Tesis sobre el Asistente de Pruebas LEAN enfocada en la Teoría de Tipos

Kevin Mateo Cárdenas

2023

1. Introducción

Lean es un sistema de demostración formal que se utiliza para verificar la correctitud matemática y de software. Fue desarrollado por Microsoft Research y es una combinación de lógica clásica y lógica intuitionista, permitiendo a los usuarios especificar y verificar teoremas y programas a través de un sistema de tipos dependientes y tácticas automáticas.

La teoría de tipos es un sistema que permite a los programadores y matemáticos especificar las propiedades de los datos en sus programas y teoremas. Este sistema es esencial para la programación y las matemáticas, ya que ayuda a evitar errores y a garantizar la consistencia de los datos.

Lean utiliza un sistema de tipos dependientes, que permite a los usuarios especificar relaciones complejas y precisas entre los tipos. Esto es muy útil en la verificación formal, ya que permite una mayor precisión y seguridad en la verificación de teoremas y programas.

La teoría de tipos tiene sus raíces en la década de 1930 con la obra de Alonzo Church en lambda cálculo. Desde entonces, ha evolucionado a través de la inclusión de sistemas de tipos más avanzados, como los tipos dependientes y la lógica intuicionista. Lean combina estos avances en teoría de tipos con tácticas automáticas, lo que lo convierte en una herramienta única y poderosa para la verificación formal de teoremas y programas.

En resumen, Lean es un sistema de demostración formal avanzado que combina lógica clásica y intuitionista con tipos dependientes y tácticas automáticas, permitiendo a los usuarios especificar y verificar teoremas y programas de manera precisa y segura. La teoría de tipos es una parte fundamental de este sistema, y su inclusión en Lean lo

convierte en una herramienta única y poderosa para la verificación formal.

2. Reseña Historica.

El proyecto formalista de Hilbert fue un movimiento matemático y filosófico en el siglo XIX y principios del siglo XX, liderado por David Hilbert. El objetivo principal de este proyecto era establecer la matemática sobre una base completamente formal y rigurosa, lo que permitiría la eliminación de las incertidumbres y controversias en la matemática.

Un aspecto clave del proyecto formalista de Hilbert era la creación de un sistema formal de demostración, que permitiría la verificación automática de teoremas matemáticos. Este sistema utilizaría una combinación de lógica y teoría de tipos para verificar la consistencia y la corrección de los teoremas.

Desde entonces, la idea de un sistema formal de demostración ha sido objeto de mucha investigación y desarrollo. Uno de los primeros sistemas de demostración formal fue el sistema NQTHM de Samual R. Buss, que utilizaba una lógica intuitionista para verificar teoremas matemáticos.

En la década de 1990, los sistemas de demostración formal evolucionaron para incluir también la verificación de programas de software. Uno de los primeros sistemas de demostración formal de software fue PVS de Sam Owre, John Rushby y Natarajan Shankar.

En los últimos años, el desarrollo de sistemas de demostración formal ha continuado a un ritmo acelerado, y ha surgido un nuevo sistema llamado Lean. Lean fue desarrollado por Microsoft Research y es un sistema de demostración formal que combina lógica clásica y intuitionista con tipos dependientes y tácticas automáticas.

Lean es único en su enfoque en la teoría de tipos dependientes, lo que permite a los usuarios especificar relaciones complejas y precisas entre los tipos. Esto aumenta la precisión y seguridad en la verificación formal de teoremas y programas. Además, Lean incluye tácticas automáticas, lo que permite a los usuarios realizar verificaciones más rápidamente y eficientemente.

En resumen, la historia de los asistentes de pruebas comienza con el proyecto formalista de Hilbert, que buscaba establecer la matemática sobre una base formal y rigurosa. Desde entonces, ha habido una evolución continua

en la investigación y desarrollo desistemas de demostración formal, incluyendo NQTHM, PVS y finalmente Lean. Lean se destaca por su enfoque en la teoría de tipos dependientes y tácticas automáticas, lo que lo convierte en una herramienta poderosa y precisa para la verificación formal de teoremas y programas.

3. Teoría de Tipos

La Teoría de Tipos es una herramienta fundamental en la matemática y la lógica formal, que permite la clasificación y verificación de la consistencia de objetos matemáticos. En otras palabras, es un sistema que permite determinar si un objeto matemático cumple con ciertas propiedades y si está en el lugar correcto en el sistema de clasificación matemática.

Esta teoría se desarrolló como una forma de lidiar con las ambigüedades y los paradoxos que surgieron en la lógica y la matemática en el siglo XIX. En particular, la Teoría de Tipos permite evitar la aparición de los paradoxos de Russell, que surgen al tratar de clasificar los objetos matemáticos de manera circular.

En la teoría de tipos, cada objeto matemático es asignado a un tipo, que es una clase abstracta de objetos con propiedades similares. Por ejemplo, los números enteros son un tipo, los números reales son otro tipo, y las funciones son un tercer tipo. Esta clasificación permite la manipulación coherente de los objetos matemáticos y ayuda a prevenir errores en los cálculos y la construcción de teoremas.

La Teoría de Tipos también es relevante en el contexto de la programación, donde se utiliza para verificar la corrección de los programas y evitar errores de tiempo de ejecución. Por ejemplo, en un lenguaje de programación tipado estático, los objetos se clasifican en tipos específicos, como números enteros o cadenas de caracteres, y el compilador verifica que las operaciones realizadas en estos objetos sean coherentes con sus tipos.

En el ámbito de la lógica formal, la Teoría de Tipos es una parte integral de la Lógica de Primer Orden, que es un marco lógico que permite la formalización y verificación de la matemática y la programación.

En conclusión, la Teoría de Tipos es una herramienta fundamental para la matemática y la informática, que permite la clasificación y verificación de objetos matemáticos y la prevención de errores en los cálculos y programas. Esta teoría es esencial para el desarrollo de sistemas y teoremas formales en la matemática y la informática.

3.1. Teoría de Tipos en Lean

La teoría de tipos es una parte fundamental de Lean y es utilizada para verificar la corrección de los programas. En Lean, todas las variables y expresiones tienen un tipo asociado que describe el tipo de valor que pueden tomar. Esto permite a Lean comprobar que los programas son correctos antes de ejecutarlos, detectando errores en tiempo de compilación en lugar de en tiempo de ejecución.

La teoría de tipos es importante en la verificación formal de programas porque ayuda a prevenir errores comunes y asegura que los programas funcionen correctamente. Por ejemplo, si una variable está asignada a un valor de un tipo equivocado, Lean lo detectará y mostrará un error antes de ejecutar el programa. Esto evita la ejecución de programas incorrectos y ahorra tiempo y esfuerzo al identificar y corregir errores de manera temprana.

Además, la teoría de tipos también es valiosa para la verificación formal de programas porque permite a Lean demostrar la corrección de los programas. Los usuarios pueden especificar pre- y post-condiciones para sus programas y utilizar Lean para demostrar que sus programas cumplen con estas condiciones. Esto aumenta la confianza en la correctitud de los programas y permite a los usuarios verificar la corrección de programas complejos con mayor facilidad.

En resumen, la teoría de tipos es un aspecto fundamental de Lean y es crucial en la verificación formal de programas. Su uso permite a Lean prevenir errores comunes y aumentar la confianza en la correctitud de los programas mediante la verificación y demostración de su corrección.

4. Sintaxis y Semántica de Lean

LEAN es un lenguaje de programación basado en la teoría de tipos, lo que significa que se utiliza para determinar el tipo de datos y la corrección de los mismos en tiempo de compilación. La sintaxis de LEAN se compone de tres elementos principales: declaraciones, expresiones y términos.

Las declaraciones en LEAN permiten definir nuevos tipos de datos, funciones, constantes y variables. Por ejemplo, se puede declarar un nuevo tipo de datos llamado ”personaçon las propiedades ”nombreτ .^{ed}ad”. Estas declaraciones se escriben utilizando la palabra clave ”definir”seguida de la estructura que se desea definir.

Las expresiones en LEAN permiten evaluar valores y realizar operaciones. Por ejemplo, se puede evaluar la edad de una persona y compararla con otra edad. Estas expresiones se escriben utilizando operadores matemáticos y

funciones.

Los términos en LEAN permiten hacer inferencias y demostraciones formales. Por ejemplo, se puede demostrar que dos personas son iguales si tienen el mismo nombre y edad. Estos términos se escriben utilizando la notación matemática y los teoremas previamente definidos.

La semántica en LEAN describe cómo se interpretan las declaraciones, expresiones y términos en el lenguaje. Por ejemplo, se puede definir una función que tome una persona y devuelva su edad. La semántica describe cómo se evalúa esta función y qué resultado se espera.

En resumen, la sintaxis y la semántica en LEAN son clave para comprender y usar este lenguaje de programación basado en la teoría de tipos. La sintaxis permite definir estructuras y realizar operaciones, mientras que la semántica describe cómo se interpretan estas estructuras y operaciones en el lenguaje. Al aprender la sintaxis y la semántica de LEAN, se pueden escribir programas formales y seguros que utilicen la teoría de tipos para garantizar la corrección de los datos.

Por ejemplo, la siguiente es una demostración sencilla de un teorema en Lean:

```
1  theorem add_assoc (a b c : nat) : (a + b) + c = a + (b + c) :=
2  begin
3    induction c with d hd,
4    { rw add_zero, refl },
5    { rw add_succ,
6      rw hd,
7      refl }
8  end
```

En este código, estamos definiendo un teorema llamado .ªdd assoc"que toma tres argumentos, a, b y c, y demuestra que $(a+b)+c = a+(b+c)$.

Utilizamos la táctica de inducción sobre c con la variable auxiliar d y la hipótesis hd (hipotesis inductiva) para escribir la demostración. En la base del caso, reescribimos c como 0 utilizando add_zero y utilizamos la táctica refl para demostrar que las dos expresiones son iguales. En el caso inductivo, reescribimos c como $d+1$ utilizando add_succ, aplicamos la hipótesis inductiva hd y utilizamos refl para demostrar que las dos expresiones son iguales, la

táctica refl es una técnica de prueba muy útil en sistemas de verificación formal como Lean, ya que permite demostrar propiedades obvias de un cálculo de manera sencilla y automática. Al mismo tiempo, esto permite una verificación más rigurosa y confiable de los programas y teoremas que se escriben en el sistema.

4.1. El pequeño Teorema de Fermat

Demostraremos el pequeño teorema de Fermat con la sintaxis de LEAN:

```
theorem Fermat_little_theorem (p : nat) (a : nat) : a^p = a [mod p] :=
begin
  have H : a^p = a * a^(p-1) [mod p],
    by rw [pow_succ, mul_mod_right],
  rw [H, pow_one, mul_one]
end
```

En este código, estamos definiendo un teorema llamado ”Fermat little theorem”que toma dos argumentos, p y a, y demuestra que $a^p = a[mod p]$.

Utilizamos la táctica begin ... end para escribir la demostración. Dentro de ella, primero definimos una propiedad intermedia H que iguala a^p con $a * a^(p-1)[mod p]$. Luego, utilizamos la táctica rw (rewrite) para reescribir a^p en términos de H. Finalmente, reescribimos $a * a^(p-1)$ como a utilizando la propiedad pow_one y mul_one.

4.2. Infinitos primos.

El teorema de los números primos infinitos establece que existen infinitos números primos. Aquí hay una posible demostración en el lenguaje de programación Lean:

```
import data.nat.basic

theorem prime_infinite :     n :    ,     p :    , p > n     prime p :=
begin
  intros n,
  have h := exists_greater_prime n,
  use (nat.next_prime n),
  split,
  { exact nat.next_prime_gt n },
  { exact h }
```

```
11      end
```

En este código, se importa la biblioteca data.nat.basic, que contiene definiciones y teoremas sobre los números naturales. Luego, se define el teorema prime_infinite, que establece que para cualquier número natural n, existe un número primo p mayor que n.

La demostración utiliza la táctica üse"para especificar la existencia de un número primo y "split"para separar una afirmación compuesta en dos afirmaciones individuales. Finalmente, se utiliza la función "nat.next_prime_gt"para demostrar que "nat.next_prime"*n* es mayor que *n*, y la hipótesis h para demostrar que es primo.

5. Uso de Lean en Matemáticas

Lean se utiliza en el ámbito de las matemáticas como una herramienta de verificación formal para demostrar teoremas y propiedades matemáticas. Los usuarios pueden escribir demostraciones en un lenguaje claro y eficiente, y Lean verifica que la demostración sea válida y cumpla con las reglas matemáticas adecuadas.

Lean se utiliza para verificar la corrección de teoremas en áreas como la teoría de números, la geometría, la lógica matemática y muchas más. Además, se puede utilizar para desarrollar librerías matemáticas complejas, como librerías de álgebra lineal y teoría de grupos.

Una de las aplicaciones prácticas más importantes de Lean en matemáticas es su capacidad para verificar la corrección de software matemático, como cálculo simbólico y numérico, sistemas de álgebra computacional y mucho más. Al utilizar Lean para verificar el software matemático, los usuarios pueden estar seguros de que los resultados son precisos y que el software está funcionando correctamente.

En resumen, Lean es una herramienta valiosa en el ámbito de las matemáticas, y se utiliza para verificar la corrección de teoremas, desarrollar librerías matemáticas y verificar el software matemático. Su combinación de claridad y eficiencia hacen que sea una herramienta popular y valiosa para los matemáticos y los científicos de la computación.

6. Ventajas y Desventajas de Lean

6.1. Ventajas de Lean.

- Verificación automática: Lean utiliza la teoría de tipos y la verificación formal para garantizar la corrección de las demostraciones matemáticas y los programas. Esto reduce el margen de error y la necesidad de revisión humana.
- Lenguaje claro y eficiente: Lean utiliza un lenguaje claro y conciso para escribir demostraciones matemáticas, lo que facilita la comprensión y el mantenimiento del código.
- Comunidad activa: Lean tiene una comunidad activa de usuarios y desarrolladores que contribuyen al desarrollo del software y proporcionan soporte y recursos adicionales.
- Integración con otros sistemas: Lean se integra fácilmente con otros sistemas de verificación formal y herramientas matemáticas, lo que permite una mayor flexibilidad y eficiencia en el trabajo.

6.2. Desventajas de Lean:

- Curva de aprendizaje: Aprender a utilizar Lean puede ser un desafío para algunos usuarios, especialmente si no tienen experiencia previa en verificación formal o programación.
- Desempeño: Aunque Lean es muy eficiente en la verificación de demostraciones matemáticas, puede ser más lento que otros sistemas de verificación formal en la verificación de programas más grandes y complejos.
- Limitaciones en la verificación: Lean no es capaz de verificar todos los teoremas y programas matemáticos, y puede requerir una demostración adicional por parte del usuario para garantizar la corrección de algunos resultados.
- En comparación con otros sistemas de verificación formal, Lean ofrece una combinación única de verificación automática, claridad en el lenguaje y integración con otros sistemas. Sin embargo, la curva de aprendizaje y las limitaciones en la verificación son factores a considerar al elegir un sistema de verificación formal.

7. Conclusión

En conclusión, Lean es un sistema de verificación formal de programas y matemáticas diseñado para ser fácil de usar y eficiente. La teoría de tipos es un aspecto fundamental de Lean y es crucial en la verificación formal de programas, permitiendo prevenir errores comunes y aumentar la confianza en la correctitud de los programas. La

sintaxis y semántica de Lean son claras y concisas, y se asemejan a la notación matemática convencional.

Desde su lanzamiento en 2013, Lean ha sido ampliamente utilizado en una variedad de aplicaciones y ha demostrado ser una herramienta valiosa para la verificación formal de programas y teoría de tipos. Lean continúa evolucionando y mejorando, y se espera que siga siendo una herramienta valiosa en el futuro.

Este trabajo se enfocó en la teoría de tipos en Lean, y su importancia en la verificación formal de programas. Se espera que esta tesis brinde una visión clara de la función de la teoría de tipos en Lean y su relevancia en la verificación formal de programas y en la enseñanza y investigación en teoría de tipos.

Referencias

[1] Institute for Computing and Radboud Information Science, Faculty of Science. Proof assistants: History, ideas and future., volume 1. University Nijmegen, 2014.

[2] J. Roger Hindley. Lambda-calculus and combinators, volume 1. Cambridge, 2010.

[3] Rob Nederpelt. Type theory and formal proof, volume 1. Cambridge University Press, 2014.

[4] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions., volume 1. College Publications, 2015, Studies in Logic (Mathematical logic and foundations), 2014.

[5] Microsoft Research. Theorem Proving in Lean. [En línea]. Disponible en: https://leanprover.github.io/.

[6] Benjamin C. Pierce. Software Foundations. Editorial: Cambridge University Press.

[7] Adam Chlipala. Certified Programming with Dependent Types. Editorial: Cambridge University Press.

[8] Benjamin C. Pierce. Types and Programming Languages. Editorial: MIT Press.

[9] Leonardo de Moura, Jeremy Avigad, Soonho Kong. The Lean Theorem Prover: An Introduction. Editorial: Communications of the ACM.