Algebraic methods for the analysis of arithmetic circuits in zero-knowledge proofs

---

Métodos algebraicos para el análisis de circuitos aritméticos en pruebas de conocimiento nulo

Pedro Palacios Almendros

Doble Grado en Ingeniería Informática - Matemáticas

Facultad de Matemáticas

Universidad Complutense de Madrid

Trabajo de Fin de Grado
del Grado en Matemáticas

Madrid, 2022-2023

Directores:

Albert Rubio Gimeno

Clara Rodríguez Núñez

Abstract

In this work, an algebraic method has been developed to verify the safety of Circom circuits, a domain-specific language for defining arithmetic circuits for zero-knowledge proofs. The method is based on the reduction of the Circom circuit verification problem (for a fixed input or for any input) to the unsatisfiability of polynomial systems with coefficients in a finite field problem. Computational methods based on Gröbner bases have been studied to determine the unsatisfiability of such polynomial systems. A modular algorithm has also been implemented in order to deal with large circuits that otherwise would not be tractable using purely algebraic methods. Finally, the modular algorithm has been run on several Circom open source repositories with practical applicability, and the results have been analyzed.

Keywords

Circom, arithmetic circuit safety, zero-knowledge proof, Gröbner bases, finite fields, polynomial systems.

Resumen

En este trabajo se ha desarrollado un método algebraico para verificar la seguridad de circuitos Circom, un lenguaje de dominio específico para definir circuitos aritméticos para pruebas de conocimiento nulo. El método se basa en reducir el problema de la verificación de circuitos Circom (para una entrada fija o para cualquier entrada) al problema de insatisfacibilidad de sistemas polinómicos con coeficientes en un cuerpo finito. Se han estudiado métodos computacionales basados en bases de Gröbner para determinar la insatisfacibilidad de dichos sistemas polinómicos. También se ha creado un algoritmo modular para poder tratar circuitos grandes que de otra forma no serían abordables con métodos puramente algebraicos. Finalmente, se ha ejecutado el algoritmo modular sobre varios repositorios de código abierto escritos en Circom con aplicabilidad práctica, y se han analizado los resultados.

Palabras clave

Circom, seguridad de circuitos aritméticos, pruebas de conocimiento nulo, bases de Gröbner, cuerpos finitos, sistemas polinómicos.

Índice general

Capítulo I

Introducción

En el mundo actual, la criptografía tiene más importancia que nunca debido a sus aplicaciones en transacciones financieras, comunicaciones gubernamentales y encriptación de datos personales al acceder a páginas web.

La criptografía ha sido usada desde tiempos inmemoriales, pero su uso ha explotado desde la introducción de Internet. Las técnicas criptográficas usadas actualmente requieren conceptos matemáticos complejos: curvas elípticas, cuerpos finitos, geometría algebraica, etc.

Debido al rápido avance de la tecnología y la alta demanda de soluciones criptográficas en ámbitos muy diversos es necesario crear un puente entre los matemáticos que desarrollan la teoría y los informáticos que las implementan en sistemas reales.

Entre estos sistemas criptográficos, unos de los más útiles son las pruebas de conocimiento nulo, que permiten a un demostrador probar una afirmación a un verificador sin desvelar ningún dato más allá de la veracidad de la afirmación dada. En concreto las ZK-SNARKs permiten crear pruebas de conocimiento nulo eficientes a partir de circuitos aritméticos, cuyo funcionamiento se basa en sistemas polinómicos con coeficientes en cuerpos finitos. Debido a la alta complejidad del desarrollo de ZK-SNARKs con polinomios, lenguajes de dominio específico como Circom han sido creados para poder describir declarativamente un circuito aritmético y generar todos los artefactos necesarios para una prueba de conocimiento nulo, incluyendo el sistema polinómico.

Circom permite a informáticos con poca experiencia previa en criptografía diseñar circuitos aritméticos para generar pruebas de conocimiento nulo. Sin embargo, cierto conocimiento específico sigue siendo necesario para escribir circuitos aritméticos seguros, y algunos programadores inexpertos pueden fácilmente crear circuitos inseguros.

Debido al papel crucial de la criptografía, las consecuencias de los problemas de seguridad y el auge de los ciberataques, es más importante que nunca poder verificar la seguridad de protocolos criptográficos, y en concreto los basados en circuitos aritméticos diseñados con Circom.

I.1. Objetivos

La finalidad de este trabajo es desarrollar un método algebraico para verificar la seguridad de circuitos Circom y realizar una implementación práctica que permita su aplicación en los circuitos Circom de gran tamaño usados en la práctica. Para ello plantearemos los siguientes objetivos:

1. Estudiar métodos algebraicos para analizar la satisfacibilidad de sistemas de ecuaciones polinómicas y su aplicación en cuerpos finitos $\mathbb{F}_p$.

2. Aplicar dichas técnicas al problema práctico de la verificación de seguridad de circuitos Circom para pruebas de conocimiento nulo.
3. Diseñar un algoritmo modular utilizando heurísticas combinadas con técnicas algebraicas para poder verificar eficientemente circuitos Circom de gran tamaño.
4. Implementar dicho algoritmo modular en Rust de forma que pueda integrarse con el compilador de Circom.
5. Realizar una evaluación experimental de dicho algoritmo utilizando repositorios de código abierto usados en la práctica.

I.2. Estructura del trabajo

En el capítulo II se introducen los conceptos de pruebas de conocimiento nulo, circuitos aritméticos y Circom que utilizaremos en el resto del trabajo.

En el capítulo III se define la seguridad de un circuito Circom y se expone una nueva forma de reducir el problema de la seguridad de Circom a la satisfacibilidad de un sistema polinómico con coeficientes en un cuerpo finito $\mathbb{F}_p$.

En el capítulo IV se exponen técnicas algebraicas para determinar la insatisfacibilidad de un sistema de polinomios utilizando bases de Gröbner, centrándonos en cuerpos finitos.

En el capítulo V se diseña un algoritmo modular que combina heurísticas con los métodos algebraicos desarrollados anteriormente para poder tratar circuitos grandes de forma eficiente.

En el capítulo VI se realiza una evaluación del algoritmo modular implementado en el lenguaje Rust utilizando repositorios de código abierto escritos en Circom y se interpretan los resultados.

Finalmente, en el capítulo VII se enumeran todas las conclusiones a las que se ha llegado en el trabajo y se exploran algunas líneas de trabajo futuro.

En el apéndice A se incluye un enlace al repositorio con todo el código fuente en Rust del algoritmo modular implementado.

En el apéndice B se muestra una demostración del teorema de la base de Hilbert, necesaria para demostrar la terminación de cierto algoritmo del capítulo IV para computar bases de Gröbner.

En el apéndice C se estudian dos ejemplos de verificación de circuitos Circom.

En el apéndice D se estudian dos circuitos inseguros que se encontraron en la práctica utilizando el algoritmo modular de verificación desarrollado en este trabajo.

I.3. Plan de trabajo

Las actividades realizadas siguieron este orden: Primero, investigar sobre conceptos básicos de ZK-SNARKs y Circom. Segundo, investigar formas de convertir el problema de seguridad de Circom a un problema algebraico. Tercero, estudiar técnicas algebraicas basadas en bases de Gröbner para resolver dicho problema. Cuarto, diseñar un algoritmo modular e implementarlo en Rust. Finalmente, realizar experimentos de prueba.

Capítulo II

Conceptos básicos

Este capítulo es una introducción a los conceptos subyacentes a Circom, sus aplicaciones y sus herramientas básicas. Primero introducimos las pruebas de conocimiento nulo y los circuitos aritméticos, después estudiamos los ZK-SNARK y sistemas R1CS y finalmente exploramos el compilador Circom y algunas de sus características más relevantes para nuestro trabajo.

II.1. Pruebas de conocimiento nulo

Una prueba de conocimiento nulo [15] es un protocolo criptográfico que permite a una de las dos partes (el *demostrador*), probar una afirmación a la otra parte (el *verificador*) sin desvelar ningún otro dato más allá de la veracidad de la afirmación expuesta.

Estas pruebas son una herramienta muy importante en criptografía, como se puede ver en los siguientes ejemplos de aplicación:

- "Demostrar que se conoce la clave privada asociada a una clave pública". Esto permite a una persona autenticarse sin revelar su clave privada.
- "Demostrar que el salario de un solicitante de crédito está en cierto rango sin desvelar el valor exacto". Esto permite una mayor privacidad a la hora de realizar operaciones financieras, aún garantizando requisitos puestos por alguna de las dos partes. El banco ING implementó [22] en Ethereum [25] una prueba de concepto de esta tecnología.
- "Demostrar que el hash de un bloque de transacciones en el blockchain [26] no incurre en balances negativos". Esto permite demostrar la validez de las transacciones del bloque sin revelar los destinatarios, los remitentes o el valor de las transacciones contenidas en dicho bloque. La criptomoneda Monero [20] utiliza tecnología similar para garantizar la privacidad en sus transacciones.

Un tipo muy importante de pruebas de conocimiento nulo son las pruebas de conocimiento nulo no interactivas [9] (*NIZK* por sus siglas en inglés *Non-Interactive Zero-Knowledge proofs*), en las que el demostrador no interactúa con el verificador durante la generación de la prueba. Varios verificadores pueden, a posteriori, comprobar la veracidad de la afirmación sin necesidad de comunicarse con el demostrador.

Esta propiedad permite a los *smart-contracts* [27] (programas almacenados en el blockchain que se ejecutan en reacción a transacciones y permiten automatizar contratos) verificar las pruebas de conocimiento nulo, pues la ejecución de los smart-contracts está aislada del mundo exterior.

Algunas propiedades deseables en protocolos NIZK son las siguientes:

- Bajo coste computacional para generar la prueba.
- Pequeño tamaño de la prueba.
- Bajo coste computacional para verificar la prueba.

Los dos últimos factores son muy relevantes al utilizar smart-contracts, pues el tiempo empleado en la verificación de la prueba y el tamaño de la prueba cuesta recursos en el blockchain y dinero.

II.2. Circuitos aritméticos

Un tipo importante de afirmaciones computacionales que se pueden demostrar utilizando pruebas de conocimiento nulo es la satisfacibilidad de circuitos aritméticos [24]. En general trabajaremos en el cuerpo finito de p elementos $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$ con p primo, que es típicamente un número primo grande, de más de 250 bits. Para ilustrar la magnitud de estos primos, mostramos un ejemplo de un primo p utilizado asiduamente en Ethereum para operaciones con curvas elípticas

$$p = 21888242871839275222246405745257275088548364400416034343698204186575808495617$$

Intuitivamente, un circuito aritmético está formado por un conjunto de cables que toman valores en $\mathbb{F}_p$ y están conectados a puertas que realizan algún tipo de operación. Nosotros vamos a trabajar con dos posibles puertas: la suma de dos elementos de $\mathbb{F}_p$ y la multiplicación de dos elementos de $\mathbb{F}_p$, ambas realizadas módulo p.

A los cables del circuito se les llama señales, y pueden ser de tres tipos:

- **Entrada** (*input* en inglés): Son las entradas al circuito. Pueden ser públicas (las conoce tanto el demostrador como el verificador) o privadas (las conoce únicamente el demostrador). Representaremos cada entrada como i_j, y al vector de todas las entradas como $\bar{i}$.
- **Intermedia** (*intermediate* en inglés): Son las salidas de las puertas intermedias del circuito. Asumiremos que son siempre privadas, el verificador no las conoce. Representaremos cada intermedia como x_j y al vector de todas las intermedias como $\bar{x}$.
- **Salida** (*output* en inglés): Son las salidas del circuito. Nosotros asumiremos que siempre son públicas, es decir, que tanto el demostrador como el verificador conocen el valor de las salidas. Representaremos cada salida como o_j y al vector de todas las salidas como $\bar{o}$.

El demostrador ejecutará el circuito aritmético usando los inputs tanto públicos como privados, y encontrará una asignación para cada señal que haga que se satisfaga el circuito. A este conjunto de asignaciones se le llama *testigo* o *witness*. El demostrador publicará el output (que es público), las inputs públicas (pero no las privadas) y una prueba (idealmente pequeña en tamaño) de que conoce un testigo que satisfaga el circuito. Dicha prueba no deberá revelar las asignaciones a las variables privadas. En la sección II.3 veremos algunas herramientas que nos permiten realizar estas tareas, pero antes estudiemos un circuito de ejemplo.

En la figura II.1 podemos ver un circuito de ejemplo, en el que las entradas son $\{i_0, i_1, i_2, i_3\}$, las señales intermedias son $\{x_0, x_1\}$ y la salida es o_0.

En este caso un testigo sería una tupla $(i_0, i_1, i_2, i_3, x_0, x_1, o_0)$ tal que el circuito se satisface. Si asumimos que las entradas públicas son $\{i_0, i_2\}$, estamos trabajando en $\mathbb{F}_7$, y las señales públicas

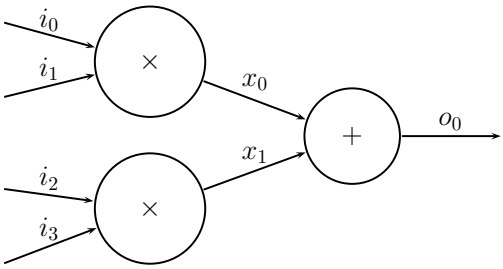

Figura II.1: Circuito aritmético de ejemplo

toman los valores $i_0 = 2, i_2 = 3, o_0 = 6$, un testigo para el circuito sería $(i_0 = 2, i_1 = 5, i_2 = 3, i_3 = 1, x_0 = 3, x_1 = 3, o_0 = 6)$.

II.3. R1CS y ZK-SNARKs

Intentemos buscar una representación algebraica del circuito aritmético de la figura II.1. Podemos expresar el circuito como un sistema de ecuaciones que deben cumplir las diferentes señales. El circuito de la figura II.1 se transformaría en el sistema de ecuaciones II.1.

$$\begin{cases} x_0 = i_0 \cdot i_1 \\ x_1 = i_2 \cdot i_3 \\ o_0 = x_0 + x_1 \end{cases} \tag{II.1}$$

De hecho, podemos transformar todos los circuitos en un sistema de ecuaciones polinómicas, añadiendo una variable por cada señal del circuito y una restricción por cada operación del circuito. Como únicamente tenemos operaciones de multiplicación y de suma, las restricciones serán a lo sumo polinomios de grado 2.

Podemos generalizar ligeramente estos sistemas de ecuaciones, definiendo de paso la notación que usaremos.

Definición II.1. *(**Señal**) Llamaremos señal a una variable que toma valores en $\mathbb{F}_p$.*

Definición II.2. *(**Restricción**) Dadas las señales $\bar{s} = (s_1, \ldots, s_n) \in \mathbb{F}_p^n$ llamamos restricción a una ecuación de la siguiente forma:*

$$(a_1 s_1 + ... + a_n s_n) \cdot (b_1 s_1 + ... + b_n s_n) + (c_1 s_1 + ... + c_n s_n) = 0$$

donde a_j, b_j, c_j son constantes en $\mathbb{F}_p$. De forma abreviada, si denotamos el producto escalar de $\bar{u}, \bar{v} \in \mathbb{F}_p^n$ como $\langle \bar{u}, \bar{v} \rangle = u_1 v_1 + \cdots + u_n v_n$, entonces podemos expresar la restricción como:

$$\langle \bar{a}, \bar{s} \rangle \cdot \langle \bar{b}, \bar{s} \rangle + \langle \bar{c}, \bar{s} \rangle = 0$$

donde $\bar{a}, \bar{b}, \bar{c}$ son constantes en $\mathbb{F}_p^n$.

Normalmente por convención se asume que la señal s_1 toma siempre el valor constantemente 1 en vez de ser una variable libre, para poder tener términos independientes en las restricciones y poder expresar restricciones como $x = 2$.

Como se deduce directamente la definición, todas las restricciones serán cuadráticas, lineales o constantes.

Definición II.3. *(**R1CS**) Llamaremos lista de restricciones, Rank 1 Constraint System (R1CS) o simplemente restricciones al sistema de ecuaciones formado por varias restricciones sobre las mismas señales $\bar{s} = (s_1, \ldots, s_n) \in \mathbb{F}_p^n$, que se tienen que satisfacer simultáneamente. Dichos sistemas de ecuaciones tienen la siguiente forma:*

$$\begin{cases} (a_1^1 s_1 + ... + a_n^1 s_n) \cdot (b_1^1 s_1 + ... + b_n^1 s_n) + (c_1^1 s_1 + ... + c_n^1 s_n) = 0 \\ (a_1^2 s_1 + ... + a_n^2 s_n) \cdot (b_1^2 s_1 + ... + b_n^2 s_n) + (c_1^2 s_1 + ... + c_n^2 s_n) = 0 \\ \vdots \\ (a_1^k s_1 + ... + a_n^k s_n) \cdot (b_1^k s_1 + ... + b_n^k s_n) + (c_1^k s_1 + ... + c_n^k s_n) = 0 \end{cases}$$

donde a_j^i, b_j^i, c_j^i son constantes en $\mathbb{F}_p$. Nótese que el superíndice no expresa potencia ni multiplicación, sino que denota constantes posiblemente distintas. De forma abreviada podemos expresar el sistema como

$$\begin{cases} \langle \bar{a}^1, \bar{s} \rangle \cdot \langle \bar{b}^1, \bar{s} \rangle + \langle \bar{c}^1, \bar{s} \rangle = 0 \\ \langle \bar{a}^2, \bar{s} \rangle \cdot \langle \bar{b}^2, \bar{s} \rangle + \langle \bar{c}^2, \bar{s} \rangle = 0 \\ \vdots \\ \langle \bar{a}^k, \bar{s} \rangle \cdot \langle \bar{b}^k, \bar{s} \rangle + \langle \bar{c}^k, \bar{s} \rangle = 0 \end{cases}$$

donde $\bar{a}^i, \bar{b}^i, \bar{c}^i$ son constantes en $\mathbb{F}_p^n$.

Definición II.4. *(**Witness**) Dado un sistema R1CS con señales $\bar{s} = (s_1, \ldots, s_n) \in \mathbb{F}_p^n$, decimos que una tupla $\bar{w} \in \mathbb{F}_p^n$ es un testigo o witness del sistema si al sustituir en el sistema las señales $\bar{s}$ por $\bar{w}$, el sistema de ecuaciones se satisface.*

Definición II.5. *(**Prueba de conocimiento nulo asociada a un R1CS**) Dado un sistema R1CS con señales $\bar{s} = (s_1, \ldots, s_n) \in \mathbb{F}_p^n$, algunas de ellas públicas y otras privadas, una prueba de conocimiento nulo asociada al R1CS es aquella que permite al demostrador probar la posesión de un testigo que satisface el sistema R1CS para ciertos valores de las señales de entrada, publicando únicamente el valor de las señales públicas y una prueba (idealmente pequeña), sin revelar información sobre las señales privadas.*

Expresar un circuito aritmético como R1CS nos aporta varias ventajas. En primer lugar, nos permite optimizar y simplificar las restricciones mediante el uso de operaciones algebraicas. En segundo lugar, nos permite expresar ciertas operaciones como inversos, condicionales o pertenencia de un punto a una curva elíptica de una forma mucho más sencilla añadiendo restricciones extra *a mano*, como veremos en la sección II.4.

Otra gran ventaja es que existen herramientas que a partir de un sistema R1CS, una lista de las señales públicas y un programa para calcular un testigo dadas las señales de entrada, permiten generar automáticamente el código para el demostrador y verificador de la prueba de conocimiento nulo asociado a dicho sistema R1CS.

Uno de dichos protocolos criptográficos es ZK-SNARK [16] (Zero Knowledge Succint Non-interactive Arguments of Knowledge). Los protocolos ZK-SNARK tienen las siguientes propiedades clave:

- **Concisión**. Las pruebas generadas por ZK-SNARKs suelen ser muy pequeñas, pesando alrededor de 200 bytes [17] independientemente del número de restricciones.
- **Verificación eficiente**. Las pruebas generadas se pueden verificar eficientemente, haciéndolas aptas para smart-contracts.

- **No interactividad**. El verificador no tiene que interactuar con el demostrador.
- **Probabilista**. Para garantizar las propiedades anteriores, se realiza una verificación probabilista. Con muy alta probabilidad, si la prueba es verificada de forma positiva, esta será válida.

Como la satisfacibilidad de circuitos en $\mathbb{F}_p$ es un problema NP-completo, se puede generar una prueba de conocimiento nulo asociada a un R1CS para cualquier problema en NP.

Se pueden generar pruebas ZK-SNARK a partir de cualquier R1CS satisfacible. De hecho, existen implementaciones como SnarkJS [19] que dada una lista de restricciones R1CS y un programa para calcular un testigo generan el código para el demostrador y el verificador (generando incluso código Solidity [28] para poder verificar en un smart-contract).

II.4. Circom

Hemos visto que se pueden generar pruebas ZK-SNARK a partir de cualquier R1CS satisfacible, y por tanto a partir de cualquier circuito aritmético. Sin embargo, generar las restricciones a mano es muy tedioso y puede llevar a errores.

Existen herramientas como Circom [7] que permiten generar un sistema R1CS a partir de la compilación de un lenguaje de dominio específico para describir circuitos aritméticos. Circom también permite añadir restricciones extra (no las directamente generadas por un circuito aritmético) al sistema R1CS, que como veremos más adelante, será muy útil.

Observación II.6. *(**Circuitos**). A los programas escritos en el lenguaje Circom se les denomina "circuitos", que se pueden confundir con los circuitos aritméticos. No son equivalentes, pues los programas Circom permiten añadir restricciones extra. A lo largo de este texto nos referiremos como circuito a los programas Circom, y si tenemos que referirnos a los circuitos aritméticos, siempre les pondremos el apellido "aritméticos".*

Circom no solo produce una lista R1CS a partir de un circuito Circom, sino que también genera un programa en WebAssembly [18] (un formato de instrucciones binario que se puede ejecutar en el navegador) o C++ para calcular eficientemente un testigo para el R1CS y genera una lista clasificando las señales en inputs, intermedias y outputs y en públicas y privadas. Es decir, Circom genera todos los artefactos necesarios para generar un verificador y demostrador para una prueba ZK-SNARK.

Pasamos ahora a realizar una breve introducción a las herramientas que nos aporta Circom. Una exposición más en detalle de Circom se puede encontrar en [7].

Circuitos Circom básicos

Circom permite describir circuitos aritméticos (y generar restricciones extra) de una forma modular. Permite crear módulos que realizan una cierta función, llamados `templates`, que se pueden instanciar varias veces para reutilizar código. Circom también permite expresar simultáneamente operaciones en el programa para generar los testigos y en la lista de restricciones para la prueba ZK-SNARK. Para ello, se utiliza la asignación dual entre señales `<==`, que genera una asignación a la variable asignada en el código WebAssembly generado, y genera la restricción asociada a dicha asignación en el R1CS. Por tanto, el lado derecho de la asignación

puede ser a lo sumo una expresión cuadrática sobre las señales. Además, todas las señales del lado derecho de la asignación deben haber sido asignadas previamente, para poder generar la asignación en WebAssembly. Más aún, las señales son inmutables, por lo que Circom generará un error si se intenta asignar dos veces a una misma señal. En el código II.1 se puede observar una implementación en Circom del circuito aritmético mostrado en la figura II.1.

Es necesario declarar qué módulo forma el componente principal que se usará en la prueba, y esa es la función de la línea 14. Además, incluye una lista de las señales de entrada públicas, el resto de señales de entrada se considerarán privadas, todas las intermedias son privadas y las señales de salida son todas públicas.

Código II.1 Implementación en Circom del circuito de la figura II.1

```
1  pragma circom 2.0.0;
2
3  template Circ(){
4     signal input in[4];
5     signal x0;
6     signal x1;
7     signal output out;
8
9     x0 <== in[0] * in[1];
10    x1 <== in[2] * in[3];
11    out <== x0 + x1;
12 }
13
14 component main {public [in]} = Circ();
```

Incluso con circuitos sencillos se puede apreciar la utilidad de añadir las restricciones extra a la lista R1CS antes mencionada. Por ejemplo, en el código II.2 implementamos el módulo `IsBinary`, que garantiza que una señal sea binaria insertando una restricción, y devuelve el valor de la entrada. Para ello utilizamos el operador `===`, que añade una restricción a la lista R1CS, por lo que debe ser una restricción a lo sumo cuadrática. En el código WebAssembly generado, la restricción se traduce en una aserción. El programa comprueba si la restricción se cumple con los valores de la señal. Si no se cumple, se aborta la generación del testigo y se señala un error.

Código II.2 Implementación del módulo `IsBinary`

```
1  pragma circom 2.0.0;
2
3  template IsBinary(){
4      signal input in;
5      signal output out;
6
7      (in-1)*in === 0;
8      out <== in;
9  }
10
11 component main {public [in]} = IsBinary();
```

Componentes y templates

Combinando módulos simples podemos conseguir circuitos complejos de forma estructurada. En Circom también es posible crear módulos *genéricos*, es decir, módulos que pueden tomar

ciertos parámetros en tiempo de instanciación. A un módulo instanciado se le llama `component`. En el código II.3 se puede observar cómo utilizar módulos genéricos para crear un circuito que multiplique matrices de cualquier tamaño.

Código II.3 Implementación de un multiplicador de matrices

```
1  pragma circom 2.0.0;
2
3  template DotProduct(n) {
4      signal input a[n];
5      signal input b[n];
6      signal output o;
7
8      signal acul[n];
9      var acc = 0;
10
11     for(var i = 0; i < n; i++) {
12         acul[i] <== a[i] * b[i];
13         acc += acul[i];
14     }
15     o <== acc;
16 }
17
18 template MatMult(m, n, k) {
19     signal input A[m][n];
20     signal input B[n][k];
21     signal output O[m][k];
22
23     component dp[m][k];
24
25     for(var i = 0; i < m; i++) {
26         for (var j = 0; j < k; j++) {
27             dp[i][j] = DotProduct(n);
28             for (var k = 0; k < n; k++) {
29                 dp[i][j].a[k] <== A[i][k];
30                 dp[i][j].b[k] <== B[k][j];
31             }
32             O[i][j] <== dp[i][j].o;
33         }
34     }
35 }
36
37 component main { public [A, B]} = MatMult(2, 3, 4);
```

Cada módulo genérico se convierte en un módulo concreto para cada valor distinto que toman los parámetros genéricos, por lo que podemos reducir nuestro análisis a módulos sin genéricos.

En la línea 23 del código II.3 podemos ver el uso de *arrays* de componentes, que se pueden instanciar más adelante, tal y como hacemos en la línea 27.

En la línea 9 también podemos observar el uso de variables `var`. En la traducción a WebAssembly, se comportan como variables mutables estándar, pero en la generación de restricciones almacenan una expresión simbólica en términos de señales o una constante. Esto nos permite en la línea 15 asignar a `o` la suma de todos los `acul[i]`, generando la restricción correspondiente en el R1CS.

Las condiciones tanto de los bucles como de los condicionales `if` solo pueden depender de

variables conocidas en tiempo de compilación, es decir, no pueden depender de señales ni de variables que dependan de señales, pues las restricciones han de ser generadas en tiempo de compilación, donde no se conocen los valores de las señales de entrada.

La compilación de un circuito Circom es la generación de todos los artefactos necesarios para las siguientes etapas de la generación de pruebas ZK-SNARK, es decir, el sistema de restricciones R1CS, el código WebAssembly para generación de witness y la lista de señales públicas.

Diferencias entre restricciones y generación de witness

Hasta ahora todos nuestros circuitos asignaban a las señales mediante la asignación dual `<==`, que genera tanto una asignación en el código WebAssembly como una restricción en la lista R1CS. Sin embargo, en ciertas situaciones, es necesario separar la computación en WebAssembly de las restricciones que representa el circuito. Uno de los ejemplos clave es la implementación del módulo `IsZero` en la librería estándar de Circom, CircomLib [2], que está mostrada en el código II.4.

Este módulo tiene una única entrada y una única salida. La salida será 1 cuando la entrada sea 0 y 0 cuando la entrada sea no nula.

Código II.4 Implementación de `IsZero`

```
1 pragma circom 2.0.0;
2
3 template IsZero() {
4     signal input in;
5     signal output out;
6
7     signal inv;
8
9     inv <-- in!=0 ? 1/in : 0;
10    out <== -in*inv +1;
11
12    in*out === 0;
13 }
```

En la línea 9 utilizamos por primera vez el operador inseguro `<--`, que únicamente genera una asignación en el código WebAssembly, pero no genera ninguna restricción. No podemos utilizar el operador dual `<==`, pues a la derecha de la asignación utilizamos una expresión no cuadrática (un condicional). Vemos que en el código WebAssembly, las líneas 9 y 10 ya determinan correctamente la salida, pues la línea 9 asigna a `inv` 0 o el inverso de `in` y la línea 10 asigna a `out` 1 si $\texttt{in} = \texttt{inv} = 0$ y $-\texttt{in} \cdot \texttt{in}^{-1} + 1 = 0$ en caso contrario.

Más interesante es el comportamiento del circuito en la generación de restricciones, que también debe ser correcta, pues es la base que se usa para generar la prueba. La línea 9 no tiene ningún efecto en la generación de restricciones, por lo que la señal `inv` no tiene ningún valor fijado y podría tomar cualquier valor. Para solucionarlo, se añade la restricción de la línea 12. Dicha línea garantiza que si $\texttt{in} \neq 0$, entonces $\texttt{out} = 0$ y tomando $\texttt{inv} = \texttt{in}^{-1}$ se satisface la restricción de la línea 10. Si $\texttt{in} = 0$, entonces por la línea 10, $\texttt{out} = -0 \cdot \texttt{inv} + 1 = 1$, como queríamos.

Una nota importante es que si $\texttt{in} = 0$, entonces la señal `inv` podría tomar cualquier valor y seguiría satisfaciendo todas las restricciones de la lista R1CS, pero no supone mayor problema, pues es una variable intermedia que no saldrá del módulo.

Capítulo III

Verificación de seguridad

En este capítulo primero motivamos la importancia de la seguridad en circuitos Circom, después definimos formalmente nuestro concepto de seguridad e introducimos un nuevo resultado: una reducción del problema de la seguridad de los circuitos Circom a la insatisfacibilidad de un cierto sistema de ecuaciones, problema que podemos resolver computacionalmente como veremos en el capítulo IV.

III.1. Motivación para la seguridad

Como hemos visto en el capítulo anterior, Circom permite disociar la generación de código WebAssembly de la generación de restricciones mediante el uso del operador `<--`, lo cual conlleva graves riesgos, pues puede provocar que la lista de restricciones R1CS (que no olvidemos es la usada para generar el demostrador y el verificador ZK-SNARK) esté subrestringida, es decir, que el testigo generado no sea la única solución válida del sistema de restricciones.

Esto puede tener graves consecuencias: supongamos que cierto módulo tiene como función verificar que se conoce la clave privada asociada a una clave pública para autorizar un pago y su salida es una señal binaria que indica si la entrada privada i_0 es la clave privada asociada a la entrada pública i_1, que representa la clave pública. Si el sistema de restricciones está subrestringido, es posible que existan varias soluciones (testigos) para el R1CS con la misma entrada. Es decir, si la entrada es una clave privada inválida, que no es la asociada a la clave pública, aparte de existir un testigo cuya salida indica que la clave privada es inválida, podría existir también otro testigo (indeseado) que afirma erróneamente que dicha clave privada es válida. Esto permitiría a un posible atacante autorizar el pago mediante una "clave privada" falsa que el verificador acepta como válida, a pesar de no ser la clave privada correcta.

Más aún, es sencillo que a un programador se le olvide declarar una restricción necesaria, sin darse cuenta generando un sistema subrestringido. El paradigma de las restricciones polinómicas es radicalmente distinto al paradigma algorítmico iterativo al que están acostumbrados los programadores, acentuando aún más este problema de seguridad. Por ejemplo, en el apéndice D se puede ver un ejemplo de un módulo inseguro de la librería estándar de Circom, que el algoritmo modular introducido en el capítulo V es capaz de detectar.

III.2. Definición de seguridad

El objetivo último de este trabajo es hallar formas automáticas de probar que un circuito Circom es seguro, para lo que es necesario tener una definición formal de la seguridad. Seguiremos las definiciones sobre circuitos Circom y sus propiedades dadas en [7].

Definición III.1. ***(Sistema de ecuaciones polinómico)*** *Llamaremos sistema de ecuaciones polinómico o simplemente sistema de ecuaciones a un subconjunto finito de $\mathbb{F}_p[\bar{s}] = \mathbb{F}_p[s_1, \ldots, s_n]$, el conjunto de polinomios sobre las variables $\bar{s}$. Representaremos el conjunto de todos los sistemas de ecuaciones polinómicos sobre las variables $\bar{s}$ como $\mathfrak{S}(\mathbb{F}_p[\bar{s}])$. Diremos que una constante $\bar{s}^0$ satisface o es solución de un sistema de ecuaciones $\mathcal{S}$ si $p(\bar{s}^0) = 0, \forall p \in \mathcal{S}$.*

Observación III.2. ***(Restricción vs. ecuación)*** *Para nosotros, las restricciones siempre serán cuadráticas, lineales o constantes, mientras que las ecuaciones polinómicas pueden ser polinomios de cualquier grado.*

Definición III.3. ***(Circuito Circom)*** *Llamamos $\mathfrak{C}^{n\times t\times m}$ al conjunto de circuitos Circom válidos que se pueden crear con n señales de entrada, t señales intermedias y m señales de salida. Llamamos $W : \mathfrak{C}^{n\times t\times m} \times \mathbb{F}_p^n \to \mathbb{F}_p^t \times \mathbb{F}_p^m$ a la función parcial que dado un circuito y sus variables de entrada, nos devuelve las variables intermedias y de salida que genera el programa de cómputo de witness en WebAssembly generado por Circom, es decir, $W(C, \bar{i}) = (\bar{x}, \bar{o})$. Dado un circuito $C \in \mathfrak{C}^{n\times t\times m}$, llamamos $\mathcal{C}(C) \in \mathfrak{S}(\mathbb{F}_p[\bar{i}, \bar{x}, \bar{o}])$ al sistema de ecuaciones polinómicas sobre las señales del circuito generado por Circom (las restricciones de la lista R1CS) cuyas variables son las entradas $\bar{i}$, intermedias $\bar{x}$ y salidas $\bar{o}$. Como las restricciones son ecuaciones polinómicas a lo sumo cuadráticas, en concreto son ecuaciones polinómicas.*

Es importante notar que W es una función parcial, pues no todos los circuitos Circom tienen un witness para todas las entradas, puede haber restricciones contradictorias entre sí o alguna condición adicional necesaria sobre las entradas para que estas sean válidas. Estos problemas son detectados por el código de generación de witness, pues todas las restricciones se convierten en aserciones en WebAssembly, y se aborta el proceso de generación de la prueba. Por ello, podemos restringirnos únicamente a las entradas que generen un witness válido.

Definición III.4. ***(Circuito Circom correcto)*** *Decimos que un circuito Circom $C \in \mathfrak{C}^{n\times t\times m}$ es correcto si para cualquier entrada $\bar{i}^0 \in \mathbb{F}_p^n$ tal que $\mathcal{C}(C)$ es satisfacible sustituyendo las entradas $\bar{i}$ por $\bar{i}^0$, existe la imagen de $(C, \bar{i}^0)$ a través de la función W y llamando $W(C, \bar{i}^0) = (\bar{x}_w, \bar{o}_w)$, entonces $(\bar{i}^0, \bar{x}_w, \bar{o}_w)$ es solución de $\mathcal{C}(C)$.*

A partir de ahora, salvo que digamos lo contrario, asumiremos que los programas Circom con los que tratamos son correctos. En [7] se introduce la noción de circuito Circom seguro y fuertemente seguro para todas las entradas. En el capítulo V desarrollaremos un algoritmo para probar que un circuito es seguro para una cierta entrada fija, por lo que extendemos la definición de seguridad para reflejar esto.

Los programas Circom fuertemente seguros son aquellos en los que la entrada fija las señales intermedias y las salidas, y los programas Circom seguros son aquellos en los que la entrada fija las salidas, pero las señales intermedias pueden variar. Más formalmente:

Definición III.5. . ***(Programa Circom seguro)*** *Decimos que un programa Circom correcto $C \in \mathfrak{C}^{n\times t\times m}$ es* ***seguro*** *para cierta entrada fija $\bar{i}^0 \in \mathbb{F}_p^n$ tal que existe $W(C, \bar{i}^0)$ si cuando llamamos $(\bar{x}_w, \bar{o}_w) \coloneqq W(C, \bar{i}^0) \in \mathbb{F}_p^{t\times m}$, entonces todos los vectores de señales en $\{\bar{i}^0\} \times \mathbb{F}_p^{t\times m}$ que satisfacen todas las restricciones de $\mathcal{C}(C)$ son de la forma $(\bar{i}^0, \bar{x}', \bar{o}_w)$. De igual forma, decimos que es* ***fuertemente seguro*** *para cierta entrada fija $\bar{i}^0 \in \mathbb{F}_p^n$ tal que existe $W(C, \bar{i}^0) = (\bar{x}_w, \bar{o}_w)$ si el único vector de $\{\bar{i}^0\} \times \mathbb{F}_p^{t\times m}$ que satisface todas las restricciones de $\mathcal{C}(C)$ es $(\bar{i}^0, \bar{x}_w, \bar{o}_w)$. Si un circuito es seguro (respectivamente fuertemente seguro) para toda entrada $\bar{i} \in \mathbb{F}_p^n$ tal que existe $W(C, \bar{i})$, diremos que el circuito es seguro (respectivamente fuertemente seguro).*

Un ejemplo de módulo seguro pero no fuertemente seguro es `IsZero`, definido en el código II.4 como veremos en el ejemplo III.11.

III.3. De la seguridad a la satisfacibilidad de ecuaciones

En esta sección introducimos el primer resultado novedoso de este trabajo. Para poder desarrollar algoritmos para verificar automáticamente la seguridad de un circuito, reduciremos el problema de la seguridad a la insatisfacibilidad de un sistema de ecuaciones, que podemos analizar con técnicas algebraicas que estudiaremos en el capítulo IV.

Para ello, dado un circuito Circom C, vamos a hallar un sistema de ecuaciones polinómicas $\mathcal{S}$, tal que si el sistema $\mathcal{S}$ no tiene ninguna solución, entonces podamos garantizar alguna propiedad de seguridad de C. Para hallar dicho sistema $\mathcal{S}$, primero necesitamos definir algunos conceptos.

Definición III.6. ***(Sustitución)*** *Dado un sistema de ecuaciones polinómicas $\mathcal{S} \in \mathfrak{S}(\mathbb{F}_p[\bar{s}, \bar{x}])$, un vector de variables $\bar{s} \in \mathbb{F}_p^n$ y otro vector de variables $\bar{s}' \in \mathbb{F}_p^n$, definimos*

$$\mathcal{S}' = \mathcal{S}[\bar{s} \to \bar{s}'] = \mathcal{S}[s_1 \to s_1', \ldots, s_n \to s_n'] \in \mathfrak{S}(\mathbb{F}_p[\bar{s}', \bar{x}])$$

como el sistema de ecuaciones resultante de sustituir simultáneamente en el sistema $\mathcal{S}$ todas las apariciones de las variables $\bar{s}$ por $\bar{s}'$. Análogamente definimos la sustitución de variables por constantes, dejando de ser dichos símbolos variables a resolver y pasando a ser constantes fijas.

El primer paso para verificar la seguridad (que consiste en la no existencia de dos soluciones *distintas* con la misma entrada) es codificar que dos vectores de señales son distintos utilizando una ecuación polinómica. Veamos como hacerlo añadiendo unas variables auxiliares $\bar{u}$ que tendrán como función poder ser el inverso multiplicativo de cualquier número no nulo.

Definición y Teorema III.7. ***(Prohibición)*** *Dado un vector de variables $\bar{x} \in \mathbb{F}_p^n$ y otro vector de variables $\bar{y} \in \mathbb{F}_p^n$, definimos el sistema $\mathcal{P}(\bar{x}, \bar{y}) \in \mathfrak{S}(\mathbb{F}_p[\bar{x}, \bar{y}, \bar{u}])$:*

$$\mathcal{P}(\bar{x}, \bar{y}) = \{((x_1 - y_1) \cdot u_1 - 1) \cdot ((x_2 - y_2) \cdot u_2 - 1) \cdots ((x_n - y_n) \cdot u_n - 1) = 0\}$$

Entonces:

1. *Si $(\bar{x}', \bar{y}', \bar{u}')$ es solución de $\mathcal{P}(\bar{x}, \bar{y})$, entonces $\bar{x}' \neq \bar{y}'$, donde la desigualdad es en sentido vectorial, es decir, existe algún índice i tal que $x_i' \neq y_i'$.*
2. *Dadas dos $\bar{x}', \bar{y}' \in \mathbb{F}_p^n$ tal que $\bar{x}' \neq \bar{y}'$, entonces $\exists \bar{u}'$ tal que $(\bar{x}', \bar{y}', \bar{u}')$ es solución de $\mathcal{P}(\bar{x}, \bar{y})$.*

Demostración.

1. Veamos que existe un i tal que $x_i' \neq y_i'$. Supongamos que no ocurre, por lo que para todo i se da que $x_i' - y_i' = 0$ y al ser $(\bar{x}', \bar{y}', \bar{u}')$ solución de $P(\bar{x}, \bar{y})$, tenemos que $(0 \cdot u_1' - 1) \cdot (0 \cdot u_2' - 1) \cdots (0 \cdot u_n' - 1) = 0 \iff (-1)^n = 0$, llegando a una contradicción.
2. Supongamos que existe un i tal que $x_i' \neq y_i'$, por lo que $x_i' - y_i' \neq 0$ y por ser $\mathbb{F}_p$ un cuerpo, $\exists u_i' := (x_i' - y_i')^{-1}$. Finalmente, tenemos que $(x_i' - y_i')\, u_i - 1 = 1 - 1 = 0$, por lo que $(\bar{x}', \bar{y}', \bar{u}')$ es solución de $P(\bar{x}, \bar{y})$, pudiendo elegir valores arbitrarios para u_j para $j \neq i$. □

Abusaremos de la notación y permitiremos crear sistemas de ecuaciones prohibición $\mathcal{P}(\bar{x}, \bar{x}^0)$ donde $\bar{x}^0$ sea un vector de constantes en vez de variables, que tiene la misma definición y propiedades que las vistas en el teorema III.7.

Ahora estamos en condiciones de hallar los sistemas polinómicos $\mathcal{S}$ tal que su satisfacibilidad es equivalente a alguna propiedad de seguridad de un circuito Circom.

En el caso de la seguridad para cierta entrada fija añadiremos a las restricciones del circuito Circom (en las que sustituimos las entradas por la entrada fija dada) la restricción prohibición que garantice que alguna salida (o en el caso de seguridad fuerte, también alguna intermedia) sea distinta a su valor en el testigo producido por el código WebAssembly.

Teorema III.8. ***(Seguridad para cierta entrada)*** *Dado un circuito Circom correcto* $C \in \mathfrak{C}^{n\times t\times m}$ *con* $\mathcal{C}(C) \in \mathfrak{S}(\mathbb{F}_p[\bar{i}, \bar{x}, \bar{o}])$ *y cierta entrada fija* $\bar{i}^0 \in \mathbb{F}_p^n$ *tal que* $W(C, \bar{i}^0) = (\bar{x}_w, \bar{o}_w)$*, entonces:*

1. *C es **seguro** para $\bar{i}^0$ si y solo si $\mathcal{S}$ no tiene solución, donde*

$$\mathcal{S} = \mathcal{C}(C)[\bar{i} \to \bar{i}^0] \cup \mathcal{P}(\bar{o}, \bar{o}_w) \in \mathfrak{S}(\mathbb{F}_p[\bar{x}, \bar{o}, \bar{u}])$$

2. *C es **fuertemente seguro** para $\bar{i}^0$ si y solo si $\mathcal{S}$ no tiene solución, donde*

$$\mathcal{S} = \mathcal{C}(C)[\bar{i} \to \bar{i}^0] \cup \mathcal{P}((\bar{x}, \bar{o}), (\bar{x}_w, \bar{o}_w)) \in \mathfrak{S}(\mathbb{F}_p[\bar{x}, \bar{o}, \bar{u}])$$

Demostración. Demostramos únicamente la primera afirmación, al ser la segunda demostración análoga pero ampliando la lista de variables prohibidas de $\mathcal{P}(\bar{o}, \bar{o}_w)$ a $\mathcal{P}((\bar{x}, \bar{o}), (\bar{x}_w, \bar{o}_w))$.

Probemos que $\mathcal{S}$ tiene solución $\iff$ C no es seguro:

- $\Rightarrow$ Supongamos que $\mathcal{S}$ tiene una solución $(\bar{x}', \bar{o}', \bar{u}')$, lo que por construcción indica que $(\bar{i}^0, \bar{x}', \bar{o}')$ es solución de $\mathcal{C}(C)$ y por tanto un witness válido del circuito. Sin embargo, $(\bar{o}', \bar{u}')$ también es solución de $\mathcal{P}(\bar{o}, \bar{o}_w)$, por lo que por el teorema III.7, $\bar{o}' \neq \bar{o}_w$, por lo que C no es seguro.
- $\Leftarrow$ Supongamos que C no es seguro para $\bar{i}^0$, por lo que existe $(\bar{i}^0, \bar{x}', \bar{o}')$ que satisface $\mathcal{C}(C)$ y cumple $\bar{o}' \neq \bar{o}_w$. Pero por el teorema III.7, esta última condición nos indica que $\exists \bar{u}'$ tal que $(\bar{o}', \bar{u}')$ es solución de $\mathcal{P}(\bar{o}, \bar{o}_w)$. Por tanto, $(\bar{x}', \bar{o}', \bar{u}')$ es una solución de $\mathcal{S}$. □

Encontramos ahora una caracterización de una propiedad de seguridad más fuerte, que un circuito Circom sea seguro para todas las entradas, no solo para una. Para ello, añadiremos dos *copias* de las restricciones del circuito Circom, cada una de ellas con sus variables intermedias y salidas distintas, pero compartiendo las mismas entradas. Después, utilizando el polinomio prohibición, garantizaremos que alguna salida (o también alguna variable intermedia en el caso de seguridad fuerte) difiera entre las dos copias.

Teorema III.9. ***(Seguridad para todas las entradas)*** *Dado un circuito Circom correcto* $C \in \mathfrak{C}^{n\times t\times m}$ *con* $\mathcal{C}(C) \in \mathfrak{S}(\mathbb{F}_p[\bar{i}, \bar{x}, \bar{o}])$*, entonces:*

1. *C es **seguro** si y solo si $\mathcal{S}$ no tiene solución, donde*

$$\mathcal{S} = \mathcal{C}(C)[\bar{x} \to \bar{x}^1, \bar{o} \to \bar{o}^1] \cup \mathcal{C}(C)[\bar{x} \to \bar{x}^2, \bar{o} \to \bar{o}^2] \cup \mathcal{P}(\bar{o}^1, \bar{o}^2) \in \mathfrak{S}(\mathbb{F}_p[\bar{i}, \bar{x}^1, \bar{x}^2, \bar{o}^1, \bar{o}^2, \bar{u}])$$

2. *C es **fuertemente seguro** si y solo si $\mathcal{S} \in \mathfrak{S}(\mathbb{F}_p[\bar{i}, \bar{x}^1, \bar{x}^2, \bar{o}^1, \bar{o}^2, \bar{u}])$ no tiene solución, donde*

$$\mathcal{S} = \mathcal{C}(C)[\bar{x} \to \bar{x}^1, \bar{o} \to \bar{o}^1] \cup \mathcal{C}(C)[\bar{x} \to \bar{x}^2, \bar{o} \to \bar{o}^2] \cup \mathcal{P}((\bar{x}^1, \bar{o}^1), (\bar{x}^2, \bar{o}^2))$$

Demostración. Demostramos únicamente la primera afirmación, al ser la segunda demostración análoga pero ampliando la lista de variables prohibidas de $\mathcal{P}(\bar{o}^1, \bar{o}^2)$ a $\mathcal{P}((\bar{x}^1, \bar{o}^1), (\bar{x}^2, \bar{o}^2))$.

Probemos que $\mathcal{S}$ tiene solución $\iff$ C no es seguro:

- $\Rightarrow$ Supongamos que $\mathcal{S}$ tiene una solución $(\bar{i}', \bar{x}'^1, \bar{x}'^2, \bar{o}'^1, \bar{o}'^2, \bar{u}')$. Entonces, por la construcción de $\mathcal{S}$, $(\bar{i}', \bar{x}'^1, \bar{o}'^1)$ y $(\bar{i}', \bar{x}'^2, \bar{o}'^2)$ satisfacen $\mathcal{C}(C)$, por lo que ambas son witness válidas del circuito. Además, por el teorema III.7, como $(\bar{o}'^1, \bar{o}'^2)$ satisfacen $\mathcal{P}(\bar{o}^1, \bar{o}^2)$, entonces $\bar{o}'^1 \neq \bar{o}'^2$ y por lo tanto las dos witnesses tienen salidas distintas, por lo que C no es seguro.
- $\Leftarrow$ Supongamos que C no es seguro, por lo que existen $(\bar{i}', \bar{x}'^1, \bar{o}'^1)$ y $(\bar{i}', \bar{x}'^2, \bar{o}'^2)$ dos witnesses de C tal que $\bar{o}'^1 \neq \bar{o}^2$. Por ser witness del circuito, tenemos que $(\bar{i}, \bar{x}'^1, \bar{o}'^1)$ y $(\bar{i}, \bar{x}'^2, \bar{o}'^2)$ satisfacen $\mathcal{C}(C)$. Además, como $\bar{o}'^1 \neq \bar{o}'^2$, por el teorema III.7 $\exists \bar{u}'$ tal que $(\bar{o}'^1, \bar{o}'^2, \bar{u}')$ es solución de $\mathcal{P}(\bar{o}^1, \bar{o}^2)$. Juntándolo todo, tenemos que $(\bar{i}', \bar{x}'^1, \bar{x}'^2, \bar{o}'^1, \bar{o}'^2, \bar{u}')$ es solución de $\mathcal{S}$. □

Por tanto, si encontramos una forma de determinar que un sistema polinómico con coeficientes en $\mathbb{F}_p$ no tiene solución, como haremos en el capítulo IV, habremos conseguido el objetivo de crear un algoritmo que automáticamente garantice la seguridad de un circuito: primero construimos el sistema $\mathcal{S}$ de los teoremas III.8 o III.9 y determinamos si tiene una única solución.

Observación III.10. ***(Completitud)*** *Nótese que un algoritmo que no sea completo también tendría un gran interés. Debido a la magnitud del problema a resolver, con circuitos que implementan primitivas criptográficas complejas de varios cientos de miles de restricciones es muy improbable encontrar un algoritmo completo que sea capaz de tratar las entradas en un tiempo razonable.*

Un algoritmo que determine la insatisfacibilidad de un sistema de ecuaciones de forma correcta pero produzca falsos negativos (es decir, clasifique sistemas como satisfacibles cuando son insatisfacibles) también sería de gran interés, pues sería un algoritmo correcto pero no completo para garantizar la seguridad de ciertos circuitos Circom.

Ejemplo III.11. *El circuito* `IsZero` *descrito en el código II.4 no es fuertemente seguro para todas las entradas.*

Demostración. El sistema polinómico $\mathcal{S}$ del teorema III.9 es

$$\begin{cases} \texttt{out1} + \texttt{inp} \cdot \texttt{inv1} - 1 & = 0 \\ \texttt{inp} \cdot \texttt{out1} & = 0 \\ \texttt{out2} + \texttt{inp} \cdot \texttt{inv2} - 1 & = 0 \\ \texttt{inp} \cdot \texttt{out2} & = 0 \\ ((\texttt{out1} - \texttt{out2}) \cdot u_1 - 1) \cdot ((\texttt{inv1} - \texttt{inv2}) \cdot u_2 - 1) & = 0 \end{cases}$$

Una solución para $\mathcal{S}$ es $\texttt{inp} = 0$, $\texttt{out1} = \texttt{out2} = 1$, $\texttt{inv1} = 1$, $\texttt{inv2} = 0$, $u_1 = 0$, $u_2 = 1$, por lo que $\mathcal{S}$ es satisfacible y por el teorema III.9 el circuito `IsZero` no es fuertemente seguro.

En cambio, el sistema es insatisfacible si incluimos únicamente la salida en el polinomio prohibición. Esta insatisfacibilidad se puede determinar utilizando métodos algebraicos que veremos en el capítulo IV. □

Capítulo IV

Métodos algebraicos

En los teoremas III.8 y III.9 redujimos el problema de demostrar la seguridad de un circuito Circom a discernir si un sistema de ecuaciones polinómico tiene alguna solución en un cuerpo finito $\mathbb{F}_p$. En este capítulo estudiamos las herramientas algebraicas necesarias para decidir la satisfacibilidad de un sistema de ecuaciones polinómico. Primero damos una introducción a los ideales y variedades, después examinamos algunos resultados sobre el algoritmo de división en un anillo de polinomios en varias variables, introducimos las bases de Gröbner como herramienta para determinar la insatisfacibilidad y finalmente estudiamos el algoritmo de Buchberger para poder obtener computacionalmente una base de Gröbner.

Este estudio se basa en los resultados obtenidos en los libros [5, 10], aunque se ha condensado el material y reescrito para adaptarse a las contingencias de nuestra aplicación concreta.

IV.1. Ideales y variedades

En vez de trabajar con sistemas de ecuaciones, vamos a trabajar con objetos algebraicos llamados *ideales*, que nos permitirán expresar dichos sistemas de una forma más útil computacionalmente.

Notación IV.1. ***(Anillo de polinomios)*** *Llamaremos $\mathbb{K}[x_1, \ldots, x_n]$ al anillo conmutativo de polinomios sobre las variables $x_1, \ldots, x_n$ con coeficientes en el cuerpo $\mathbb{K}$.*

Definición IV.2. ***(Ideal)*** *Diremos que un subconjunto I de un anillo conmutativo $\mathcal{A}$ es un ideal si:*

1. $0 \in I$
2. *Si $f, g \in I$, entonces $f + g \in I$*
3. *Si $f \in I$ y $h \in \mathcal{A}$, entonces $f \cdot h \in I$*

Definición IV.3. *Sean $f_1, \ldots, f_m \in \mathcal{A}$, siendo $\mathcal{A}$ un anillo conmutativo. Definimos el ideal generado por $f_1, \ldots, f_m$ como el conjunto*

$$\langle f_1, \ldots, f_m \rangle = \left\{ \sum_{i=1}^{m} f_i \cdot g_i : g_i \in \mathcal{A} \right\}$$

Es inmediato comprobar que $I = \langle f_1, \ldots, f_m \rangle$ es un ideal. Diremos que $\{f_1, \ldots, f_m\}$ es una base de I y que I es finitamente generado.

De aquí en adelante, salvo mención expresa, trabajaremos en el anillo $\mathbb{K}[x_1, \ldots, x_n]$, y cuando nos refiramos a un ideal nos referiremos siempre a ideales de dicho anillo.

Definamos ahora la variedad asociada a un ideal finitamente generado, que es lo que nos permitirá enlazar los ideales con los sistemas de ecuaciones polinómicos.

Definición IV.4. *Dado un ideal de $\mathbb{K}[x_1, \ldots, x_n]$ finitamente generado $I = \langle f_1, \ldots, f_m \rangle$, definimos la variedad $\mathbf{V}(I)$ asociada al ideal como el conjunto*

$$\mathbf{V}(I) = \{x \in \mathbb{K}^n : f_i(x) = 0 \ \forall i \in \{1, \ldots, m\}\}$$

es decir, el conjunto de puntos de $\mathbb{K}^n$ donde se anulan todos los polinomios de la base simultáneamente. Es fácil comprobar que la definición anterior no depende de la base elegida para el ideal.

Definición IV.5. ***(Ideal asociado a un sistema de ecuaciones)*** *Si tenemos un sistema de ecuaciones polinómicas sobre un cuerpo $\mathbb{K}$*

$$\begin{cases} f_1(\bar{x}) & = 0 \\ f_2(\bar{x}) & = 0 \\ & \vdots \\ f_m(\bar{x}) & = 0 \end{cases}$$

definimos el ideal asociado al sistema de ecuaciones como $I = \langle f_1, \ldots, f_m \rangle$. Por la propia definición se deduce que $\mathbf{V}(I)$ es el conjunto de soluciones del sistema polinómico.

Por tanto, el problema de los teoremas III.8 y III.9 se reduce a decidir si $\mathbf{V}(I) = \emptyset$ dado un ideal finitamente generado I.

Teorema IV.6. *Sea $f_1, \ldots, f_m$ un ideal finitamente generado. Si $1 \in I$, entonces $V(I) = \emptyset$*

Demostración. Si $1 \in I$, por definición de base, $1 = f_1 g_1 + \cdots + f_m g_m$ para ciertos $g_i \in \mathbb{K}[x_1, \ldots, x_n]$. Si $V(I) \neq \emptyset$, entonces $\exists x \in V(I)$, y aplicando los polinomios de la igualdad anterior en x llegamos a que $1 = \underbrace{f_1(x)}_{0} g_1(x) + \cdots + \underbrace{f_m(x)}_{0} g_m(x) = 0$, una contradicción. □

Observación IV.7. ***(Completitud)*** *El teorema IV.6 es la implicación trivial de la forma débil del Nullstellensatz, demostrado por Hilbert. La otra implicación también se da en cuerpos algebraicamente cerrados. En [14] se demuestra una versión débil del Nullstellensatz para cuerpos finitos, que se reduce a lo siguiente:*

Sea $\mathbb{K} = \mathbb{F}_p$, y sea $I = \langle f_1, \ldots, f_m, x_1^p - x_1, \ldots, x_n^p - x_n \rangle$. Entonces, $1 \notin I$ si y solamente si existe una solución simultánea para todas las ecuaciones $f_i(\bar{x}) = 0$ en $\mathbb{F}_p^n$.

Como vemos, hay que añadir al ideal los polinomios $x_i^p - x_i$, que "codifican" los ceros que añade el cuerpo finito. Estos polinomios tienen el primo p del cuerpo en el exponente. Recordemos que en nuestro problema, p puede tener más de 250 bits, haciendo el problema intratable computacionalmente. Por tanto, renunciaremos a la completitud al no añadir los polinomios $x_i^p - x_i$ extra, conformándonos con poder demostrar la seguridad correctamente, tal y como comentamos en la observación III.10. Además, en el capítulo VI veremos que en la práctica no hemos encontrado ningún circuito en el que se pierda completitud por no añadir los polinomios $x_i^p - x_i$.

En las siguientes secciones desarrollaremos métodos para poder decidir si un polinomio dado (estando nosotros interesados en el polinomio constantemente uno) pertenece a un ideal, hallando

así un algoritmo correcto (pero no completo) para garantizar la seguridad de ciertos circuitos Circom.

IV.2. Algoritmo de división en $\mathbb{K}[x_1, \ldots, x_n]$

Para decidir si un polinomio está en un ideal, nos gustaría poder *dividir* un polinomio $f \in \mathbb{K}[x_1, \ldots, x_n]$ por otros polinomios $f_1, \ldots f_m \in \mathbb{K}[x_1, \ldots, x_n]$, es decir, expresar f como

$$f = q_1 f_1 + \cdots + q_m f_m + r$$

donde $q_1, \ldots, q_m, r \in \mathbb{K}[x_1, \ldots, x_n]$, y nos gustaría poner alguna restricción sobre el resto r. Antes de desarrollar un algoritmo de este estilo, recordemos el algoritmo de la división en polinomios de una única variable, para encontrar alguna manera de generalizarlo a polinomios de varias variables. Para ello, necesitamos algunas definiciones.

Definiciones IV.8. *Sea* $p = \sum_{i=0}^{m} a_i x^i \in \mathbb{K}[x]$ *con* $a_m \neq 0$*, entonces llamamos* ***grado*** *a* $\deg(p) = m$ *y* ***término director*** *a* $\mathrm{TD}(p) = a_m x^m$*. En el caso de que* $p = 0$*, definimos* $\mathrm{TD}(0) = 0$ *y diremos que* $\deg(0)$ *no está definido.*

Algoritmo de división en $\mathbb{K}[x]$

Algoritmo 1 Algoritmo de división en $\mathbb{K}[x]$

Entrada: $a, b \in \mathbb{K}[x]$, $b \neq 0$
Salida: Cociente $q \in \mathbb{K}[x]$, resto $r \in \mathbb{K}[x]$

```
q ← 0
r ← a
while r ≠ 0 and deg(r) ≥ deg(b) do        ▷ El invariante del bucle es a = b · q + r
    tmp ← TD(r) / TD(b)
    q ← q + tmp
    r ← r − tmp · b
end while
```

Teorema IV.9. *El algoritmo 1 termina y computa una división en* $\mathbb{K}[x]$*, es decir, al finalizar se cumple que* $a = b \cdot q + r$ *y o bien* $r = 0$ *o bien* $\deg(r) < deg(b)$*.*

Demostración. ▪ **Corrección**. Comenzaremos demostrando que en toda iteración del bucle se cumple el invariante $a = b \cdot q + r$. Esto es trivialmente cierto al comienzo del programa, ya que $a = 0 \cdot b + a$, y si se cumple al comienzo de una iteración del bucle, al final también:

$$b \cdot \underbrace{\left(q + \frac{\mathrm{TD}(r)}{\mathrm{TD}(b)}\right)}_{q'} + \underbrace{\left(r - \frac{\mathrm{TD}(r)}{\mathrm{TD}(b)} b\right)}_{r'} = \underbrace{b \cdot q + r}_{a} + \underbrace{b\frac{\mathrm{TD}(r)}{\mathrm{TD}(b)} - \frac{\mathrm{TD}(r)}{\mathrm{TD}(b)} b}_{0} = a$$

Por tanto, al terminar el algoritmo se cumplirá que $a = b \cdot q + r$, y como hemos salido del bucle, también se cumplirá la negación de la condición, por lo que o bien $r = 0$ o bien $\deg(r) < \deg(b)$.

- **Terminación**. La observación clave es que en cada iteración del bucle, o bien r se convierte en 0, o bien el grado de r decrece estrictamente, ya que el término de mayor grado de r se convierte en 0 al realizar la operación $r - b \cdot \frac{\mathrm{TD}(r)}{\mathrm{TD}(b)}$. Como hemos definido el grado de un polinomio como un número natural, por la buena ordenación de los naturales no puede existir una secuencia infinita de grados de r estrictamente decrecientes, por lo que en algún momento se deberá salir del bucle. □

Una propiedad muy importante de la división en $\mathbb{K}[x]$ es que el cociente y el resto son únicos, sin importar el algoritmo que se use para realizar la división.

Teorema IV.10. *Sean $a, b \neq 0 \in \mathbb{K}[x]$, entonces, existen q y r en $\mathbb{K}[x]$ tal que $a = b \cdot q + r$ y o bien $r = 0$ o bien $\deg(r) < deg(b)$. Además, esta descomposición es única.*

Demostración. Hemos demostrado la existencia en el teorema IV.9, estudiemos ahora la unicidad de la descomposición. Supongamos que no es única, por lo que $a = b \cdot q + r = b \cdot q' + r'$ con r y r' satisfaciendo la desigualdad del grado. Sacando factor común, $b(q - q') = r' - r$. Como $b \neq 0$, si $r = r'$, entonces $q = q'$ y hemos terminado. Supongamos que $r \neq r'$, por lo que $q - q' \neq 0$ y $\deg(r' - r) < \deg(b)$ por la desigualdad del grado de los restos de la división. Sin embargo,

$$\deg(r - r') = \deg(b(q - q')) = \deg(b) + \deg(q - q') \geq \deg(b)$$

por lo que llegamos a una contradicción, y $r = r'$ y $q = q'$. □

Orden monomial

Hemos visto que una parte importante del algoritmo de la división en polinomios de una variable es el concepto de grado y el concepto de coeficiente director, que lleva implícito un orden entre monomios. Por tanto, para generalizar el algoritmo de la división a polinomios de varias variables, primero necesitaremos generalizar el concepto de grado y el concepto de orden entre monomios.

Definición IV.11. ***(Monomio)*** *Un monomio en $\mathbb{K}[x_1, \ldots, x_n]$ es un producto de variables $x_1, x_2, \ldots, x_n$ elevados a exponentes $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{N}_0$, es decir, el término $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$. Por notación, si denotamos la tupla $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{N}_0^n$, definimos el monomio $x^\alpha := x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$.*

Tal y como hemos visto en la definición IV.11, podemos hacer una biyección entre los monomios y las tuplas de $\mathbb{N}^0$, lo que nos será muy útil para definir órdenes entre monomios. Definiremos la suma entre tuplas de $\mathbb{N}^0$ como la suma elemento a elemento.

Una propiedad muy deseable en el orden de monomios será que se respeten las operaciones del anillo de los polinomios. La suma de monomios no es un monomio, por lo que no podremos compararlos directamente. La multiplicación de dos monomios sí que es un monomio, por lo que sería interesante que si $x^\alpha > x^\beta$, entonces $x^{\alpha+\gamma} = x^\alpha \cdot x^\gamma > x^\beta \cdot x^\gamma = x^{\beta+\gamma}$.

Finalmente, nos gustaría que el orden fuese un buen orden, propiedad que utilizamos para demostrar la terminación de la división en $\mathbb{K}[x]$. Estamos listos para la definición formal.

Definición IV.12. ***(Orden total)*** *Siendo $S \neq \emptyset$, diremos que $\geq$ es una relación de orden total si satisface las siguientes propiedades:*

1. ***Reflexiva**: Para todo $e \in S$, $e \geq e$.*

2. ***Antisimétrica**: Para todo $x, y \in S$, $x \geq y \wedge y \geq x \implies x = y$.*
3. ***Transitiva**: Para todo $x, y, z \in S$, $x \geq y \wedge y \geq z \implies x \geq z$.*
4. ***Total**: Para todo $x, y \in S$, o bien $x \geq y$ o bien $y \geq x$.*

Nótese que una relación de orden $\geq$ induce una relación de orden $>$ y viceversa, donde $x > y \iff x \geq y \wedge x \neq y$.

Dada la biyección vista entre los monomios y $\mathbb{N}_0^n$, podemos identificar los órdenes entre monomios y los órdenes entre tuplas de $\mathbb{N}_0^n$.

Definición IV.13. ***(Orden monomial)** Un orden monomial $<$ es una relación de orden en $\mathbb{N}_0^n$ (definiendo por tanto una relación de orden entre monomios) que cumple las siguientes propiedades:*

1. *$\leq$ es un orden total.*
2. *$\forall \alpha, \beta, \gamma \in \mathbb{N}_0^n$, si $\alpha > \beta$, entonces $\alpha + \gamma > \beta + \gamma$.*
3. *$\mathbb{N}_0^n$ es un conjunto bien ordenado con $<$, es decir, $\forall S \neq \emptyset \subseteq \mathbb{N}_0^n$, entonces $\exists m \in S$ tal que para todo $e \in S, m \leq e$.*

Vamos a demostrar ahora un lema que utilizaremos para caracterizar los buenos órdenes.

Lema IV.14. *Un orden total $\leq$ en un conjunto S es un buen orden si y solo si toda sucesión (a_n) donde $a_n \in S$ y $a_0 > a_1 > a_2 > \cdots$ termina, es decir, es finita.*

Demostración. $\Rightarrow$ Supongamos que la sucesión (a_n) no termina, es decir, es infinita. Sea $A = \{a_n\} \neq \emptyset$. El conjunto A no es vacío y no tiene elemento mínimo, por lo que llegamos a una contradicción.

$\Leftarrow$ Si $<$ no es un buen orden, existe un conjunto $A \neq \emptyset \subseteq S$ que no tiene elemento mínimo. Elegimos $a_0 \in A$. Como A no tiene elemento mínimo, en concreto a_0 no es un elemento mínimo, por lo que $\exists a_1 \in A$ tal que $a_0 > a_1$. Repitiendo esta construcción hallamos una secuencia infinita $a_0 > a_1 > a_2 > \cdots$. □

Ejemplo IV.15. ***(Orden lexicográfico)** Sea $\alpha, \beta \in \mathbb{N}_0^n$. Diremos que $\alpha >_{lex} \beta$ si el primer elemento no nulo del vector $\alpha - \beta$ es positivo. Este orden en $\mathbb{N}_0^n$ induce un orden monomial en $\mathbb{K}[x_1, \ldots, x_n]$.*

Demostración. Veamos que $>_{lex}$ satisface todas las propiedades de la definición IV.13.

1. Que $>_{lex}$ es un orden total en $\mathbb{N}_0^n$ se deduce de que $>$ es un orden total en $\mathbb{N}_0$.
2. Esta propiedad se deduce de que $(\alpha + \gamma) - (\beta + \gamma) = \alpha - \beta$, por lo que el primer elemento no nulo de un vector será positivo si y solo si lo es del otro, y por tanto si $\alpha >_{lex} \beta$, tendremos que $\alpha + \gamma >_{lex} \beta + \gamma$.
3. Por el lema IV.14 basta con comprobar que no existe ninguna sucesión de desigualdades $a_0 >_{lex} a_1 >_{lex} \cdots$. Como $\mathbb{N}_0$ es un conjunto bien ordenado con $>$, cada una de las dimensiones del vector debe estabilizarse en algún momento, no pueden descender infinitamente. Como $\mathbb{N}_0$ tiene únicamente n dimensiones, un número finito de dimensiones, la sucesión no puede continuar de forma infinita, sino que tiene que parar en algún momento. □

Una vez definido un orden monomial, podemos dar algunas definiciones que usaremos en algoritmos posteriores, como el término director.

En la definición IV.11 vimos que podemos identificar cada $\alpha \in \mathbb{N}_0^n$ con el monomio x^α. Por tanto, si llamamos $a_\alpha \in \mathbb{K}$ al coeficiente asociado al monomio x^α, podemos representar cualquier polinomio multivariable $f \in \mathbb{K}[x_1, \cdots, x_n]$ como la suma $f = \sum_\alpha a_\alpha x^\alpha$ con cada $a_\alpha \neq 0$.

Definiciones IV.16. *Sea $f = \sum_\alpha a_\alpha x^\alpha \neq 0 \in \mathbb{K}[x_1, \cdots, x_n]$. Definimos*

- *El **multigrado** es* $\operatorname{multideg}(f) = \max\{\alpha \in \mathbb{N}_0^n : a_\alpha \neq 0\}$
- *El **coeficiente director** es* $\operatorname{CD}(f) = a_{\operatorname{multideg}(f)}$
- *El **término director** es* $\operatorname{TD}(f) = a_{\operatorname{multideg}(f)} x^{\operatorname{multideg}(f)}$
- *El **monomio director** es* $\operatorname{MD}(f) = x^{\operatorname{multideg}(f)}$

A partir de ahora asumiremos que junto a nuestros polinomios tenemos siempre un orden monomial establecido. Nótese que distintos órdenes monomiales pueden cambiar el resultado de los algoritmos e incluso la eficiencia de estos. Por ello, siempre asumiremos que hemos escogido un orden monomial cualquiera fijo común para todos los algoritmos y teoremas.

Algoritmo de división multivariable

Como citamos anteriormente, nuestro objetivo es obtener un algoritmo para descomponer un polinomio a en una combinación lineal de polinomios $b_1, \ldots, b_s$ más un resto r. Para ello generalizaremos el algoritmo de la división univariable al algoritmo 2, viendo si es posible dividir el término director del dividendo por el de algún divisor, comprobando los divisores secuencialmente. La única novedad es que es posible que el término director del dividiendo no sea divisible por ningún divisor, pero sí sean divisibles otros términos del dividendo. En ese caso, añadimos directamente el término director del dividendo al resto.

Observación IV.17. *En principio, el resultado del algoritmo 2 puede depender del orden en el que se pasan los divisores $b_1, \ldots, b_s$, por lo que toma como argumento una tupla **ordenada** $B = (b_1, \ldots, b_s)$. También depende del orden monomial usado.*

Visto el algoritmo, una caracterización adecuada del resto sería que ningún término de r sea divisible por $\operatorname{TD}(b_1), \ldots, \operatorname{TD}(b_s)$. Demostremos la corrección y terminación del algoritmo 2 en el teorema IV.18.

Teorema IV.18. *El algoritmo 2 termina y computa una división en $\mathbb{K}[x_1, \ldots, x_n]$, es decir, al finalizar se cumple que $a = b_1 \cdot q_1 + \cdots + b_s \cdot q_s + r$ y ningún término de r es divisible por $\operatorname{TD}(b_i)$ para ningún $i \in \{1, \ldots, s\}$. Además, si $b_i q_i \neq 0$, entonces $\operatorname{multideg}(b_i q_i) \leq \operatorname{multideg}(a)$ para todo i.*

Demostración.

- **Corrección**. Primero demostremos que en cada iteración del bucle `while` se cumple el invariante $a - d = b_1 \cdot q_1 + \cdots + b_s \cdot q_s + r$. Antes de entrar al bucle, el invariante es cierto, pues $d = a$ y $q_1 = \cdots = q_s = r = 0$, por lo que $a - d = 0 = b_1 \cdot q_1 + \cdots + b_s \cdot q_s + r$. Si el invariante se cumple antes de una iteración, también se cumple después: si no se realiza ninguna división $q_i' = q_i$, $r' = r + \operatorname{TD}(d)$, $d' = d - \operatorname{TD}(d)$, por lo que

$$a - d' = a - d + \operatorname{TD}(d) = b_1 \cdot \underbrace{q_1}_{q_1'} + \cdots + b_s \cdot \underbrace{q_s}_{q_s'} + \underbrace{r + \operatorname{TD}(d)}_{r'}$$

Algoritmo 2 Algoritmo de división en $\mathbb{K}[x_1, \ldots, x_n]$

Entrada: Dividendo $a \in \mathbb{K}[x_1, \ldots, x_n]$, divisores $b_1, \ldots, b_s \neq 0 \in \mathbb{K}[x_1, \ldots, x_n]$
Salida: Cocientes $q_1, q_2, \ldots, q_s \in \mathbb{K}[x_1, \ldots, x_n]$, resto $r \in \mathbb{K}[x_1, \ldots x_n]$

```
for i = 1 to s do
    q_i ← 0
end for
d ← a                                   ▷ d es el dividendo intermedio en cada paso
r ← 0
while d ≠ 0 do          ▷ El invariante del bucle es a − d = b_1·q_1 + ··· + b_s·q_s + r
    division_exitosa ← false
    for i = 1 to s do
        if TD(b_i) | TD(d) then
            tmp ← TD(d)/TD(b_i)
            q_i ← q_i + tmp
            d ← d − tmp · b_i
            division_exitosa ← true
            break for loop                  ▷ Salimos del bucle for
        end if
    end for
    if ¬division_exitosa then
        r ← r + TD(d)
        d ← d − TD(d)
    end if
end while
```

En caso de que se realice una división con el polinomio b_i, entonces $q'_j = q_j$ para todo $j \neq i$, $q'_i = q_i + \frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)}$, $d' = d - \frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)} b_i$ y $r' = r$. Por tanto,

$$\begin{aligned} a - d' &= a - d + \frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)} b_i = b_1 \cdot q_1 + \cdots + b_s \cdot q_s + r + \frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)} b_i \\ &= b_1 \cdot \underbrace{q_1}_{q'_1} + \cdots + b_i \underbrace{\left(q_i + \frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)} \right)}_{q'_i} + \cdots + b_s \cdot \underbrace{q_s}_{q'_s} + \underbrace{r}_{r'} \end{aligned}$$

Como se mantiene el invariante, y al salir del bucle tenemos $d = 0$, se cumple la igualdad buscada. Además, el resto cumple la propiedad buscada, pues únicamente se añaden términos cuando no se ha podido realizar una división, y ninguno de los monomios de esos términos pueden ser divisibles por ningún $\mathrm{TD}(b_i)$, pues si no se hubiese realizado una división con dicho b_i.

Por último, se cumple que si $b_i q_i \neq 0$, entonces $\mathrm{multideg}(b_i q_i) \leq \mathrm{multideg}(a)$, pues en todo momento $\mathrm{TD}(d) \leq \mathrm{TD}(a)$, pues a d en cada paso el mayor término que le restamos es su término director para hacerlo 0, y todo término de q_i es de la forma $\frac{\mathrm{TD}(d)}{\mathrm{TD}(b_i)}$.

- **Terminación**. Es consecuencia de que $\mathbb{N}_0^n$ esté bien ordenado con el orden monomial $<$. En cada iteración del bucle, o d se hace 0, o su multigrado decrece estrictamente, pues tanto si se realiza división como si no, el coeficiente asociado al término director de d se convierte en 0. □

IV.3. Bases de Gröbner

Cabe preguntarse si el resto de la división en $\mathbb{K}[x_1, \ldots, x_n]$ es único, tal y como pasa en $\mathbb{K}[x]$. Desafortunadamente, la respuesta es negativa, como se puede ver en el ejemplo IV.19.

Ejemplo IV.19. *Consideremos el anillo de polinomios $\mathbb{K}[x, y]$ con el orden monomial $<_{lex}$. Sean $a = xy^2 - y$, $b_1 = xy - 1$, $b_2 = y^2$. Entonces, el resultado de dividir a entre (b_1, b_2) es $(q_1 = y, q_2 = 0, r = 0)$, y el resultado de dividir a entre (b_2, b_1) es $(q_1 = x, q_2 = 0, r = -y)$. Es decir, $a = y \cdot b_1 + 0 \cdot b_2 + 0 = 0 \cdot b_1 + x \cdot b_2 - y$*

Por tanto, observamos que aunque un polinomio pertenezca a cierto ideal, dividir dicho polinomio entre los generadores del ideal no siempre resultará en resto nulo. Vamos a buscar algún tipo de bases donde esto no suceda, es decir, que el resto de dividir un polinomio por los generadores del ideal sea 0 si y solo si dicho polinomio pertenece al ideal. La clave de estas bases estará en la relación entre los términos directores de los polinomios de la base y los términos directores del ideal. Necesitamos algunas definiciones previas.

Definición IV.20. ***(Ideal generado por un conjunto)*** *Sea $S \neq \emptyset \subseteq \mathbb{K}[x_1, \ldots, x_n]$ un conjunto posiblemente infinito. Definimos el ideal generado por S como el ideal que contiene todas las combinaciones lineales finitas de elementos en S:*

$$\langle S \rangle = \left\{ \sum_{i=1}^{m} f_i \cdot g_i : m \in \mathbb{N}, f_i \in S, g_i \in \mathbb{K}[x_1, \ldots, x_n] \right\}$$

Definición IV.21. ***(Ideal de términos directores)*** *Sea $I \neq \{0\} \subseteq \mathbb{K}[x_1, \ldots, x_n]$. Definimos $\mathrm{TD}(I)$ como el conjunto de términos directores de todos los polinomios no nulos de I:*

$$\mathrm{TD}(I) = \{a_\alpha x^\alpha : \exists f \in I \setminus \{0\} \text{ tal que } \mathrm{TD}(f) = a_\alpha x^\alpha\}$$

Definimos $\langle \mathrm{TD}(I) \rangle$ como el ideal generado por el conjunto $\mathrm{TD}(I)$.

Definición IV.22. ***(Base de Gröbner)*** *Sea $I \neq \{0\} \subseteq \mathbb{K}[x_1, \ldots, x_n]$ un ideal. Diremos que una base $\{f_1, \ldots, f_m\}$ de I con $f_i \neq 0$ es una base de Gröbner si se cumple que*

$$\langle \mathrm{TD}(I) \rangle = \langle \mathrm{TD}(f_1), \ldots, \mathrm{TD}(f_m) \rangle$$

Observación IV.23. *Nótese que si $I = \langle f_1, \ldots, f_m \rangle$, entonces $f_i \in I$, por lo que $\mathrm{TD}(f_i) \in \langle \mathrm{TD}(I) \rangle$, y siempre será cierta la desigualdad $\langle \mathrm{TD}(f_1), \ldots, \mathrm{TD}(f_m) \rangle \subseteq \langle \mathrm{TD}(I) \rangle$*

El ideal $\langle \mathrm{TD}(f_1), \ldots, \mathrm{TD}(f_m) \rangle \subseteq \langle \mathrm{TD}(I) \rangle$ de la definición IV.22 es un ideal muy especial, pues todos sus generadores son monomios. Esto tiene una consecuencia bastante importante que usaremos en demostraciones futuras.

Lema IV.24. *Sea $I \subseteq \mathbb{K}[x_1, \ldots, x_n]$ tal que $I = \langle a_1 x^{\alpha_1}, \ldots, a_m x^{\alpha_m} \rangle$. Entonces, si $c \in \mathbb{K}$, un término $cx^\alpha \in I \iff \exists i \in \{1, \ldots, m\}$ tal que $a_i x^{\alpha_i}$ divide a cx^α.*

Demostración. $\Leftarrow$ Por pertenecer $a_i x^{\alpha_i}$ al ideal I, también lo hará cualquier múltiplo suyo, en concreto cx^α.

⇒ Si $cx^\alpha \in I$, entonces $cx^\alpha = \sum_{i=1}^m g_i a_i x^{\alpha_i}$ con $g_i \in \mathbb{K}[x_1, \dots, x_n]$. Descomponiendo cada $g_i = \sum_j b_{i,j} x^{\beta_{i,j}}$ en sus términos,

$$cx^\alpha = \sum_{i=1}^m \left(a_i x^{\alpha_i} \sum_j b_{i,j} x^{\beta_{i,j}} \right) = \sum_{i,j} a_i b_{i,j} x^{\beta i,j} x^{\alpha_i}$$

Agrupando todos los términos de la derecha del mismo multigrado, llegamos a que todos los términos son divisibles por al menos un $a_i x^{\alpha_i}$, por lo que también lo será cx^α. □

La definición IV.22 puede parecer arbitraria, pero es la condición clave para poder demostrar en la proposición IV.25 la unicidad del resto que buscamos.

Proposición IV.25. *Sea $I \neq \{0\} \subseteq \mathbb{K}[x_1, \dots, x_n]$ un ideal. Si $\{f_1, \dots, f_m\}$ es una base de Gröbner de I, entonces, para cualquier $a \in \mathbb{K}[x_1, \dots, x_n]$, existen $q_1, \dots, q_m \in \mathbb{K}[x_1, \dots, x_n]$ y $r \in \mathbb{K}[x_1, \dots, x_n]$ tal que $a = q_1 f_1 + \cdots + q_m f_m + r$ y ningún término de r es divisible por ningún $\mathrm{TD}(f_i)$. Además, r es único bajo estas condiciones.*

Demostración. La existencia viene dada por la aplicación del algoritmo de división 2 al dividir a entre la tupla $(f_1, \dots, f_m)$. Veamos la unicidad. Supongamos que

$$a = \underbrace{q_1 f_1 + \cdots + q_m f_m}_{g} + r = \underbrace{q_1' f_1 + \cdots + q_m' f_m}_{g'} + r'$$

Por tanto, $r - r = g' - g \in I$, pues $g, g' \in I$. Supongamos que $r \neq r'$ y lleguemos a una contradicción. Como $r - r' \in I$, entonces $\mathrm{TD}(r - r') \in \langle \mathrm{TD}(I) \rangle = \langle \mathrm{TD}(f_1), \dots, \mathrm{TD}(f_m) \rangle$. Por el lema IV.24, esto significa que $\exists i$ tal que $\mathrm{TD}(f_i) \mid (r - r')$. Eso significa que $r - r' = h \cdot \mathrm{TD}(f_i)$, con $h = \sum_{\beta_j} d_{\beta_j} x^{\beta_j}$ y $h, d_{\beta_j} \neq 0$. Por tanto,

$$r - r' = \sum_{\beta_j} d_{\beta_j} x^{\beta_j} \, \mathrm{TD}(f_i)$$

Consideremos ahora el sumando $d_{\beta_0} x^{\beta_0} \, \mathrm{TD}(f_i) \neq 0$. Como r' no tiene ningún término divisible por $\mathrm{TD}(f_i)$, el sumando no puede cancelarse al sumarle r', por lo que es un sumando de r, y por tanto $\mathrm{TD}(f_i)$ divide a un término de r, llegando a una contradicción. □

Podemos utilizar esta propiedad para garantizar cuando un polinomio pertenece a un ideal, si tenemos una base de Gröbner de dicho ideal, resolviendo el problema que buscabamos.

Corolario IV.26. *Sea $I \neq \{0\} \subseteq \mathbb{K}[x_1, \dots, x_n]$ un ideal. Si $\{f_1, \dots, f_m\}$ es una base de Gröbner de I, entonces, para cualquier $a \in \mathbb{K}[x_1, \dots, x_n]$, se cumple que $a \in I$ si y solo si el resto de dividir a entre $(f_1, \dots, f_m)$ es 0.*

Demostración. ⇐ Si el resto de la división es 0, entonces $a = q_1 f_1 + \cdots + q_m f_m$, y cada $f_i \in I$, por lo que $a \in I$.

⇒ Como $a \in I$, podemos expresarlo como combinación lineal de la base del ideal, es decir,

$$a = \underbrace{g_1}_{q_1} f_1 + \cdots + \underbrace{g_m}_{q_m} f_m + \underbrace{0}_{r}$$

Por la proposición IV.25 el resto de la división es único, y hemos encontrado una división con resto 0, por lo que el resto de dividir a entre $(f_1, \dots, f_m)$ también debe ser 0. □

Ahora vamos a centrarnos en encontrar un algoritmo para hallar bases de Gröbner de un ideal dado. Para ello, vamos a comenzar estudiando por qué una base de un ideal puede no ser una base de Gröbner, que se deberá a la cancelación de los términos directores de los polinomios de una base al tomar alguna combinación lineal. Para determinar estas cancelaciones, definamos algunos términos.

Definición IV.27. *Sean $x^\alpha, x^\beta \in \mathbb{K}[x_1, \dots, x_n]$ dos monomios. Definimos entonces el mínimo común múltiplo de x^α y x^β como* $\operatorname{mcm}(x^\alpha, x^\beta) = x^\gamma$*, donde $\gamma_i = \max\{\alpha_i, \beta_i\}$. Es inmediato ver que cualquier múltiplo simultáneo de x^α y x^β también será múltiplo de* $\operatorname{mcm}(x^\alpha, x^\beta)$.

Ejemplo IV.28. *Usando $<_{lex}$ y los polinomios $f_1 = xy - 1$, $f_2 = y^2$, con $f_1, f_2 \in \mathbb{K}[x, y]$ del ejemplo IV.19, se puede comprobar que $\{f_1, f_2\}$ no es una base de Gröbner de $I = \langle f_1, f_2\rangle$.*

Demostración. Sabemos que no puede ser base de Gröbner porque el resto no es único, pero veamos explícitamente que $\langle \mathrm{TD}(I)\rangle \neq \langle \mathrm{TD}(f_1), \mathrm{TD}(f_2)\rangle$. Sea

$$g := \frac{x^{\operatorname{mcm}(\mathrm{MD}(f_1),\mathrm{MD}(f_2))}}{\mathrm{TD}(f_1)} f_1 - \frac{x^{\operatorname{mcm}(\mathrm{MD}(f_1),\mathrm{MD}(f_2))}}{\mathrm{TD}(f_2)} f_2 = \frac{xy^2}{xy}(xy - 1) - \frac{xy^2}{y^2} y^2 = -y$$

Vemos que $\mathrm{TD}(g) = -y$, pero ni $\mathrm{TD}(f_1) = xy$ ni $\mathrm{TD}(f_2) = y^2$ dividen a $\mathrm{TD}(g)$, por lo que por el lema IV.24, $\mathrm{TD}(g) \notin \langle \mathrm{TD}(f_1), \mathrm{TD}(f_2)\rangle$, pero sin embargo $g \in I$, por lo que $\mathrm{TD}(g) \in \langle \mathrm{TD}(I)\rangle$. □

La clave está en la construcción de g en el ejemplo IV.28, que es una expresión diseñada para cancelar términos directores.

Definición IV.29. *Sean $f, g \neq 0 \in \mathbb{K}[x_1, \dots, x_n]$, definimos el polinomio-S de f y g como*

$$S(f, g) = \frac{x^{\operatorname{mcm}(\mathrm{MD}(f),\mathrm{MD}(g))}}{\mathrm{TD}(f)} f - \frac{x^{\operatorname{mcm}(\mathrm{MD}(f),\mathrm{MD}(g))}}{\mathrm{TD}(g)} g$$

En la siguiente proposición demostraremos que efectivamente los polinomios-S producen cancelaciones.

Proposición IV.30. *Sean $f, g \neq 0 \in \mathbb{K}[x_1, \dots, x_n]$. Si $x^\alpha = \operatorname{mcm}(\mathrm{MD}(f), \mathrm{MD}(g))$ y $S(f, g) \neq 0$, entonces* $\operatorname{multideg}(S(f, g)) < \alpha$.

Demostración. Por definición, $S(f, g) = \frac{x^\alpha}{\mathrm{TD}(f)} f - \frac{x^\alpha}{\mathrm{TD}(g)} g$. Si descomponemos f en $f = \mathrm{TD}(f) + f'$ y $g = \mathrm{TD}(g) + g'$, llegamos a que

$$S(f, g) = \frac{x^\alpha(\mathrm{TD}(f) + f')}{\mathrm{TD}(f)} - \frac{x^\alpha(\mathrm{TD}(g) + g')}{\mathrm{TD}(g)} = \cancel{x^\alpha} + \frac{x^\alpha}{\mathrm{TD}(f)} f' - \cancel{x^\alpha} + \frac{x^\alpha}{\mathrm{TD}(g)} g'$$

Veamos que $\frac{x^\alpha}{\mathrm{TD}(f)} f' < x^\alpha$, y la demostración para g es idéntica. Si f' es 0 hemos terminado, si no, por ser $\frac{x^\alpha}{\mathrm{TD}(f)}$ un monomio, tenemos que $\mathrm{TD}\left(\frac{x^\alpha}{\mathrm{TD}(f)} f'\right) = \frac{x^\alpha}{\mathrm{TD}(f)} \mathrm{TD}(f') < x^\alpha$, pues $\mathrm{TD}(f) > \mathrm{TD}(f')$. □

Una propiedad clave de los polinomios-S es que se pueden expresar todas las cancelaciones de términos directores entre polinomios del mismo multigrado como una combinación lineal de polinomios-S.

Proposición IV.31. *Sean* $f_1, \dots, f_m \in \mathbb{K}[x_1, \dots, x_n]$ *no nulos tal que* $\text{multideg}(f_1) = \cdots = \text{multideg}(f_m) = \alpha$. *Si* $\text{multideg}\left(\sum_{i=1}^m f_i\right) < \alpha$, *entonces existen* $c_{i,j} \in \mathbb{K}$ *tal que*

$$\sum_{i=1}^{m} f_i = \sum_{1 \le i,j \le m} c_{i,j} S(f_i, f_j)$$

Demostración. Como todos los f_i tienen el mismo multigrado α, podemos expresar $\text{TD}(f_i) = a_i x^\alpha$. Además, como la suma no tiene multigrado α, el coeficiente asociado a α en la suma debe ser 0, es decir, $\sum_{i=1}^m \text{TD}(f_i) = 0$, por lo que $\sum_{i=1}^m a_i = 0$, es decir $a_m = -\sum_{i=1}^{m-1} a_i$. Además como $\text{MD}(f_1) = \cdots = \text{MD}(f_m) = x^\alpha$, $S(f_i, f_j) = \frac{1}{a_i} f_i - \frac{1}{a_j} f_j$. Por tanto,

$$\begin{aligned}\sum_{i=1}^{m-1} a_i S(f_i, f_m) &= a_1 \left(\frac{1}{a_1} f_1 - \frac{1}{a_m} f_m\right) + \cdots + a_{m-1} \left(\frac{1}{a_{m-1}} f_{m-1} - \frac{1}{a_m} f_m\right) \\ &= \sum_{i=1}^{m-1} f_i - \frac{1}{a_m} f_m \underbrace{(a_1 + \cdots + a_{m-1})}_{-a_m} = \sum_{i=1}^{m} f_i\end{aligned}$$

□

Con esta proposición estamos listos para un teorema muy importante de esta sección, que nos da una forma computacional de verificar que una base es una base de Gröbner.

Teorema IV.32. ***(Criterio de Buchberger)*** *Sea* $I \neq \{0\} \subset \mathbb{K}[x_1, \dots, x_n]$ *un ideal. Una base* $\mathcal{B} = \{f_1, \dots, f_m\}$ *de* I *con* $f_i \neq 0$ *es una base de Gröbner si y solo si para todo* $1 \le i \neq j \le m$ *el resto de la división entre* $S(f_i, f_j)$ *y* $\mathcal{B}$ *(donde para la división se toman los elementos de* $\mathcal{B}$ *en algún orden) es 0.*

Demostración. $\Rightarrow$ Si $\mathcal{B}$ es una base de Gröbner, como $S(f_i, f_j) \in I$ por el corolario IV.26 el resto de la división será 0.

$\Leftarrow$ Sea $g \neq 0 \in I$. Veamos que $\text{TD}(g) \in \langle \text{TD}(f_1), \dots, \text{TD}(f_n) \rangle$. Como $g \in I$, podemos expresar g como una o posiblemente varias combinaciones lineales de los f_i. Por el buen orden de $<$ en $\mathbb{Z}_0^n$, existe una de esas representaciones $g = \sum_{i=1}^m h_i f_i$ con $h_1, \dots, h_m \in \mathbb{K}[x_1, \dots, x_n]$ tal que $\alpha = \text{máx}\{\text{multideg}(h_i f_i) : h_i f_i \neq 0\}$ es mínimo. Como g es una suma de $h_i f_i$, se da que $\text{multideg}(g) \le \alpha$.

En el caso de que se alcance la igualdad, g tendrá un término de multigrado α, que es el mismo multigrado que algún $h_i f_i$ para el que se alcance el máximo de la definición de α. Como tienen el mismo multigrado, $\text{TD}(h_i f_i) \mid \text{TD}(g)$, lo que por el lema IV.24 significa que $\text{TD}(g) \in \langle \text{TD}(f_1), \dots, \text{TD}(f_n) \rangle$ y hemos terminado.

Supongamos ahora que $\text{multideg}(g) < \alpha$. Usaremos que el resto de $S(f_i, f_j)$ entre $\mathcal{B}$ es 0 para encontrar una combinación lineal de g con los f_i que tenga un menor α, contradiciendo la minimalidad. Para ello, separemos los sumandos de g entre los que alcanzan α y los que no, separando además $\text{TD}(h_i)$ en el caso $\text{multideg}(h_i f_i) = \alpha$:

$$g = \sum_{\text{multideg}(h_i f_i) = \alpha} \text{TD}(h_i) f_i + \sum_{\text{multideg}(h_i f_i) = \alpha} (h_i - \text{TD}(h_i)) f_i + \sum_{\text{multideg}(h_i f_i) < \alpha} h_i f_i \quad \text{(IV.1)}$$

Todos los elementos no nulos de la segunda suma tienen multigrado estrictamente menor que α, pues hemos eliminado el término director de h_i. Por definición, todos los elementos

de la tercera suma también tienen multigrado estrictamente menor que α, y hemos asumido que $\text{multideg}(g) < \alpha$, por lo que la primera suma tiene también multigrado estrictamente menor que α. Sin embargo, cada uno de los sumandos $\text{TD}(h_i)f_i$ tiene multigrado exactamente α, por lo que estamos en las condiciones de la proposición IV.31, y podemos expresar la primera suma como

$$\sum_{\text{multideg}(h_i f_i)=\alpha} \text{TD}(h_i)f_i = \sum_{i,j} c_{i,j} S\left(\text{TD}(h_i)f_i, \text{TD}(h_j)f_j\right)$$

con $c_{i,j} \in \mathbb{K}$. Vamos a intentar expresar $S\left(\text{TD}(h_i)f_i, \text{TD}(h_j)f_j\right)$ en términos de $S(f_i, f_j)$. En primer lugar notamos que como $\text{multideg}(h_i f_i) = \alpha$, entonces $\text{MD}(\text{TD}(h_k)f_k) = x^\alpha$ para $k \in \{i, j\}$. Por tanto,

$$S(\text{TD}(h_i)f_i, \text{TD}(h_j)f_j) = \frac{x^\alpha}{\text{TD}(\text{TD}(h_i)f_i)} \text{TD}(h_i)f_i - \frac{x^\alpha}{\text{TD}(\text{TD}(h_j)f_j)} \text{TD}(h_j)f_j \qquad \text{(IV.2)}$$

Observamos que $\text{TD}(\text{TD}(h_k)f_k) = \text{TD}(h_k)\,\text{TD}(f_k)$, por lo que $\frac{1}{\text{TD}(f_k)} = \frac{\text{TD}(h_k)}{\text{TD}(\text{TD}(h_k)f_k)}$ para $k \in \{i, j\}$. Llamando $x^{\beta_{i,j}} = \text{mcm}(\text{MD}(f_i), \text{MD}(f_j))$, multiplicando y dividiendo en IV.2 por $x^{\beta_{i,k}}$ obtenemos

$$\begin{aligned} S(\text{TD}(h_i)f_i, \text{TD}(h_j)f_j) &= x^{\alpha-\beta_{i,j}} \left(\frac{x^{\beta_{i,j}}\,\text{TD}(h_i)}{\text{TD}(\text{TD}(h_i)f_i)} f_i - \frac{x^{\beta_{i,j}}\,\text{TD}(h_j)}{\text{TD}(\text{TD}(h_j)f_j)} f_j \right) \\ &= x^{\alpha-\beta_{i,j}} S(f_i, f_j) \end{aligned}$$

Como el resto de dividir $S(f_i, f_j)$ entre $\mathcal{B}$ es 0, el algoritmo 2 de división nos proporciona unos cocientes q_k tal que $S(f_i, f_j) = \sum_{k=1}^m q_k f_k$. Además, tal y como vimos en el teorema IV.18, se cumple que $\text{multideg}(q_k f_k) \leq \text{multideg}(S(f_i, f_j))$ para todos los $q_k f_k \neq 0$. Multiplicando por $x^{\alpha-\beta_{i,j}}$ obtenemos que

$$\text{multideg}(x^{\alpha-\beta_{i,j}} q_k f_k) \leq \text{multideg}(x^{\alpha-\beta_{i,j}} S(f_i, f_j))$$

Por la proposición IV.30 tenemos que $\text{TD}(S(f_i, f_j)) < \text{mcm}\left(\text{MD}(f_i), \text{MD}(f_j)\right) = \beta_{i,j}$ por lo que $\text{multideg}(x^{\alpha-\beta_{i,j}} q_k f_k) \leq \text{multideg}(x^{\alpha-\beta_{i,j}} S(f_i, f_j)) < \alpha$, lo que significa que podemos reescribir la primera suma de IV.1 como

$$\sum_{\text{multideg}(h_i f_i)=\alpha} \text{TD}(h_i)f_i = \sum_{i,j} \sum_{k=1}^m c_{i,k} x^{\alpha-\beta_{i,j}} q_k f_k$$

donde cada término tiene multigrado estrictamente menor que α y es una combinación lineal de f_k. Esto contradice la minimalidad de α, terminando la demostración. □

IV.4. Algoritmo de Buchberger

La principal aplicación del criterio de Buchberger, demostrado en el teorema IV.32 es la creación de un algoritmo para hallar una base de Gröbner a partir de una base de un ideal, tal y como se puede ver en el algoritmo 3.

Notación IV.33. *Sea $S = (b_1, \ldots, b_m)$ y a, donde $a, b_1, \ldots, b_m \in \mathbb{K}[x_1, \ldots, x_n]$, entonces denotamos* $\text{resto}(a, S)$ *como el resto producido por el algoritmo 2 de división.*

Algoritmo 3 Algoritmo de Buchberger

Entrada: $\{f_1, \ldots, f_m\} \subseteq \mathbb{K}[x_1, \ldots, x_m]$ con $f_i \neq 0$
Salida: Base de Gröbner $\mathcal{B}$ del ideal $I = \langle f_1, \ldots, f_m \rangle$

```
B ← [f_1, ..., f_m]                                   ▷ Vector
I ← {(i, j) : 1 ≤ i < j ≤ m}      ▷ I representa el conjunto de índices de S(f_i, f_j) a probar.
k ← m                                                 ▷ k es el tamaño del vector B
while S ≠ ∅ do
    Elegir (i, j) ∈ S
    I ← I \ {(i, j)}
    (f_i, f_j) ← (B[i], B[j])                         ▷ f_i es el elemento i-ésimo de B.
    r ← resto(S(f_i, f_j), B)
    if r ≠ 0 then
        k ← k + 1
        B.append(r)                                   ▷ Añadir r al final del vector B
        I ← I ∪ {(i, k) : 1 ≤ i ≤ k − 1}
    end if
end while
```

Comencemos demostrando la corrección del algoritmo, ya que la terminación es un poco más delicada.

Teorema IV.34. ***(Corrección del algoritmo de Buchberger)*** *Si el algoritmo 3 termina, entonces el resultado $\mathcal{B}$ es una base de Gröbner del ideal $I = \langle f_1, \ldots, f_m \rangle$.*

Demostración. Sea $\mathcal{B} = \{g_1, \ldots, g_s\}$ al finalizar el algoritmo. Veamos primero que $\mathcal{B} \subseteq I$, que es trivialmente cierto al comienzo, pues $f_1, \ldots, f_m \in I$ y lo es en cada paso, pues al expandir $\mathcal{B}$ añadimos el resto r de dividir $S(g_i, g_j)$ entre $\mathcal{B}' \subseteq I$, siendo $\mathcal{B}'$ el valor intermedio de $\mathcal{B}$ al principio de la iteración. Como $g_i, g_j \in \mathcal{B}' \subseteq I$, entonces $S(g_i, g_j) \in I$. Como el divisor y los dividendos están en I, podemos expresar el resto como combinación lineal de ellos, y por tanto $r \in I$, y $\mathcal{B} \subseteq I$. Además, $\{f_1, \ldots, f_m\} \subseteq \mathcal{B}$ y es una base de I, por lo que $\mathcal{B}$ también será una base de I.

Veamos que al terminar el bucle `while` tenemos que $\text{resto}(S(g_i, g_j), \mathcal{B}) = 0$ para todo $i \neq j$, por lo que aplicando el teorema IV.32 estaría demostrado que $\mathcal{B}$ es una base de Gröbner.

Notemos primero que si $\text{resto}(S(g_i, g_j), \mathcal{B}) = 0$ y expandimos $\mathcal{B}$ a $\mathcal{B}'$ añadiendo elementos al final de la tupla, entonces $\text{resto}(S(g_i, g_j), \mathcal{B}') = 0$, pues en el bucle `for` del algoritmo 2 de la división comprobaremos primero los polinomios de $\mathcal{B}$, y en el mismo orden y como el resto era 0, en toda iteración se producía una división, por lo que se sale del bucle `for` mediante la instrucción `break` antes de que los nuevos polinomios añadidos a la base puedan afectar al resultado.

Notamos también que $S(g_j, g_i) = -S(g_i, g_j)$, por la construcción de los polinomios-S. Además, si $\text{resto}(S(g_i, g_j), \mathcal{B}) = 0$, entonces también tendremos que $\text{resto}(-S(g_i, g_j), \mathcal{B}) = 0$, pues la única decisión que afecta al resto es si el término director de algún g_i divide al término director del dividendo, y al trabajar en un cuerpo $\mathbb{K}$, los coeficientes no afectan a la divisibilidad de los términos, que está únicamente determinada por los monomios.

Por tanto, basta con demostrar que $\text{resto}(S(g_i, g_j), \mathcal{B}') = 0$ para el $\mathcal{B}'$ de alguna iteración y para $i < j$. En primer lugar, entramos al bucle con cada (i, j) en esas condiciones, pues inicializamos

S con todas las tuplas requeridas, y al añadir un elemento al final de S, añadimos todas las tuplas requeridas con los elementos anteriores. Además, al final de cada iteración del bucle nos aseguramos que $\text{resto}(S(g_i, g_j), \mathcal{B}') = 0$, donde $\mathcal{B}'$ es la tupla $\mathcal{B}$ al finalizar esa iteración. Si $r = 0$, claramente se cumple. Si no es 0, estamos añadiendo el resto de la división a $\mathcal{B}'$, por lo que el resto de la nueva división entre $S(g_i, g_j)$ y $\mathcal{B}'$ sí será 0. □

La clave para demostrar la terminación del algoritmo está en comprobar que si el algoritmo no termina tendríamos una cadena infinita de inclusiones estrictas de ideales.

Para demostrar que esto es imposible, es necesario utilizar el teorema de la base de Hilbert y que el anillo $\mathbb{K}[x_1, \ldots, x_n]$ es Noetheriano. Una demostración puede hallarse en el apéndice B.

Teorema IV.35. ***(Terminación del algoritmo de Buchberger)*** *El algoritmo 3 termina en tiempo finito.*

Demostración. Supongamos que el algoritmo no termina en tiempo finito. Eso significa que debemos añadir infinitas veces elementos al conjunto S, lo que significa que debemos añadir infinitas veces el resto r a la base $\mathcal{B}$. Podemos definir una cadena de inclusiones de conjuntos

$$\mathcal{B}_1 \subsetneq \mathcal{B}_2 \subsetneq \mathcal{B}_3 \subsetneq \cdots$$

Cada uno de los $\mathcal{B}_{i+1}$ se forma añadiendo a $\mathcal{B}_i$ el resto de una división entre $\mathcal{B}_i$. Viendo el algoritmo 2, notamos que si el resto es no nulo es porque el monomio director de ningún divisor divide al monomio director del dividendo, que pasa a añadirse al resto. En concreto, el monomio director del resto no es dividido por el monomio director de ningún dividendo. Por el lema IV.24, esto significa que si $\mathcal{B}_i = \{g_1, \ldots, g_m\}$, entonces el resto r de cualquier división entre $\mathcal{B}_i$ satisfará que $\text{TD}(r) \notin \langle \text{TD}(\mathcal{B}_i) \rangle := \langle \text{TD}(g_1), \ldots, \text{TD}(g_m) \rangle$. Como $\mathcal{B}_{i+1} = \mathcal{B}_i \cup \{r\}$, esto significa que $\text{TD}(\mathcal{B}_i) \subsetneq \text{TD}(\mathcal{B}_{i+1})$. Como el algoritmo no termina, esto sucede infinitas veces, generando la cadena ascendente de ideales

$$\text{TD}(\mathcal{B}_1) \subsetneq \text{TD}(\mathcal{B}_2) \subsetneq \text{TD}(\mathcal{B}_3) \subsetneq \cdots$$

lo cual entra en contradicción con que $\mathbb{K}[x_1, \ldots, x_n]$ sea Noetheriano por el corolario B.4, por lo que el algoritmo debe terminar en tiempo finito. □

Observación IV.36. ***(Complejidad computacional)*** *Aún usando los algoritmos más eficientes para computar bases de Gröbner que se conocen en la actualidad (los algoritmos F4 y F5 de Faugère [11, 12], que son bastante más eficientes que el algoritmo 3) los grados y coeficientes de los polinomios intermedios pueden explotar, requiriendo cantidades excesivas de tiempo y memoria.*

De hecho, se ha demostrado que al computar la base de Gröbner para un ideal cuyos polinomios todos tienen grado igual o menor que d, es posible que aparezcan polinomios intermedios de grado proporcional a 2^{2^d} [21], es decir, crecimiento doblemente exponencial. El número de variables también es un factor crítico.

Por lo tanto, en la práctica, las bases de Gröbner son útiles para un número reducido de polinomios de grado reducido. En el capítulo V veremos como evitar parcialmente estas restricciones y poder garantizar la seguridad de circuitos grandes.

Capítulo V

Algoritmo modular

En este capítulo desarrollamos un algoritmo modular para demostrar la seguridad de circuitos Circom grandes, combinando heurísticas con los métodos algebraicos del capítulo IV.

V.1. Motivación para un algoritmo modular

En el capítulo III estudiamos como convertir el problema de la verificación de circuitos Circom (consistente en demostrar que para las entradas dadas existen unos únicos valores para las salidas que puedan ser solución del sistema R1CS) a un problema de insatisfacibilidad de un sistema polinómico. En el capítulo IV vimos un algoritmo no completo para verificar la insatisfacibilidad de un sistema polinómico. Sin embargo, también vimos en la observación IV.36 que la complejidad de decidir la insatisfacibilidad puede crecer muy rápidamente con el número de variables, grado de los polinomios y número de restricciones.

Por tanto, aunque para demostrar la seguridad de un circuito podríamos hallar su sistema R1CS, añadir el polinomio de prohibición del teorema III.8 y decidir la satisfacibilidad utilizando bases de Gröbner, la aplicabilidad de ese método estaría limitada a los circuitos más pequeños.

En lugar de eso, vamos a desarrollar un algoritmo modular que demuestre por separado propiedades de seguridad para subcomponentes más pequeños utilizando las bases de Gröbner y después una inductivamente dichas demostraciones de los subcomponentes para demostrar la seguridad del circuito completo.

Para ello, este algoritmo tendrá la siguiente filosofía:

- **Aplicabilidad práctica**: El objetivo no es obtener un algoritmo completo, sino un algoritmo que utilice **heurísticas** basadas en las particularidades de los sistemas que aparecen en los circuitos Circom prácticos para poder demostrar la mayoría de ellos de forma eficiente. Esto significa **sacrificar completitud**, que con los métodos que hemos presentado es computacionalmente infactible.
- **Seguridad**: Nos vamos a centrar en demostrar la seguridad (no seguridad fuerte) para una **única entrada** dada. Eso significa que nuestro algoritmo recibirá el circuito Circom y un *testigo* y garantizará que la salida del componente principal es única (está *fija*) para la entrada recibida. Vamos a demostrar únicamente la seguridad para una entrada dada debido al número de optimizaciones agresivas que podemos realizar, con el objetivo de poder verificar circuitos grandes. Una vez hemos demostrado que cierta señal está fija, podemos sustituir las apariciones de dicha señal por su valor, simplificando en muchos casos el problema de forma notable.
- **Modularidad**: Vamos a intentar reducir al máximo posible el tamaño de los sistemas de

ecuaciones a los que aplicamos el método de las bases de Gröbner. En concreto, al tratar un componente, vamos a asumir que **cada subcomponente suyo es seguro** y repetiremos el proceso para todos los subcomponentes para demostrar la seguridad modularmente. En el componente padre, cada subcomponente se tratará como una caja negra segura, y luego verificaremos por separado cada subcomponente con el mismo algoritmo para verificar que efectivamente es seguro. Esta asunción suele funcionar en los circuitos reales y nos permite dividir la verificación de seguridad en subproblemas más manejables, sacrificando completitud por eficiencia.

V.2. Grafo de verificación

El algoritmo modular trabaja sobre un grafo que representa el circuito Circom. Este grafo es la entrada del algoritmo. Dos ejemplos de grafos de verificación se pueden ver en la figura V.1.

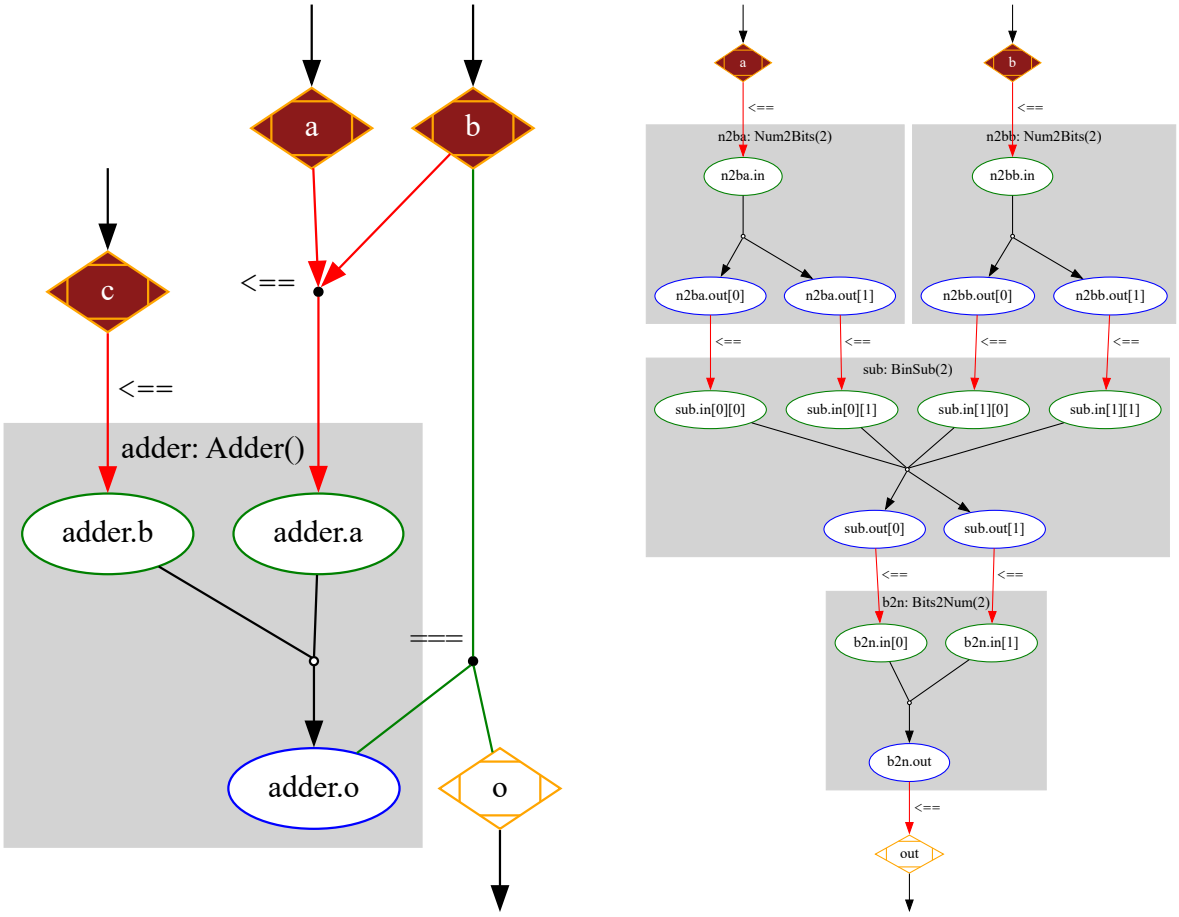

Figura V.1: Dos ejemplos de grafos de verificación

Los grafos de la figura V.1 han sido creados automáticamente a partir de un circuito Circom por una herramienta escrita en Rust [6] con el propósito de ayudar a la depuración. También puede mostrar estados intermedios del grafo de verificación en distintas etapas del algoritmo.

Definición V.1. ***(Grafo de verificación)*** *Llamaremos grafo de verificación de un componente a un conjunto de tres hipergrafos distintos (grafos cuyas aristas pueden relacionar cualquier número de vértices).*

Todos los hipergrafos tienen el mismo conjunto de vértices, que son las señales del componente que se está verificando que pueden ser o bien entradas, salidas, señales intermedias, señales de entrada de un subcomponente o señales de salida de un subcomponente. Nótese que las señales intermedias de cada subcomponente no están representadas en el grafo de verificación del componente padre, lo estarán en el grafo de verificación del subcomponente.

Cada hipergrafo tiene un conjunto de aristas diferente. El primero de ellos consiste en las aristas dirigidas de las asignaciones duales `<==`*. Existe una arista por cada una de dichas asignaciones. El conjunto de vértices "cabeza" son las señales que aparecen en el lado derecho de la asignación* `<==` *y el único vértice "cola" es la señal asignada. El segundo consiste en las aristas no dirigidas formadas por las restricciones de igualdad* `===`*. El conjunto de vértices relacionados por cada restricción de igualdad son las señales que aparecen en dicha restricción. El tercer tipo de aristas consiste en las aristas dirigidas formadas por los diferentes subcomponentes. Existe una arista por subcomponente instanciado. El conjunto de vértices cabeza de cada arista son las entradas del subcomponente y el conjunto de vértices cola las salidas del subcomponente.*

Podemos interpretar este conjunto de tres hipergrafos como un gran único hipergrafo en el que hay tres tipos diferentes de aristas, dos dirigidos y uno no dirigido.

El algoritmo irá modificando este grafo eliminando los nodos que ya hayan sido *fijados* (de los que se haya demostrado unicidad). El objetivo es eliminar del grafo todas las salidas, y así habremos logrado fijar su valor para cierta entrada dada y demostrar la seguridad del circuito para dicha entrada.

Una vez demostremos la unicidad de una señal, podemos sustituir la señal por su valor en el testigo computado por el programa WebAssembly (pues sabemos que es una solución válida, y como es única, es forzosamente esa) y propagarla para así demostrar la unicidad de más señales.

Además del grafo de verificación, mantendremos también una lista de nodos fijados que todavía no han sido procesados y eliminados del grafo, una lista de subcomponentes para verificar (pues verificaremos la seguridad de cada subcomponente independientemente, y la seguridad del componente padre estará condicionada a la seguridad de sus subcomponentes) y una lista de sistemas polinómicos para los que demostrar la unicidad de ciertas variables utilizando el teorema III.8 y las técnicas del capítulo IV basadas en bases de Gröbner. Intentaremos que el tamaño de cada uno de los sistemas polinómicos sea lo más reducido posible.

V.3. Algoritmo

El primer paso es, dado el grafo de verificación de cierto componente de entrada, determinar qué nodos están inicialmente fijos y añadirlos a la lista de nodos fijados. Son los siguientes:

- Las **señales de entrada**, pues queremos verificar que las salidas se fijen para una entrada determinada fija.
- Las señales objeto de una asignación `<==` en el que el lado derecho es **constante**. Después de fijar el lado izquierdo, eliminamos dicha asignación.
- Las señales que sean la **única señal** de una restricción `===` lineal donde el resto de elementos son constantes y el coeficiente asociado a dicha señal es no nulo. Después de fijar la señal, borramos la restricción.

- Las salidas de subcomponentes que no tienen entradas (componentes cuya función es devolver una constante). Además, añadimos dicho subcomponente a la lista de componentes a demostrar recursivamente.

Una vez tenemos las señales iniciales fijas, utilizamos el algoritmo 4 para propagarlas.

Algoritmo 4 Propagación de señales fijadas

Entrada: $\mathcal{G}$ grafo de verificación, x señal fijada, w testigo del circuito
Salida: F conjunto de valores a fijar, C lista de subcomponentes a verificar

```
F ← ∅
C ← ∅
for all a in G.asignaciones_duales tal que x ∈ a.rhs do          ▷ Asignaciones <==
    a.sustituir_señal(x → w[x])        ▷ Sustituimos el valor constante del testigo de la señal
    if a.rhs es constante then
        F ← F ∪ {a.lhs}                                          ▷ Fijamos la señal asignada
        G.eliminar_asignacion_dual(a)
    end if
end for
for all r in G.restricciones tal que x ∈ r do                    ▷ Restricciones ===
    r.sustituir_señal(x → w[x])        ▷ Sustituimos el valor constante del testigo de la señal
    if r es lineal y contiene una única señal s y su coeficiente es no nulo then
        F ← F ∪ {s}                                      ▷ Fijamos la señal de la restricción
        G.eliminar_restriccion(r)
    end if
end for
if ∃c ∈ G.componentes tal que x ∈ c.inputs then
    c.inputs ← c.inputs \ {x}
    if c.inputs = ∅ then                                 ▷ Todas las entradas han sido fijadas
        F ← F ∪ c.outputs                                          ▷ Fijamos todas las salidas
        C ← C ∪ {c}
        G.eliminar_componente(c)
    end if
end if
F ← F ∩ G.nodos        ▷ Nos quedamos con los nodos que no hayan sido fijados previamente
G.eliminar_nodo(x)     ▷ Borra toda referencia a la señal, como la pertenencia a componentes
```

Teorema V.2. *El algoritmo 4 es correcto.*

Demostración. En el caso de las asignaciones <==, si tras sustituir el nuevo valor fijado (que podemos hacer por haber demostrado que ese valor es fijo) el lado derecho de la asignación es constante, entonces el lado izquierdo también será fijo (y será igual a su valor computado en el testigo), pues es la única solución a la restricción generada por Circom. Lo mismo sucede si tras sustituir el valor fijado, una señal s es la única en una restricción ===, esta es lineal y el coeficiente asociado c es no nulo. Es decir, la restricción se puede expresar como $cs = a$ para $c, a \in \mathbb{F}_p$ fijos. Como $c \neq 0, s = c^{-1}a$, quedando el valor de s fijo. Finalmente, debido a la heurística que estamos usando, sacrificando completitud en pos de ganar eficiencia práctica, asumimos que todos los subcomponentes son seguros. Entonces, si todas las entradas de un subcomponente han sido fijadas, las salidas también lo estarán. Añadimos dicho subcomponente a una lista de

componentes a verificar recursivamente, para posteriormente verificar que efectivamente dicho componente es seguro. □

Después de propagar iterativamente los valores fijados hasta que no quede ningún valor fijo es posible que ya hayamos fijado todas las salidas del componente, en cuyo caso hemos terminado la demostración y el componente es seguro (condicionado a la seguridad de los subcomponentes).

Sin embargo también es posible que no hayamos fijado todas las salidas pero que haya varias señales conectadas por restricciones `===` que formen un sistema polinómico que garantice la unicidad de algunas de sus señales. Podemos demostrar la unicidad de esas señales usando bases de Gröbner y después sustituir dichas señales por sus valores en el testigo.

Si incluimos una entrada o salida de un subcomponente en cierta *componente conexa* de restricciones `===`, incluiremos todas las entradas y salidas de dicho subcomponente, por simplicidad al generar el sistema polinómico del que demostraremos la unicidad de ciertas variables utilizando las herramientas algebraicas del capítulo IV.

El algoritmo que computa una componente conexa es el algoritmo 5 que añade recursivamente elementos a una componente conexa utilizando una búsqueda en profundidad (DFS).

Algoritmo 5 Creación de componente conexa

Entrada: $\mathcal{G}$ grafo de verificación, S conjunto de la componente conexa, x señal a visitar
Salida: Nueva componente conexa, posiblemente ampliada

```
function DFS(𝒢, S, x)
    if x ∈ S then
        return S
    end if
    S ← S ∪ {x}
    for all r in 𝒢.restricciones tal que x ∈ r do          ▷ Restricciones ===
        for all y in r.señales tal que y ≠ x do
            S ← S ∪ DFS(𝒢, S, y)
        end for
    end for
    if ∃c componente tal que x ∈ c.inputs ∪ c.outputs then
        for all y in c.inputs ∪ c.outputs do
            S ← S ∪ DFS(𝒢, S, y)
        end for
    end if
    return S
end function
```

Dada una componente conexa, podemos generar un sistema polinómico para demostrar la unicidad de ciertas señales de dicha componente conexa usando bases de Gröbner, para poder después sustituir las señales por sus valores en el testigo. El método de las bases de Gröbner necesita como entrada un conjunto de ecuaciones polinómicas y una lista de señales de las que demostrar la unicidad, junto con sus valores en el testigo (para generar la restricción de prohibición vista en el teorema III.8).

El conjunto de ecuaciones polinómicas es el formado por las restricciones `===` y las asignaciones

<== entre señales de la componente conexa. También debemos marcar un conjunto de señales para las que demostrar la unicidad (fijar) utilizando bases de Gröbner. Las salidas del componente principal serán unas de esas señales, pues son el objetivo principal. También marcaremos como variables a fijar aquellas que formen parte del lado derecho de una asignación <== que salga fuera de la componente conexa.

El algoritmo 6 calcula esos datos para una componente conexa dada, que se pueden utilizar junto al teorema III.8 y los algoritmos vistos en el capítulo IV para demostrar la unicidad del conjunto de señales marcadas a fijar y luego sustituir las señales por sus valores en el testigo.

Algoritmo 6 Sistema polinómico para una componente conexa

Entrada: $\mathcal{G}$ grafo de verificación, S componente conexa, w testigo
Salida: C conjunto de ecuaciones, F conjunto de señales a fijar

```
C ← ∅
for all r in G.restricciones tal que r.señales ⊆ S do        ▷ Restricciones ===
    C ← C ∪ {r}
end for
for all a in G.asignaciones_duales tal que a.señales ⊆ S do        ▷ Asignaciones <==
    C ← C ∪ {a.lhs − a.rhs = 0}
end for
for all c in G.componentes tal que c.inputs ∪ c.outputs ⊆ S do
    C ← C ∪ c.expandir_R1CS()    ▷ Añadimos las ecuaciones del R1CS del subcomponente
end for
F ← S ∩ G.outputs      ▷ Marcamos las salidas del componente principal como señales a fijar
F ← F ∪ {x ∈ S : ∃a ∈ G.restricciones tal que x ∈ a.rhs ∧ a.lhs ∉ S}
```

Es posible que haya más de una componente conexa de restricciones === y que algunas dependan de otras por asignaciones <==, así que debemos establecer alguna heurística para elegir qué componente conexa será tratada primero usando bases de Gröbner. Primero, calcularemos todas las posibles componentes conexas de restricciones ===. Si existe alguna componente conexa que no tenga ninguna asignación <== entrante desde fuera, entonces la tratamos con bases de Gröbner. Si no existe ninguna, escogemos alguna asignación <== que no satisfaga la condición y combinamos todas las componentes conexas de las señales participantes en la asignación en una sola, repitiendo este proceso varias veces si es necesario.

Ahora estamos en condiciones de estudiar el algoritmo completo de verificación modular, cuyo pseudocódigo puede verse en el algoritmo 7. Hay que recordar que este algoritmo no es completo, hemos utilizado algunas heurísticas a cambio de poder tratar circuitos grandes. Sin embargo, sí que es correcto, como podemos ver en el teorema V.3.

Teorema V.3. ***(Corrección del algoritmo modular)****. El algoritmo 7 termina y es correcto.*

Demostración. Veamos primero que el algoritmo **termina**. Al comienzo de cada iteración del bucle principal propagamos los nodos fijados y en concreto eliminamos del grafo $\mathcal{G}$ todos los nodos fijados. Es por tanto suficiente con demostrar que en cada iteración que no terminemos o bien borraremos al menos un nodo de $\mathcal{G}$ o que al menos se marcará un elemento como fijado para demostrar que el algoritmo termina (pues al grafo G nunca se añaden nodos). Si $\mathcal{G}.outputs = \emptyset$, hemos acabado. Si hay alguna salida pero no hay ninguna restricción ===, también hemos acabado. En caso de que sí que haya alguna restricción ===, entonces extendemos la componente

Algoritmo 7 Verificación modular

Entrada: $\mathcal{G}$ grafo de verificación, w testigo
Salida: Resultado de la verificación

Inicializar F el conjunto de nodos fijados y C conjunto de subcomponentes a verificar como se describió al principio de la sección
while true **do**
  while $F \neq \emptyset$ **do**
    Elegir $x \in F$
    $(F', C') \leftarrow$ propagar_señal_fijada$(\mathcal{G}, x, w)$ ▷ Algoritmo 4
    $F \leftarrow (F \cup F') \setminus \{x\}$
    $C \leftarrow C \cup C'$
  end while
  if $\mathcal{G}.outputs = \emptyset$ **then** ▷ Todas las salidas han sido fijadas
    Verificar recursivamente los subcomponentes de la lista C
    if Todos los subcomponentes son seguros **then**
      return "El componente es seguro"
    end if
  else ▷ Hay al menos una salida sin fijar después de la propagación
    if $\mathcal{G}.restricciones \neq \emptyset$ **then** ▷ Hay alguna restricción ===
      $A \leftarrow \mathcal{G}.señales$ ▷ Nodos sin componente conexa todavía
      $B \leftarrow \emptyset$ ▷ Conjunto de componentes conexas
      while $A \neq \emptyset$ **do**
        Elegir $x \in A$
        $S \leftarrow$DFS$(\mathcal{G}, \emptyset, x)$ ▷ Algoritmo 5
        $A \leftarrow A \setminus S$
        $B \leftarrow B \cup \{A\}$
      end while
      while true **do** ▷ Hasta tener una componente conexa suficientemente grande
        if $\exists CC \in B$ tal que $\nexists$ asignación <== entrante desde otra c.c. **then**
          Creamos un sistema polinómico a fijar para CC según el algoritmo 6
          Añadimos al sistema polinómico el polinomio prohibición (definición III.7)
            para las señales a fijar y sus valores en el testigo w
          if El sistema no tiene ninguna solución utilizando bases de Gröbner **then**
            Borrar de $\mathcal{G}$ todos los nodos de CC no fijados
            Añadir a F los nodos fijados de CC
            break
          else
            return "El componente es posiblemente inseguro"
          end if
        else
          Elegir una asignación dual a que no satisfaga la condición
          Combinar todas las componentes conexas de $a.lhs$ y $a.rhs$ en una sola
        end if
      end while
    else ▷ No hay ninguna restricción === que pueda fijar la salida
      return "El componente es posiblemente inseguro"
    end if
  end if
end while

conexa hasta que sea suficientemente grande para no tener dependencias entrantes. Ese bucle también acaba, pues en cada iteración se obtiene una componente conexa más grande, y cuando la componente conexa es todo el grafo $\mathcal{G}$ se satisface la condición de que no tenga dependencias externas. Si el sistema a fijar tiene alguna solución utilizando bases de Gröbner, hemos acabado. Finalmente, si no, borraremos de $\mathcal{G}$ todos los nodos no fijados y marcaremos como fijados los nodos a fijar. Como toda componente conexa tiene al menos un elemento, al menos se borrará o marcará como fijado un elemento de $\mathcal{G}$ en cada iteración en la que el algoritmo no termine, por lo que el algoritmo acabará terminando.

Demostremos ahora la **corrección**, es decir, que si todas las salidas del componente han sido eliminadas del grafo $\mathcal{G}$, entonces el componente es seguro (es decir, para cierta entrada, las salidas están fijas). Las salidas solo han podido ser eliminadas del grafo mediante la propagación del algoritmo 4, pues las salidas del componente siempre se consideran un nodo a fijar en las componentes conexas.

Hemos demostrado en 4 que la propagación de señales fijas es correcta, así que resta comprobar que las señales marcadas como fijas por un procedimiento diferente a la propagación son también fijas. Las señales marcadas como fijas al comienzo del algoritmo son aquellas enumeradas al comienzo de sección: señales de entrada (que son fijas por definición de la propiedad de seguridad que queremos demostrar, que es que para una entrada fija hay una única salida), señales objeto de una asignación dual por parte de constantes, única señal en una restricción `===` con coeficiente no nulo (por lo que se puede multiplicar por el inverso del coeficiente y despejar dicha señal, fijándola) y salidas de subcomponentes sin entradas (que es la heurística que hemos tomado, verificamos que todos los subcomponentes sean seguros recursivamente, así que sus salidas deben estar fijadas).

El otro lugar donde se marcan nodos como fijos es al procesar una componente conexa. Como hemos utilizado el polinomio de prohibición, aplicando el teorema III.8 (a un "circuito" cuyo R1CS es generado por el algoritmo 6 y cuyas "salidas" son los nodos a fijar del algoritmo 6) y el teorema IV.6, podemos garantizar que si el procedimiento de bases de Gröbner nos indica que el sistema polinómico no tiene solución, las señales a fijar tendrán una única solución, es decir, estarán fijas. Además, las variables a fijar según el algoritmo 6 son las necesarias: las salidas del componente y las señales que formen parte del lado derecho de alguna asignación hacia afuera del componente, que son las únicas formas que tienen las señales de la componente conexa de afectar a nodos fuera de la componente conexa (pues la componente conexa incluye todas las señales conectadas por componentes y restricciones `===` entre sí). Por tanto, todas las señales marcadas como fijas lo son, y la propagación también es correcta, por lo que si una salida del componente ha sido marcada como fija, es realmente fija. □

Observación V.4. ***(Optimización binaria)*** *En la práctica, muchos de los sistemas a tratar utilizando las bases de Gröbner están formados por señales binarias. Según la definición III.7, el término del polinomio de prohibición para una variable x y su valor w en el testigo es $((x - w) \cdot u - 1)$, donde una nueva variable u es necesaria para que el término se pueda anular para cualquier $x \neq w$. Sin embargo, si x es binaria, es decir, tenemos una restricción $x(x-1) = 0$, entonces podemos simplificar el término del polinomio de prohibición a $(x - (1 - w))$, ya que si una variable binaria no es 0, forzosamente debe ser 1 y viceversa. Por tanto, podemos ahorrarnos la variable extra u y disminuir en 1 el grado del polinomio de prohibición por cada variable binaria, que como vimos en la observación III.10 puede afectar en gran manera al rendimiento.*

Capítulo VI

Resultados

En este capítulo se analizan los resultados de ejecutar una implementación del algoritmo modular descrito en el capítulo V. Se intentan verificar varios repositorios Circom de código abierto utilizados en la práctica, para analizar e interpretar los resultados.

Para realizar las pruebas se ha implementado el algoritmo modular en el lenguaje de programación Rust [6], ampliamente usado en criptografía debido a sus garantías de seguridad, y es el mismo lenguaje en el que está escrito el compilador del lenguaje Circom, lo que facilita la integración. Para la computación de bases de Gröbner, tras considerar diversas herramientas de álgebra computacional, hemos decidido finalmente utilizar CoCoA [8].

VI.1. Proyectos

Para probar el sistema se han usado repositorios de código abierto que contienen código real que se utiliza en diferentes sistemas criptográficos. Se han utilizado las entradas del conjunto de pruebas de cada uno de los repositorios para cada circuito.

En la tabla VI.1 podemos ver para cada repositorio el número total de circuitos probados, cuántos han sido verificados correctamente, cuántos han sido verificados condicionalmente a que el módulo `Num2Bits(N)` (que veremos más adelante) sea seguro para un N grande, cuántos no han podido ser verificados debido a una heurística fallida (por ejemplo, se ha intentado demostrar que un subcomponente es seguro cuando es falso, y se deberían haber expandido sus restricciones en el componente padre, ya que aunque el subcomponente sea inseguro, el uso concreto como parte del componente padre puede hacer el total seguro) y cuántos circuitos han sido marcados correctamente como inseguros (es decir, cuántos errores de seguridad se han encontrado).

Repositorio	Total	Verificados	Condicionados a `Num2Bits`	Heurística fallida	Inseguros
CircomLib [2]	27	9	15[1]	2	1
Circom-ECDSA [1]	10	4	0	6	0
CircomLib-ML [3]	30	15	15	0	0
ED25519 [4]	11	2	0	4	5

Tabla VI.1: Resultado de intentos de verificación de varios repositorios Circom

Exploramos ahora más en detalle cada uno de los repositorios que se han probado:

[1] Dos pruebas están condicionadas también al componente `BinSub` además de `Num2Bits`. Sin embargo, después de propagar las entradas, el sistema de ecuaciones de `BinSub` es idéntico a `Num2Bits`.

- **CircomLib** [2] es la librería estándar oficial de Circom, que ofrece múltiples circuitos para funcionalidades generales y comunes en situaciones prácticas.

  Tras ejecutar el reportorio de tests de CircomLib, hemos sido capaces de verificar la seguridad de casi todos los módulos asumiendo que un subcomponente llamado `Num2Bits` es seguro. Como veremos en el apéndice C.1, dicho módulo recibe como entrada un elemento de N bits de $\mathbb{F}_p$ y devuelve N señales binarias con la descomposición en binario de dicho número. Muchos de los circuitos de prueba de CircomLib tienen como subcomponente una instancia de `Num2Bits(N)` para un $N > 250$, que supera por mucho las capacidades del método de bases de Gröbner. En un tiempo razonable (menos de un minuto), únicamente hemos podido verificar la seguridad de `Num2Bits(N)` para $N \leq 12$.

  Sin embargo, hemos sido capaces de verificar todos los circuitos de prueba (asumiendo la seguridad de `Num2Bits`) excepto alguno relacionado con puntos en curvas elípticas. Dichos circuitos instancian componentes sin salidas, cuyo objetivo es simplemente generar restricciones sobre señales del componente padre. Esto choca con nuestras heurísticas, que asumen que todos los subcomponentes son seguros, y se pueden tratar individualmente de forma aislada. Sin embargo, este tipo de circuitos son una minoría en CircomLib, y hemos sido capaces de verificar la gran mayoría de circuitos de pruebas (condicionado a `Num2Bits`).

  Al verificar CircomLib, el algoritmo modular ha detectado correctamente un circuito inseguro, el módulo `Decoder`. Se puede encontrar más información sobre este error de seguridad en el apéndice D.1.

- **Circom-ECDSA** [1] es un repositorio que incluye una muestra de concepto de la implementación de algoritmos de firmas digitales mediante curvas elípticas en Circom.

  Al contrario que el conjunto de pruebas de CircomLib, Circom-ECDSA utiliza mucho la instanciación de componentes simplemente para añadir restricciones al componente padre, por lo que no hemos sido capaces de verificar la mitad de circuitos de prueba. Sí que hemos sido capaces de verificar la otra mitad de circuitos, condicionados a la seguridad de `Num2Bits` de nuevo. Circom-ECDSA tiene una librería para operar con números de tamaño arbitrario (BigInts), y algunos de sus circuitos resultan en sistemas polinómicos interesantes a la hora de la verificación de seguridad. En el apéndice C.2 estudiamos uno de ellos, `Split`.

- **CircomLib-ML** [3] es una librería de Machine Learning implementada usando Circom. Permite expresar redes neuronales y garantizar que la clasificación se ha producido siguiendo las etapas de dichas redes neuronales sin filtrar las entradas, que pueden ser secretas.

  CircomLib-ML sí que satisface las heurísticas propuestas para los subcomponentes. La mitad de los circuitos de prueba han sido verificados completamente, y la otra mitad han sido verificados condicionados a la seguridad de `Num2Bits(N)` para N grande.

- **ED25519-Circom** [4] es una prueba de concepto de una implementación de operaciones sobre curvas elípticas y verificación de firmas para el esquema de firma digital ED25519.

  En este repositorio se han encontrado varios errores de seguridad, casi todos por la confusión entre asignaciones inseguras `<--` y asignaciones duales `<==` por parte de los autores. En el apéndice D.2 estudiamos el caso de uno de ellos, `fulladder`.

  También ha habido algunos circuitos que no han podido ser verificados porque se incumple la heurística que asume que todos los subcomponentes son seguros, y el resto han podido ser verificados satisfactoriamente.

VI.2. Interpretación de los resultados

En general, hemos visto que el algoritmo modular ha sido de gran ayuda. La propagación de constantes fijadas ha simplificado muchos problemas, y solamente algunos módulos específicos han debido ser resueltos utilizando bases de Gröbner, consiguiendo verificar un gran número de circuitos con aplicación real.

A pesar de perder la completitud de la insatisfacibilidad de los sistemas polinómicos al pasar a un cuerpo finito $\mathbb{F}_p$ por no añadir los polinomios $x^p - x$ (por ser computacionalmente infactible para un p gigante) que vimos en la observación IV.7, en ningún ejemplo práctico hemos encontrado el caso de un sistema polinómico a fijar que sí fuese insatisfacible pero los métodos desarrollados con bases de Gröbner no pudiesen garantizar la insatisfacibilidad.

Sí que nos hemos encontrado problemas de completitud en la práctica en el algoritmo modular. La asunción de que todos los subcomponentes son seguros por sí mismos y se deben fijar primero las entradas es demasiado fuerte. Como vimos en Circom-ECDSA, algunos componentes instancian otros componentes simplemente para imponer restricciones sobre algunas señales. Se podría diseñar una heurística para detectar esos casos y en lugar de verificar recursivamente el subcomponente, expandir sus restricciones en el componente padre, para tenerlas en cuenta en el cómputo de componentes conexas de restricciones `===`. Una posible heurística sería marcar como componentes que simplemente imponen restricciones aquellos que o bien no tienen salidas, o sus salidas se utilizan exclusivamente en restricciones `===`.

El ejemplo de `Num2Bits` también muestra la importancia de trabajar modularmente. Las bases de Gröbner son efectivas para un número reducido de señales y grado de polinomios, tal y como hemos visto. Hemos encontrado muchos circuitos de los que hemos podido demostrar la seguridad para cierta entrada asumiendo que el módulo `Num2Bits` para un número grande de bits es seguro. Todas las salidas de este módulo son binarias. Además de la optimización comentada en la observación V.4, se podría considerar utilizar otras técnicas específicas para casos binarios, que tal vez podrían comportarse mejor que la técnica general usando bases de Gröbner. Una de estas posibles técnicas sería utilizar resolvedores SMT (Satisfiability Modulo Theories) [23].

Capítulo VII

Conclusiones

Hemos estudiado los conceptos básicos de las pruebas de conocimiento nulo ZK-SNARK y el lenguaje de diseño de circuitos aritméticos Circom. En particular, hemos hallado una nueva forma de reducir el problema de seguridad de circuitos Circom (para una entrada fijada y para todas las entradas) a la insatisfacibilidad de un sistema polinómico con coeficientes en un cuerpo finito.

Se han estudiado métodos algebraicos para decidir la insatisfacibilidad de dichos sistemas polinómicos utilizando bases de Gröbner, aunque hemos descubierto que su aplicabilidad práctica es limitada debido a la gran complejidad computacional.

Hemos implementado un algoritmo modular basado en heurísticas y los métodos algebraicos descritos anteriormente para poder verificar circuitos más grandes sacrificando completitud.

Los resultados han sido prometedores, pues hemos sido capaces de verificar la seguridad de una buena parte de circuitos de repositorios de código abierto con aplicación práctica, con un gran número de restricciones, detectando incluso un módulo inseguro en CircomLib. Sin embargo, también nos hemos encontrado con problemas demasiado grandes para bases de Gröbner, como `Num2Bits`, que necesitan técnicas más especializadas para ser verificados eficientemente. Igualmente hemos descubierto la necesidad de implementar alguna heurística para determinar si se debe asumir la seguridad de un subcomponente o si se deben expandir sus restricciones al componente padre.

VII.1. Trabajo futuro

Basándonos en los resultados del trabajo realizado, habría algunas líneas de investigación futura interesantes:

- Estudiar algoritmos para resolver bases de Gröbner más eficientemente que el algoritmo de Buchberger, como los algoritmos F4 o F5 de Faugère.
- Estudiar otras herramientas algebraicas diferentes a las bases de Gröbner (y posiblemente más eficientes) para decidir la insatisfacibilidad de un sistema polinómico.
- Utilizar técnicas de SMT (Satisfiability Modulo Theories) para verificar circuitos grandes como `Num2Bits` de forma más eficiente.
- Expandir el algoritmo modular para poder verificar la seguridad para todas las entradas, no para una única entrada fijada.
- Crear heurísticas para detectar si un componente debe asumirse como seguro o si por el contrario debería ser expandido en el componente padre, para poder tratar más circuitos.

Bibliografía

[1] Circom-ECDSA. URL: https://github.com/0xPARC/circom-ecdsa (8/6/2023).

[2] CircomLib. URL: https://github.com/iden3/circomlib (8/6/2023).

[3] CircomLib-ML. URL: https://github.com/socathie/circomlib-ml (8/6/2023).

[4] ED25519-Circom. URL:https://github.com/Electron-Labs/ed25519-circom(8/6/2023).

[5] Adams, W. and Loustaunau, P. (1994). *An Introduction to Gröbner Bases*, volume 3. American Mathematical Society.

[6] Balasubramanian, A., Baranowski, M. S., Burtsev, A., Panda, A., Rakamarić, Z., and Ryzhyk, L. (2017). System Programming in Rust. pages 156–161. ACM.

[7] Belles-Munoz, M., Isabel, M., Munoz-Tapia, J. L., Rubio, A., and Baylina, J. (2022). Circom: A Circuit Description Language for Building Zero-knowledge Applications. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18.

[8] Bigatti, A. and Robbiano, L. (2006). CoCoA: a system for computations in commutative algebra. pages 6–6. ACM.

[9] Blum, M., Feldman, P., and Micali, S. (2019). Non-interactive zero-knowledge and its applications.

[10] Cox, D. A., Little, J., and O'Shea, D. (2015). *Ideals, Varieties, and Algorithms*. Springer International Publishing.

[11] Faugère, J. C. (2002). A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). pages 75–83. ACM.

[12] Faugére, J.-C. (1999). A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139:61–88.

[13] Fernando, J. F. and Gamboa, J. M. (2017). *Estructuras algebraicas: divisibilidad en anillos conmutativos*.

[14] Ghorpade, S. (2019). A note on Nullstellensatz over finite fields.

[15] Goldreich, O. and Oren, Y. (1994). Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32.

[16] Groth, J. (2010). Short Non-interactive Zero-Knowledge Proofs.

[17] Groth, J. (2016). On the Size of Pairing-Based Non-interactive Arguments.

[18] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the web up to speed with WebAssembly. pages 185–200. ACM.

[19] Iden3. JavaScript and Pure Web Assembly implementation of zkSNARK and PLONK schemes. URL: https://github.com/iden3/snarkjs (8/6/2023).

[20] Koe, Alonso, K. M., and Noether, S. (2020). Zero to Monero.

[21] Mayr, E. W. and Meyer, A. R. (1982). The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329.

[22] Morais, E., van Wijk, C., and Koens, T. (2018). Zero Knowledge Set Membership.

[23] Moura, L. D. and Bjørner, N. (2011). Satisfiability Modulo Theories: Introduction and Applications. *Communications of the ACM*, 54:69–77.

[24] Shpilka, A. and Yehudayoff, A. (2009). Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5:207–388.

[25] Tikhomirov, S. (2018). Ethereum: State of Knowledge and Research Perspectives.

[26] Vujicic, D., Jagodic, D., and Randic, S. (2018). Blockchain technology, bitcoin, and Ethereum: A brief overview. pages 1–6. IEEE.

[27] Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., and Wang, F.-Y. (2018). An Overview of Smart Contract: Architecture, Applications, and Future Trends. pages 108–113. IEEE.

[28] Wohrer, M. and Zdun, U. (2018). Smart contracts: security patterns in the ethereum ecosystem and solidity. pages 2–8. IEEE.

Apéndice A

Código fuente

El algoritmo modular descrito en el capítulo V ha sido implementado en el lenguaje de programación Rust. Este lenguaje ha sido escogido por ser el lenguaje en el que el compilador Circom está escrito, lo que permite una gran interoperabilidad. También ha sido escogido por sus grandes garantías de seguridad, así como ser un lenguaje lo suficientemente eficiente para gestionar grandes volúmenes de datos.

El código Rust genera sistemas polinómicos para los que se debe demostrar su insatisfacibilidad, tal y como vimos en el capítulo IV. Para demostrar la insatisfacibilidad, se crean archivos en el programa de álgebra computacional CoCoA [8], que se comunica con el código en Rust.

El código fuente del algoritmo modular puede encontrarse en el siguiente repositorio de GitHub: `https://github.com/palmenros/zksnark-safety-verificator`.

Apéndice B

Teorema de la base de Hilbert

En este apéndice se estudia una demostración del teorema de la base de Hilbert y un corolario que garantiza que los anillos de polinomios en varias variables $\mathbb{K}[x_1, \ldots, x_n]$ son Noetherianos.

Esto es necesario para demostrar la terminación del algoritmo 3 de Buchberger para hallar una base de Gröbner, vista en el teorema IV.35.

Comencemos definiendo el concepto de anillo Noetheriano.

Definición B.1. ***(Anillo Noetheriano)***. *Sea $\mathcal{A}$ un anillo conmutativo. Diremos que $\mathcal{A}$ es Noetheriano si satisface la condición de la cadena ascendente, es decir, si toda cadena numerable de inclusión de ideales*

$$I_1 \subseteq I_2 \subseteq I_3 \subseteq \ldots$$

se acaba estabilizando, es decir, $\exists n \geq 1$ tal que $\forall m \geq n, I_m = I_n$.

Proposición B.2. *Sea $\mathcal{A}$ un anillo conmutativo. Entonces $\mathcal{A}$ es Noetheriano si y solo si todo ideal de $\mathcal{A}$ es finitamente generado.*

Demostración. $\Rightarrow$ Supongamos por contradicción que existe un ideal I de $\mathcal{A}$ que no es finitamente generado. Elegimos $f_1 \in \mathcal{A}$, y como no es finitamente generado, $\langle f_1 \rangle \subsetneq \mathcal{A}$. Supongamos que tenemos $J = \langle f_1, \ldots, f_n \rangle \subsetneq \mathcal{A}$. Como $J \neq \mathcal{A}$, $\exists f_{n+1} \in \mathcal{A}$ tal que $f_{n+1} \notin J$. Además, como $\mathcal{A}$ no es finitamente generado, $\langle f_1, \ldots, f_n \rangle \subsetneq \langle f_1, \ldots, f_n, f_{n+1} \rangle \subsetneq \mathcal{A}$. Repitiendo, hallamos una cadena estrictamente ascendente

$$\langle f_1 \rangle \subsetneq \langle f_1, f_2 \rangle \subsetneq \cdots \subsetneq \langle f_1, \ldots, f_n \rangle \subsetneq \cdots$$

lo que entra en contradicción con que $\mathcal{A}$ sea Noetheriano.

$\Leftarrow$ Sea $I_1 \subseteq I_2 \subseteq I_3 \subseteq \ldots$ una cadena numerable de ideales. Sea $I = \bigcup_{i=1}^{\infty} I_i$. Veamos que I es un ideal. Como $0 \in I_1, 0 \in I$. Sean $f, g \in I$, por lo que $\exists i, j$ tal que $f \in I_i, g \in I_j$. Sea $k = \max\{i, j\}$, por lo que $f, g \in I_k \implies f + g \in I_k \subseteq I$. Además, por ser I_i un ideal, si $h \in \mathcal{A}$, $f \cdot h \in I_i \subseteq I$. Por tanto, I es un ideal, por lo que es finitamente generado. Es decir, $I = \langle f_1, \ldots, f_s \rangle$. Cada $f_i \in I_{\alpha_i}$, sea $n = \max\{\alpha_1, \ldots, \alpha_s\}$, entonces $f_1, \ldots, f_s \in I_n = I$ por lo que $f_1, \ldots, f_s \in I_m$ para todo $m \geq n$ y por tanto $I_n = I \subseteq I_m \subseteq I$ por lo que $I_m = I = I_n$ para todo $m \geq n$. □

La demostración aquí presentada del teorema de la base de Hilbert sigue la línea de la demostración original y de [13].

Teorema B.3. ***(Teorema de la base de Hilbert)*** *Sea $\mathcal{A}$ un anillo conmutativo Noetheriano, entonces el anillo $\mathcal{A}[x]$ es también Noetheriano.*

Demostración. Supongamos por reducción al absurdo que existe un ideal $I \subseteq \mathcal{A}[x]$ no finitamente generado. Escogemos $f_1 \in I \setminus \{0\}$ de forma que $\deg(f_1) = \min\{\deg(f) : f \in I \setminus \{0\}\}$, posible por estar $\mathbb{N}_0$ bien ordenado. Inductivamente, si tenemos un ideal $J_n = \langle f_1, \ldots, f_n \rangle$ podemos elegir un $f_{n+1} \in I \setminus \langle f_1, \ldots, f_n \rangle$ de forma que $\deg(f_{n+1}) = \min\{\deg(f) : f \in I \setminus J_n\}$, por ser I no finitamente generado, definiendo así $J_{n+1} = \langle f_1, \ldots, f_{n+1} \rangle$. Por la minimalidad del grado de cada f_n, tenemos que $\deg(f_i) \leq \deg(f_{i+1})$ para todo $i \geq 1$. Sea $a_n = \mathrm{CD}(f_n) \in \mathcal{A}$ y sea $L_n = \langle a_1, \ldots, a_n \rangle \subseteq \mathcal{A}$. Tenemos por tanto una cadena

$$\langle a_1 \rangle \subseteq \langle a_1, a_2 \rangle \subseteq \langle a_1, a_2, a_3 \rangle \cdots$$

Por ser $\mathcal{A}$ Noetheriano, $\exists n \geq 1$ tal que $L_k = L_n$ para todo $k \geq n$. En concreto eso quiere decir que existen $c_1, \ldots c_n \in \mathcal{A}$ tal que $a_{n+1} = \sum_{i=1}^{n} c_i a_i$. Consideremos ahora el polinomio

$$g = f_{n+1} - \sum_{i=1}^{n} c_i x^{\deg(f_{n+1}) - \deg(f_i)} f_i$$

que está bien definido pues si $1 \leq i \leq n$, entonces $\deg(f_i) \leq \deg(f_{n+1})$. Además, $g \in I \setminus J_n$, pues el sumatorio de la derecha pertenece a J_n, y si $g \in J_n \implies f_{n+1} \in J_n$, lo cual es falso por construcción de los f_i. Finalmente, por construcción, $\mathrm{CD}(f_{n+1}) = a_{n+1}$ se cancela con el sumatorio de la derecha, por lo que $\deg(g) < \deg(f_{n+1})$, lo cual contradice la minimalidad del grado de los f_i, probando que todo ideal $I \subseteq \mathcal{A}[x]$ es finitamente generado. □

Corolario B.4. *Si $\mathbb{K}$ es un cuerpo, el anillo $\mathbb{K}[x_1, \ldots, x_n]$ es Noetheriano.*

Demostración. En primer lugar, observemos que $\mathbb{K}$ es un anillo Noetheriano. Sus dos únicos ideales son $\{0\}$ y $\mathbb{K} = \langle 1 \rangle$, ambos finitamente generados. Efectivamente, sea un ideal I. Si $a \neq 0 \in I$, entonces $a \cdot a^{-1} = 1 \in I \implies \forall b \in \mathbb{K}$, $b = 1 \cdot b \in I$.

Por el teorema B.3, como $\mathbb{K}$ es Noetheriano, $\mathbb{K}[x]$ también lo es. Aún más, como $\mathbb{K}[x_1, \ldots, x_n] \cong (\mathbb{K}[x_1, \ldots, x_{n-1}])[x_n]$ para $n \geq 2$, podemos aplicar inductivamente el teorema B.3 para demostrar que $\mathbb{K}[x_1, \ldots, x_n]$ es Noetheriano. □

Apéndice C

Ejemplos de verificación de algunos circuitos

En este apéndice se estudia la verificación de seguridad para dos circuitos concretos con aplicabilidad real que resultan en sistemas polinómicos interesantes: `Num2Bits` y `Split`.

C.1. Num2Bits

Un módulo de CircomLib [2] ampliamente usado por múltiples circuitos es `Num2Bits`, cuya implementación puede verse en el código C.1.

Código C.1 Implementación del módulo `Num2Bits`

```
1  template Num2Bits(N) {
2      signal input in;
3      signal output out[N];
4      var lc1=0;
5
6      var e2=1;
7      for (var i = 0; i<N; i++) {
8          out[i] <-- (in >> i) & 1;
9          out[i] * (out[i] -1 ) === 0;
10         lc1 += out[i] * e2;
11         e2 = e2+e2;
12     }
13
14     lc1 === in;
15 }
```

El módulo tiene como entrada un número `in` de N bits en $\mathbb{F}_p$ y su salida es la representación binaria de dicho número.

Veamos un ejemplo para $N = 4$ y $\texttt{in} = 13$. Tras realizar la propagación del algoritmo V, resulta el sistema de ecuaciones

$$\begin{cases} 13 - \texttt{out}[0] - 2 \cdot \texttt{out}[1] - 4 \cdot \texttt{out}[2] - 8 \cdot \texttt{out}[3] & = 0 \\ (\texttt{out}[0] - 1) \cdot \texttt{out}[0] & = 0 \\ (\texttt{out}[1] - 1) \cdot \texttt{out}[1] & = 0 \\ (\texttt{out}[2] - 1) \cdot \texttt{out}[2] & = 0 \\ (\texttt{out}[3] - 1) \cdot \texttt{out}[3] & = 0 \end{cases}$$

Vemos que a cada salida se le asigna una restricción que la fuerza a ser binaria (o 0 o 1), y luego tenemos una restricción que construye el número deseado (`in`) a partir de la salida

binaria. Aplicando la optimización descrita en la observación V.4, la restricción del polinomio de prohibición resulta ser

$$(\texttt{out}[0] - 0) \cdot (\texttt{out}[1] - 1) \cdot (\texttt{out}[2] - 0) \cdot (\texttt{out}[3] - 0) = 0$$

El principal problema con el módulo `Num2Bits` es que es instanciado para parámetros de N enormes, con $N > 250$, para luego realizar operaciones con los bits resultantes. Como comentamos en la observación IV.36, el método de cálculo de bases de Gröbner escala muy mal con el grado de los polinomios. En un tiempo razonable (menos de un minuto), solo hemos sido capaces de demostrar la seguridad de `Num2Bits(N)` para $N \leq 12$. Sin embargo, otras técnicas como SMT (Satisfiability Modulo Theory) pueden tener un mejor comportamiento para circuitos de este estilo.

C.2. Split

Otro módulo con un sistema polinómico interesante resultante al verificar su seguridad es `Split` en Circom-ECDSA [1]. Su implementación puede encontrarse en el código C.2.

Código C.2 Implementación del módulo `Split`

```
template Split(N, M) {
    assert(N <= 126);
    signal input in;
    signal output small;
    signal output big;

    small <-- in % (1 << N);
    big <-- in \ (1 << N);

    component n2b_small = Num2Bits(N);
    n2b_small.in <== small;
    component n2b_big = Num2Bits(M);
    n2b_big.in <== big;

    in === small + big * (1 << N);
}
```

La función del módulo `Split(N, M)` es descomponer un número `in` $\in \mathbb{F}_p$ de $N + M$ bits en dos números `small` de N y `big` de M bits tal que $\texttt{in} = \texttt{small} + 2^n \cdot \texttt{big}$.

Los dos componentes `Num2Bits` fuerzan a que efectivamente `small` y `big` tengan N y M bits respectivamente. Sin embargo, estos componentes fuerzan restricciones sobre las señales `small` y `big`, por lo que no se puede demostrar la seguridad modularmente para cada subcomponente como indicamos en el capítulo VI.

Para poder verificar la seguridad de este circuito, debemos tratar como un único sistema de polinomios todas las restricciones del componente y subcomponentes. Para $N = 2, M = 3, \texttt{in} = 23$, tras propagar los valores de la entrada, queda el sistema polinómico

$$\begin{cases} \texttt{n2b\_small.in} - \texttt{n2b\_small.out}[0] - 2 \cdot \texttt{n2b\_small.out}[1] & = 0 \\ (-1 + \texttt{n2b\_small.out}[0]) \cdot \texttt{n2b\_small.out}[0] & = 0 \\ (-1 + \texttt{n2b\_small.out}[1]) \cdot \texttt{n2b\_small.out}[1] & = 0 \\ \\ \texttt{n2b\_big.in} - \texttt{n2b\_big.out}[0] - 2 \cdot \texttt{n2b\_big.out}[1] - 4 \cdot \texttt{n2b\_big.out}[2] & = 0 \\ (-1 + \texttt{n2b\_big.out}[0]) \cdot \texttt{n2b\_big.out}[0] & = 0 \\ (-1 + \texttt{n2b\_big.out}[1]) \cdot \texttt{n2b\_big.out}[1] & = 0 \\ (-1 + \texttt{n2b\_big.out}[2]) \cdot \texttt{n2b\_big.out}[2] & = 0 \\ \\ \texttt{small} - \texttt{n2b\_small.in} & = 0 \\ \texttt{big} - \texttt{n2b\_big.in} & = 0 \\ \\ -23 + \texttt{small} + 4 \cdot \texttt{big} & = 0 \end{cases}$$

Las tres primeras ecuaciones corresponden al subcomponente `n2b_small`, y las siguientes 4 al subcomponente `n2b_big`. Las siguientes dos asignan a los subcomponentes `Num2Bits` sus entradas. La última es la restricción que realmente garantiza la salida que queremos, que es expresar 23 como $\texttt{small} + 2^2 \cdot \texttt{big}$.

La restricción asociada al polinomio de prohibición es

$$((\texttt{small} - 3) \cdot u_1 - 1) * ((\texttt{big} - 5) \cdot u_2 - 1) = 0$$

El sistema polinómico una vez añadida la restricción de prohibición es lo suficientemente pequeño para tratarlo con bases de Gröbner y demostrar que no tiene ninguna solución, por lo que la salida queda fija para dicha entrada.

Apéndice D

Ejemplos de circuitos inseguros encontrados

En este apéndice se estudian dos casos de circuitos inseguros detectados gracias a la herramienta desarrollada en el capítulo V y las pruebas realizadas.

D.1. Decoder

El componente `Decoder` es parte del repositorio oficial de CircomLib [2], la librería estándar de componentes Circom. Se puede encontrar la implementación original Circom de dicho módulo en el código D.1.

Código D.1 Implementación del módulo `Decoder`

```
template Decoder(W) {
    signal input inp;
    signal output out[W];
    signal output success;
    var lc=0;

    for (var i=0; i<W; i++) {
        out[i] <-- (inp == i) ? 1 : 0;
        out[i] * (inp-i) === 0;
        lc = lc + out[i];
    }

    lc ==> success;
    success * (success -1) === 0;
}
```

Vemos que el módulo tiene una entrada $\texttt{inp} \in \mathbb{F}_p$ y W salidas binarias $\texttt{out}[i]$. W es un parámetro del módulo, y se puede elegir cuantas salidas existen. También existe una salida binaria `success`.

El funcionamiento deseado del módulo es el siguiente: si $0 \leq \texttt{inp} < W$, entonces $\texttt{out}[\texttt{inp}] = 1$, $\texttt{out}[i] = 0 \; \forall i \neq \texttt{inp}$ y $\texttt{success} = 1$. En caso contrario, $\texttt{success} = 0$ y $\texttt{out}[i] = 0 \; \forall i$. Intuitivamente, se pone el elemento de `out` en la posición `inp` a 1 si existe y el resto a 0. Si el elemento existe, entonces $\texttt{success} = 1$ y si no, $\texttt{success} = 0$.

Podemos ver que el testigo que computa el circuito mediante las asignaciones inseguras `<--` es el descrito por el comportamiento anterior. Sin embargo, ese comportamiento no es forzado al sistema de restricciones R1CS asociado, que también permite la alternativa de fallar con $\texttt{success} = 0$ aún sí `inp` está en un rango válido.

Veamos un ejemplo con $W = 3$ e $\texttt{inp} = 2$. El sistema R1CS generado por Circom es

$$\begin{cases} \texttt{inp} \cdot \texttt{out}[0] & = 0 \\ (\texttt{inp} - 1) \cdot \texttt{out}[1] & = 0 \\ (\texttt{inp} - 2) \cdot \texttt{out}[2] & = 0 \\ (\texttt{success} - 1) \cdot \texttt{success} & = 0 \\ \texttt{out}[0] + \texttt{out}[1] + \texttt{out}[2] - \texttt{success} & = 0 \end{cases}$$

Vemos que efectivamente la solución computada por el testigo de Circom es válida ($\texttt{inp} = 2, \texttt{out} = [0, 0, 1], \texttt{success} = 1$) pero la solución alternativa $\texttt{inp} = 2, \texttt{out} = [0, 0, 0], \texttt{success} = 0$ también satisface el sistema, por lo que el módulo no es seguro.

Nuestro algoritmo modular descrito en el capítulo V marcó que este módulo no era seguro. Tras ejecutar la propagación descrita dicho capítulo, el sistema de ecuaciones resultante tras añadir el polinomio prohibición es:

$$\begin{cases} \texttt{out}[2] - \texttt{success} & = 0 \\ (\texttt{success} - 1) \cdot \texttt{success} & = 0 \\ ((\texttt{out}[2] - 1) \cdot u_3 - 1) \cdot (\texttt{success} - 0) & = 0 \end{cases}$$

Vemos que este sistema no es insatisfacible, pues $\texttt{success} = 0, \texttt{out}[2] = 0, u_3 = -1$ es una solución.

D.2. FullAdder

El subcomponente `fulladder` es parte del repositorio ED25519-Circom [4], y se usa como subcomponente de un sumador binario.

Su implementación se puede encontrar en el código D.2.

Código D.2 Implementación del módulo `fulladder`

```
template fulladder() {
    signal input bit1;
    signal input bit2;
    signal input carry;

    signal output val;
    signal output carry_out;

    val <-- (bit1 + bit2 + carry) % 2;
    val * (val - 1) === 0;
    carry_out <-- (bit1 + bit2 + carry) \ 2;
    carry_out * (carry_out - 1) === 0;
}
```

Podemos ver que el problema es que a `val` únicamente se asigna con una asignación insegura `<--`, que solamente genera dicha asignación en el código WebAssembly para generar el testigo, pero no genera una restricción equivalente en el sistema de restricciones R1CS.

Esto significa que el sistema de ecuaciones para este subcomponente es:

$$\begin{cases} \texttt{val} \cdot (\texttt{val} - 1) & = 0 \\ \texttt{carry\_out} \cdot (\texttt{carry\_out} - 1) & = 0 \end{cases}$$

La única restricción que impone ese sistema es que ambas salidas `val` y `carry_out` sean binarias, pero pueden tomar cualquier valor, y no tienen ninguna relación con las entradas.

Olvidar restricciones debido al uso erróneo de asignaciones inseguras `<--` es algo común tanto en el repositorio ED25519-Circom como en programadores inexpertos, debido al cambio radical de paradigma requerido para trabajar con polinomios.