Colored Coins:

“En el inicio solo existia Bitcoin.”

Los colored coins fue una de las primeras creaciones de la comunidad blockchain para crear tokens. Evidentemente esto surgio mucho antes de la existencia de Ethereum y todas las demas tecnologias de blockchain que tenemos ahora. Es un metodo para representar y administrar activos del mundo real con la blockchain de Bitcoin. Generalmente utiliza el lenguaje de script de Bitcoin para codificar la informacion necesaria para realizar las transacciones. Basicamente son bitcoins transferidos en una transaccion con informacion adicional [1].

Nota: Se asume conocimientos tecnicos minimos acerca del funcionamiento de una blockchain y del modelo UTXO.

Los distintos protocolos:

La mayor diferencia entre los protocolos de colored coins es entre aquellos que usan OP\_RETURN y aquellos que no. Estos ultimos necesitan de crear una pseudo estructura para las transacciones, de forma tal que se codifique la informacion necesaria para el funcionamiento del protocolo pero garantizando que la misma sea aceptada por la red Bitcoin. Sin embargo, todos los protocolos son similares en cuanto a los pasos generales.

De que hablo cuando hablo de los tipos de transacciones:

Como se decia anteriormente, casi todos los protocolos son similares en cuanto a como crean los colored coins y como estos son transferidos, generalmente la diferencia esta en como se reconocen los colored coins en una transaccion. Normalmente, existen dos tipos fundamentales de transacciones, las genesis transactions y las transfer transactions. Como su propio nombre lo indica la primera es utilizada para la creacion de una cantidad especifica de colored coins y su posterior transferencia a algunas cuentas (mint), ademas, tambien puede tener la informacion suficiente acerca de quien es el creador de las colored coins permitiendo en el futuro solamente a este la emision de nuevas monedas del mismo color [2], [3]. Por otro lado la transaccion de transferencia como su propio nombre lo indica es aquella utilizada por un usuario para transferir una cantidad de sus colored coins a otra cuenta. Si no se cumple la diferenciacion correcta de las transacciones los colored coins de las mismas pueden perderse [3].

BC (Order-based Coloring):

En [2] se explica el primer algoritmo de colored coins. De forma simple este requiere que los inputs de una transaccion esten ordenados por color y que los uncolored coins (bitcoins normales) esten al final, que el mismo orden de los colores sea usado tanto para los inputs como para los outputs y que para cada color el valor total de sus inputs sea igual al valor total de sus outputs. Solo los bitcoins pueden tener un valor mayor en los inputs que en sus outputs (para pagar fees).

Segun [2] para parsear la transaccion se itera sobre cada color en orden. Por cada color se suman los valores de los inputs de ese color, despues se itera por los outputs en orden y se asignan los mismos al color hasta que la suma del valor de los outputs sea igual a la de los inputs. Como todos los outputs del primer color aparecen primero y la suma de los valores de estos es igual a la suma de los valores de los inputs del mismo color (que tambien aparecen primero), llegaremos a un output cuyo valor al sumarlo con el de los anteriores hara la suma igual a la de los inputs de ese color. Posteriormente nos movemos al siguiente output y al siguiente color [2]. Una vez que hayamos terminado con todos los colores solo quedaran los bitcoins, recordar que la suma total de los outputs de los bitcoins debe ser menor estricto que la suma de sus inputs (por el pago de los fees). Notar que esto es un estandar de transaccion de Bitcoin y que la misma se mantiene valida ante la red.

Ejemplo:

|  |  |  |
| --- | --- | --- |
| Index | Inputs | Outputs |
| 0 | 13 Red | 3 |
| 1 | 6 Green | 6 |
| 2 | 4 Green | 4 |
| 3 | 9 Blue | 10 |
| 4 | 2 Blue | 3 |
| 5 | 8 Uncolored | 8 |
| 6 |  | 5 |
| 7 |  | 2 |

First color: Red. Total Red input value: 13.

Output #0 is Red. Total Red output value: 3.

Output #1 is Red. Total Red output value: 9.

Output #2 is Red. Total Red output value: 13.

Red is complete.

Next color: Green. Total Green input value: 10.

Output #3 is Green. Total Green output value: 10.

Green is complete.

Next color: Blue. Total Blue input value: 11.

Output #4 is Blue. Total Blue output value: 3.

Output #5 is Blue. Total Blue output value: 11.

Blue is complete.

Total uncolored input value: 8.

Output #6 is uncolored. Total uncolored output value: 5.

Output #7 is uncolored. Total uncolored output value: 7.

No more outputs.

Total uncolored output value is less than input value.

Difference of 1 BTC is transaction fee.

EPOBC (Enhanced padded order-based coloring):

Intento de crear un modelo conceptual formal:

En lo general los protocolos se basan en que los colored coins son un concepto que permiten, dada una tecnologia blockchain determinada, ciertas propiedades con los output (coins) en un contexto de blockchains tipo Bitcoin [4]. Como esta propiedad esta asociada al output de una transaccion, una vez este output es gastado, la propiedad pasa al output de la nueva transaccion. La forma en que esta propiedad es traspasada es dada por un conjunto de reglas especiales (color kernels). De esta forma se puede decir que la parte que puede gastar el output es el dueno de dicha propiedad.

En [4] se define colorvalue = f(txOut, Q) donde, txOut: transaction output, Q: color. Una funcion que asigna al output de una transaccion, en un grafo dirigido de transacciones, una propiedad. Se define el color kernel como CK: (tx, tx\_in\_cvalues)→tx\_out\_cvalues, se aplica a cada transaccion independientemente, tx: una transaccion, tx\_in\_cvalues: es un vector de colorvalues de las outputs que son referenciadas por los inputs de la transaccion y tx\_out\_cvalues: es un vector de colorvalues de los outputs de las transacciones. En caso de una transaccion de regalo a un minero el colorvalue se asume null [4].

### Transaction tags:

Segun [7],los tipos de transacciones de colored coins son codificados usando un tag (etiqueta) en el campo de nSequence ([5]) del primer input. nSequence es un campo que siempre esta presente pero que se utiliza en ciertos casos, es un entero de 4 bytes de los cuales se pueden usar los 6 bits menos significativos para codificar el tipo de transaccion [3]. En este caso se usa la secuencia de bits 110011 para las transacciones de transferencia y la 100101 para las genesis.

## Padding:

Segun [3] la politica de los nodos de Bitcoin requiere que cada output tenga una cantidad minima de satoshis (la cual puede variar en el tiempo). Para satisfacer esta politica se utiliza el padding. Tener presente que 1 atomo de un color value es un satoshi. El valor de cada output se calcula como:

output value = padding + color value

El padding puede ser 0 (si el color value es mayor que la cantidad requerida por los nodos) o 2^pc donde 1<= pc <= 63 y pc es un padding code contenido en nSequence [6]. El padding es fijo cuando se crea la transaccion y se codifica en el campo de nSequence entre los 6 a 12 bits menos significativos [3].

## Transaccion genesis:

Una transaccion genesis crea nuevas colored coins las cuales estan asignadas a la primera output, el valor del primer output es usado para saber cuantos colored coins fueron emitidos.

color\_value = output\_value – padding

Los outputs restantes tienen color value null y el hash de la transaccion se usa para identificar al nuevo color [3].

## Transacciones de transferencia:

A continuacion un algoritmo que describe la forma de computar el color value para un output de una transaccion especifica [3]:

1. Verificar el tag de la transaccion (si es un genesis, tratarlo como tal y terminar, de otra forma asignar null y terminar).
2. Buscar las inputs que hace match con el output que se esta analizando (leer abajo el algoritmo para ello).
3. Buscar el color y color value de ese input (se puede hallar recursivamente o se puede tener en una cache).
4. Si todas las inputs que hacen match estan coloreadas y son del mismo color y la suma de sus color values es mayor que output\_value – padding, entonces output\_value – padding es el color value de la output que estamos analizando y su color es el mismo que el color de las inputs.
5. De otra forma asignar null.

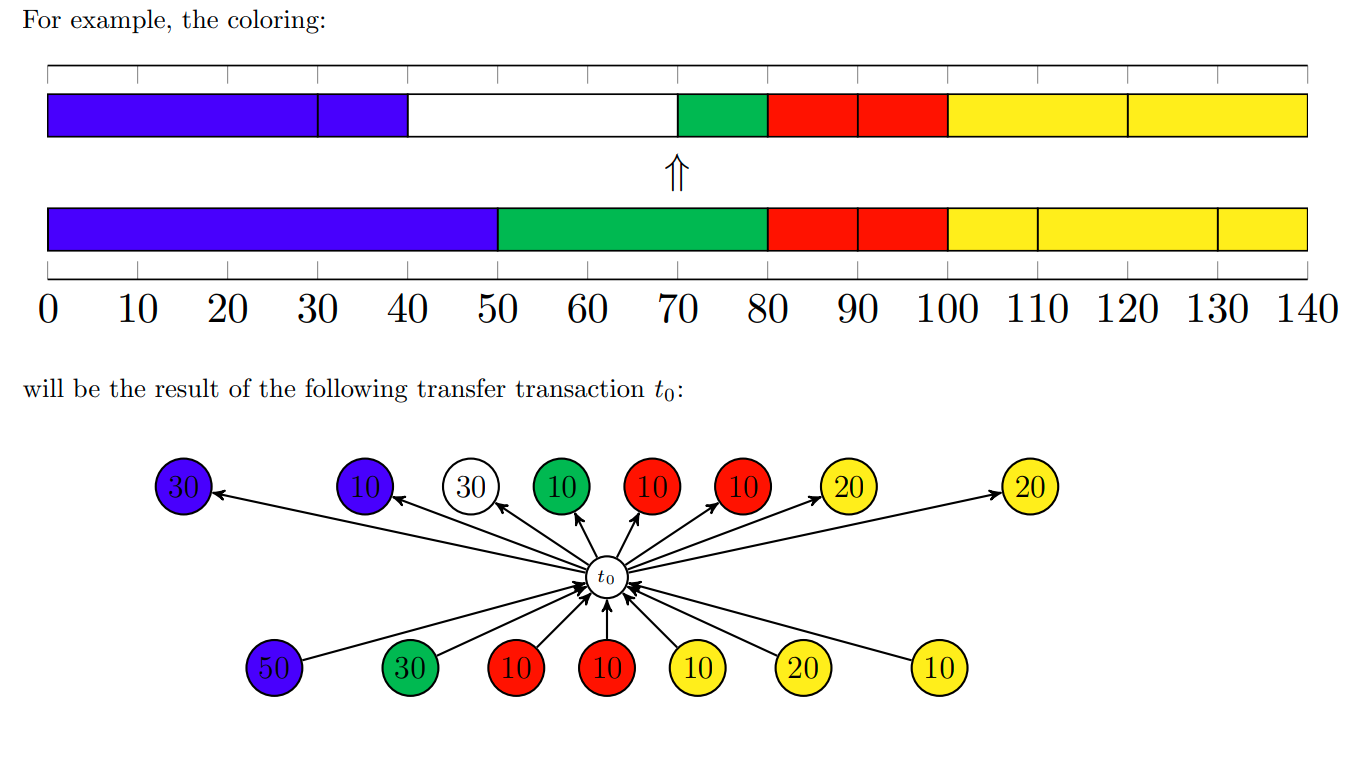
Normalmente color value es un par (color, value), donde el hash de la transaccion genesis es el color y el valor un numero entero [3].

poner un ejemplo de esto, quizas un video con una animacion?????

usar formulas matematicas para darle mejor formato a las ecuaciones

El algoritmo del paso dos utilizado para el match entre inputs y outputs depende del orden de las mismos. Siempre se trata de que los colores esten en el mismo orden:

1. Por cada output conputar un valor sin el padding (value\_wop) como: output\_value – padding, donde padding es el padding de la transaccion donde se encuentra el output.
2. Si existe un output con value\_wop < 0 el cual precede al output que se esta analizando, retornar una lista vacia (no existe un input que haga match, lo cual significa que el output no puede ser coloreado).
3. Computar la suma de los value\_wop para los outputs precedentes al que se esta analizando.
4. Hacer input\_running\_sum = 0 e iterar por cada input.
5. Si la input no viene de una transaccion con un tag de EPOBC terminar y retornar la lista de los inputs que se tengan hasta el momento (generalmente vacia).
6. Computar los valores de la input (value\_wop) sin el padding: input\_value – padding donde el padding es el de la transaccion referenciada por el input.
7. Si el value\_wop del input es negativo, terminar y retornar la lista obtenida.
8. Si el input se solapa con el output agregarlo a la lista. Solaparse entre dos segmentos de input y output puede verse de la siguiente manera: sean los segmentos [input\_running\_sum, input\_running\_sum + input\_value\_wop] y [output\_preceding\_sum, output\_preceding\_sum + output\_value\_wop] se puede saber si se solapan si se cumple que: ((input\_running\_sum < (output\_preceding\_sum + output\_value\_wop)) y ((input\_running\_sum + input\_value\_wop) > output\_preceding\_sum).
9. Hacer input\_running\_sum += input\_value\_wop.

Figure 1: Ejemplo del calculo de los color values usando EPOBC [16].

Colored Coin Protocol (CCP):

Algunas consideraciones iniciales:

Bitcoin script:

Segun [7] el lenguaje de script de Bitcoin es basado en pila y procesado de izquierda a derecha, no es Turing completo, no posee ciclos. Un script es escencialmente una lista de instrucciones almacenadas en cada output de una transaccion [8], que describe como puede, la persona que desea gastar los bitcoins transferidos, tener acceso a estos [7]. Como se explica en [2], una transaccion se considera valida si el script no lanza error y el item en el tope de la pila es True (no cero) cuando el script finaliza. La parte que originalmente envia los bitcoins dicta las operaciones del script que deben suceder en orden de liberarlos para ser utilizados en otra transaccion. La parte que desea gastar estos bitcoins debe proporcionar al script los inputs que son necesarios para que este se ejecute con un True en el tope de la pila. Ejemplo de lo anterior: una transaccion tipica de bitcoins hacia una direccion D posee un script que obliga al destinatario a probar que su llave publica, cuando es hasheada, es igual a la direccion D y, ademas, que la firma que prueba la posesion de la llave privada se corresponde con la llave publica que se provee. Los scripts de Bitcoin permite agregar flexibilidad para cambiar los parametros que son necesarios para gastar unos bitcoins que han sido transferidos. Por ejemplo, podrian ser utilizados para requerir dos llaves privadas, una combinacion de estas o ninguna llave.

Ejemplos de instrucciones (opcodes):

Segun [7]:

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Input | Output | Descripcion |
| OP\_0, OP\_FALSE | Nothing. | (empty value) | An empty array of bytes is pushed onto the stack. (This is not a no-op: an item is added to the stack.) |
| OP\_PUSHDATA2 | (special) | data | The next two bytes contain the number of bytes to be pushed onto the stack in little endian order. |
| OP\_IF |  |  | <expression> if [statements] [else [statements]]\* endif  If the top stack value is not False, the statements are executed. The top stack value is removed. |
| [OP\_RETURN](https://en.bitcoin.it/wiki/OP_RETURN) | Nothing | *fail* | Marks transaction as invalid. Since bitcoin 0.9, a standard way of attaching extra data to transactions is to add a zero-value output with a scriptPubKey consisting of OP\_RETURN followed by data. Such outputs are provably unspendable and specially discarded from storage in the UTXO set, reducing their cost to the network. Since [0.12](https://bitcoin.org/en/release/v0.12.0" \l "relay-any-sequence-of-pushdatas-in-opreturn-outputs-now-allowed), standard relay rules allow a single output with OP\_RETURN, that contains any sequence of push statements after the OP\_RETURN provided the total scriptPubKey length is at most 83 bytes. |
| OP\_DUP | x | xx | Duplicates the top stack item. |
| OP\_SHA256 | in | hash | The input is hashed using SHA-256. |

Ejemplo de script:

Transaccion estandar:

Hacer un video de ejemplo para que se vea la ejecucion

Nota: scriptSig es en el input de la transaccion y scriptPubKey es en el output de la misma, en este caso la output no gastada [8].

scriptPubKey: OP\_DUP OP\_HASH160 <pubKeyHash> OP\_EQUALVERIFY OP\_CHECKSIG

scriptSig: <sig> <pubKey>.

Ejecucion:

Primeramente se agregan a la pila los datos del scriptSig:

Pila:

<pubKey>

<sig>

Se ejecuta OP\_DUP:

Pila:

<pubKey>

<pubKey>

<sig>

Se ejecuta OP\_HASH160:

Pila:

<pubKeyHash1>

<pubKey>

<sig>

Pila:

<pubKeyHash>

<pubKeyHash1>

<pubKey>

<sig>

Se ejecuta OP\_EQUALVERIFY:

Pila:

<pubKey>

<sig>

Se ejecuta OP\_CHECKSIG:

Pila:

True

Se necesita poner unos datos especificos para liberar los fondos, no es necesaria una clave publica, cualquier persona que tenga los datos puede liberar los fondos [7]:

scriptPubKey: <data> OP\_HASH256 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000 OP\_EQUAL

scriptSig:

OP\_RETURN opcode y la mecanica del corazon:

Segun [8] y [9] solamente es posible un opcode de este tipo por transaccion, esto significa que solo un output de la transaccion podra contener este opcode y, ademas, que el mismo es no gastable e invalido, no almacenandose en la base de datos de UTXO [10]. Esto nos abre la puerta de poner datos arbitrarios en una transaccion y, en la misma, enviar bitcoins a otras cuentas (usando los otros outputs). Esta transaccion sera valida para la red de Bitcoin pero, ademas, podrian existir en esa misma red nodos especiales que sepan interpretar los datos arbitrarios, dandole un valor especial a los bitcoins que han sido transferidos.

Esquema del CCP:

Segun [11] este protocolo utiliza dos tipos de opcodes en la blockchain de Bitcoin para almacenar las reglas para manipular los activos (colored coins): OP\_RETURN opcode y las direcciones multifirma (opcional).

Multifirma en Bitcoin, mas espacio que en el espacio:

Bitcoin permite m de N direcciones multifirma con m<N de forma tal que m firmas de N sean necesarias para la ejecucion correcta del script asociado al output y la liberacion de los fondos del mismo [11]. Esto significa que es posible tener scripts que permitan 1 de N firmas para la liberacion de los fondos y las restantes N-1 firmas se pueden utilizar para almacenar datos necesarios para el protocolo. De esta forma el output permanece “gastable” puesto que existe al menos una llave privada que los libera y una vez liberados puede ser eliminada esa output del conjunto de UTXOs en la base de datos de los nodos. La notacion (1|N) hace referencia a 1 firma necesaria de N posibles.

Caso OP\_RETURN:

Los datos asociados a la manipulacion de los activos se pueden almacenar despues de este comando. Sin embargo, como la cantidad de datos en este opcode es fija se puede almacenar el hash de estos datos y los mismos se podrian almacenar en BitTorrent u otra red similar.

En [11] se usa la siguiente logica para almacenar los datos necesarios para el protocolo:

1. Si no se almacenan metadatos para la transaccion, todos los datos se pueden guardar en el espacio del input del OP\_RETURN.
2. Si se almacenan metadatos se almacena el SHA1 del info\_hash (un hash necesario para la localizacion de los archivos que se almacenan en BitTorrent) del torrent en el OP\_RETURN.
3. Si se necesita el SHA256 de los metadatos este debera almacenarse en una llave de la multifirma.

### Dandole candela:

Segun [11] una transaccion para quemar cierta cantidad de colored coins es posible, estos se restaran de la cantidad total en circulacion, lo cual significaria que serian “no gastables”.

De codificar una transaccion de emision y otros demonios:

Algo a notar en este ultimo protocolo es que a diferencia de los anteriores en los que solo es posible emitir los colored coins en una transaccion genesis, imposibilitando que sea emitida otra cantidad en otro momento, es posible tener varias transacciones genesis, en este caso llamadas “de emision”. En [11] se codifica la transaccion de emision de la siguiente manera:

| **Bytes** | **Description** | **Comments** | **Stored in** |
| --- | --- | --- | --- |
| 2 | Protocol Identifier | 0x4343 ASCII representation of the string CC ("Colored Coins") | OP\_RETURN |
| 1 | Version Number | Currently 0x02 | OP\_RETURN |
| 1 | **Issuance** OP\_CODEs |  | OP\_RETURN |
| 20 | SHA1 Torrent Hash | (optional), only when metadata is included | OP\_RETURN or (1|**3**) Multisig |
| 32 | SHA256 of metadata | (optional), only when metadata is included Allows for torrent metadata verification | OP\_RETURN or (1|**2**) or (1|**3**) Multisig |
| 1-7 | Amount of **issued units** | Encoded with the [Encoding Scheme](Number Encoding) | OP\_RETURN |
| 2-9 (per instruction) | [Transfer Instructions] (ver abajo) | Encoding the flow of assets from inputs to outputs | OP\_RETURN |
| 1 | Issuance Flags | At the moment only 6 bits are used | OP\_RETURN |

## Codificando la transaccion de transferencia [11] y eso que tu sabes:

| **Bytes** | **Description** | **Comments** | **Stored in** |
| --- | --- | --- | --- |
| 2 | Protocol Identifier | 0x4343 ASCII representation of the string CC ("Colored Coins") | OP\_RETURN |
| 1 | Version Number | Currently 0x02 | OP\_RETURN |
| 1 | **Transfer** OP\_CODEs |  | OP\_RETURN |
| 20 | SHA1 Torrent Hash | (optional), only when metadata is included | OP\_RETURN or (1|**3**) Multisig |
| 32 | SHA256 of metadata | (optional), only when metadata is included Allows for torrent metadata verification | OP\_RETURN or (1|**2**) or (1|**3**) Multisig |
| 2-9 (per instruction) | Transfer Instructions (ver abajo) | Encoding the flow of assets from inputs to outputs | OP\_RETURN |

Instrucciones de transferencias instruyendo:

Segun [12] se define las instrucciones de transferencia en un lenguaje de script el cual codifica el flujo de los activos desde las inputs hasta las outputs. Cada instrucción de transferencia esta construida de 5 piezas:

| Command | Type | Memory | Meaning |
| --- | --- | --- | --- |
| [Skip](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Transfer Instructions" \l "skip) | Boolean | 1 bit | 0 => Stay on input after processing 1 => skip to next input after processing |
| [Range](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Transfer Instructions" \l "range) \* | Boolean | 1 bit | 0 => Output size is 5 bits and understood literally 1 => Output size is 13 bits and understood as specifying a range |
| [Percent](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Transfer Instructions" \l "percent) | Boolean | 1 bit | 0 => Amount is understood literally 1 => Amount is understood as percent (1 byte) |
| [Output](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Transfer Instructions" \l "output) \* | Integer | 5 bits (Range=0)  13 bits (Range=1) | Specific Output index between 0..31  (Range=1) Range of outputs between index 0 and the specified index (maximum 8191) |
| [Amount](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Transfer Instructions" \l "amount) | Integer | [1-7](Number Encoding) bytes (Percent=0)  1 Byte (Percent=1) | Number of units to be transferred  Percent of units to be transferred |
| \* In a Burn transaction case: Range = 0 and Output = 31 are interpreted as a burn instruction. |  |  |  |

### Skip

Establece si la siguiente instrucción procesara la misma input (skip=0) o la siguiente (skip=1).

### Range

Especifica si Output sera el indice de la output a la que se aplicara la instrucción o un rango de outputs (indices empiezan en 0).

### Output

Especifica la output a la que se le aplicara la instrucción (o rango de instrucciones):

Range = 0: hace la transferencia al output especificado por el indice.

Range = 1: hace la misma transferencia a todos los outputs en el rango especificado por el indice.

### Amount

Especifica la cantidad de unidades (o porcentaje de las mismas si Percent = 1) a ser transferidas:

Percent = 0: Amount se entiende tal cual.

Percent = 1: Amount representa un porciento.

Descripcion de algunos parametros de las transacciones:

## Cantidad: Relevante solo en caso de las transacciones de emision.

Flags de emision [11]: Especifica informacion adicional de un activo creado. Se usan 6 bits, los primeros 3 describen la divisibilidad del activo, el siguiente si este esta bloqueado (no se pueden emitir mas en un futuro) o no (si es posible emitir mas en un futuro) y los ultimos dos la politica de agregacion. La divisibilidad solamente es relevante en el caso de las transacciones de emision. Ejemplo de divisibilidad:

1. divisibilidad 0: enteros 1, 2, …
2. divisibilidad 1: flotantes 0.1, 0.2, 1.5, 2.09, ...

#### Politica de agregacion:

En [11] se utiliza un flag para asegurar que los activos (colored coins) sean “agregables” (todas las colored coins son iguales) o “dispersos” (cada colored coin es unica). En el caso de los dispersos cada instancia de los mismos son tratados por una instrucción de transferencia distinta. Estas politicas tambien se codifican en las transacciones.

Capacidad de procesamiento:

En [11] se establece un limite de la cantidad de activos que pueden transferirse en cada transaccion, a diferencia de otros algoritmos, y el mismo es variable en dependencia de ciertos parametros que se codifiquen en la transaccion. Por ejemplo, si no tenemos metadata y cada instrucción de transferencia utiliza un minimo de 2 bytes se pueden procesar hasta 38 instrucciones de transferencia en una sola transaccion de transferencia (estas instrucciones se encuentran codificadas en el input del OP\_RETURN como vimos anteriormente). Se pueden ejecutar 37 en las de emision si la cantidad emitida utiliza un byte [11].

Relgas de transferencia es la parte interesante de toda la sopa:

Algo muy interesante con respecto a las reglas de transferencia es que estas permiten codificar mas de un activo en una misma output (input), lo cual ayuda a ahorrar espacio. Las reglas según [12] utilizadas para la transferencia de los colored coins entre las inputs y las outputs son:

1. Comenzando con el primer activo de la primera input en la transaccion (el orden es determinado recursivamente).
2. Reglas de validacion de la instrucción de transferencia:
   1. La input actual debe ser un indice valido.
   2. Output debe indicar un indice existente (irrelevante en caso de que sea una transaccion de quema).
   3. Amount no puede exceder la cantidad del activo que se esta procesando, a menos que: el activo sea “agregable” y existe un activo siguiente a este con el mismo assetID y su cantidad puede satisfacer la cantidad que falta en la isntruccion.
   4. En caso de una transferencia invalida, todos los activos son enviados a la ultima output.
3. Rango:
   1. 0 mueve la cantidad especificada del activo actual a el output del indice especificado en Output.
   2. 1 mueve la cantidad especificada del activo actual a todos los output en el rango de los indices de 0 a Output.
4. Quema: la cantidad especificada es reducida del activo procesado y no se envia a ninguna output. Queda como fee.
5. Todos los activos cuyas cantidades no se hayan transferido completamente (existe en la instrucción una cantidad menor que la existente en la input para ese activo) seran enviados a la ultima output.

Recordar que es necesario, como ya se explico anteriormente, que cada output tenga una cantidad minima de satoshis para que la transaccion sea valida.

## Ejemplo ejemplarizante [12]:

* We have 3 assets of types A, B, C
* The first input in our transaction has 10 units of asset A and 12 units of asset B
* The second input has 100 units of asset C

The following chain of instructions

0 0 0 0 3 0 0 0 1 7 0 0 1 1 50 1 0 0 2 3 0 1 0 49 2

means:

1. 0 0 0 0 3 Transfer 3 units of asset A from the first input to the first output (index 0). Stay on first input.
2. 0 0 0 1 7 Transfer the remaining 7 units of asset A from the first input to the second output (index 1). Stay on first input.
3. 0 0 1 1 50 Transfer 6 units of asset B to second output. 6 units because 6 is 50% of 12 and we are now processing asset B because the previous instructions exhausted asset A in the first input. Stay on first input.
4. 1 0 0 2 3 Transfer 3 more units of asset B to the third output (index 2). Skip to second input.
5. 0 1 0 49 2 Transfer 2 units of asset C from the second input to each output, starting from the first output all the way to the 50th output (index 49).
6. 3 units of asset B are implicitly transferred to the last output, since it had 12 units and only 9 were transferred explicitly.

Unos datos muy estaticos:

Id del activo o coge tu assetId aqui:

Segun [13] los assetId son un identificador inmutable que es creado con la emision del activo y derivado de la transaccion de emision, de la divisibilidad y la politica de agregacion. Como ya se explico anteriormente existen dos tipos de acitvos, bloqueados y los que no son bloqueados.

Los assetId no bloqueados, desbloqueados y unbloqueados:

Son similares a direcciones de Bitcoin [13] solo que comienzan con la letra U. Esta direccion es derivada de la clave publica del script que se encuentra en el output referenciado por el primer input de la transaccion de emision. Esto da la posibilidad de crear varias transacciones de emision para el mismo activo por parte de un mismo emisor, ya que todo aquel activo que se origine de un output que contenga la misma llave publica es considerado el mismo.

Los assetId que vienen con bloqueo de fabrica:

Son similares a direcciones de Bitcoin, solo que comienzan con la letra capital L. Se derivan de una forma similar a los anteriores solo que en vez de utilizar la calve publica en el script del output referenciado por el primer input de la transaccion de emision se utiliza el id de dicha transaccion y el indice del output referenciado por el primer input. A diferencia de los no bloqueados no es posible emitirlos nuevamente ya que cualquier nueva transaccion de emision tendra un identificador distinto y por tanto el assetId sera distinto [13].

# Metadatos para los colored coins:

Este protocolo acepta metadatos, los cuales son almacenados generalmente en BitTorrent. Estos metadatos se pueden almacenar en cualquier transaccion [14]. Aceptan campos estaticos que tambien son opcionales y ademas de ello permiten la encriptacion de los mismos [13].

Reglas que no pueden romperse:

Esta parte de los metadatos darian la posibilidad de crear contratos inteligentes [15]. Segun [15] se soportan hasta 4 tipos distintos:

* [Fees](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Rules" \l "fees): Se paga un fee a una direccion especifica en cada transferencia.
* [Expiration](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Rules" \l "expiration): El activo expira despues de un tiempo.
* [Minters](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Rules" \l "minters): Algunas direcciones que pueden emitir este activo.
* [Holders](https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Rules" \l "holders): Las direcciones que pueden recibir este activo

[1] ‘Latest News on Colored Coins’, *Cointelegraph*. https://cointelegraph.com/tags/colored-coins (accessed Apr. 24, 2022).

[2] M. Rosenfeld, ‘Overview of Colored Coins’. 2013. Accessed: Apr. 25, 2022. [Online]. Available: https://www.semanticscholar.org/paper/Overview-of-Colored-Coins-Rosenfeld/198ad7cafc43c1cf0271e4daafdc4638775f8b49

[3] ‘EPOBC\_simple · chromaway/ngcccbase Wiki’, *GitHub*. https://github.com/chromaway/ngcccbase (accessed Apr. 25, 2022).

[4] ‘Conceptual model · chromaway/ngcccbase Wiki’, *GitHub*. https://github.com/chromaway/ngcccbase (accessed Apr. 25, 2022).

[5] ‘Transaction - Bitcoin Wiki’. https://en.bitcoin.it/wiki/Transaction (accessed Apr. 26, 2022).

[6] ‘EPOBC · chromaway/ngcccbase Wiki’, *GitHub*. https://github.com/chromaway/ngcccbase (accessed Apr. 26, 2022).

[7] ‘Script - Bitcoin Wiki’. https://en.bitcoin.it/wiki/Script (accessed Apr. 24, 2022).

[8] almel, ‘Answer to “Explanation of what an OP\_RETURN transaction looks like”’, *Bitcoin Stack Exchange*, Jul. 17, 2014. https://bitcoin.stackexchange.com/a/29555 (accessed Apr. 25, 2022).

[9] 六四事 C. S. П. К., ‘Answer to “Explanation of what an OP\_RETURN transaction looks like”’, *Bitcoin Stack Exchange*, Jan. 14, 2021. https://bitcoin.stackexchange.com/a/101451 (accessed Apr. 25, 2022).

[10] ‘OP\_RETURN - Bitcoin Wiki’. https://en.bitcoin.it/wiki/OP\_RETURN (accessed Apr. 25, 2022).

[11] ‘Coloring Scheme · Colored-Coins/Colored-Coins-Protocol-Specification Wiki’, *GitHub*. https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification (accessed Apr. 27, 2022).

[12] ‘Transfer Instructions · Colored-Coins/Colored-Coins-Protocol-Specification Wiki’, *GitHub*. https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification (accessed Apr. 27, 2022).

[13] ‘Static Data · Colored-Coins/Colored-Coins-Protocol-Specification Wiki’, *GitHub*. https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification (accessed Apr. 28, 2022).

[14] ‘Metadata · Colored-Coins/Colored-Coins-Protocol-Specification Wiki’, *GitHub*. https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification (accessed Apr. 28, 2022).

[15] ‘Rules · Colored-Coins/Colored-Coins-Protocol-Specification Wiki’, *GitHub*. https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification (accessed Apr. 28, 2022).

[16] ChromaWay, ‘EPOBC - Enhanced Padded Order-Based Coloring’. [Online]. Available: http://www.cs.technion.ac.il/~idddo/epobc.pdf