Tesis de Grado
de Ingeniería en Informática

Verificación de smart contracts en Marlowe para la blockchain Cardano

Director: Dr. Ing. Mariano G. Beiró
mbeiro@fi.uba.ar

Co-director: Phd. Simon Thompson (Kent University, IOHK)
S.J.Thompson@kent.ac.uk

Alumno: Julián Ferres, *(Padrón #101.483)*
jferres@fi.uba.ar

Facultad de Ingeniería, Universidad de Buenos Aires

30 de abril de 2022

Índice general

Capítulo 1

Introducción

Capítulo 2

Escritura de contratos financieros en Marlowe para Cardano

2.1. El modelo UTXO

Para poder entender la estructura de los contratos en Cardano, es importante tener comprensión de como se lleva a cabo la contabilidad en la misma. Tradicionalmente, pensamos en las transferencias de dinero entre dos cuentas bancarias, o quizás direcciones de Internet en el caso de la moneda digital.

La plataforma Cardano, asi como otras plataformas de criptomonedas como Bitcoin, utilizan en su lugar un enfoque contable conocido como UTXO (Unspent transaction output, o 'Salida de transacción no utilizadas').

El modelo UTXO [Zahnentferner, 2018][Brünjes and Gabbay, 2020] documenta el flujo de dinero no de cuenta a cuenta, sino de **transacción a transacción**. Cada transacción tiene entradas (de dónde proviene el dinero que se gasta) y salidas (hacia donde se dirige este dinero).

Cabe aclarar que una transacción puede también contener otros datos, como veremos en la sección 2.1.1.

Consideremos el gráfico de flujo de dinero en la figura 2.1. Las lineas negras representan outputs no gastados de las transacciones, y las lineas rojas representan dichos outputs siendo utilizados como inputs de transacciones posteriores.

Las cajas sin etiquetas representan una transacción (que contiene varios inputs y outputs). Los certificados azules denotan los outputs no gastados disponibles en nuestra ilustración.

Al comienzo del gráfico de flujo, Alice tiene 100 Ada en 'outputs sin utilizar' previos al comienzo de nuestro análisis. Este dinero proviene de una o más transacciones pasadas, que exceden el alcance del gráfico. Simplificamos el mismo con una simple caja (etiquetada con su nombre y el dinero correspondiente).

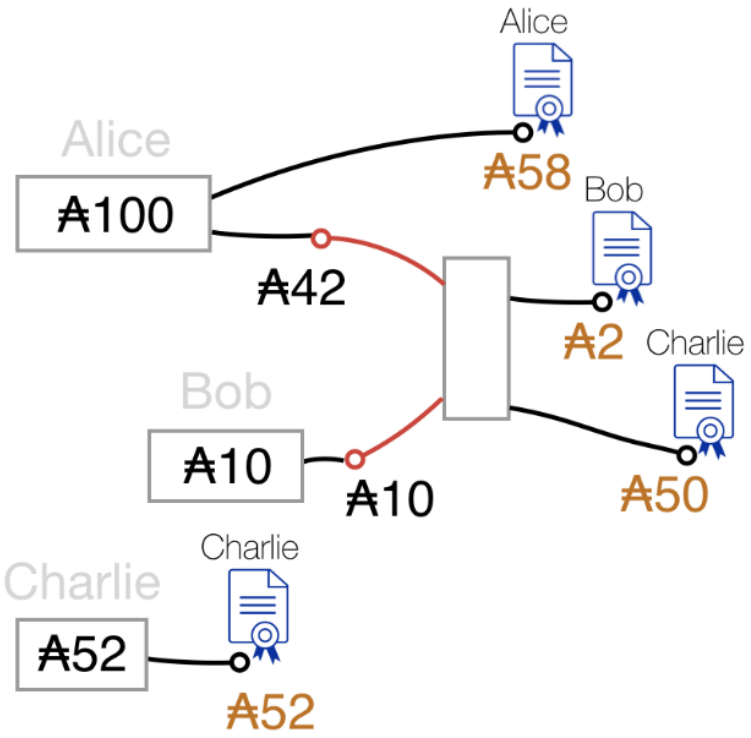

Figura 2.1: Flujo de dinero en el modelo UTXO

Dicha caja tiene dos lineas negras (outputs) saliendo de ella, siendo la suma del valor de las mismas 100 Ada:

- Un output de 58 Ada permanece sin ser utilizado y es parte de los outputs sin utilizar al final del análisis.
- Un output de 42 Ada se utiliza como parte de la nueva transacción.

Por su parte, Bob tiene 10 Ada de previos outputs sin utilizar. Los utiliza a todos en la nueva transacción. La transacción que ilustramos tiene dos inputs: 42 de Alice y 10 de Bob. La misma también tiene dos outputs: 2 para Bob y 50 para Charlie.

Vemos también que Charlie tiene 52 Ada provenientes de outputs previos a nuestro gráfico, totalizando 102 Ada que puede utilizar en transacciones futuras. Bob termina con solo un output de 2 Ada, y Alice con un total de 58 Ada.

El modelo anterior muestra estrictamente el flujo de dinero entre varios participantes. En esta versión simplificada, por ejemplo, las transacciones ilustradas no pagan comisiones. Sin embargo, en este modelo simplificado, vemos que los outputs deben gastarse en su totalidad. Es decir, un registro de un output no gastado no puede ser modificado (esta acción se adecua mas a los modelos contables basados en cuentas), solo podría utilizarse de forma completa.

Para mantener la integridad de la contabilidad, las nuevas transacciones debe tener todas las outputs no gastados (totalizando la cantidad correcta de outputs no gastados) utilizados como entrada.

En nuestro ejemplo anterior, la nueva transacción elimina (utilizándolas como entrada) a los outputs no gastados de valor 42 de Alice y 10 de Bob, para un total de 52 Ada. Esto implica que la transacción esta obligada a totalizar 52 Ada como outputs sin gastar (que de hecho cumple, con 2 para Bob y 50 para Charlie).

Un aspecto a destacar es que Bob tiene un 'unspent output' como entrada y uno como salida. Esto se podría interpretar como un 'cambio' (de 2 Ada) para esta transacción. Dicho concepto es similar al que utilizamos en el día a día al realizar pagos en efectivo: Si un producto cuesta \$98 y tenemos un billete de \$100, no podemos fraccionar dicho billete. Tenemos que pagar con todo el billete y recibir \$2 de cambio.

Dado que no existe una forma real de gastar parte de un 'unspent output', así es como el modelo UTXO trata el gasto parcial: agregando una salida de 'cambio'.

Cabe destacar que este modelo contable hace que sea conveniente distribuir el flujo de efectivo de varios contribuyentes a varios destinatarios haciendo que el mismo fluya hacia un fondo común, en este caso, la transacción, antes de enviarse a los beneficiarios finales. Esto representa, en un sentido muy general, el objetivo de los 'smart contracts' o contratos inteligentes.

Veamos más formalmente lo que sucede durante una transacción en el modelo UTXO.Para el modelo de transacciones basico que analizaremos, podemos referirnos a las definiciones en la figura 2.2:

Primitive types

$$
\begin{array}{rcll}
txid & \in & \mathsf{TxId} & \text{transaction id} \\
ix & \in & \mathsf{Ix} & \text{index} \\
addr & \in & \mathsf{Addr} & \text{address} \\
c & \in & \mathsf{Coin} & \text{currency value}
\end{array}
$$

Derived types

$$
\begin{array}{rcrcll}
tx & \in & \mathsf{Tx} & = & (inputs, outputs) & \in \mathbb{P}(\mathsf{TxIn}) \times (\mathsf{Ix} \mapsto \mathsf{TxOut}) & \text{transaction} \\
txin & \in & \mathsf{TxIn} & = & (txid, ix) & \in \mathsf{TxId} \times \mathsf{Ix} & \text{transaction input} \\
txout & \in & \mathsf{TxOut} & = & (addr, c) & \in \mathsf{Addr} \times \mathsf{Coin} & \text{transaction output} \\
utxo & \in & \mathsf{UTxO} & = & txin \mapsto txout & \in \mathsf{TxIn} \mapsto \mathsf{TxOut} & \text{unspent transaction outputs} \\
b & \in & \mathsf{Block} & = & tx & \in \mathbb{P}(\mathsf{Tx}) & \text{block} \\
pending & \in & \mathsf{Pending} & = & tx & \in \mathbb{P}(\mathsf{Tx}) & \text{pending transactions}
\end{array}
$$

Functions

$$
\begin{array}{rcll}
\mathsf{txid} & \in & \mathsf{Tx} \to \mathsf{TxId} & \text{compute transaction id} \\
\mathsf{ours} & \in & \mathsf{Addr} \to \mathbb{B} & \text{addresses that belong to the wallet}
\end{array}
$$

Filtered sets

$$
\begin{aligned}
\mathsf{Addr}_{\mathsf{ours}} &= \{a \mid a \in \mathsf{Addr},\ \mathsf{ours}\, a\} \\
\mathsf{TxOut}_{\mathsf{ours}} &= \mathsf{Addr}_{\mathsf{ours}} \times \mathsf{Coin}
\end{aligned}
$$

Figura 2.2: Definiciones básicas en el modelo UTXO

Antes de analizar la estructura de las transacciones, haremos un pequeño repaso sobre como la contabilidad se lleva a cabo en el 'libro mayor' o ledger. El registro que contiene la información sobre el ledger es llamado `UTXO`. Este registro es un map finito, donde la key o clave is un par formado por el id de la transaccion y un indice, `TxIn = TxId * Ix`. El id de la transacción puede ser calculado en base a una transacción completada para procesar, y es un identificador único de la transacción.

El índice `Ix` es necesario debido a que puede haber mas de un output en dicha transacción, y cada uno de los mismos tiene que tener un identificador único dentro de el conjunto de 'outputs' dentro de una transacción.

Los valores o values en el mapa son pares formados por un 'coin value' y una direccion, y el tipo de los mismos es `TxOut = Addr * Coin`. Cabe destacar que las direcciones de los usuarios son siempre claves públicas, y los fondos en ellas pertenecen a la entidad que puede probar que posee la clave privada correspondiente. Las direcciones de 'script' (smart contract o contrato inteligente) se comportan de manera ligeramente distinta, debido a que no tienen un dueño directo.

Para poder comprender la escturctura de la transacción en sí, analicemos primero los 'outpus'. Una transacción puede distribuir el dinero que está gastando a varias direcciones diferentes. Los outputs, (valores de tipo `TxOut`) se almacenan en una transacción como valores en un mapa finito. Las claves del mapa son indices únicos dentro del contexto del mapa, de manera tal que la combinación del id de la transacción y dicho indice identifica de forma global a dicho output. En el modelo UTXO, se relaciona a los valores de salida con las entradas de las cuales provienen, por medio de este identificador global compuesto.

Los inputs, cuyo orden no es relevante, son un conjunto y no una lista. Los elementos de este conjunto no contienen ni el valor de la moneda a gastar, ni la dirección de donde proviene el dinero. Esta es la principal distinción entre el modelo contrable tradicional y el UTXO:el dinero que se gasta solo referencia a los outputs no gastadas de transacciones previamente procesadas en el 'ledger' que reside actualmente en la blockchain. Cada elemento del mencionado set de inputs es un par formado por el id de la transacción y un indice que, como se explicó anteriormente, identifica de forma única el output no gastado en la UTXO.

Procesar una transacción implica actualizar el `UtxO` en el ledger de manera tal que los fondos gastados por la transacción que se está procesando estén disponibles para que los gasten los propietarios de las direcciones de las salidas de la transacción. Es decir, todas las entradas correspondientes a inputs de la transacción procesada se eliminan del ledger `UtxO`.

Adicionalmente, todos los valores de `TxOut` en el mapa finito de las salidas de la transacción se agregan a la `UTXO`, con la clave del mapa finito que consiste en el id de la transacción que se procesa, y el valor del índice es el mismo que en el mapa finito de salidas de esta transacción. Es decir, si `tx` contiene un par formado por el conjunto de entrada y el mapa de salidas `(ins, outs)` con id `id`, y `ix |->(a, c)` es una entrada de `outs`, la `UTXO` va a tener la entrada `(id, ix) |->(a, c)` agregada. En este párrafo, utilizamos la notación `k |->v` para referirnos a una

entrada del mapa finito con clave `k` y valor `v`.

Veamos como se refleja dicha actualización del ledger en terminos de notación matemática (que puede ser reflejada en código con relativa facilidad). Las siguientes son tres formas de filtrar el mapa finito de `UTXO`. El primero filtra dicho mapa mediante un subconjunto `ins` de las claves. El segundo filtro obtiene el complemento del resultado del primer filtro (en otras palabras, todas las entradas de la `UTXO` que no son indexadas por claves en la lista de inputs). El tercero filtra el mapa mediante los valores.

$$\begin{aligned}
\texttt{ins} \lhd \texttt{utxo} &= \{i \mapsto o \mid i \mapsto o \in utxo,\ i \in ins\} && \text{restricción de dominio} \\
\texttt{ins} \lhd\!\!\!\!/\ \texttt{utxo} &= \{i \mapsto o \mid i \mapsto o \in utxo,\ i \notin ins\} && \text{exclusión de dominio} \\
\texttt{utxo} \rhd \texttt{outs} &= \{i \mapsto o \mid i \mapsto o \in utxo,\ o \in outs\} && \text{restricción de rango}
\end{aligned}$$

Utilizaremos la notación introducida para procesar una nueva transacción. En otras palabras, eliminar los ouputs no gastados correspondientes y construir un nuevo conjunto de outputs que serán agregados al `UTXO` (como se describió anteriormente). Los outputs a agregar serían computados de la siguiente manera:

$$\begin{aligned}
\text{txins} &\in \mathbb{P}(\text{Tx}) \to \mathbb{P}(\text{TxIn}) \\
\text{txins}\ txs &= \bigcup \{inputs \mid (inputs, _) \in txs\} \\
\text{txouts} &\in \mathbb{P}(\text{Tx}) \to \text{UTXO} \\
\text{txouts}\ txs &= \left\{ (\text{txid}\ tx, ix) \mapsto txout \;\middle|\; \begin{array}{lcl} tx & \in & txs \\ (_, outputs) & = & tx \\ ix \mapsto txout & \in & outputs \end{array} \right\}
\end{aligned}$$

Usando esta notación, podemos definir la actualización del `UTXO`, debido a la transacción `tx` como:

$$(\text{txins}\ tx \lhd\!\!\!\!/\ \ utxo) \cup \text{outs}\ tx$$

Hay que tener en cuenta que se debe realizar un cálculo expícito de la cantidad total de Aca en las salidas y el total de Ada en todas las entradas de una transacción como parte de la validación de la transacción. También podría haber outputs en una transacción sin inputs correspondientes; estos se deben a la recolección de recompensas.

Ahora, para validar una transacción, se realiza una serie de cálculos que involucran el Ada en la misma y el Ada en otras cuentas del ledger, para asegurarse de que no se crea ni se destruye dinero. Esto se conoce como 'propiedad contable generalizada'. El modelo contable UTXO brinda protección integrada contra el 'doble gasto' de un output determinado.

Esta protección inherente, junto con la aplicación de la propiedad contable generalizada, asegura que no se permita que ocurra ningún gasto deshonesto. Esta es una propiedad crucial del sistema contable del ledger de Cardano, en particular porque existe una cantidad fija de Ada que nunca puede cambiar.

Para finalizar, hay que tener en cuenta que una transacción incluye una gran cantidad de datos adicionales, como testigos, certificados, y scripts juntos con sus hashes. En esta sección no hemos entrado en los detalles de los tipos y cálculos específicos utilizados en la implementación del ledger de Cardano. Sin embargo, abarcamos suficiente información como para poder entender que sucede detras de escena cuando se genera una transacción en la blockchain.

2.1.1. El modelo UTXO extendido

2.2. Marlowe como DSL

Marlowe [Lamela Seijas et al., 2020] [Kondratiuk et al., 2021] es un lenguaje pequeño, con pocas sentencias soportadas que, para cada contrato, describen el comportamiento que involucra un conjunto fijo y finito de roles.

Marlowe está diseñado para crear bloques para contratos financieros: pagos o depósitos de las partes, elecciones e información del mundo real. Cuando se ejecuta un contrato, los roles que implica son satisfechos por los participantes, que son identidades en la cadena de bloques. Cada rol está representado por un token en la cadena y los roles se pueden transferir durante la ejecución del contrato, lo que significa que esencialmente se pueden intercambiar.

Los contratos se pueden construir reuniendo una pequeña cantidad de estas sentencias que, en combinación, se pueden usar para describir y modelar muchos tipos diferentes de contratos financieros. Algunos ejemplos incluyen un contrato que puede realizar un pago a un rol o a una clave pública, un contrato que puede esperar una acción por parte de uno de los roles, como un depósito de moneda, o una elección entre un conjunto de opciones.

En particular, un contrato no puede esperar indefinidamente una acción: si no se ha realizado en un tiempo determinado (conocido como *timeout*), el mismo continuará con un comportamiento alternativo, por ejemplo, reembolsar los fondos en el contrato.

Los contratos de Marlowe pueden ramificarse en función de alternativas y tienen una vida finita, al final de la cual el dinero restante retenido por el mismo se devuelve a los participantes. Esta característica garantiza que el dinero no se puede bloquear para siempre en un contrato. Dependiendo del estado actual de un contrato, puede elegir entre dos cursos de acción alternativos, que son en sí mismos contratos. Cuando no se requieran más acciones, el contrato se cerrará y se reembolsará cualquier moneda restante en el contrato.

2.2.1. Contratos en Marlowe

Un contrato en Marlowe se obtiene combinando una pequeña cantidad de sentencias o *building blocks*. Las mismas pueden llegar a describir muchos tipos de contratos financieros, como hacer un pago, hacer una observación, esperar hasta que cierta condición se cumpla, etc. Luego, el contrato se ejecuta en una cadena de bloques, como Cardano, e interactúa con el mundo exterior.

Marlowe, en sí mismo, está embebido en Haskell y se modela como una colección de tipos de datos algebraicos en Haskell [HaskellWiki, 2020], con contratos definidos por el tipo de contrato:

```
data Contract = Close
              | Pay Party Payee Token Value Contract
              | If Observation Contract Contract
              | When [Case] Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

Marlowe tiene seis maneras de construir contratos. Cinco de esos métodos — `Pay`, `Let`, `If`, `When`, and `Assert` — construyen un contrato complejo a partir de contratos más simples, y el ultimo método, `Close` es un contrato simple. En cada paso de la ejecución, además de modificar el estado y proceder hacia un nuevo contrato, podrían generarse pagos y advertencias (*warnings*).

Antes de describir los métodos exhaustivamente, es útil conocer la definición de valores, observaciones y acciones:

1. **Valores**: Incluyen cantidades que cambian con el tiempo, tales como: el *slot interval* o 'intervalo actual', el balance de cierto token en una cuenta o elecciones que se han realizado (conocidas como *valores volátiles*). Los valores pueden ser combinados usando operaciones como suma, resta, negación, etc. Los mismos pueden ser valores condicionales o una observación.

2. **Observaciones**: Valores booleanos que son obtenidos al comparar valores, y que pueden ser combinados con los operadores booleanos estándar. Además, es posible observar si alguna elección se ha realizado (para una elección en concreto). Las observaciones tendrán un valor en cada etapa de la ejecución.

3. **Acciones**: Suceden en momentos particulares durante la ejecución, por ejemplo: un depósito de dinero o elegir entre varias alternativas.

Pay

Un contrato de pago `(Pay acc payee tok val cont)` realizará un pago de valor `val` de un token `tok` desde una cuenta `acc` a un beneficiario `payee`, quien sera uno de los participantes del contrato, u otra cuenta en el mismo.

Se generarán *warnings* si el valor `val` no es positivo, o si no hay recursos suficientes en `acc` para realizar el pago en su totalidad (incluso si hay balances positivos de otros tokens en la misma). En este último caso, se realizará un pago parcial (conteniendo todo el dinero disponible). El contrato en el que continuará la ejecución es `cont`.

Close

Un contrato `Close` prevé que el contrato sea cerrado (o rescindido). La única acción que realiza es reembolsar a los titulares de cuentas que contienen un saldo positivo. Esto se realiza de a una cuenta a la vez, pero todas las cuentas se reembolsarán en una sola transacción.

If

El conditional `If obs cont1 cont2` continuará en `cont1` o `cont2`, dependiendo de la observación `obs` cuando el mismo es ejecutado.

When

Es el constructor de contratos mas complejo, con la forma `When cases timeout cont`. El mismo es activado por acciones, que pueden o no ocurrir en un *slot* en particular. Como continua el mismo tras una acción se declara en la sintaxis de `cases` del contrato.

En el contrato `When cases timeout cont`, la lista `cases` contiene una colección de casos. Cada caso es de la forma `Case ac co` donde `ac` es una acción y `co` un contrato de continuación. Cuando una acción en particular, por ejemplo `ac`, ocurre, el estado del contrato es actualizado correspondientemente y y mismo continuara su ejecución en `co`.

Para garantizar que el contrato eventualmente progresará, la ejecución de `When cases timeout cont` continuará como `cont` una vez que el slot `timeout` es alcanzado.

Let

Un contrato `Let id val cont` permite registrar un valor, en un punto particular en el tiempo, y darle nombre usando un identificador. En este caso, la expresión `val` se evalúa y se almacena con el nombre `id`. El contrato entonces continúa como `cont`.

Además de permitirnos usar abreviaturas, este mecanismo nos brinda la capacidad de capturar y guardar valores volátiles que pueden cambiar con el tiempo, por ejemplo: '*el precio actual del petróleo*', '*el slot actual, en un punto particular de la ejecución del contrato*', para ser utilizado más adelante en la ejecución del mismo.

Assert

Un contrato `Assert obs cont` no tiene ningún efecto en el estado de un contrato, que continua inmediatamente en `cont`, pero genera una advertencia cuando la observación `obs` es falsa. Puede

ser utilizado para asegurar que alguna propiedad se cumple en un momento particular de la ejecución del contrato. Esta sentencia es útil porque permite que un *análisis estático* detecte que algún `assert` es falso, para alguna ejecución específica del contrato.

2.3. El estándar ACTUS

Los contratos financieros son acuerdos legales entre dos (o más) partes sobre el futuro intercambio de dinero. Dichos acuerdos legales se definen sin ambigüedades por medio de un conjunto de términos y lógica contractual. Como resultado, los mismos pueden describirse matemáticamente y representarse digitalmente como algoritmos. Los beneficios de representar contratos financieros de esta forma son múltiples; Tradicionalmente, el procesamiento de transacciones ha sido un campo en el que se pueden lograr mejoras de eficiencia mediante la automatización de contratos.

Adicionalmente, el análisis financiero (por naturaleza del dominio) se basa en la disponibilidad de representaciones computables de estos acuerdos, donde a menudo se utilizan aproximaciones analíticas. Recientemente, el auge de las blockchain, de contabilidad distribuida y los diversos casos de uso de los contratos inteligentes han abierto nuevas posibilidades para los contratos financieros digitales.

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones. Un patrón típico es un contrato de préstamo de tipo *bullet*, donde un monto de dinero inicial se entrega, a cambio de pagos de intereses cíclicos y la devolución del dinero inicial en el vencimiento del contrato. Si bien los pagos son fijos, existen muchas variantes que determinan cómo se programan y/o pagan los pagos de intereses cíclicos. Por ejemplo, los pagos de intereses pueden ser mensuales, anuales, mediante períodos arbitrarios. Pueden además ser de tasa fija o variable, pueden usarse diferentes métodos de cálculo de fracciones anuales o puede que no haya ningún interés.

Otro patrón popular es el de amortización de préstamos, en el que, a diferencia de los préstamos *bullet*, el dinero inicial prestado puede devolverse en porciones de montos fijos o variables, y de acuerdo con cronogramas cíclicos o personalizados. Otros tipos de contratos financieros a mencionar incluyen, acciones, contratos a plazo, opciones, swaps, mejoras crediticias, acuerdos de recompra, titularización, etc.

Al centrarse en las principales características distintivas, ACTUS describe la gran mayoría de todos los contratos financieros con un conjunto de alrededor de 32 patrones generales de flujo de efectivo, también conocidos como 'tipos de contrato'.

La taxonomía ACTUS [ACTUS, 2019d] proporciona un sistema de clasificación que organiza los contratos financieros según sus patrones distintivos de flujo de dinero. Aparte de este sistema de clasificación, la taxonomía también incluye una descripción de los instrumentos del mundo real cubiertos por cada contrato.

Por otro lado, los acuerdos legales en los contratos financieros representan una lógica puramente

determinista. Es decir, un contrato financiero define un conjunto fijo de reglas y condiciones bajo las cuales, dado cualquier conjunto de variables externas, las obligaciones de flujo de efectivo pueden determinarse sin ambigüedades. Por ejemplo, en un préstamo de tasa fija, las obligaciones de flujo de efectivo se definen explícitamente.

Las propiedades de los contratos financieros descritos anteriormente sientan las bases para una descripción algorítmica estandarizada y determinista de las obligaciones de flujo de dinero que surgen de tales acuerdos. Por lo tanto, esta descripción es agnóstica de la tecnología y es compatible con todos los casos de uso necesarios para que este mismo estándar se utilice en todas las funciones financieras. Entre estas se podrían mencionar: fijación de precios, creación de acuerdos, procesamiento de transacciones, así como el análisis en general, proyecciones de liquidez, valoración, cálculos y proyecciones de pérdidas y ganancias, y medición y agregación de riesgos, etc.

Adicionalmente, este estándar crea una base formidable para las máquinas de estados financieras y los *smart contracts*. En la documentación técnica [ACTUS, 2018] es posible encontrar la descripción matemática de los contratos financieros.

2.3.1. Notación ACTUS

Antes de adentrarnos en la especificación de un contrato, es necesario poder entender algunos aspectos de la notación del mismo:

- **Atributos de contrato**: Representan los términos contractuales que definen el flujo de dinero en un contrato financiero. Estos atributos están definidos en [ACTUS, 2019c].
- **Starting date**: t_0 representa la fecha de comienzo del contrato, y marca el instante en el cual las condiciones y estado del contrato esta siendo representado. En general, partiendo desde la lógica contractual, se podrán determinar los eventos del contrato y el estado para todo $t > t_0$, pero no para $s < t_0$
- **Variables de estado**: Las variables de estado describen el estado de un contrato, para un tiempo determinado de su ciclo de vida. Algunos ejemplos de las mismas son: *Notional Principal*, *Nominal Interest Rate*, o *Contract Performance*.

  El diccionario de ACTUS [ACTUS, 2019b] define todas las variables de estado y provee información adicional sobre el tipo de dato esperado por cada una, el formato, etc.

  En general, el 'estado' representa ciertos términos de un contrato que pueden cambiar a lo largo de su ciclo de ejecución, de acuerdo a eventos programados o no programados. Las variables están escritas en su forma abreviada con la primera letra en mayúscula, en negrita e indexadas mediante el tiempo.
- **Eventos**: Un evento de contrato (o simplemente evento) e_t^k se refiere a cualquier evento programado o no programado en un momento determinado t y de un tipo determinado k.

  Los eventos del contrato marcan puntos específicos en el tiempo (durante la ejecución del

mismo) en el que se intercambian flujos flujo de efectivo o se actualizan los estados del contrato. El diccionario de eventos [ACTUS, 2019a] enumera y describe todos los tipos de eventos k definidos por el estándar ACTUS.

- **Funciones de transición de estado** Dichas funciones, conocidas en Inglés como *'State Transicion Functions'* (STF) definen la transición de las variables de estado desde el *pre-evento* hacia el *post-evento*, cuando un cierto evento e_t^k ocurre. Esto provoca que el *pre-evento* y *post-evento* reciban la notación de t^- y t^+ respectivamente.

  Estas funciones son específicas para un tipo de evento y contrato. Las mismas son escritas de acuerdo al siguiente formato **STF_[event type]_[contract type] ()**, donde [event type] y [contract type] hacen alusión al tipo de evento y contrato al cual la STF pertenece.

  Por ejemplo: La STF para un evento de tipo IP en el contrato PAM se escribe como STF_IP_PAM () y modifica (entre otras) a la variable **Ipac** desde el pre-evento $\mathbf{Ipac}_{t^-}$ al post-evento $\mathbf{Ipac}_{t^+}$.

- **Funciones de pago**: Las funciones de pago, o Payoff Functions (POF) definen como el flujo de dinero $c \in \mathbb{R}$ ocurre para un determinado evento e_t^k. El mismo es obtenido del estado actual y los terminos del contrato. Si fuera necesario, el flujo de dinero puede ser indexado con el tiempo del evento: c_t.

  Las funciones de pago (de forma analoga a las STF), son específicas para un tipo de evento y contrato, y su notación es la siguiente: **POF_[event type]_[contract type] ()**, donde [event type] y [contract type] hacen alusión al tipo de evento y contrato al cual la POF pertenece.

  Por ejemplo: La POF para un evento de tipo IP e_t^{IP} en el contrato PAM se escribe como POF_IP_PAM ().

- **Fechas/Tiempo**: Sin adentrarnos demasiado en particularidades, cabe aclarar que ACTUS utiliza el formato de fechas ISO 8601. Por lo tanto, las fechas son usualmente expresadas en el siguiente formato: [YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]. El formato no soporta husos horarios.

- **Secuencia de eventos**: Los eventos (de diferentes tipos) de un contrato pueden ocurrir en el mismo instante de tiempo t. En este caso, la secuencia de evaluación de su STF y POF es crucial para los flujos de efectivo resultantes y las transiciones de estado. Por lo tanto, se utiliza un indicador de secuencia de eventos que se puede encontrar para cada evento en el diccionario de eventos. Este implica el orden de ejecución de diferentes eventos en el mismo tiempo t.

- **Lifetime de contrato**: La vida útil de un contrato ACTUS es el período de tiempo de su existencia, desde la perspectiva del usuario que analiza. Para cada punto en el tiempo durante su vida, se puede analizar un contrato ACTUS en términos de estado actual y flujos de efectivo futuros.

2.3.2. Un contrato de ejemplo

En esta sección, recorreremos brevemente la especificación técnica ofrecida por [ACTUS, 2018].

En particular, nos centraremos en un tipo de contrato llamado *Principal at Maturity* (PAM). El propósito del contrato PAM puede ser resumido en el siguiente párrafo:

'Se efectuará un pago del valor total en la fecha de intercambio inicial (simbolizada con la variable de contrato IED) y es reembolsado en la fecha de vencimiento (MD). Dependiendo de las variables de contrato, podrían aplicarse tarifas fijas o variables.'

Al describir el contrato, la especificación técnica separa al mismo en tres tablas. Las mismas expresan de forma declarativa, que acción debe ocurrir ante determinado evento.

Las filas de las tablas representan los diferentes tipos de eventos que el contrato tolera, y en las columnas se encuentra la acción correspondiente, junto con comentarios apropiados:

- **Contract Schedule (Cronograma del contrato)**: Contiene información acerca de los eventos programados para dicho contrato. En general, se realiza la asignación a las variables de estado correspondiente. Dichas variables reciben fechas o vectores de fechas (en caso de que el tipo de evento pueda ocurrir en múltiples instantes de la vida del contrato).

  Por ejemplo, para el evento de *monitoring* (AD), un contrato podría definir $\vec{t}^{AD} = (t_0, t_1, \ldots, t_n)$, siendo $t_1, \ldots, t_n$ tiempos definidos por el usuario.

- **State Variables Initialization (Inicialización de variables de estado)**: Esta tabla contiene información acerca del estado inicial de las variables del contrato. Muchas variables son simplemente extraídas de los términos del contrato, mientas que otras tienen estructuras condicionales en su definición.

  Dichas variables serán luego utilizadas para definir pagos y funciones de transición de estado.

- **State Transition Functions and Payoff Functions (Funciones de transición de estado y de pago)**: Esta tabla reúne las funciones de transición y de pago correspondientes a un contrato, para cada tipo de evento.

A continuación, se muestran fragmentos de las 3 tablas para el contrato, extraídos de la especificación:

PAM: Contract Schedule

Event	Schedule	Comments
AD	$\vec{t}^{AD} = (t_0, t_1, ..., t_n)$	With $t_i, i = 1, 2, ...$ a custom input
IED	$t^{IED} = \texttt{IED}$	
MD	$t^{MD} = \textbf{Tmd}_{t_0}$	
PP	$\vec{t}^{PP} = \begin{cases} \varnothing & \text{if} \quad \texttt{PPEF} = \text{'N'} \\ (\vec{u}, \vec{v}) & \text{else} \end{cases}$ where $\vec{u} = S(s, \texttt{OPCL}, T^{MD})$ $\vec{v} = O^{ev}(\texttt{CID}, \texttt{PP}, t)$	with $s = \begin{cases} \varnothing & \text{if} \quad \texttt{OPANX} = \varnothing \wedge \texttt{OPCL} = \varnothing \\ \texttt{IED} + \texttt{OPCL} & \text{else if} \quad \texttt{OPANX} = \varnothing \\ \texttt{OPANX} & \text{else} \end{cases}$
PY	$\vec{t}^{PY} = \begin{cases} \varnothing & \text{if} \quad \texttt{PYTP} = \text{'O'} \\ \vec{t}^{PP} & \text{else} \end{cases}$	
FP	$\vec{t}^{FP} = \begin{cases} \varnothing & \text{if} \quad \texttt{FER} = \varnothing \vee \texttt{FER} = 0 \\ S(s, \texttt{FECL}, T^{MD}) & \text{else} \end{cases}$	with $s = \begin{cases} \varnothing & \text{if} \quad \texttt{FEANX} = \varnothing \wedge \texttt{FECL} = \varnothing \\ \texttt{IED} + \texttt{FECL} & \text{else if} \quad \texttt{FEANX} = \varnothing \\ \texttt{FEANX} & \text{else} \end{cases}$
PRD	$t^{PRD} = \texttt{PRD}$	
TD	$t^{TD} = \texttt{TD}$	
IP	$\vec{t}^{IP} = \begin{cases} \varnothing & \text{if} \quad \texttt{IPNR} = \varnothing \\ S(\ldots, T^{MD}) & \text{else} \end{cases}$	with

Figura 2.3: Cronograma del contrato PAM para algunos eventos

PAM: State Variables Initialization

State	Initialization per t_0	Comments
Tmd	$\textbf{Tmd}_{t_0} = \texttt{MD}$	
Nt	$\textbf{Nt}_{t_0} = \begin{cases} 0.0 & \text{if} \quad \texttt{IED} > t_0 \\ R(\texttt{CNTRL}) \times \texttt{NT} & \text{else} \end{cases}$	
Ipnr	$\textbf{Ipnr}_{t_0} = \begin{cases} 0.0 & \text{if} \quad \texttt{IED} > t_0 \\ \texttt{IPNR} & \text{else} \end{cases}$	

Continued on next page

Figura 2.4: Inicialización de algunas variables del contrato PAM

PAM: State Transition Functions and Payoff Functions

Event	Payoff Function	State Transition Function
AD	0.0	$\mathbf{Ipac}_{t+} = \mathbf{Ipac}_{t-} + Y(\mathbf{Sd}_{t-1}, t)\mathbf{Ipnr}_{t-}\mathbf{Nt}_{t-}$ $\mathbf{Sd}_{t+} = t$
IED	$O^{rf}(\texttt{CURS}, t)R(\texttt{CNTRL})(-1)(\texttt{NT} + \texttt{PDIED})$	$\mathbf{Nt}_{t+} = R(\texttt{CNTRL})\texttt{NT}$ $\mathbf{Ipnr}_{t+} = \begin{cases} 0.0 & \text{if } \texttt{IPNR} = \varnothing \\ \texttt{IPNR} & \text{else} \end{cases}$ $\mathbf{Ipac}_{t+} = \begin{cases} \texttt{IPAC} & \text{if } \texttt{IPAC} \neq \varnothing \\ y\mathbf{Nt}_{t+}\mathbf{Ipnr}_{t+} & \text{if } \texttt{IPANX} \neq \varnothing \wedge \texttt{IPANX} < t \\ 0.0 & \text{else} \end{cases}$ $\mathbf{Sd}_{t+} = t$ with $y = Y(\texttt{IPANX}, t)$
MD	$O^{rf}(\texttt{CURS}, t)(\mathbf{Nsc}_{t-}\mathbf{Nt}_{t-} + \mathbf{Isc}_{t-}\mathbf{Ipac}_{t-} + \mathbf{Feac}_{t-})$	$\mathbf{Nt}_{t+} = 0.0$ $\mathbf{Ipac}_{t+} = 0.0$ $\mathbf{Feac}_{t+} = 0.0$ $\mathbf{Sd}_{t+} = t$
PP	$O^{rf}(\texttt{CURS}, t)f(O^{ev}(\texttt{CID}, \texttt{PP}, t))$	$\mathbf{Ipac}_{t+} = \mathbf{Ipac}_{t-} + Y(\mathbf{Sd}_{t-}, t)\mathbf{Ipnr}_{t-}\mathbf{Nt}_{t-}$ $\mathbf{Fac}_{t+} = \begin{cases} \mathbf{Fac}_{t-} + Y(\mathbf{Sd}_{t-}, t)\mathbf{Nt}_{t-}\texttt{FER} & \text{if } \texttt{FEB} = \text{'N'} \\ \frac{Y(t^{FP-}, t)}{Y(t^{FP-}, t^{FP+})} R(\texttt{CNTRL})\texttt{FER} & \text{else} \end{cases}$ $\mathbf{Nt}_{t+} = f(O^{ev}(\texttt{CID}, \texttt{PP}, t))$

Figura 2.5: Funciones de cambio de estado y pago del contrato PAM

Capítulo 3

Verificación de programas

3.1. Concepto general, herramientas, metodologías

3.2. Isabelle

Capítulo 4

Desarrollo: verificación de contratos financieros usando Isabelle

4.1. Escritura de contratos ACTUS para Cardano

4.2. sss

4.3. sss

Capítulo 5

Conclusión

Capítulo 6

Apéndice

Bibliografía

[ACTUS, 2018] ACTUS (2018). Actus technical specification. https://www.actusfrf.org/techspecs.

[ACTUS, 2019a] ACTUS (2019a). Actus dictionary events. https://github.com/actusfrf/actus-dictionary/blob/master/actus-dictionary-event-types.json. Online; accessed 17 April 2022.

[ACTUS, 2019b] ACTUS (2019b). Actus dictionary states. https://github.com/actusfrf/actus-dictionary/blob/master/actus-dictionary-states.json.

[ACTUS, 2019c] ACTUS (2019c). Actus dictionary terms. https://github.com/actusfrf/actus-dictionary/blob/master/actus-dictionary-terms.json.

[ACTUS, 2019d] ACTUS (2019d). Actus taxonomy. https://www.actusfrf.org/taxonomy.

[Brünjes and Gabbay, 2020] Brünjes, L. and Gabbay, M. J. (2020). Utxo- vs account-based smart contract blockchain programming paradigms. *CoRR*, abs/2003.14271.

[HaskellWiki, 2020] HaskellWiki (2020). Algebraic data type. https://wiki.haskell.org/Algebraic_data_type. Online; accessed 30 April 2022.

[Kondratiuk et al., 2021] Kondratiuk, D., Seijas, P. L., Nemish, A., and Thompson, S. (2021). Standardized crypto-loans on the cardano blockchain. In *5th Workshop on Trusted Smart Contracts, Financial Cryptography and Data Security 2021*.

[Lamela Seijas et al., 2020] Lamela Seijas, P., Nemish, A., Smith, D., and Thompson, S. (2020). Marlowe: Implementing and analysing financial contracts on blockchain. In Bernhard, M., Bracciali, A., Camp, L. J., Matsuo, S., Maurushat, A., Rønne, P. B., and Sala, M., editors, *Financial Cryptography and Data Security*, pages 496–511, Cham. Springer International Publishing.

[Zahnentferner, 2018] Zahnentferner, J. (2018). Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies. Cryptology ePrint Archive, Report 2018/262. https://ia.cr/2018/262.