*La verificación formal y los asistentes de prueba como pieza clave en la industria de la computación*

Integrantes:


*Castillo Hernández Antonio - 320017438*

☎☎☎☎☎

*Luna Campos Emiliano - 320292084*

☎☎☎☎☎

*Vázquez Reyes Jesús Elías - 320010549*

☎☎☎☎☎

Universidad Nacional Autónoma De Mexico
Facultad de Ciencias



# Resumen

En el presente proyecto se abordan temas relacionados con la verificación formal como lo son su historia, aplicación mediante asistentes de prueba, y un conjunto de ejercicios enfocados en una representación binaria alternativa de los números naturales tradicionales. Inicialmente, se explora la importancia de la verificación formal para garantizar la correctitud del software crítico, destacando ejemplos históricos y la evolución de métodos formales desde Frege y Russell, hasta los avances en la lógica de Hoare. Posteriormente, se mencionan aplicaciones contemporáneas de la verificación formal, incluyendo la verificación formal de programas cuánticos, la compilación formalmente verificada, y la verificación formal en algoritmos criptográficos AES.Finalmente, se analizan los desafíos, aprendizajes y hallazgos encontrados durante la resolución de los ejercicios, así como la eficacia del trabajo realizado.

# Preliminares

En la presente investigación se contó con el apoyo de diversos estudios y análisis con un énfasis principal en la verificacion formal y asistentes de prueba. Dichos estudios nos dejaban ver que la materia prima de la verificación formal se centra en el uso de las matemáticas y la lógica formal para la resolucion de demostraciones formales, mediante herramientas especializadas las cuales fingiran como apoyo principal para lo que el usuario desee obtener de este proceso.

Dicho esto, la forma más accesible y acostumbrada de abordar este tema es hablando en primera instancia acerca de nuestro mundo actual. Claramente con el paso del tiempo la dependencia del ser humano con respecto a la tecnolgía consecuentemente ha hecho que la complejidad en la que se envuelven dos areas como la ciencia, y la misma mencionada, haya crecido drásticamente. Actualmente todo tipo de organizaciones están relacionadas con el uso de software de alguna manera; los casos ideales para ejemplificar esto son las cuentas bancarias, que sin dudarlo son un sistema de suma importancia, y en el que se han involucrado de primera mano el software para facilitar el manejo de la economía.

Hagamos una suposición terrorífica: "¿qué pasaría si algún software de algún ámbito importante fallara?"; es claro que podria afectar no solo a una sola persona, sino una región considerablemente más grande. Como estos, hay muchos otros ejemplos que han sucedido en la vida real, algunos más devastadores que otros, como la falla documentada del $Therac-25$ (en primera instancia se trataba de una máquina de radiología la cuál daba una sobredosis mortal de radiación a pacientes bajo ciertos parámetros y condiciones en específico, los cuales hacían que se detonará este error), u algunos otros errores tan sencillos como el querer indexar a un espacio de un arreglo que no existe.

Hoy en día se tiene mas noción sobre la importancia de verificar un software y los daños (ya sean mayores o menores) que puede provocar el no ponerlo a prueba, lo que nos lleva a preguntarnos cómo asegurarse de que el software realmente funcione de manera correcta. En un pronto, y con bastante lógica, es natural pensar el recrear distintas pruebas y corroborar que el resultado esperado sea ideal e identico al que se recibe, y no solamente parecido e idoneo. Sin embargo esto a su vez abre caminos que abordar, y por lo tanto tambien poder dominar. A causa de lo que antes se ha dicho y de los errores presentes en el software, ¿qué garantiza que la pruebas que se crean para verificar que el software no falle, en realidad no fallan?. Esta misma pregunta se la han hecho varias personas a lo largo de la historia, fue entonces que, bajo las ordenanzas e ideas que tenian ellos mismos, se fue conformando la idea de un sistema que utizara metodos matemáticos y lógicos para corroborar la correctitud y propiedades de los sistemas de software para su correcto funcionamiento, fue entonces se fue formando a lo que se le conoce hoy como la *verificación formal.*

Entonces, se tiene la cuestión de asegurarse que el desarrollo del software sea confiable. ¿Sobre qué clase de razonamiento se puede abarcar a tal confiabilidad?, lo oportuno es el razonamineto matemático. En efecto, la Verificación Formal es un proceso que ocupa los "métodos formales" para decidir la correctitud de algoritmos dadas especificaciones, y de

esta manera tener un completo estado de pruebas garantizado. Especificando un poco más, los "metodos formales" son técnicas basadas en las matemáticas y que al igual están estrechamente relacionadas con la lógica ya que requieren a su vez de deducciones, tener un conjunto de hechos conocidos y utilizar un conjunto reglas de inferencia establecidas para llegar a una conclusion deseada. Además, con este tipo de verificación se hace mucho más sencillo el estar simulando desmesuradamente, ya que todas las posibles entradas pueden ser exploradas algorítmicamente.

El poder utilizar la lógica (variando el orden de esta, con sus respectivas ventajas y desventajas en cada caso) para los "métodos formales" se debe precisamente al "Isomorfismo Curry-Howard" (mismo que se mencionara más adelante) que declara que los tipos son proposiciones y los programas demostraciones, apesar de que aun hay limitaciones practicas y teoricas en lengujaes funcionales en este ambito. Realmente este es un trabajo investigacion que vale la pena indagar y "empaparse" mas de el, pero que el profundizar en el queda muy fuera del alcance de este reporte de investigacion.

Hay que entender que como todo en el mundo, la verificación formal no surgio de la noche a la mañana, si no que llevo un proceso para formarse tal y como se conoce contemporaneamente. Se produjo a raiz de que muchos matematicos (y computologos mas tarde) tuvieran la necesidad tener un sistema de pruebas que los ayudará con su labor matematico de esquematización y formalización de las matematicas, mas tárde a este ámbito en particular se le conocería como lógica formal.

A finales del siglo $XIX$ y principios del $XX$, los matematicos Gottlob Frege y Bertrand Russell hicieron las primeras contribuciones importantes a la lógica formal (moderna), lo cual sento las bases para la verificación formal. Frege introdujo un sistema lógico que permitía la representación precisa de proposiciones y sus relaciones, dicho sistema es mas conocido como *lógica de fredge*.
Según Frege, el sentido de una expresión es su significado y la referencia es el objeto real al que se refiere dicha expresión es entonces que para llevar acabo este analisis Frege introdujo la noción de "juicio" o "proposición".
Sin embargo Bertrand Russell vio en el trabajo de Frege una contradicción (**Paradoja de Russell**). Dicho trabajo demostraba que el sistema lógico no era consistente y cambio e influyo a la logica matematica de obras venideras.

Después de estos eventos, a lo largo del siglo XX, los matemáticos y lógicos continuaron formalizando diversas ramas de las matemáticas. David Hilbert promovió el formalismo, proponiendo que todas las matemáticas podrían basarse en un conjunto finito de axiomas y reglas de inferencia. En 1931, Kurt Gödel demostró que en cualquier sistema axiomático suficientemente poderoso, existen proposiciones verdaderas que no pueden ser probadas dentro del sistema. Aunque esto mostró los límites del formalismo, también estimuló más investigación en la formalización y verificación de sistemas. En la década de 1960, Tony Hoare introdujo un gran avance para la "Verificacion Formal" con su lógica de Hoare, que era derivada de la logica de predicados, una forma de razonamiento formal que permite verificar la corrección de programas de computadora mediante "triples de Hoare", que especifican pre-

condiciones y postcondiciones para los programas. Durante esta misma época, Robert Floyd y Tony Hoare desarrollaron métodos para la verificación formal de programas, utilizando invariantes y condiciones de preprocesado y postprocesado para asegurar la corrección de los algoritmos.

De esa manera se fue conformando a lo que le conoce hoy como verificación formal. Todo esto claramente de manera muy resumida y sin abordar en los temas mismos a detalle. Mas alla de eso cada uno de esos escalones que representaron los avances en la materia defienden el nombre de la misma verificación formal; una manera de corroborar juiciosamente. No obstante, hasta el momento se ha hablado de manera totalmente teórica, hablando de porqué importa, qué es, cómo sirve, su contexto; en algún momento nos preguntaremos cómo se aplica, pues bien, este reporte se centra específicamente en los asistentes de prueba, justo una de las respuestas a la pregunta: una manera de utilizar la verificacion formal.

En un mundo ideal existen herramientas tan avanzadas con la capacidad de demostrar teoremas con la simple acción de picar un botón. Hasta que se hagan presente ese día, se cuentan con herramientas suficientemente capaces (evidentemente no tan mágicas ni tan poderosas) de guiar las mismas. Los asistentes de prueba son programas o sistemas informáticos que ayudan a hacer matemáticas; un momento ¿Qué no eso es lo que ya hacen y su propósito principal?; su principal ayuda ha sido en el cálculo y aritmética (con miles ramas en ellas); los asistentes de prueba ayudan en aspectos totalmente teóricos, ayudan a definir y demostrar valga la redundancia teorías.
Las pruebas comúnmente se componen de pruebas aún más pequeñas, así que estas mismas pueden ser un campo en la que se puede echar mano de los asistentes, escribir un programa que verifique de manera mecánica las pruebas es una opción válida y de la cual más adelante se mostrara que este era el principio básico de los asistentes de prueba en sus inicios. Sin embargo, una de las mejores maneras de asegurarse que las demostraciones que se quieren llevar a cabo son realmente correctas, es establecer las propias definiciones de la teoría dispuesta a trabajarse como uno las piensa, de esta manera las pruebas en las que se trabaja serán posibles de probar únicamente si pueden ser derivables bajo la lógica que se está estableciendo.

Más que eso los asistentes actuales se apoyan de los avances en cómputo para ocupar "tácticas", que son comandos que ayudan al usuario a hacer uso de técnicas representadas en un alto nivel, un ejemplo claro es la inducción. Al interactuar con el asistente de pruebas se puede hacer uso de las tácticas, que como se adelantó ya, dividen la prueba en unas cada vez más pequeñas y sencillas, al comprobar cada una de estas "minipruebas" se tiene como resultado que está probado el objetivo principal. Ahora bien, estas herramientas como todo, tienen límites y fallos. Desde luego no pueden apoyar a los usuarios a plantear los teoremas, ni mucho menos formalizarlos, es decir que el usuario aún tiene que rascarse la cabeza para poder utilizar los asistentes de prueba. También posible pasar por alto inconsistencias cuando se transcribe a papel, además de que no es tan trivial hacer estos mismos transcritos. Este también, es otro punto a profundizar, el gran abanico de posibilidades que da el utilizar las "tácticas" y profundizar en su funcionamiento a la vez que se reconoce las limitaciones de estos. Aunque igualmente son conceptos que no abarca este reporte de investigación.

Hablando un poco sobre su origen habría que remontarse casi cincuenta años atrás, alrededor de inicios de los setentas, ya que justo en esos años fue cuando en general se empezó a concebir la idea de pruebas corroboradas por la computadora. En 1967 De Bruijn introdujo el isomorfismo Curry-Howard a su verificador dando como resultado "verificación de prueba = verificación de tipos", y únicamente lo acompaño de conceptos muy básicos con el objetivo de poder agregar reglas lógicas propias, siendo este apenas un verificador de pruebas (escribía la prueba completa y la verificaba). Más adelante Martin-Lof y Milner introduciría los tipos inductivos y la recursión, en 1972, y que más adelante servirían para formar asistentes como Isabelle.

Por el medio también se han desarrollado asistentes declarativos como Mizar, que desde 1973 se ha hecho popular principalmente en países orientales. En el mismo año se desarrolló la lógica de primer orden Nqthm que permitía la agregación de lemas por parte del usuario. También cabe mencionar el PVS bastante más reciente que el resto (de la década de los 90's) que tiene una lógica clásica de orden superior tipificada. Y EA, un sistema ruso que ha permanecido igualmente desde la década de los 70's muy hermético, pero del que se sabe evoluciono hasta convertirse en un SAD (System for Automated Deduction) que verifica textos matemáticos.

¿Y esto se ha aplicado?, ¿Sirve de algo más que hacer inducción en una computadora?, a continuación se presentan algunas aplicaciones en el cómo los asistentes de prueba han aportado en problemas del mundo real.

## Aplicaciones

- **Verificación formal de programas cuánticos**

A estas alturas podría considerarse como redundante el decir que vivimos en 'la era de la tecnología', pues en todo lados se repite esta misma afirmación, aunque eso no le quita razón. Uno de los avances tecnológicos que mas dá de qué hablar son las computadoras cuánticas, las cuales se basan en ciertos principios de la superposición de la materia, y el entrelazamiento cuántico, para desarrollar una computación distinta a la tradicional; estos equipos de computo deberían ser capaces de almacenar más estados por unidad de información y operar con algoritmos que son ciertamente más eficientes a nivel numérico.

La computación cuántica despertó las ansias e interés de muchos expertos de la indudtria de la computación, tan es así, que su campo ha sufrido de un aumento considerable en cuanto a investigaciones. No obstante, tal como mencionan Lewis, Soudjani y Zuliani (2023) en su artículo "Verificación formal de programas cuánticos: teoría, herramientas y desafíos", la atención que se le da a la verificación formal de programas cuánticos ha disminuído debido a lo complicada que puede resultar la programación eficaz y correcta de algoritmos complejos en hardware cuántico, siendo que ésta es bastante propensa a fallar. Si bien no se le da tanta atención como podría, también es verdad que la verificación en programas de tipo cuántico se haun desarrollado recientemente, contando con una gran baraja de herramientas basadas en ideas puramente teóricas.

Primero hay que definir algo de suma importancia: los Qubit. Estos son la unidad de almacenamiento de información mmás básica de información, basicamente son como los bits convencionales de las computadores estándar. Por otro lado, la cualidad característica de los quibits es que son capaces de admitir una coherente superposición de ceros y unos, a diferencia del bit, que solo puede guardar un único valor en un tiempo determinado (cero o uno). El artículo de Lewis se centra en los errores causados por el software en la computación cuántica y cómo podrían ser prevenidos a traés del uso de verificación formal. Pueden haber 3 fuentes de error fundamentales:

La primera se provoca cuando se realiza una medición de qubit, ya que sus estados suelen tener una cierta cantidad de aleatoriedad. Si bien los algoritmos cuánticos frecuentemente producen el resultado correcto con alta probabilidad, como en el caso del algoritmo de Grover que busca un elemento en una base de datos, siempre existe el riesgo de obtener mediciones erróneas. Sin embargo, este problema se puede mitigar repitiendo el proceso varias veces y seleccionando el resultado más frecuentemente observado, que tiende a ser el correcto. Aunque los asistentes de prueba no pueden eliminar la aleatoriedad inherente a la medición de los estados cuánticos, son usados para verificar la lógica de los algoritmos que gestionan estas mediciones. Por ejemplo, aseguran que un algoritmo ejecute el número adecuado de repeticiones y recolecte resultados de manera eficiente para maximizar la probabilidad de obtener el resultado correcto. Los asistentes de prueba y la verficación formal verifican que la implementación del algoritmo maneje correctamente las probabilidades y seleccione el resultado más probable como el válido.

En segundo lugar, los errores de hardware, que ocurren cuando los qubits son perturbados por elementos externos o por el funcionamiento inadecuado de las puertas cuánticas, fenómeno conocido como 'infidelidad de puerta'. Además, pueden surgir errores durante la lectura si los qubits son medidos de manera incorrecta. Para contrarrestar estos errores, es útil emplear códigos de corrección de errores, lo que contribuye a la realización de computación cuántica tolerante a fallas. Los asistentes de prueba pueden no impactar directamente en los errores de hardware, pero son útiles para validar los códigos de corrección de errores cuánticos. Estos códigos son esenciales para crear sistemas cuánticos tolerantes a fallos. Los asistentes pueden verificar matemáticamente que estos códigos cumplan con sus especificaciones teóricas, garantizando así que los errores de hardware sean corregidos adecuadamente antes de que los resultados sean interpretados por el software.

Finalmente, los errores de software pueden presentarse en diversas fases, desde la codificación errónea de algoritmos hasta deficiencias en los circuitos compilados. Para abordar estos errores, es crucial desarrollar herramientas que permitan la verificación de programas y circuitos cuánticos compilados. Al minimizar los errores en el hardware y el software, se asegura que las distribuciones de resultados medidos se alineen con las expectativas teóricas. Esta es el área donde los asistentes de prueba tienen el impacto más directo. Son utilizados para probar formalmente que el software que controla los circuitos cuánticos está libre de errores lógicos. Esto incluye la verificación de la correcta implementación de algoritmos, la precisión de la compilación de los circuitos cuánticos, y la validación de que el comportamiento del software se alinea con las expectativas teóricas y las especificaciones del diseño.

- **Compilación formalmente verificada**

Existe un tipo de compilador denominado como "compilador de producción", los cuales son una herramienta diseñada para generar código ejecutable de alta calidad a partir de código fuente, optimizado para rendimiento o eficiencia en entornos de producción. A diferencia de otros tipos de compiladores, que pueden enfocarse en características experimentales o en la facilidad de depuración, los compiladores de producción se centran en estabilidad,compatibilidad y velocidad con diversos sistemas operativos. Ciertamete, estos compiladores son esenciales para muchas aplicaciones vitales de la industria, como sistemas operativos, aplicaciones de bases de datos, o software integrado en dispositivos.

A nosotros nos concierne el compilador de producción CompCert, para el lenguaje de programación C, cuya característica principal es ser el único de su tipo que "está formalmente verificado, mediante pruebas matemáticas asistidas por máquinas, para estar exento de problemas de mala compilación" (AbsInt, 2023). El código generado a partir de CompCert tiene un comportamiento que va acorde a lo esperado, es decir, de la semántica del programa fuente en C.

CompCert implementa diversas optimizaciones, y cada una ha sido verificada formalmente usando el asistente de pruebas Coq:

**1. Asignación de registros mediante la coloración de gráficas y fusión de registros iterados**

Coloración de gráficas: Este método se basa en un algoritmo de coloreo de gráficas para asignar variables del programa a registros físicos. La verificación formal de este algoritmo incluye asegurar que dos variables que se intersecan en tiempo de vida no compartan el mismo registro, a menos que se haya garantizado que no causará conflictos en tiempo de ejecución.

Fusión de registros iterados: Optimiza la asignación de registros fusionando registros cuando es posible hacerlo sin alterar la semántica del programa. La verificación formal debe demostrar que estas fusiones no introducen condiciones de carrera o cambios en el resultado final del programa.

**2. Selección de instrucciones con reducción de fuerza**

La selección de instrucciones trata de elegir la representación más eficiente de cada operación del código intermedio en las instrucciones de la arquitectura objetivo. La reducción de fuerza intenta reemplazar las operaciones costosas por equivalentes más baratas. La verificación de esta optimización garantiza que las transformaciones conservan la semántica original del programa.

**3. Propagación constante para tipos enteros y de punto flotante**

Esta optimización evalúa expresiones en tiempo de compilación cuando todas sus entradas son constantes, y propaga estos valores constantes. La verificación formal de esta optimización asegura que la propagación de constantes no altera el comportamiento del programa según su especificación original, incluso considerando las diferencias en la precisión y el manejo de los tipos de datos de punto flotante.

**4. Eliminación de subexpresiones comunes**

Identifica y elimina las instancias donde subexpresiones idénticas se calculan más de una vez, reutilizando el resultado de la primera computación. La verificación formal debe mostrar que la reutilización de estos resultados no introduce cambios semánticos en el programa.

**5. Eliminación de código muerto**

Esta optimización remueve código que no afecta el resultado del programa, como variables no usadas o códigos que nunca se ejecutan. La verificación formal de esta optimización implica demostrar que el código eliminado realmente es innecesario para el resultado final del programa.

### 6. Función en línea

Consiste en reemplazar una llamada a una función por el cuerpo de la función. La verificación formal debe asegurar que este reemplazo no altera la semántica del programa, prestando especial atención a los efectos de borde como el ámbito de las variables y la manipulación de la pila.

### 7. Eliminación de llamadas de cola

Optimiza las llamadas de funciones en posición de cola para mejorar el uso de la pila y evitar la sobrecarga de llamadas recursivas. La verificación formal demuestra que transformar las llamadas de cola en saltos no modifica la funcionalidad esperada del programa.

- **Verificación formal en algoritmos criptográficos AES**

Desde los inicios de la internet en la epoca moderna, este mismo se ah caracterizado por ser bastante inseguro, tanto asi que la creación de la World Wide Web y de cualquier otro protocolo implementado atravez de lo que hoy conocemos como internet global desde su nacimiento hasta la actualidad, siempre han requerido de parches y actualizaciones cosntantes en este ambito conocido como seguridad informatica (o ciberseguridad en terminos mas coloquiales).

Es por eso que en el ejemplo presente, la implementación de la verificación formal para algoritmos criptográficos AES **(Advanced Encryption Standard)** es una muy buena forma de asegurarnos de que los mismos no van a fallar o almenos nos asegura tener una certeza de estos son mas seguros que los algoritmos a los cuales no se les implementa la verificación formal.
Sabemos que la clase de algoritmos AES son algoritmos los cuales estan bajo un estándar internacional ampliamente utilizado tanto por gobiernos como comercialmente en la industria del sofware es asi que en el presente ejemplo se implementa una versión simplificada del algoritmo, llamada Mini-AES, que fue creada con fines académicos pero sin perder la esencia de su funcionamiento y sus propiedades criptográficas. Así mismo tambien se eligio el asistente de pruebas coq debido a que este provee un lenguaje formal mediante el cual se puede modelar la sintaxis y semántica del lenguaje que se quiere modelar, entonces llegados a este punto es oportuno mencionar que el objetivo principal del proyecto de este ejemplo es diseñar un lenguaje imperativo e implementar el algoritmo Mini-AES, probando formalmente la corrección funcional de Mini-AES, asegurando que los procedimientos de encriptar y desencriptar sean inversos entre si *(mas adelante se menciona el porque esto tiene que ser así).*

Veamos que en Criptografía se define como "texto plano" a un mensaje o información en su forma original y "texto cifrado" o "encriptado" como la codificación del texto plano en una cierta forma que aparente ser un flujo de información aleatoria o sin sentido, pero que podrá ser procesada solamente por entidades autorizadas para recuperar el texto plano.

También se define encriptación o codificación como el proceso de transformar texto plano en texto cifrado y "desencriptación" o "decodificación" como el proceso de recuperar el texto plano a partir del cifrado. Durante el proceso de encriptación y desencriptación, los algoritmos criptográficos utilizan un parámetro de entrada especial denominado clave el cual determina la salida específica del algoritmo, haciendo que el mismo texto al ser encriptado con dos claves distintas produzca salidas distintas. Se puede ver entonces al proceso de encriptación como una función que recibe de entrada un texto plano y una clave y produce como salida un texto cifrado. En el sentido opuesto, la desencriptación es una función que realiza el procedimiento inverso y, dado un texto cifrado junto con una clave, devuelve el texto plano original, es por esto mismo que ambos procedimientos, tanto el de encriptar como desencriptar tienen que ser inversos entre si.

Es así que muchos de los algoritmos de encriptación comparten estas mismas características mencionadas anteriormente. De esta misma forma el algortmo AES y todos sus derivados funcionan de manera similar. En el ejemplo que estamos tratando, tanto el algoritmo AES y mini-AES se basan fundamentalmente en la teoría de campos finitos, sin embargo dado que ese tema no compete al proyecto de investigación presente solo daremos las nocionesfundamentales para comprender la operativa de AES.
El funcionamiento del algoritmo criptográfico de AES se basa en una red de sustitución-permutación o SPN, un sistema que se utiliza para convertir bloques de texto planos en bloques cifrados. De esta manera, modifica la información original de forma que, si es interceptada, sea incomprensible. Para poder descifrarla, hará falta estar en posesión de una clave que podrá ser de 128, 192 o 256 bits.

En el cifrado AES, la información se estructura en bloques, todos de un tamaño fijo de 128 bits. Estos están compuestos de una matriz de cuatro por cuatro bytes (cada byte tiene 8 bits, de ahí los 128). Después, cada byte se va moviendo de sitio y reemplazando siguiendo una serie de instrucciones recogidas en la clave. Es justo en la longitud de esa clave (128, 192 o 256 bits) donde están las bases de la seguridad de todo el sistema.

De la misma forma, por ser una simplicación del AES, el algoritmo Mini-AES funciona de la misma forma pero aplicándose en bloques de menor tamaño y utilizando una clave de menor tamaño además de aplicar menor cantidad de rondas. Es decir para el estudio del Mini-AES se enfocan principalmente en sus diferencias con el original.Por lo cual, la verificación formal en este ejemplo se realiza utilizando la lógica de Hoare en Coq, demostrando que la implementación de Mini-AES es funcionalmente correcta. Asi mismo durante la realizacion de dicho proyecto se realizo la prueba formal de corrección del algoritmo Mini-AES, es decir, probaron que para todo bloque de texto plano (representado por una matriz) y para toda clave, si encriptamos el texto y luego desencriptamos el resultado, el bloque no cambia.

Así pues, el ejemplo presentado en esta sección es uno de los primeros acercamientos al tema sobre la verificación formal en los algoritmos de cifrado AES, así como tambien nos muestra el porque de la importancia de la verificación formal en la Ciberseguridad y Criptografía. Veamos que esta es otra de sus aplicaciones útiles de la verificación formal en ambitos que nos competen como computólogos.

# Implementación

Se pone sobre la mesa una definición y/o representación alternativa de los números naturales a través de dos constructores, además del cero, por ello podemos nombrarlo un sistema binario. Este sistema cuenta con tres reglas fundamentales:

- El cero es un número natural.
- Si n es un número natural, el doble de n también es un número natural.
- Si n es un número natural, entonces uno mas que n es un número natural.

El objetivo es comprobar la equivalencia entre ambos sistemas, ya que ciertamente los dos se utilizan para representar a los números naturales de toda la vida. Será necesario el uso de Coq para la demostración de una baraja de propiedades y teoremas. Se decidió documentar cada teorema y función en un archivo separado, es recomendado leer ese archivo primero.

Sin duda pasamos por cierta dificultad al momento de buscar la solución de los ejercicios propuestos, tan es así, que en algún momento de la implementación del sistema binari0, decidimos darnos un largo descanso de 5 días. Nuestras mentes estaban completamente sumergidas en un muy hondo y apestoso pozo llamado Coq (tal vez eso es una exageración).

El principal desafío al que hicimos frente fue decidir qué versión utilizaríamos para nuestras funciones (tales como bin2nat, nat2bin, incremneta, etc.), debido a que contábamos con diversas opciones de implementación, y claro, nuestra decisión tendría consecuencias en los ejercicios que nuestras versiones del futuro enfrentarían. Esta fue la razón principal por la que resolvimos el ejercicio 10 de una manera diferente a la que se podría esperar, ya que la definición de nat2bin tenía que incluir el constrcutor Suc en su caso recursivo para que el ejercicio 7 pudiera ser resuelto de manera más simple y sencilla, en caso de que se incluyera incrementa en lugar de Suc, el ejercicio 10.2 y 10.3 se vuelven increíblemente sencillos, pero el 7 adquiere un nivel de dificultad tal que no pudimos resolverlo de nignuna manera. Por ello, resultó más conveniente definir una segunda versión de nat2bin para llegar a un equilibrio idóneo.

Por otro lado, los aprendizajes adquiridos fueron mayúsculos, tenemos claro que éstos fueron muchos más que los problemas que pudimos encontrarnos. Si bien ya veníamos con cierto nivel de entendimiento en Coq por la práctica 6, este proyecto duplicó nuestra habilidad para el manejo de cada táctica que pudimos y se nos permitió usar, como induction, simpl, reflexivity, destruct, e incluso omega ('lia' en versiones recientes del asistente de prueba).

Además, la investigación junto a los ejercicios nos evidenció que la verificación formal no es solo una tarea académica intrascendente, sino una práctica cítica que respalda la integridad y funcionalidad de los sistemas en múltiples dominios, desde criptogrfía o sistemas computacionales avanzados, hasta definiciones matemáticas más aterrizadas. Al seguir desarrollando y aplicando las técnicas, podemos continuar mejorando la confiabilidad de los sistemas y códigos, asegurando así un trabajo lógico más seguro y eficiente.

# Conclusión

La implementación a la que se llegó está en un nivel bastante decente de eficacia; no se usaron demasiadas funciones auxiliares, las demostraciones en su mayoría son cortas y sin usar demasiadas tácticas. No obstante, consideramos que el talón de aquiles de nuestra solución es el ejercicio 10 y todos sus incisos, esto debido a que tuvimos que recurrir a estrategias y demostraciones que podrían no ser las más idóneas si se ve el panorama completo, pero es algo de lo que estuvimos concientes en el momento que decidimos excluir la función incrementa de la definición del ejercicio 6 'nat2bin', con el fin de simplificar el ejercicio 7. Sin lugar a dudas, las propiedades 1, 2 y 3 del ejercicio 10 fueron un reto mayúsuclo en la realización del proyecto, pero consideramos que nuestra alternativa resulta válida pues demostramos propiedades que son equivalentes a éstas

En líneas generales, el proyecto fue clave para apoyarnos en el entendimiento y comprensión del asistente de prueba, además de que el propuesto sistema binario alternativo de los números naturales sí es equivalente a la representación que todos conocemos, con ambos sistemas compartiendo características clave entre sí.

# Referencias


1. Indian Academy of Sciences. (2009). Proof assistants: History, ideas and future. Sādhanā Vol. 34, Part 1.

2. Tang, K., Deng, J. (2019). Learning to Prove Theorems via Interacting with Proof Assistants. Cornell University.

3. Serna. E., Morales, D. (2014). Estado del arte de la investigación en verificación formal. Ingeniería Investigación y Tecnología.

4. Anand, A. (2016). TRUST IN PROOF ASSISTANTS : OPPORTUNITIES AND LIMITATIONS. [Tesis de titulación]. Facultad de la Escuela de Graduados de la Universidad de Cornell.

5. Sheard, T. (2005). Putting Curry-Howard to work. Artículo de revisión. Association for Computing Machinery Digital Library. https://dl.acm.org/doi/abs/10.1145/1088348.1088356

6. Instituto de Matemáticas de la UNAM. (2022, 30 mayo). Asistentes de prueba y verificación formal [Vídeo]. YouTube. https://www.youtube.com/watch?v=FRcsobHjyK4.

7. Ringa Tech. (2022, 28 octubre). El software que (A veces) te mataba con radiación [Vídeo]. YouTube. https://www.youtube.com/watch?v=v5mfyj0S2Ss.

8. Lewis, M., Soudjani, S. y Zuliani, P. (2023, 16 diciembre). Formal Verification of Quantum Programs: Theory, Tools, and Challenges. Artículo de revisión. Association for Computing Machinery Digital Library. https://dl.acm.org/doi/full/10.1145/3624483d1e1130.

9. Iberdrola S.A. (s.f.). Qué es la computación cuántica: La computación cuántica y las supercomputadoras que revolucionarán la tecnología. Iberdrola. https://www.iberdrola.com/innovacion/que-es-computacion-cuantica.

10. Artturi, J. (2023, 3 febrero). What Is a Compiler?. BuiltIn. https://builtin.com/software-engineering-perspectives/compiler

11. AbsInt. (2022). CompCert: Formally verified compilation. https://www.absint.com/compcert/index.htm

12. AbsInt Angewandte Informatik GmbH. (2021, 15 junio). Can you trust your compiler? — With CompCert, you can! [Video]. YouTube. https://www.youtube.com/watch?v=REFNUw2zMlw

13. Pablo. (2022, 5 diciembre). Ni en un millón de años: así funciona el cifrado AES. El Blog de Orange. Recuperado 28 de mayo de 2024, de https://blog.orange.es/navegacion-segura/cifrado-aes/

14. Villacís, C. (2018, 8 mayo). La paradoja de Russell. ACADEMIA PLAY! Recuperado 27 de mayo de 2024, de https://academiaplay.net/paradoja-russell/

15. HistoriaUniversal.org. (2024, 8 marzo). Biografía de Gottlob Frege. Recuperado 28 de mayo de 2024, de https://historiauniversal.org/gottlob-frege/

16. ACM. (2019). Robert W. Floyd - A.M. Turing Award laureate. https://amturing.acm.org/awardwinners/floyd3720707.cfm

17. Salinas Molina, M. A. (2011). Computabilidad y máquina de Turing (Tesis de maestría, Universidad Nacional Mayor de San Marcos, Facultad de Letras y Ciencias Humanas, Unidad de Post Grado). Lima, Perú.

18. (S/f). Pruebas Formales: descifrando el codigo, pruebas formales en asignaciones de POA. Fastercapital.com. Recuperado el 3 de junio de 2024, de https://fastercapital.com/es/contenido/Pruebas-formales–Descifrando-el-codigo–Pruebas-formales-en-asignaciones-de-POA.html

19. Leveson, N. (1993). Medical Devices: The Therac-25*.

20. Harrison, J. (2013). Formal verification.

21. Schnider, P. (2014). An Introduction to Proof Assistants.

22. La logica de Hoare es una extension de la logica de predicados de primer orden... A. C. un C. de R. de C. P. la M. (s/f). 2.2 Logica de Hoare. Uma.es. Recuperado el 3 de junio de 2024, de http://www.lcc.uma.es/ jmmb/ttaadd/ttaadd2-2.pdf

23. Kern, C., & Greenstreet, M. R. (1999). Formal verification in hardware design: A survey. ACM Transactions on Design Automation of Electronic Systems, 4(2), 123–193. https://doi.org/10.1145/307988.307989