Especificación Formal de Operaciones de Blockchain usando $\mathbb{Z}$

Manuel Figueroa, ITCR, Esteban Leandro ITCR
{mfigueroacr, elc790}@gmail.com

***Resumen*— Blockchain es en esencia una base de datos distribuida, una capa pública de todas las transacciones o eventos que han sido ejecutados y compartidos entre todas las partes participantes del sistema. Cada una de las transacciones es verificada por consenso entre la mayoría de los participantes. Una vez que se ingresa información en el sistema no puede ser removida, por lo que se considera que la tecnología de blockchain es útil para mantener registros confiables y verificables de cada una de las transacciones realizadas en algún momento [1]. La especificación formal de las operaciones de cada nodo participante en la red de blockchain nos permite determinar las operaciones requeridas para lograr consenso y registrar las transacciones realizadas en el sistema.**

***Index Terms*— Blockchain, Especificación, Formal, Z.**

Nota: Se omite el uso de acentos en el texto formal.

I. Introducción

UN sistema blockchain introduce la capacidad de mantener de manera descentralizada y distribuida un registro inmutable, confiable y verificable de transacciones o eventos realizados por los participantes del sistema. Es importante destacar de que parte de la seguridad inherente del sistema se debe a las operaciones realizadas por los nodos participantes de sistema distribuido. El sistema ordena las transacciones colocándolas en grupos denominados bloques, estos bloques se conectan entre sí obteniendo el nombre de blockchain (cadena de bloques).

Para decidir cuando un bloque debe ser agregado a la cadena, el sistema crea un mecanismo de verificación criptográfica que se conoce como "*proof of work*" esto obliga a los nodos que participan a resolver un problema matemático usualmente difícil, cuya solución requiere de mucho poder computacional y que asegura que la incorporación de bloques fraudulentos sea prácticamente imposible, ya que el poder computacional requerido para lograr una cadena de bloques verificable y que sea aceptada por la totalidad de la red, sería inmenso.

II. Expresando El modelo en $\mathbb{Z}$

Para simplificar la descripción del modelo y abstraer las particularidades de implementaciones orientadas al manejo de criptomonedas, el modelo descrito está basado en la implementación de Lauri Hartikka de NaiveChain, que implementa el algoritmo básico de blockchain sin las complejidades propias de otras plataformas como Bitcoin y Ethereum [3].
Vamos a definir las entidades que debemos conocer para implementar un sistema de blockchain.

Inicialmente definimos todos los mensajes posibles del sistema en el conjunto *Reporte* que consiste en la siguiente enumeración:

$$Reporte ::= Okay \mid BloqueInvalido \mid Rechazo$$

Reportamos el conjunto de operaciones exitosas mediante el esquema *Success.*

$$Success \triangleq [r! : Reporte \mid r! = Okay]$$

A. Bloques

4 correspondientes a las transacciones o eventos ocurridos en el sistema. Para lograr este encadenamiento cada bloque debe contener además de los datos, una firma única que lo identifica y la firma del bloque anterior.

Para definir un bloque nuevo requerimos los siguientes arreglos dados, para poder registrar los datos, la firma del bloque actual y anterior y la fecha de creación del bloque.

[*FINGERPRINT*, *TIMESTAMP*, *DATA*]

No requerimos saber más detalle de la estructura interna de los datos del bloque, ni de la representación de las firmas criptográficas de cada bloque.

Entonces un bloque va a estar definido por el esquema:

Bloque

fingerprint:*FINGERPRINT*
prevblock: *FINGERPRINT*
indice: $\mathbb{N}$
timestamp: *TIMESTAMP*
data: *DATA*

Se va a requerir de una función que nos calcula el fingerprint de un bloque, vamos a describir dicha función como:

getFingerprint : *Bloque* $\longrightarrow$ *FINGERPRINT*
createFingerprint: *BloqueEntrada* $\longrightarrow$ *FINGERPRINT*

B. Nodos

Establecemos que un nodo en la cadena de blockchain es una entidad que mantiene una secuencia dada de bloques. Vamos a definir como arreglos dados la dirección IP que va a servir de identificador único para el nodo.

[*IPADDRESS*]

Status ::= *Activo* | *Inactivo*

La estructura de un nodo va a estar dada por un esquema que contiene una IP que suponemos única, un estado indicando su actividad en el sistema y una cadena que contiene la secuencia de bloques calculados al momento.

Nodo

IP: *IPADDRESS*
Estado: *Status*
cadena: seq *Bloque*
peers: $\mathbb{P}$ *IPADDRESS*

C. Creación de Bloques

Para generar un bloque nuevo ocupamos definir un subtipo *BloqueEntrada* que vamos a utilizar para calcular el hash correspondiente al bloque.

BloqueEntrada

prevblock: *FINGERPRINT*
indice: ℕ
timestamp: *TIMESTAMP*
data: *DATA*

Utilizando *BloqueEntrada* y la función *createFingerprint* podemos generar una nueva instancia de un bloque y calcular su identificador único.

CrearBloque

entrada?: *BloqueEntrada*
Nuevo!: *Bloque*

Nuevo!.*fingerprint* = *createFingerprint entrada*?
Nuevo!.*prevblock* = *entrada*?.*prevblock*
Nuevo!.*indice* = *entrada*?.*indice*
Nuevo!.*timestamp* = *entrada*?.*timestamp*
Nuevo!.*data* = *entrada*?.*data*

También requerimos definir una función que nos valide si el bloque se puede agregar a la cadena:

ValidarBloque

Ξ*Nodo*
bloque?: *Bloque*
m!: *Reporte*

---

bloque?.*prevblock* = (*last cadena*).*fingerprint*
getFingerprint(*bloque*?) = *bloque*?.*fingerprint*
bloque?.*indice* = (*last cadena*).*indice* + 1
m! = *Okay*

III. Operaciones Del Nodo

Para establecer un conjunto de operaciones realizadas por los nodos individuales del sistema de blockchain, primero tenemos que definir que responsabilidades tiene cada una de estas entidades en el sistema.

Un nodo de blockchain además de mantener un registro de la cadena de bloques tiene como responsabilidad realizar la verificación de un nuevo bloque a ser incorporado en la secuencia. También es responsable de minar bloques, esta operación hace que el nodo tenga que establecer un valor de “proof of work” para un bloque para que este pueda ser enviado a la red y que sea aceptado por los demás nodos como el siguiente nodo válido en la cadena.

A. Agregar un nodo a la cadena

La operación para agregar un nuevo bloque a la cadena de bloques del nodo actual.

AgregarCadenaOk

$\Delta Nodo$
nuevo?: *Bloque*
m!: *Reporte*

---

nuevo?.*prevblock* = (*last cadena*).*fingerprint*
nuevo?.*fingerprint* = *getFingerprint*(*nuevo*?)
cadena′ = *cadena* ⌢ ⟨*nuevo*?⟩
m! =*Okay*

B. Reemplazar cadena

Hay ocasiones en las que es necesario que el nodo escoja entre dos posibles cadenas de bloques debido a que dos nodos distintos han creado bloques con el mismo índice, en estos casos es necesario reemplazar la cadena completa por la cadena más larga de bloques disponible.

ReemplazarCadena

$\Delta Nodo$
nuevaCadena?: seq *Bloque*

#*nuevaCadena*? > #*cadena*
cadena′ = *nuevaCadena*?

C. Listar Bloques

Es común que el usuario de un nodo quiera consultar la lista completa de bloques almacenados en el sistema, para lo que se debe incluir la operación *ListBlocks.*

listarBloques == (λ *Nodo*; *b*: *seq Bloque* • *cadena*)

D. *Iniciar Sistema*

Al arrancar el sistema iniciamos creando un nuevo nodo con un bloque génesis. Dicho bloque contiene un estado inicial y solamente sirve como marcador del inicio de la cadena.

$$bloqueGenesis : Bloque$$

$$getIP : \mathbb{P}\ Nodo \longrightarrow IPADDRESS$$

InitBlockchain

$\Delta Nodo$

$$IP' = getIP(Nodo)$$
$$Estado' = Activo$$
$$cadena' = \langle\ bloqueGenesis\ \rangle$$
$$peers' = \varnothing$$

IV. Transacciones de Blockchain

Una de las principales aplicaciones de blockchain en la actualidad es el manejo de transacciones en criptomonedas donde las más famosas son BitCoin[4] y Ethereum [5]. Para especificar las operaciones de manejo de transacciones en nuestro sistema de blockchain tenemos que definir el esquema *transacción* y las operaciones asociadas.

A. *Transacción*

En el modelo de blockchain se utiliza el esquema de cifrado de llaves públicas y privadas para encriptar las transferencias que al final resultan ser simples mensajes entre un emisor y un receptor unidos al sistema de blockchain.
Al utilizar el esquema de cifrado es necesario especificar dos conceptos de específicos de transacción por lo que vamos a tener *transacciones de salida y transacciones de entrada.*

1) *Transacción de Salida*

Una transacción de salida consiste en una dirección y una cantidad específica de tokens o monedas, la dirección corresponde a una llave pública. Lo que permite que el usuario poseedor de la llave privada asociada a la llave pública referenciada en la transacción, a acceder a las monedas de la transferencia.

TSalida

$$ID : FINGERPRINT$$
$$Timestamp : TIMESTAMP$$
$$Direccion : FINGERPRINT$$
$$Cantidad : \mathbb{N}_1$$

2) Transacción de Entrada

Las transacciones de entrada tienen la información de donde vienen las 'monedas'. Cada una de las transacciones de entrada hace referencia a una *transacción de salida* anterior donde dichas 'monedas'. Después de desbloquearse quedan disponibles para ser utilizadas en otras transacciones de salida. La firma prueba que únicamente el usuario con la llave privada referenciada en la transacción de salida.

$$
\begin{array}{|l}
\textit{TEntrada} \\
\hline
IDOut : FINGERPRINT \\
Firma : FINGERPRINT \\
\hline
\end{array}
$$

3) Transacción General

Definimos una transacción como un esquema que contiene un arreglo de transacciones de entrada y de transacciones de salida.

$$
\begin{array}{|l}
\textit{Transaccion} \\
\hline
ID : FINGERPRINT \\
Salidas : \mathbb{P}\ TSalida \\
Entradas : \mathbb{P} TEntrada \\
\hline
\end{array}
$$

Cuando un usuario posee 'monedas' en el sistema, en realidad lo que tiene es una lista de transacciones de salida no gastadas de las que puede disponer porque la llave pública a la que hacen referencia coincide con la llave privada que poseen.

B. Billetera Electrónica

El concepto de billetera electrónica en un sistema de blockchain consiste según lo descrito anteriormente en la generación de un par de llaves pública y privada que nos va a servir para asociar las transacciones a un usuario determinado.
Para generar la llave privada requerimos de una función especial que nos genera la llave privada. Además a partir del Wallet generado podemos calcular el *public_key* utilizando la función *getPublicFromWallet*.

$$
\begin{array}{|l}
\hline
getPrivateKey : FINGERPRINT \\
getPublicFromWallet : FINGERPRINT \\
\hline
\end{array}
$$

C. Operaciones con transacciones.

1) Obtener el balance de una billetera Electrónica

Una operación común es intentar obtener el balance actual de una billetera electrónica. Como se describió anteriormente el concepto de posesión de las 'monedas' es en realidad el tener una o más transacciones de salida sin gastar asociadas a la llave pública del usuario. Entonces de la lista general de transacciones sin gastar en el sistema, simplemente sumamos aquellas en las que la llave coincide con la de la billetera.

$$
\begin{array}{|l}
\hline
getWalletBalance : FINGERPRINT \longrightarrow \mathbb{Z} \\
\hline
\end{array}
$$

2) Enviar Transacción

Para enviar una transacción de un usuario a otro es necesario conocer la llave pública del destinatario que vamos a utilizar para crear una nueva transacción de salida que contiene el monto que deseamos enviar.

Es importante notar que las transacciones de salida que tengamos asociadas a nuestro wallet deben ser gastadas de manera completa, por lo que si tenemos una transacción que corresponde a un valor mayor del que de deseamos enviar entonces debemos partir el contenido en dos transacciones una por el monto que se desea enviar y otra con el resto a nuestra propia dirección para recuperar el monto sobrante.

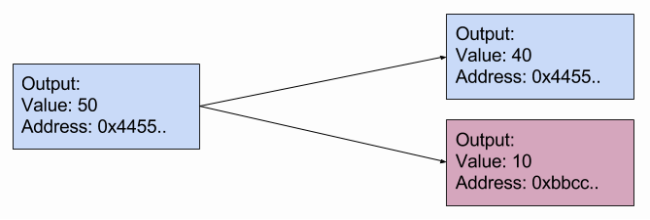

Figura 1 Tomado de [3]

De igual manera las transacciones de salida se pueden "acumular", para enviar un monto mayor por ejemplo el usuario tiene 3 transacciones de salida por 10,20,30 tokens respectivamente y desea enviar 55 tokens a otra dirección.

Como el total de transacciones de salida suma 60 entonces se redirigen 5 tokens de vuelta al *wallet* de usuario.

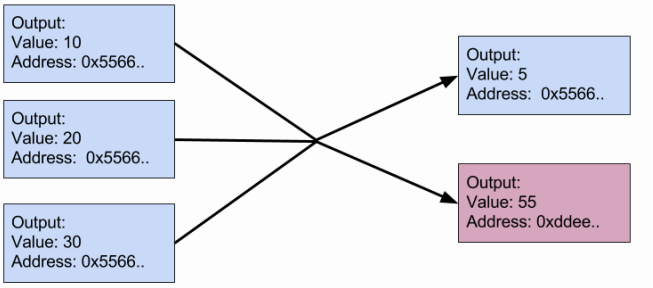

Figura 2 Tomado de [3]

El detalle de la operación de envío de transacciones se detalla en el siguiente esquema:

Definimos la operación *Monto Disponible* para verificar si el *wallet* tiene suficientes tokens asignados para realizar la transacción:

MontoDisponible

$\Xi Transaccion$

Wallet?: *FINGERPRINT*

Monto?: $\mathbb{Z}$

Monto? > 0

Monto? $\leqslant$ *getWalletBalance*(*Wallet*?)

Además, requerimos de una función que nos devuelva el conjunto de transacciones de salida para un usuario en particular acorde a un monto que deseamos transferir. Asumimos que dicha función se encarga de manejar el restante para dejarlo asociado a la cuenta del usuario según el esquema de transacciones visto anteriormente.

$$getTSalidas : \{user: FINGERPRINT; monto: \mathbb{Z}\} \longrightarrow \mathbb{P} TSalida$$

EnviarTransaccionOk

$\Delta Transaccion$
Wallet?: *FINGERPRINT*
Receiver: *FINGERPRINT*
Monto?: $\mathbb{Z}$
R!: *Reporte*

MontoDisponible
Monto? > 0
Receiver ≠ *Wallet*?
R! = *Okay*
$\forall t: TSalida \mid t \in getTSalidas(Wallet?, Monto?) \bullet t.Direccion = Receiver$

Como condiciones de error es que el usuario no tenga fondos suficientes para realizar la transacción.

FondoInsuficiente

$\Xi Transaccion$
Wallet?: *FINGERPRINT*
Monto?: $\mathbb{Z}$
R! : *Reporte*

Monto? > 0
Monto? > *getWalletBalance*(*Wallet*?)
R! = *Rechazo*

3) Operaciones Globales

La operación global de las transacciones la podemos expresar con la siguiente expresión *RealizarTransferencia:*

$$RealizarTransferencia \triangleq (EnviarTransaccionOk \land Success) \lor FondoInsuficiente$$

V. Conclusiones

Los sistemas blockchain al ser una base de datos distribuida requieren de mecanismos complejos de comunicación y sincronización para mantener la consistencia de los datos entre los nodos participantes.
Además, es necesario el uso de complejas funciones de encriptación para mantener la seguridad del sistema.
Este trabajo pretendía explorar las capacidades y limitaciones del lenguaje formal $\mathbb{Z}$ en la expresión y especificación del manejo de un sistema distribuido basado en la tecnología blockchain. Concretamente tratamos de especificar las operaciones de una implementación básica del sistema, NaiveChain[3] y asi explorar todas las operaciones y entidades necesarias para el manejo de una arquitectura distribuida de blockchain.
Los formalismos en $\mathbb{Z}$ permiten describir de manera completa y en muchos casos libre de interpretaciones los requerimientos de las operaciones necesarias en un sistema informático, por lo que creemos en el poder del uso de las especificaciones formales para lograr la consistencia y fiabilidad máxima posible en sistemas de alto impacto y donde la correctitud lógica de los algoritmos sea vital.

Verificado con Z Word Tools 3.3.0.1 25th August 2018

Referencias

[1] M. Crosby *et al*, "BlockChain Technology: Beyond Bitcoin," *Applied Innovation Review*, *Berkeley*., no. 2, pp. 6-19, Jun. 2016.
[2] Kass, "Creating Your First Blockchain with Java. Part 1.", Medium, 2019. [Online]. Available: https://medium.com/programmers-blockchain/create-simple-blockchain-java-tutorial-from-scratch-6eeed3cb03fa. [Accessed: 24- Nov- 2019]
[3] L. Hartikka, "A blockchain in 200 lines of code," Medium, 29-Dec-2017. [Online]. Available: https://medium.com/@lhartikk/a-blockchain-in-200-lines-of-code-963cc1cc0e54. [Accessed: 20-Oct-2019].
[4] Nakamoto, Satoshi. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at https://metzdowd.com.
[5] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf, 2014.