interactivo, donde se puede realizar la escritura y verificación de la demostración de teoremas.
- 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 formaliza a continuación una propiedad

Demostración interactiva

Utilizando las constantes, definiciones, funciones y los axiomas de campo para el sistemas de los números reales, se formaliza la demostración de la siguiente propiedad.

Ejemplo

Se demuestra que

$$\text{si } x = y \text{ entonces } x + z = y + z, \text{ para todo } x, y, z \in \mathbb{R}. \tag{1}$$

La formalización de la demostración, se realiza usando razonamiento ecuacional.

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y = { }


El objetivo a demostrar  x + z ≡ y + z
```

Demostración interactiva

Utilizando las constantes, definiciones, funciones y los axiomas de campo para el sistemas de los números reales, se formaliza la demostración de la siguiente propiedad.

Ejemplo

Se demuestra que

$$\text{si } x = y \text{ entonces } x + z = y + z, \text{ para todo } x, y, z \in \mathbb{R}. \qquad (1)$$

La formalización de la demostración, se realiza usando razonamiento ecuacional.

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y = { }


El objetivo a demostrar  x + z ≡ y + z
```

Demostración interactiva

Utilizando las constantes, definiciones, funciones y los axiomas de campo para el sistemas de los números reales, se formaliza la demostración de la siguiente propiedad.

Ejemplo

Se demuestra que

$$\text{si } x = y \text{ entonces } x + z = y + z, \text{ para todo } x, y, z \in \mathbb{R}. \tag{1}$$

La formalización de la demostración, se realiza usando razonamiento ecuacional.

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y = { }

El objetivo a demostrar  x + z ≡ y + z
```

Demostración interactiva

Ejemplo

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ { } ⟩
  y + z ∎

congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst { } { } { } ⟩
  y + z ∎
```

Demostración interactiva

Ejemplo

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ { } ⟩
  y + z ∎

congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst { } { } { } ⟩
  y + z ∎
```

Demostración interactiva

Ejemplo

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst { } x=y { } ⟩
  y + z ∎
```

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst (λ w → (x + z ≡ w + z)) x=y (refl (x + z)) ⟩
  y + z ∎
```

Demostración interactiva

Ejemplo

```
congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst { } x=y { } ⟩
  y + z ∎

congruence-r : {x y z : ℝ} → x ≡ y → x + z ≡ y + z
congruence-r {x} {y} {z} x=y =
  x + z ≡⟨ subst (λ w → (x + z ≡ w + z)) x=y (refl (x + z)) ⟩
  y + z ∎
```

Demostración automática

La demostración automática está referida a la utilización de programas de razonamiento que implementan métodos y técnicas para demostrar propiedades y teoremas de diferentes teorías formales.

Preliminares

- La TPTP es una biblioteca de problemas para sistemas que prueban teoremas automáticamente para la lógica clásica, además el lenguaje TPTP, hace parte de esta biblioteca y es soportado por varios de los ATPs.
- Apia [Sicard-Ramírez 2015, cap. 6] es un programa que traslada la representación que realiza Agda de las formulas de primer orden al lenguaje TPTP, así mismo es una interfaz entre Agda y los ATPs.

Demostración automática

La demostración automática está referida a la utilización de programas de razonamiento que implementan métodos y técnicas para demostrar propiedades y teoremas de diferentes teorías formales.

Preliminares

- La TPTP es una biblioteca de problemas para sistemas que prueban teoremas automáticamente para la lógica clásica, además el lenguaje TPTP, hace parte de esta biblioteca y es soportado por varios de los ATPs.
- Apia [Sicard-Ramírez 2015, cap. 6] es un programa que traslada la representación que realiza Agda de las formulas de primer orden al lenguaje TPTP, así mismo es una interfaz entre Agda y los ATPs.

Demostración automática

La demostración automática está referida a la utilización de programas de razonamiento que implementan métodos y técnicas para demostrar propiedades y teoremas de diferentes teorías formales.

Preliminares

- La TPTP es una biblioteca de problemas para sistemas que prueban teoremas automáticamente para la lógica clásica, además el lenguaje TPTP, hace parte de esta biblioteca y es soportado por varios de los ATPs.
- Apia [Sicard-Ramírez 2015, cap. 6] es un programa que traslada la representación que realiza Agda de las formulas de primer orden al lenguaje TPTP, así mismo es una interfaz entre Agda y los ATPs.

Demostración automática

Para realizar demostraciones automáticas de propiedades se usa una versión extendida del programa Agda, se emplea el programa Apia y los demostradores automáticos. Los pasos son los siguientes:

- Adicionar los ATP-pragma de las definiciones y axiomas necesarios de la formalización en Agda.

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

- Compilar el archivo empleando Agda.

```
$ agda ejemplo.agda
```

- Ejecutar Apia, en el archivo `ejemplo.agda`, seleccionando los ATPs de preferencia.

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

- Verificar si los ATPs seleccionados demostraron o no la formula.

Demostración automática

Para realizar demostraciones automáticas de propiedades se usa una versión extendida del programa Agda, se emplea el programa Apia y los demostradores automáticos. Los pasos son los siguientes:

- Adicionar los ATP-pragma de las definiciones y axiomas necesarios de la formalización en Agda.

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

- Compilar el archivo empleando Agda.

```
$ agda ejemplo.agda
```

- Ejecutar Apia, en el archivo `ejemplo.agda`, seleccionando los ATPs de preferencia.

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

- Verificar si los ATPs seleccionados demostraron o no la formula.

Demostración automática

Para realizar demostraciones automáticas de propiedades se usa una versión extendida del programa Agda, se emplea el programa Apia y los demostradores automáticos. Los pasos son los siguientes:

- Adicionar los ATP-pragma de las definiciones y axiomas necesarios de la formalización en Agda.

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

- Compilar el archivo empleando Agda.

```
$ agda ejemplo.agda
```

- Ejecutar Apia, en el archivo `ejemplo.agda`, seleccionando los ATPs de preferencia.

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

- Verificar si los ATPs seleccionados demostraron o no la formula.

Demostración automática

Para realizar demostraciones automáticas de propiedades se usa una versión extendida del programa Agda, se emplea el programa Apia y los demostradores automáticos. Los pasos son los siguientes:

- Adicionar los ATP-pragma de las definiciones y axiomas necesarios de la formalización en Agda.

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

- Compilar el archivo empleando Agda.

```
$ agda ejemplo.agda
```

- Ejecutar Apia, en el archivo `ejemplo.agda`, seleccionando los ATPs de preferencia.

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

- Verificar si los ATPs seleccionados demostraron o no la formula.

Demostración automática

Para realizar demostraciones automáticas de propiedades se usa una versión extendida del programa Agda, se emplea el programa Apia y los demostradores automáticos. Los pasos son los siguientes:

- Adicionar los ATP-pragma de las definiciones y axiomas necesarios de la formalización en Agda.

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

- Compilar el archivo empleando Agda.

```
$ agda ejemplo.agda
```

- Ejecutar Apia, en el archivo `ejemplo.agda`, seleccionando los ATPs de preferencia.

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

- Verificar si los ATPs seleccionados demostraron o no la formula.

Demostración automática

Consideraciones

Durante este proceso Apia llama indistintamente los ATPs seleccionados, si uno de ellos demuestra la propiedad, entonces el proceso se detiene y Apia informa cual ATP realizó la demostración. Si una propiedad no es demostrada empleando la metodología presentada en esta tesis, esto se puede asociar a uno de los siguientes casos:

- 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.
- Los ATPs elegidos ninguno logra hacer la demostración, en este caso se puede usar un ATP diferente.
- No sabe si la propiedad se puede demostrar automáticamente.

Demostración automática

Consideraciones

Durante este proceso Apia llama indistintamente los ATPs seleccionados, si uno de ellos demuestra la propiedad, entonces el proceso se detiene y Apia informa cual ATP realizó la demostración. Si una propiedad no es demostrada empleando la metodología presentada en esta tesis, esto se puede asociar a uno de los siguientes casos:

- 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.
- Los ATPs elegidos ninguno logra hacer la demostración, en este caso se puede usar un ATP diferente.
- No sabe si la propiedad se puede demostrar automáticamente.

Demostración automática

Consideraciones

Durante este proceso Apia llama indistintamente los ATPs seleccionados, si uno de ellos demuestra la propiedad, entonces el proceso se detiene y Apia informa cual ATP realizó la demostración. Si una propiedad no es demostrada empleando la metodología presentada en esta tesis, esto se puede asociar a uno de los siguientes casos:

- 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.
- Los ATPs elegidos ninguno logra hacer la demostración, en este caso se puede usar un ATP diferente.
- No sabe si la propiedad se puede demostrar automáticamente.

Demostración automática

Consideraciones

Durante este proceso Apia llama indistintamente los ATPs seleccionados, si uno de ellos demuestra la propiedad, entonces el proceso se detiene y Apia informa cual ATP realizó la demostración. Si una propiedad no es demostrada empleando la metodología presentada en esta tesis, esto se puede asociar a uno de los siguientes casos:

- 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.
- Los ATPs elegidos ninguno logra hacer la demostración, en este caso se puede usar un ATP diferente.
- No sabe si la propiedad se puede demostrar automáticamente.

Demostración automática

Ejemplo

La propiedad (1) fue demostrada de forma interactiva. Para demostrar automáticamente esta propiedad, se postula y se emplea el ATP-pragma para indicarle a los ATPs que la intenten demostrar.

Demostración automática.

```
postulate congruence-r : {x y z : ℝ} → x ≡ y
                                     → x + z ≡ y + z

{-# ATP prove congruence-r #-}
```

Demostración automática

Ejemplo

La propiedad (1) fue demostrada de forma interactiva. Para demostrar automáticamente esta propiedad, se postula y se emplea el ATP-pragma para indicarle a los ATPs que la intenten demostrar.

Demostración automática.

```
postulate congruence-r : {x y z : ℝ} → x ≡ y
                                     → x + z ≡ y + z

{-# ATP prove congruence-r #-}
```

Límite de una función

Formalización de límite

- Para formalizar el concepto de límite de una función se procedió en primer lugar a formalizar el valor absoluto de un número real y en segundo lugar, la distancia entre dos puntos, luego se demostraron algunas de sus propiedades.
- La formalización de la definición de límite fue adaptada de la librería estándar de Coq.
- Luego de haber realizado esta adaptación se demostraron algunas propiedades.

Límite de una función

Formalización de límite

- Para formalizar el concepto de límite de una función se procedió en primer lugar a formalizar el valor absoluto de un número real y en segundo lugar, la distancia entre dos puntos, luego se demostraron algunas de sus propiedades.
- La formalización de la definición de límite fue adaptada de la librería estándar de Coq.
- Luego de haber realizado esta adaptación se demostraron algunas propiedades.

Límite de una función

Formalización de límite

- Para formalizar el concepto de límite de una función se procedió en primer lugar a formalizar el valor absoluto de un número real y en segundo lugar, la distancia entre dos puntos, luego se demostraron algunas de sus propiedades.
- La formalización de la definición de límite fue adaptada de la librería estándar de Coq.
- Luego de haber realizado esta adaptación se demostraron algunas propiedades.

Conclusiones

A partir del trabajo realizado se obtuvieron los siguientes resultados:

- Se presentó de forma axiomática el sistema de los números reales, utilizando el asistente de pruebas Agda, para realizar demostraciones interactivas.
- Se emplearon los demostradores automáticos de teoremas como E y Vampire, así como el programa Apia, para demostraciones automáticas.

Conclusiones

A partir del trabajo realizado se obtuvieron los siguientes resultados:

- Se presentó de forma axiomática el sistema de los números reales, utilizando el asistente de pruebas Agda, para realizar demostraciones interactivas.
- Se emplearon los demostradores automáticos de teoremas como E y Vampire, así como el programa Apia, para demostraciones automáticas.

Conclusiones

- 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. Por ejemplo, el caso de la formalización del axioma de tricotomía.
- 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, tales como: definiciones que no están en lógica de primer orden, funciones de orden superior, trasladar al lenguaje TPTP la funcionalidad de la palabra reservada `with` y cuando se emplean varios universos en la formalización.

Conclusiones

- 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. Por ejemplo, el caso de la formalización del axioma de tricotomía.
- 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, tales como: definiciones que no están en lógica de primer orden, funciones de orden superior, trasladar al lenguaje TPTP la funcionalidad de la palabra reservada **`with`** y cuando se emplean varios universos en la formalización.

Repositorio

La formalización, los ejemplos mostrados en la presentación y otros más se encuentran en el siguiente repositorio en GitHub:

```
https://github.com/jechev28/real-numbers-formalisation
```

Referencias I

- Apostol, T. M. (1974). Mathematical Analysis. 2.ª ed. Addison-Wesley.
- 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.
- Harrison, J. (2015). HOL Light Tutorial. Intel Corporation.
- 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.
- Nipkow, T., Paulson, L. C. y Wenzel, M. (2016). A Proof Assistant for Higher-Order Logic. Springer-Verlag.
- 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.

Referencias II

- Sicard-Ramírez, A. (2015). Reasoning about Functional Programs by Combining Interactive and Automatic Proofs. Tesis doct. PEDECIBA Informática. Universidad de la República. Uruguay.
- 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.
- The Coq Development Team (2016). Reference Manual: Version 8.5.

Gracias!