Verificación de smart contracts en Marlowe para la blockchain Cardano

Julián Ferres

Facultad de Ingeniería
Universidad de Buenos Aires.

3 de agosto de 2022

.UBAfiuba
FACULTAD DE INGENIERÍA

Índice de Contenidos

Índice de contenidos

Índice de contenidos

Cadenas de bloques o *Blockchains*

Las cadenas de bloques, conocidas en inglés como blockchains, son estructuras de datos en las cuales la información se divide en conjuntos (bloques) que cuentan con información adicional relativa a bloques previos de la cadena.

Cadenas de bloques o *Blockchains*

Las cadenas de bloques, conocidas en inglés como blockchains, son estructuras de datos en las cuales la información se divide en conjuntos (bloques) que cuentan con información adicional relativa a bloques previos de la cadena.

Con esta organización relativa, y con ayuda de técnicas criptográficas, la información de un bloque solo puede ser alterada modificando todos los bloques anteriores.

Cadenas de bloques o *Blockchains*

Las cadenas de bloques, conocidas en inglés como blockchains, son estructuras de datos en las cuales la información se divide en conjuntos (bloques) que cuentan con información adicional relativa a bloques previos de la cadena.

Con esta organización relativa, y con ayuda de técnicas criptográficas, la información de un bloque solo puede ser alterada modificando todos los bloques anteriores.

Esta propiedad facilita su aplicación en un entorno distribuido, de manera tal que la cadena de bloques puede modelar una base de datos pública no relacional, que contenga un registro histórico irrefutable de información.

Cadenas de bloques o *Blockchains*

Las cadenas de bloques, conocidas en inglés como blockchains, son estructuras de datos en las cuales la información se divide en conjuntos (bloques) que cuentan con información adicional relativa a bloques previos de la cadena.

Con esta organización relativa, y con ayuda de técnicas criptográficas, la información de un bloque solo puede ser alterada modificando todos los bloques anteriores.

Esta propiedad facilita su aplicación en un entorno distribuido, de manera tal que la cadena de bloques puede modelar una base de datos pública no relacional, que contenga un registro histórico irrefutable de información.

En la práctica esta técnica ha permitido la implementación de un registro contable o ledger distribuido que soporta y garantiza la seguridad de transacciones y dinero digital.

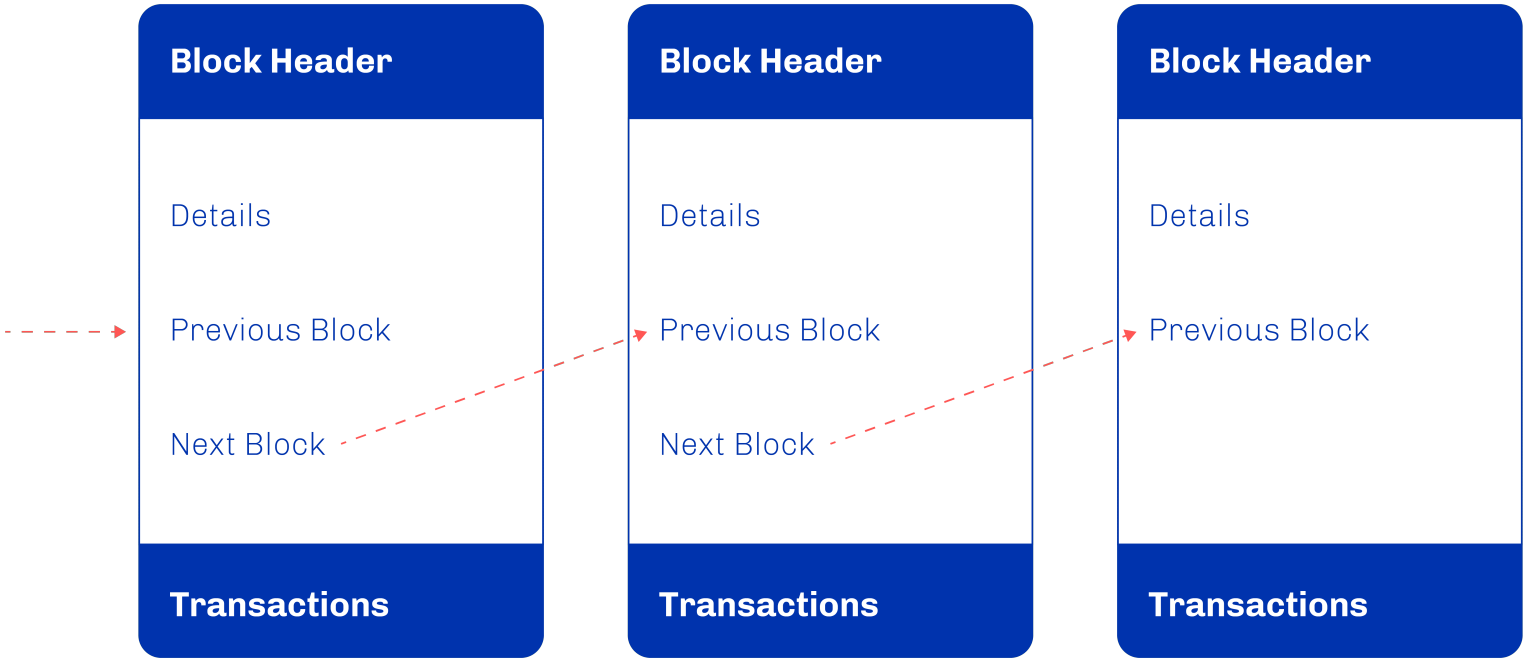


Figura: Representación simplificada de los datos en un bloque de la cadena. Extraída de [**?**].

Criptomonedas

Las criptomonedas son activos digitales que se almacenan en el ledger y están diseñadas para servir como medio de intercambio de bienes o servicios.

Criptomonedas

Las criptomonedas son activos digitales que se almacenan en el ledger y están diseñadas para servir como medio de intercambio de bienes o servicios.

Los ledgers de blockchain son utilizados como tecnología subyacente para la creación de criptomonedas en un entorno descentralizado.

Criptomonedas

Las criptomonedas son activos digitales que se almacenan en el ledger y están diseñadas para servir como medio de intercambio de bienes o servicios.

Los ledgers de blockchain son utilizados como tecnología subyacente para la creación de criptomonedas en un entorno descentralizado.

Los protocolos de blockchain utilizan técnicas criptográficas rigurosas para permitir la creación de criptomonedas, asegurar y verificar la propiedad de las mismas y los registros de movimiento de fondos.

Criptomonedas

Las criptomonedas son activos digitales que se almacenan en el ledger y están diseñadas para servir como medio de intercambio de bienes o servicios.

Los ledgers de blockchain son utilizados como tecnología subyacente para la creación de criptomonedas en un entorno descentralizado.

Los protocolos de blockchain utilizan técnicas criptográficas rigurosas para permitir la creación de criptomonedas, asegurar y verificar la propiedad de las mismas y los registros de movimiento de fondos.

El precio de la criptomoneda no está controlado por un gobierno o una institución financiera centralizada. Se define por su valor, la correlación con las cifras del mundo real y está impulsado por la oferta y la demanda del mercado.

Smart contracts

Un contrato inteligente o *smart contract* es un acuerdo digital automatizado.

Smart contracts

Un contrato inteligente o *smart contract* es un acuerdo digital automatizado.

Los mismos están escritos en código, rastrean, verifican y ejecutan las transacciones de un contrato entre varias partes.

Smart contracts

Un contrato inteligente o *smart contract* es un acuerdo digital automatizado.

Los mismos están escritos en código, rastrean, verifican y ejecutan las transacciones de un contrato entre varias partes.

Las transacciones del contrato se ejecutan automáticamente mediante el código del smart contract cuando se cumplen las condiciones predeterminadas. Esencialmente, un contrato inteligente es un programa cuyas entradas y salidas son acciones en una cadena de bloques.

Smart contracts

Un contrato inteligente o *smart contract* es un acuerdo digital automatizado.

Los mismos están escritos en código, rastrean, verifican y ejecutan las transacciones de un contrato entre varias partes.

Las transacciones del contrato se ejecutan automáticamente mediante el código del smart contract cuando se cumplen las condiciones predeterminadas. Esencialmente, un contrato inteligente es un programa cuyas entradas y salidas son acciones en una cadena de bloques.

Los smart contracts son auto-ejecutables y no requieren las acciones o la presencia de terceros. El código del contrato inteligente se almacena y distribuye en la *blockchain*, lo que lo hace transparente e irreversible.

Índice de contenidos

Cardano es una plataforma blockchain descentralizada de tercera generación, cuya criptomoneda es llamada *ada*.

Cardano es una plataforma blockchain descentralizada de tercera generación, cuya criptomoneda es llamada *ada*.

- **La primera generación** de blockchains (con Bitcoin como gran representante) ofrece ledgers descentralizados para la transferencia segura de criptomonedas.

Cardano es una plataforma blockchain descentralizada de tercera generación, cuya criptomoneda es llamada *ada*.

- **La primera generación** de blockchains (con Bitcoin como gran representante) ofrece ledgers descentralizados para la transferencia segura de criptomonedas. Sin embargo, tales *blockchains* no proporcionaron un entorno funcional para la liquidación de acuerdos complejos y el desarrollo de aplicaciones descentralizadas (dApps).

.UBAfiuba
FACULTAD DE INGENIERÍA

Cardano es una plataforma blockchain descentralizada de tercera generación, cuya criptomoneda es llamada *ada*.

- **La primera generación** de blockchains (con Bitcoin como gran representante) ofrece ledgers descentralizados para la transferencia segura de criptomonedas. Sin embargo, tales *blockchains* no proporcionaron un entorno funcional para la liquidación de acuerdos complejos y el desarrollo de aplicaciones descentralizadas (dApps).
- **La segunda generación** (cuyo ejemplo más conocido es Ethereum) proporcionó soluciones mejoradas para redactar y ejecutar contratos inteligentes, desarrollar aplicaciones y crear diferentes tipos de tokens.

Cardano es una plataforma blockchain descentralizada de tercera generación, cuya criptomoneda es llamada *ada*.

- **La primera generación** de blockchains (con Bitcoin como gran representante) ofrece ledgers descentralizados para la transferencia segura de criptomonedas. Sin embargo, tales *blockchains* no proporcionaron un entorno funcional para la liquidación de acuerdos complejos y el desarrollo de aplicaciones descentralizadas (dApps).
- **La segunda generación** (cuyo ejemplo más conocido es Ethereum) proporcionó soluciones mejoradas para redactar y ejecutar contratos inteligentes, desarrollar aplicaciones y crear diferentes tipos de tokens. Sin embargo, la segunda generación de cadenas de bloques a menudo enfrenta problemas en términos de escalabilidad.

.UBAfiuba FACULTAD DE INGENIERÍA

Cardano como *blockchain* de 3ra generación

Cardano se concibe como la cadena de bloques de tercera generación.
La misma combina las propiedades de las generaciones anteriores y ofrece las siguientes propiedades:

Cardano como *blockchain* de 3ra generación

Cardano se concibe como la cadena de bloques de tercera generación.
La misma combina las propiedades de las generaciones anteriores y ofrece las siguientes propiedades:

- **Escalabilidad**: Rendimiento de transacciones, escala de datos, ancho de banda de la red.

Cardano como *blockchain* de 3ra generación

Cardano se concibe como la cadena de bloques de tercera generación.
La misma combina las propiedades de las generaciones anteriores y ofrece las siguientes propiedades:

- **Escalabilidad**: Rendimiento de transacciones, escala de datos, ancho de banda de la red.
- **Funcionalidad**: Además del procesamiento de transacciones, la cadena de bloques debe proporcionar todos los medios para la liquidación de acuerdos comerciales.

Cardano como *blockchain* de 3ra generación

Cardano se concibe como la cadena de bloques de tercera generación.
La misma combina las propiedades de las generaciones anteriores y ofrece las siguientes propiedades:

- **Escalabilidad**: Rendimiento de transacciones, escala de datos, ancho de banda de la red.
- **Funcionalidad**: Además del procesamiento de transacciones, la cadena de bloques debe proporcionar todos los medios para la liquidación de acuerdos comerciales.
- **Desarrollo e Integración**: Es importante asegurarse que la blockchain esté en constante desarrollo en términos de mantenibilidad y sea interoperable con otras blockchains e instituciones financieras.

Ada como criptomoneda de Cardano

Cada ledger de blockchain tiene su criptomoneda subyacente o moneda nativa. Ada es la moneda nativa o principal en Cardano. Esto significa que ada es la principal unidad de pago en Cardano.

Ada como criptomoneda de Cardano

Cada ledger de blockchain tiene su criptomoneda subyacente o moneda nativa. Ada es la moneda nativa o principal en Cardano. Esto significa que ada es la principal unidad de pago en Cardano.

Cardano también admite la creación de **tokens nativos**: activos digitales que se crean para fines específicos.

Ada como criptomoneda de Cardano

Cada ledger de blockchain tiene su criptomoneda subyacente o moneda nativa. Ada es la moneda nativa o principal en Cardano. Esto significa que ada es la principal unidad de pago en Cardano.

Cardano también admite la creación de **tokens nativos**: activos digitales que se crean para fines específicos.

Por lo tanto los usuarios, desarrolladores y empresas pueden usar la cadena de bloques de Cardano para crear tokens que representen una huella de valor.

Ada como criptomoneda de Cardano

Cada ledger de blockchain tiene su criptomoneda subyacente o moneda nativa. Ada es la moneda nativa o principal en Cardano. Esto significa que ada es la principal unidad de pago en Cardano.

Cardano también admite la creación de **tokens nativos**: activos digitales que se crean para fines específicos.

Por lo tanto los usuarios, desarrolladores y empresas pueden usar la cadena de bloques de Cardano para crear tokens que representen una huella de valor.

Un token puede ser **fungible** (intercambiable) o **no fungible** (único) y actuar como unidad de pago, recompensa, activo comercial o contenedor de información.

El lenguaje Marlowe

Marlowe es un lenguaje de dominio específico creado por IOHK. El mismo posee pocas sentencias soportadas que describen el comportamiento de un conjunto fijo y finito de roles en un contrato.

Marlowe está diseñado para crear bloques para contratos financieros: pagos o depósitos de las partes, elecciones e información del mundo real.

Índice de contenidos

Contratos Financieros

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.

Contratos Financieros

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 mediante algoritmos.

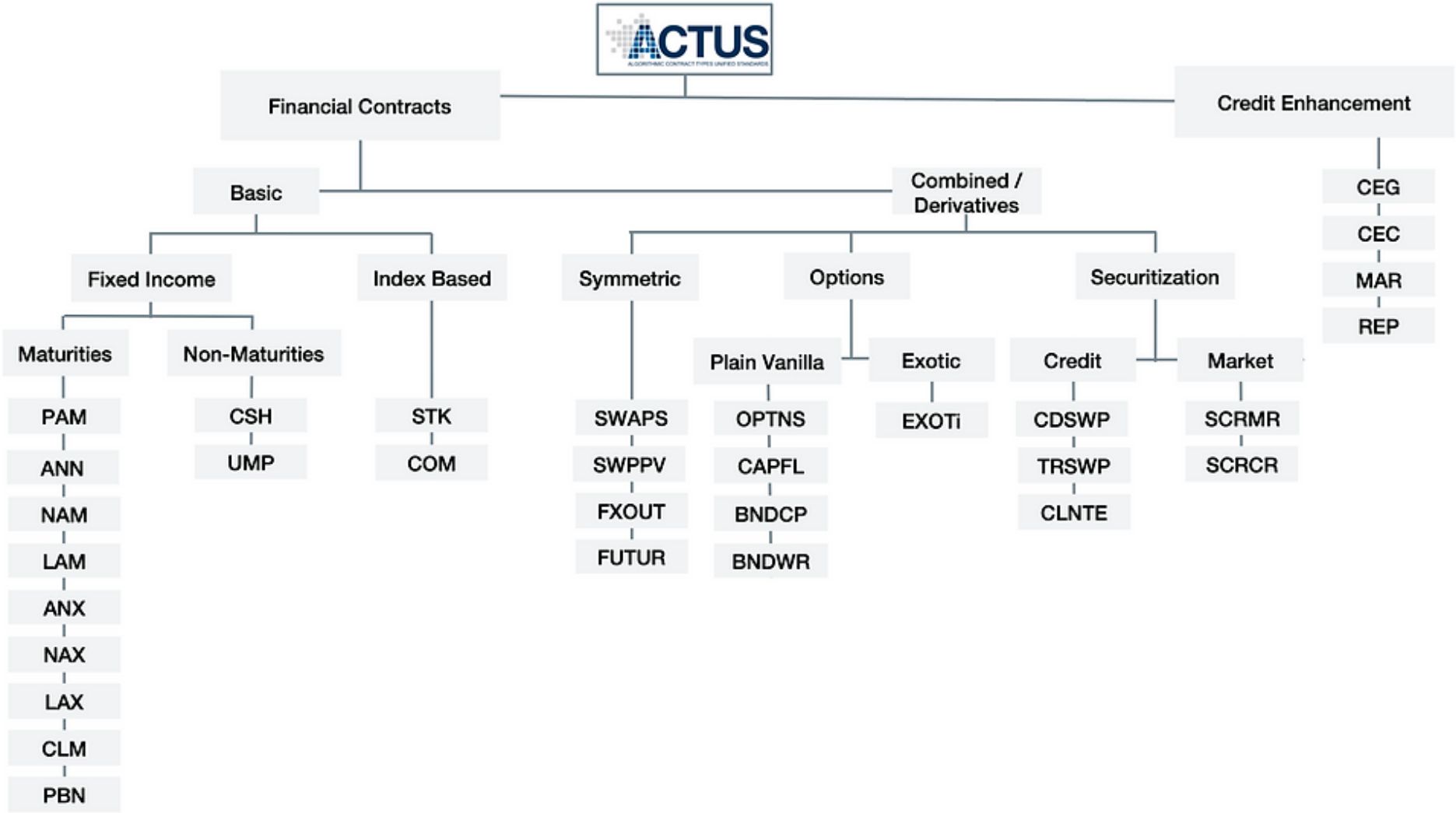


Figura: Taxonomía ACTUS

.UBAfiuba
FACULTAD DE INGENIERÍA

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

> *"Se entrega un monto de dinero inicial, a cambio de pagos de intereses cíclicos y la devolución del dinero inicial en el vencimiento del contrato."*

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

> *"Se entrega un monto de dinero inicial, 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:

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

> *"Se entrega un monto de dinero inicial, 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:

- Los pagos de intereses pueden ser mensuales, anuales, mediante períodos arbitrarios.

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

> *"Se entrega un monto de dinero inicial, 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:

- Los pagos de intereses pueden ser mensuales, anuales, mediante períodos arbitrarios.
- Las tasas pueden ser de fijas o variables.

En general, el intercambio de flujos de efectivo entre partes sigue ciertos patrones.
Un patrón típico es un contrato de préstamo de tipo *bullet*:

> *"Se entrega un monto de dinero inicial, 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:

- Los pagos de intereses pueden ser mensuales, anuales, mediante períodos arbitrarios.
- Las tasas pueden ser de fijas o variables.
- Pueden usarse diferentes métodos de cálculo de fracciones anuales o que no haya ningún interés.

.UBAfiuba FACULTAD DE INGENIERÍA

Índice de contenidos

Concepto general, herramientas y metodologías

Los asistentes de pruebas formales son herramientas de software diseñadas para ayudar a sus usuarios a realizar pruebas, especialmente en cálculo lógico.

Por lo general, los llamamos asistentes de demostración o demostradores interactivos de teoremas.

Concepto general, herramientas y metodologías

Los asistentes de pruebas formales son herramientas de software diseñadas para ayudar a sus usuarios a realizar pruebas, especialmente en cálculo lógico.

Por lo general, los llamamos asistentes de demostración o demostradores interactivos de teoremas.

La principal fortaleza de los asistentes de prueba es que ayudan a desarrollar pruebas altamente confiables e inequívocas de enunciados matemáticos, usando lógica precisa. Se pueden usar para probar resultados arbitrariamente avanzados, y no solo ejemplos simples.

Algunos asistentes de pruebas

Hay una gran cantidad de asistentes de prueba en desarrollo o uso alrededor del mundo. A continuación presentamos una lista de los principales, clasificados por sus fundamentos lógicos:

Algunos asistentes de pruebas

Hay una gran cantidad de asistentes de prueba en desarrollo o uso alrededor del mundo. A continuación presentamos una lista de los principales, clasificados por sus fundamentos lógicos:

- **Teoría de conjuntos**: Isabelle/ZF, Metamath, Mizar
- **Teoría simple de tipos**: HOL4, HOL Light, Isabelle/HOL
- **Teoría dependiente de tipos**: Agda, Coq, Lean, Matita, PVS
- **Lógica de primer orden, de tipo Lisp**: ACL2

Índice de contenidos

Índice de contenidos

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

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.

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.
- **Starting date**: t_0 representa la fecha de comienzo del contrato, y marca el instante en el cual las condiciones y estado del contrato están siendo representados.

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.
- **Starting date**: t_0 representa la fecha de comienzo del contrato, y marca el instante en el cual las condiciones y estado del contrato están siendo representados.
- **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:

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.
- **Starting date**: t_0 representa la fecha de comienzo del contrato, y marca el instante en el cual las condiciones y estado del contrato están siendo representados.
- **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.

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.
- **Starting date**: t_0 representa la fecha de comienzo del contrato, y marca el instante en el cual las condiciones y estado del contrato están siendo representados.
- **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.

  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.

- **Eventos**: Un evento de contrato 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 de efectivo o se actualizan los estados del contrato.

- **Eventos**: Un evento de contrato 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 de efectivo o se actualizan los estados del contrato.

- **Eventos**: Un evento de contrato 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 de efectivo o se actualizan los estados del contrato.
- **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. Se puede analizar un contrato ACTUS en términos de estado actual y flujos de efectivo futuros para cada punto del tiempo.

- **Funciones de transición de estado**: Dichas funciones, conocidas en Inglés como *'State Transition 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.

- **Funciones de transición de estado**: Dichas funciones, conocidas en Inglés como *'State Transition 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.

- **Funciones de transición de estado**: Dichas funciones, conocidas en Inglés como *'State Transition 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 $\mathbf{Ipac}_{t^+}$.

.UBAfiuba FACULTAD DE INGENIERÍA

- **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 términos del contrato. Si fuera necesario, el flujo de dinero puede ser indexado con el tiempo del evento: c_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 términos 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 análoga 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.

- **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 términos 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 análoga 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().

Índice de contenidos

Introducción

En esta sección veremos como se desarrolló la escritura de tres contratos ACTUS para la blockchain Cardano, bajo la supervisión de IOHK.

Cabe destacar que durante esta tarea tuve la colaboración de Yves Hauser, con quien conversamos sobre decisiones de diseño e implementación de los contratos correspondientes. Yves fue el responsable de integrar mis cambios a la rama *master* del repositorio de `marlowe-cardano` [**?**].

Estructura del proyecto ACTUS en Cardano

A grandes rasgos, la estructura del generador de contratos ACTUS tiene las siguientes partes:

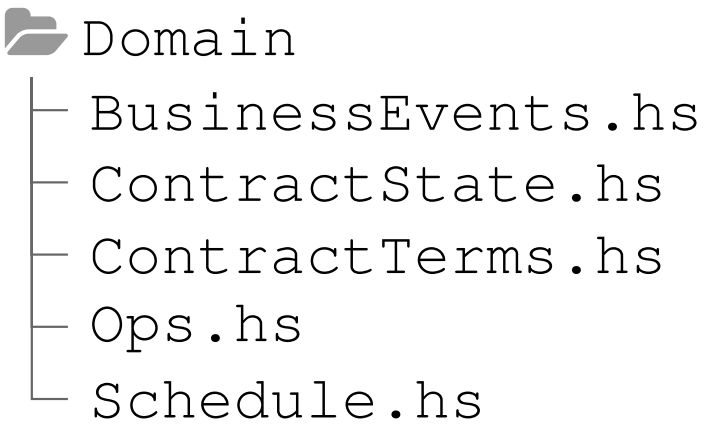
- **Domain**: El mismo está conformado por los archivos que modelan el dominio de los contratos ACTUS.

Estructura del proyecto ACTUS en Cardano

A grandes rasgos, la estructura del generador de contratos ACTUS tiene las siguientes partes:

- **Domain**: El mismo está conformado por los archivos que modelan el dominio de los contratos ACTUS.

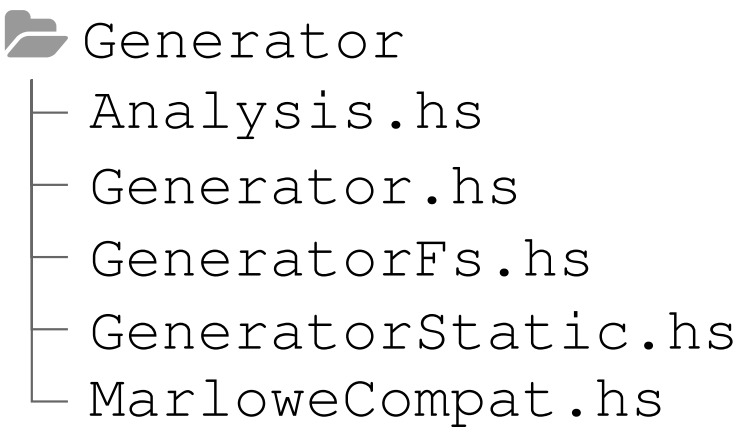
  Entre ellos se pueden destacar:



- **Generator**: En este directorio se implementan los diferentes generadores y compatibilidad hacia el lenguaje Marlowe.

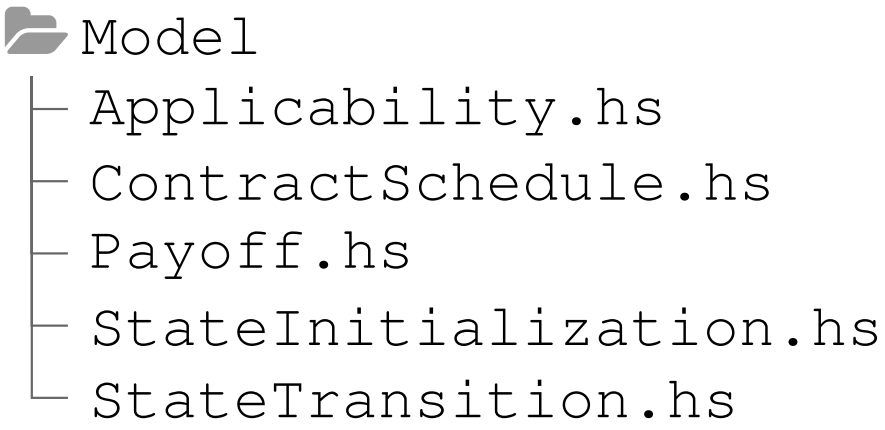
- **Generator**: En este directorio se implementan los diferentes generadores y compatibilidad hacia el lenguaje Marlowe.

  La estructura de dicho directorio es la siguiente:



- **Model**: En este directorio se encuentran los archivos que modelan la lógica expuesta por el estándar ACTUS, tales como el *scheduling*, la inicialización de variables de estado y funciones de transición de estado y de pago.

- **Model**: En este directorio se encuentran los archivos que modelan la lógica expuesta por el estándar ACTUS, tales como el *scheduling*, la inicialización de variables de estado y funciones de transición de estado y de pago.



- **Model**: En este directorio se encuentran los archivos que modelan la lógica expuesta por el estándar ACTUS, tales como el *scheduling*, la inicialización de variables de estado y funciones de transición de estado y de pago.

```
Model
├── Applicability.hs
├── ContractSchedule.hs
├── Payoff.hs
├── StateInitialization.hs
└── StateTransition.hs
```

- **Utility**: En este directorio se encuentran algunos archivos con funciones que se utilizan para aislar la lógica del cálculo de fechas, que suele tornarse complejo y repetitivo durante los contratos.

.UBAfiuba FACULTAD DE INGENIERÍA

Índice de contenidos

Índice de contenidos

Marlowe está diseñado para soportar la ejecución de contratos financieros en la blockchain, específicamente en Cardano.

Marlowe está diseñado para soportar la ejecución de contratos financieros en la blockchain, específicamente en Cardano.

Los contratos se construyen *reuniendo una pequeña cantidad de constructores* que se pueden combinar para describir muchos tipos diferentes de contratos financieros.

Marlowe está diseñado para soportar la ejecución de contratos financieros en la blockchain, específicamente en Cardano.

Los contratos se construyen *reuniendo una pequeña cantidad de constructores* que se pueden combinar para describir muchos tipos diferentes de contratos financieros.

Antes de describir estos constructores, debemos analizar el enfoque general para modelar contratos en Marlowe, y el contexto en el que se ejecutan los mismos (la blockchain de Cardano).

- Contratos

Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Participantes y roles

- Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Participantes y roles
- Valores y *tokens*

- Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Participantes y roles
- Valores y *tokens*
- Cuentas

- Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Participantes y roles
- Valores y *tokens*
- Cuentas
- Pasos y Estados

- Contratos

```
data Contract = Close
              | Pay party payee token value contract
              | If observation contract1 contract2
              | When [Case] timeout contract
              | Let valueId value contract
              | Assert observation contract
```

- Participantes y roles
- Valores y *tokens*
- Cuentas
- Pasos y Estados
- Simulación omnisciente (*Marlowe Playground*)

Breve ejemplo de un contrato de Intercambio

```
When [Case (Deposit "alice" "alice" ada price)
        (When [ Case aliceChoice
                    (When [ Case bobChoice
                                (If (aliceChosen `ValueEQ` bobChosen)
                                     agreement
                                     arbitrate)
                            ]
                            60
                            arbitrate)
                ]
                40
                Close)
     ]
     10
     Close
```

Ejecución de un contrato Marlowe

Ejecutar un contrato de Marlowe en la cadena de bloques de Cardano significa restringir las transacciones generadas por el usuario a la lógica del contrato.

Ejecución de un contrato Marlowe

Ejecutar un contrato de Marlowe en la cadena de bloques de Cardano significa restringir las transacciones generadas por el usuario a la lógica del contrato.

Una **transacción** contiene una **lista ordenada de *entradas* o *acciones***. El intérprete de Marlowe se ejecuta durante la validación de transacciones.

Procesando una transacción

- Primero, se evalúa el contrato *paso* a *paso* hasta que no se puede cambiar más sin procesar alguna entrada, una condición que se llama *quiescent* (o inactiva).
  En esta etapa, avanza a través cualquier `When` cuyos timeouts hayan sido superados, y todos los constructores `If`, `Let`, `Pay` y `Close`, sin procesar ninguna entrada.

Procesando una transacción

- Primero, se evalúa el contrato *paso* a *paso* hasta que no se puede cambiar más sin procesar alguna entrada, una condición que se llama *quiescent* (o inactiva).
  En esta etapa, avanza a través cualquier `When` cuyos timeouts hayan sido superados, y todos los constructores `If`, `Let`, `Pay` y `Close`, sin procesar ninguna entrada.

- A continuación, se procesa la primera entrada, y luego el contrato se vuelve a procesar hasta la inactividad. Este proceso se repite hasta que se procesan todas las entradas. En cada paso, el contacto actual y el estado cambiarán, es posible que se procesen algunas entradas y se realicen pagos.

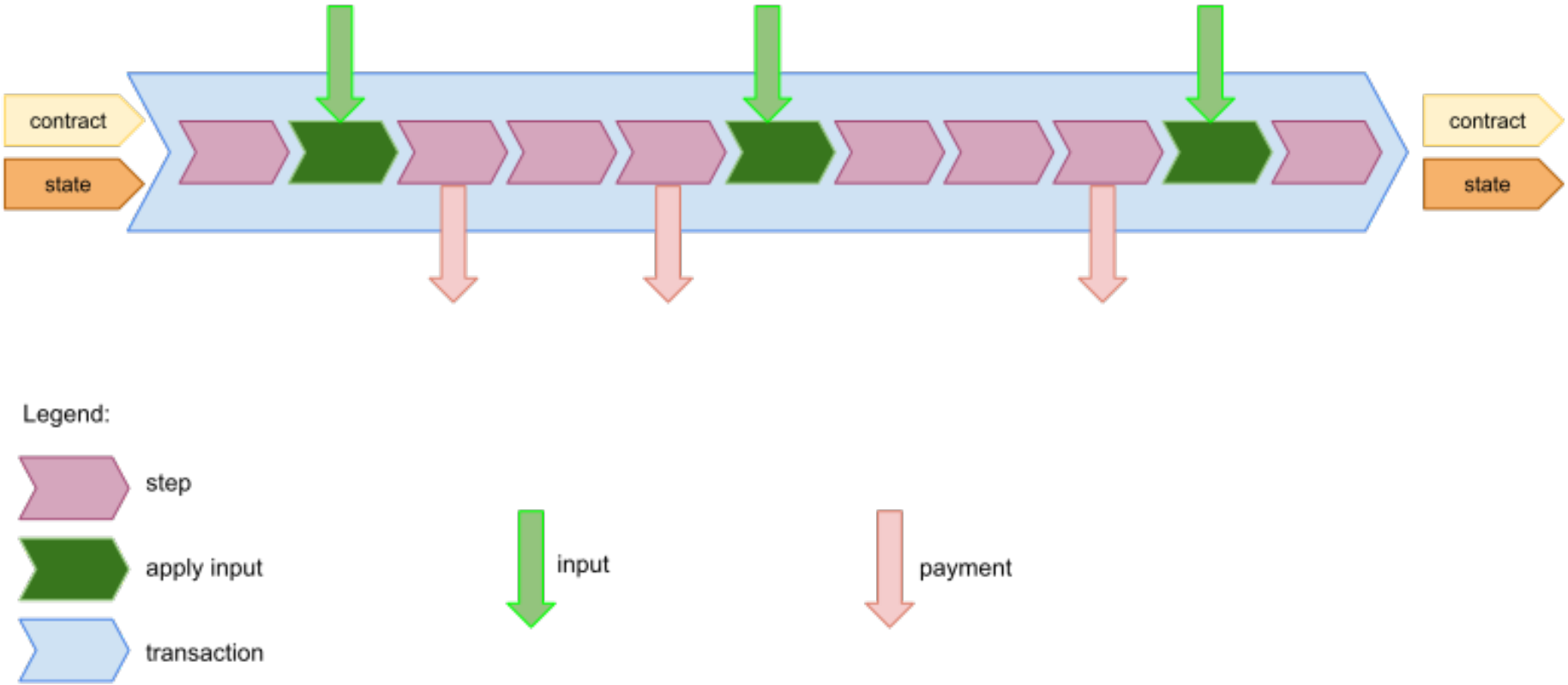
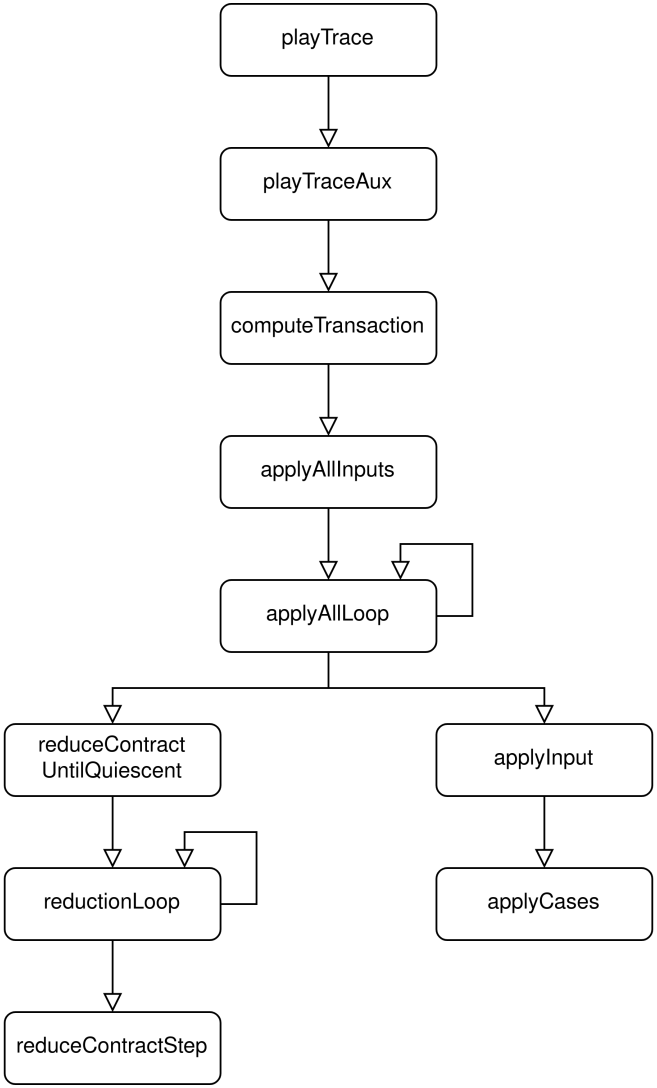
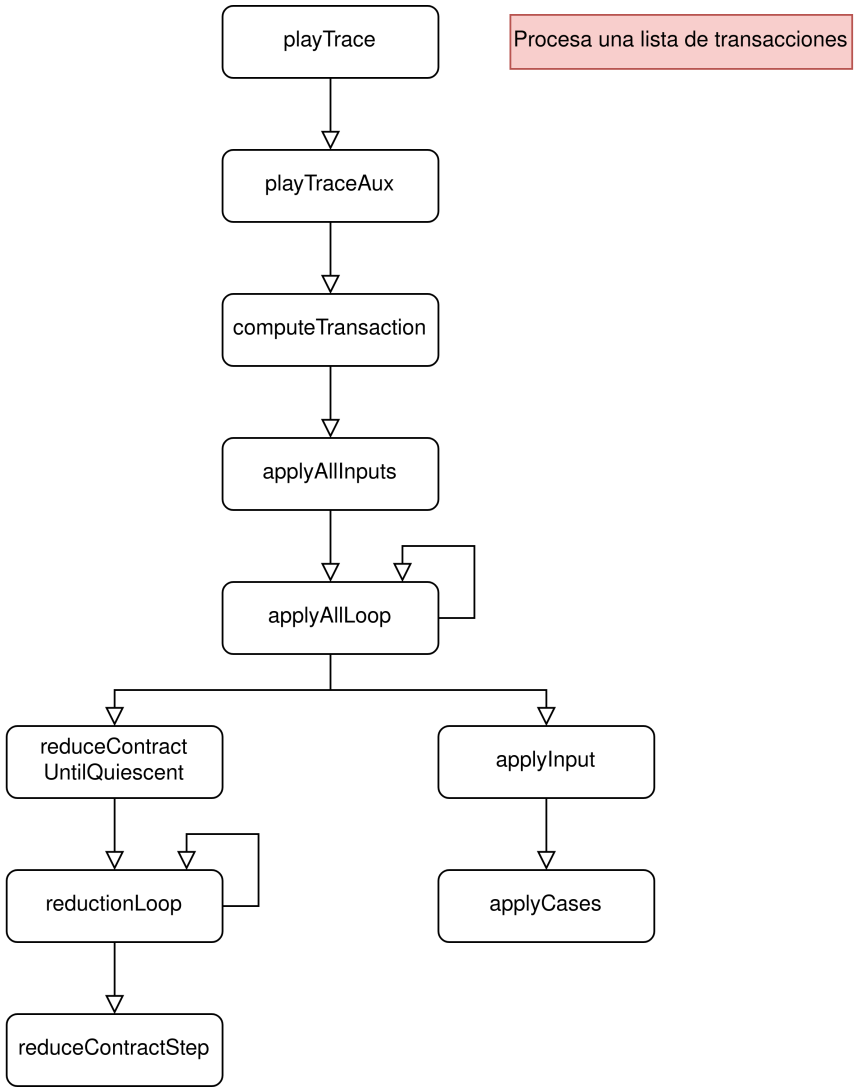


Figura: Diagrama de ejemplo de una transacción. Extraído de la documentación oficial del Modelo de Marlowe.

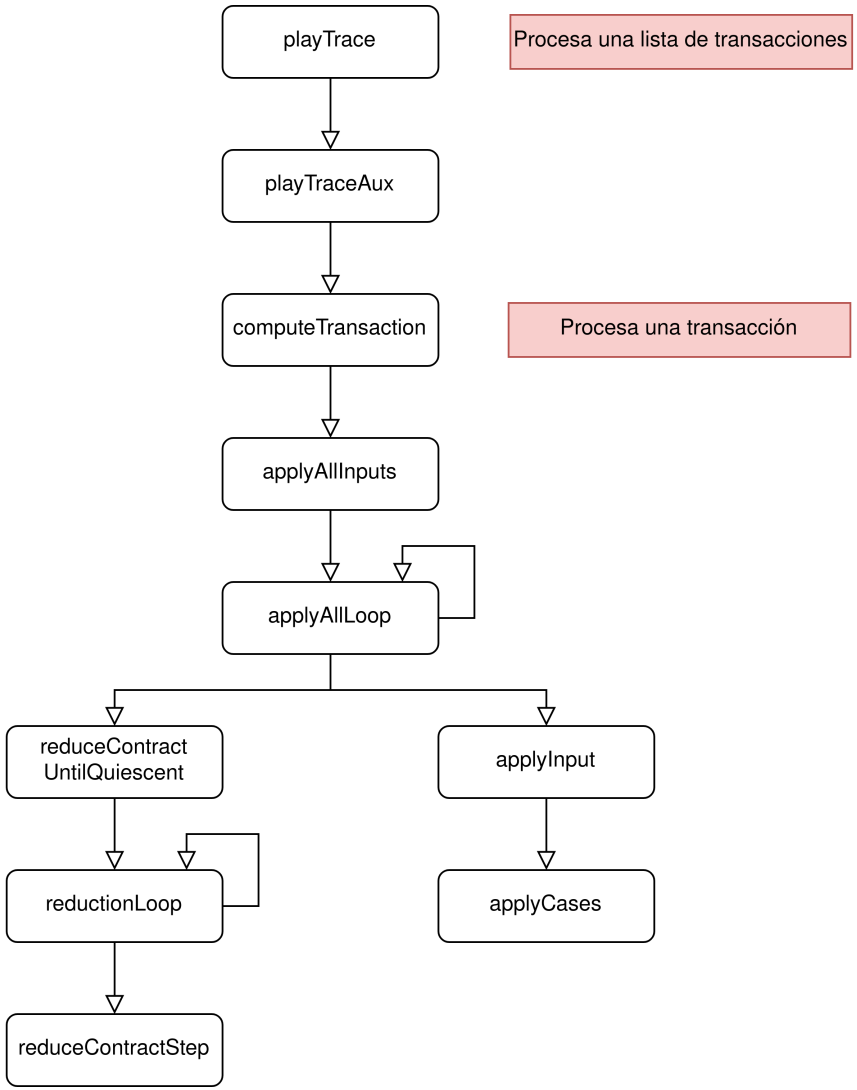
Funciones utilizadas en el modelo



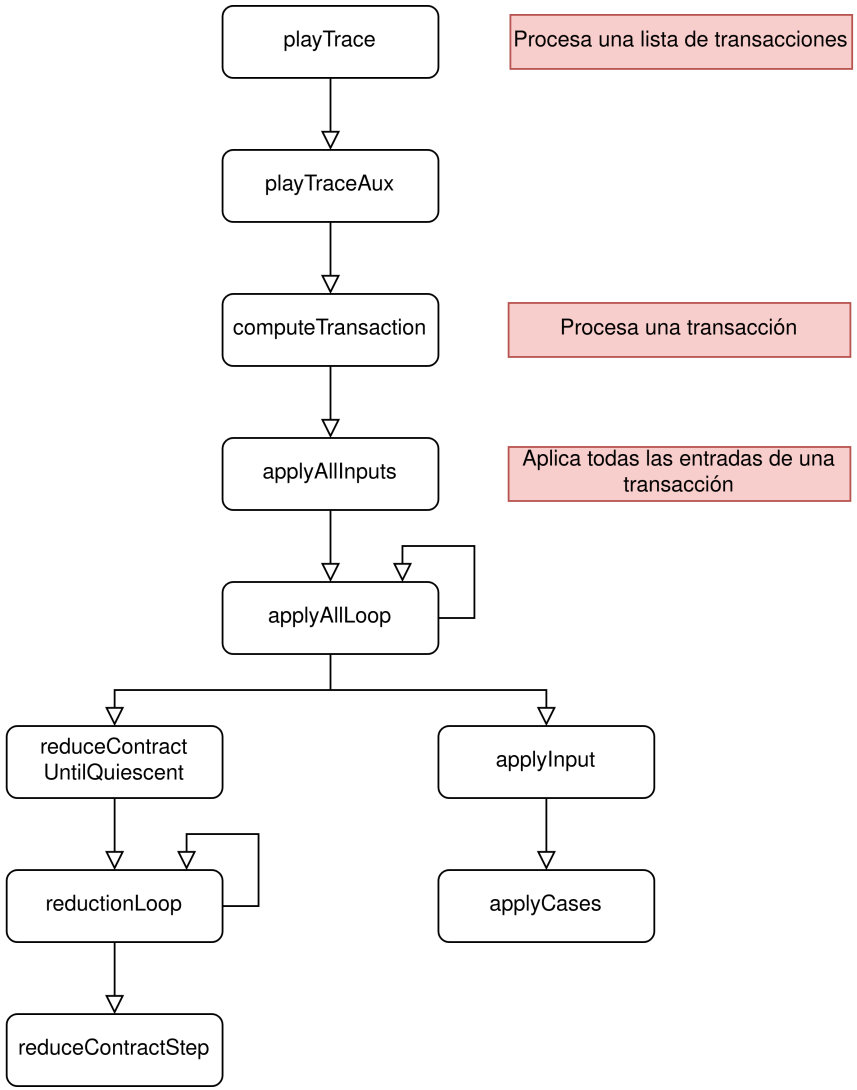
Funciones utilizadas en el modelo



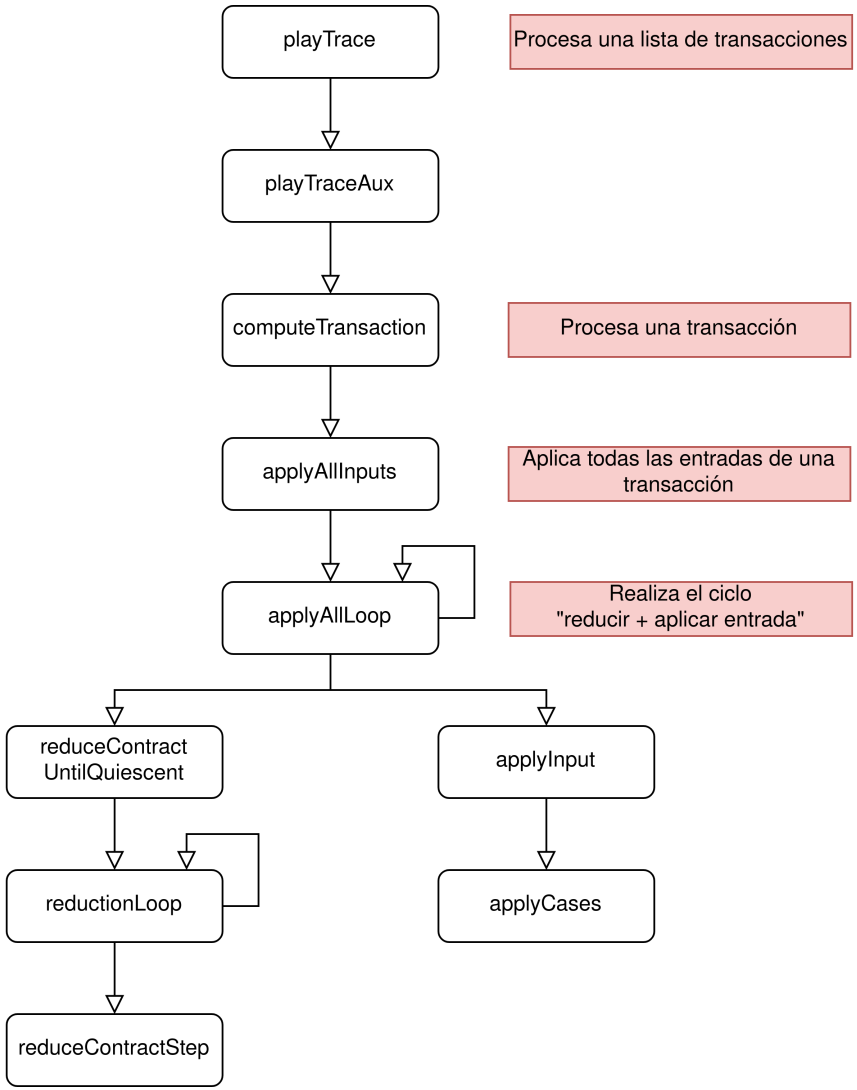
Funciones utilizadas en el modelo



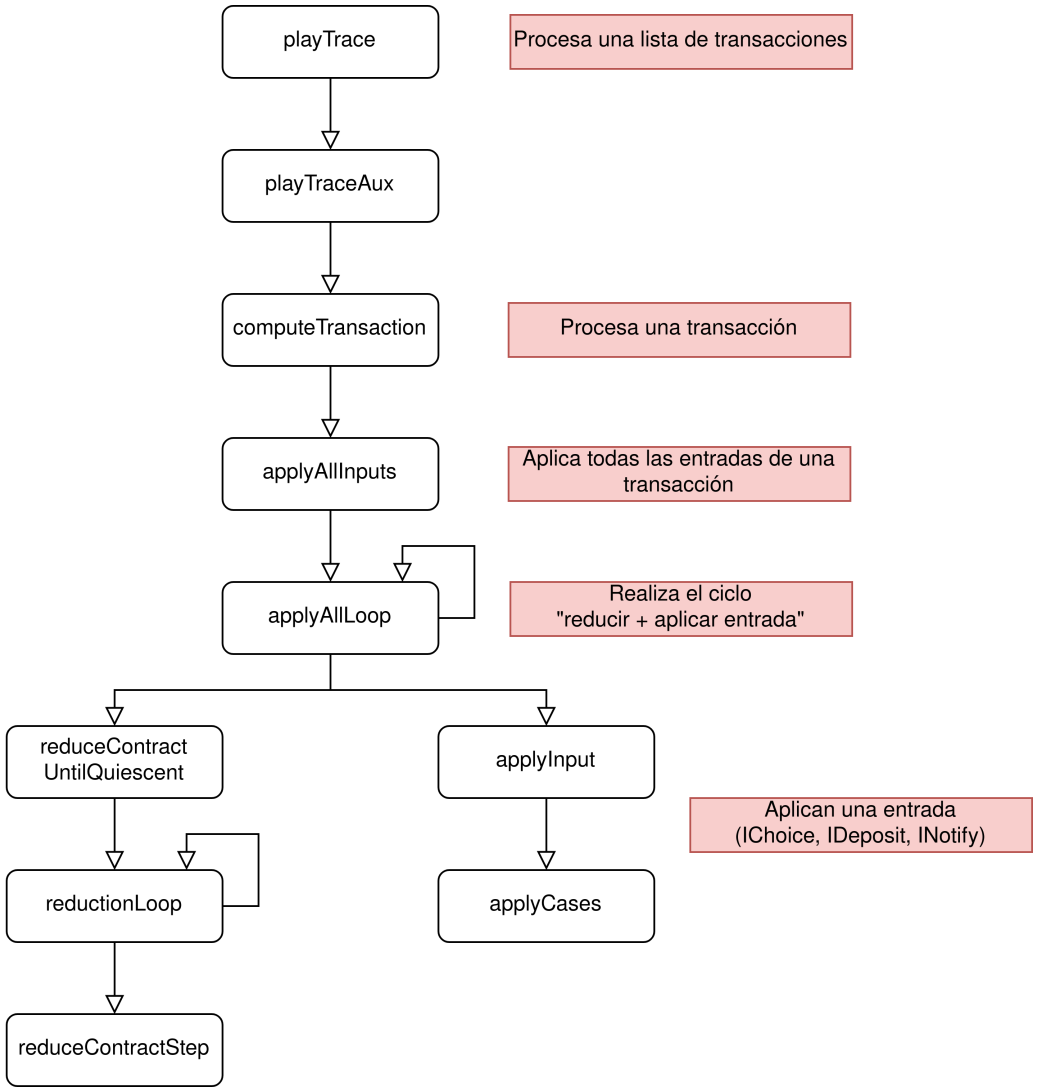
Funciones utilizadas en el modelo



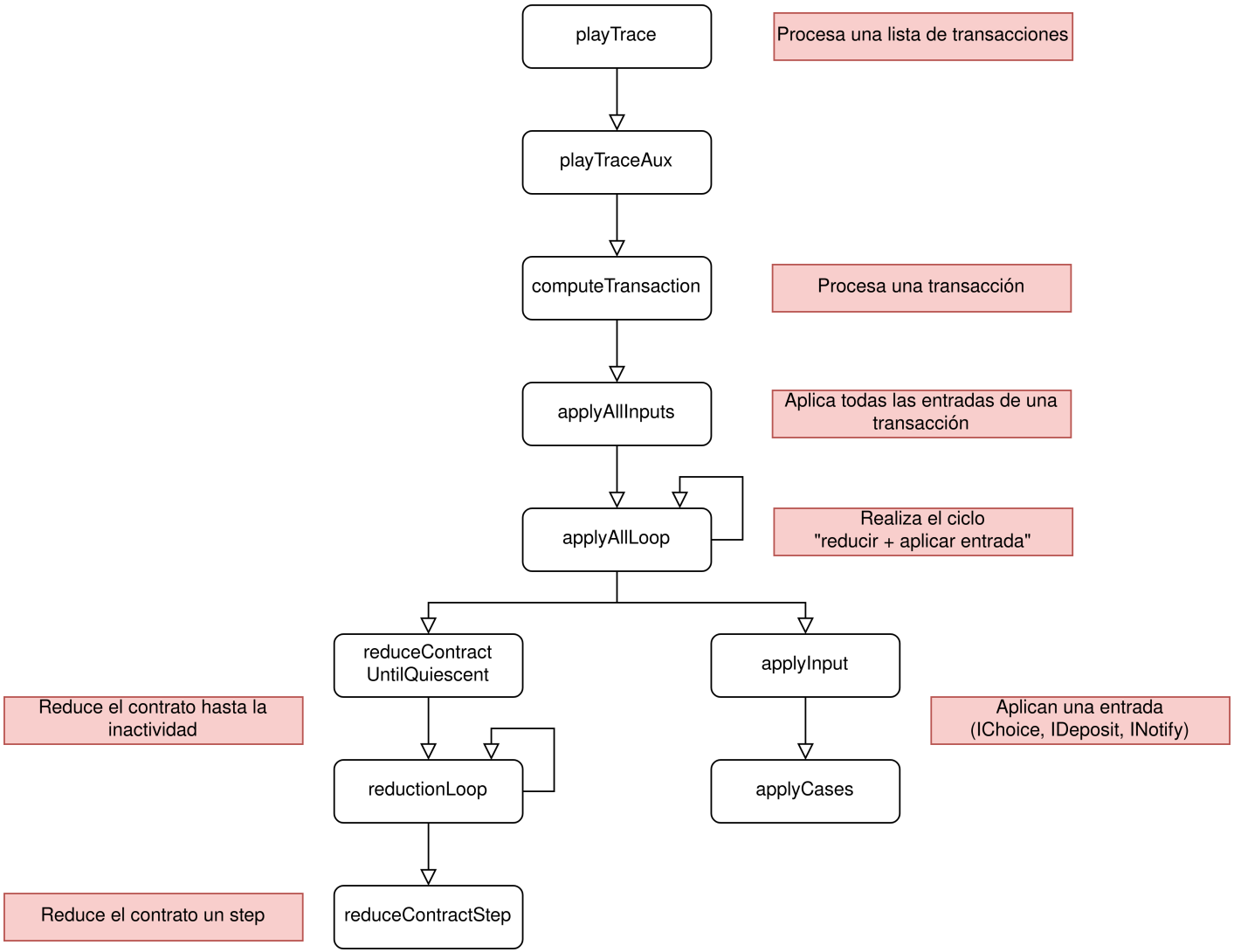
Funciones utilizadas en el modelo



Funciones utilizadas en el modelo



Funciones utilizadas en el modelo



Índice de contenidos

Prueba de minSlot no decreciente en COM

Supongamos que queremos probar la siguiente propiedad:

> *"Evaluar applyAllInputs en el contrato COM4, para cualquier environment y state dado, produce un estado con minSlot mayor al actual."*

Este tipo de prueba evita la 'vuelta al pasado' por parte del contrato, que podrían llevar a caminos de ejecución no deseados.

```
lemma applyAllInputsDoesNotDecreaseMinSlot :
"applyAllInputs env sta COM4 inputList =
    ApplyAllSuccess reduced warnings payments newstate cont ⟹
  (minSlot sta) ≤ (minSlot newstate)"
  apply auto
  apply (cases inputList)
  apply auto
  by (smt (z3) ApplyAllResult.distinct(1) (* propuesto por Sledgehammer *)
                 ApplyResult.case(1)
                 ApplyResult.case(2)
                 ApplyResult.exhaust
                 COM4_def
                 applyAllLoop_preserves_minSlot
                 applyCases_preserves_minSlot applyInput.simps(1))
```

Código 9: Prueba del lema de *minSlot* para un contrato COM.

Índice de contenidos

El contrato Auction

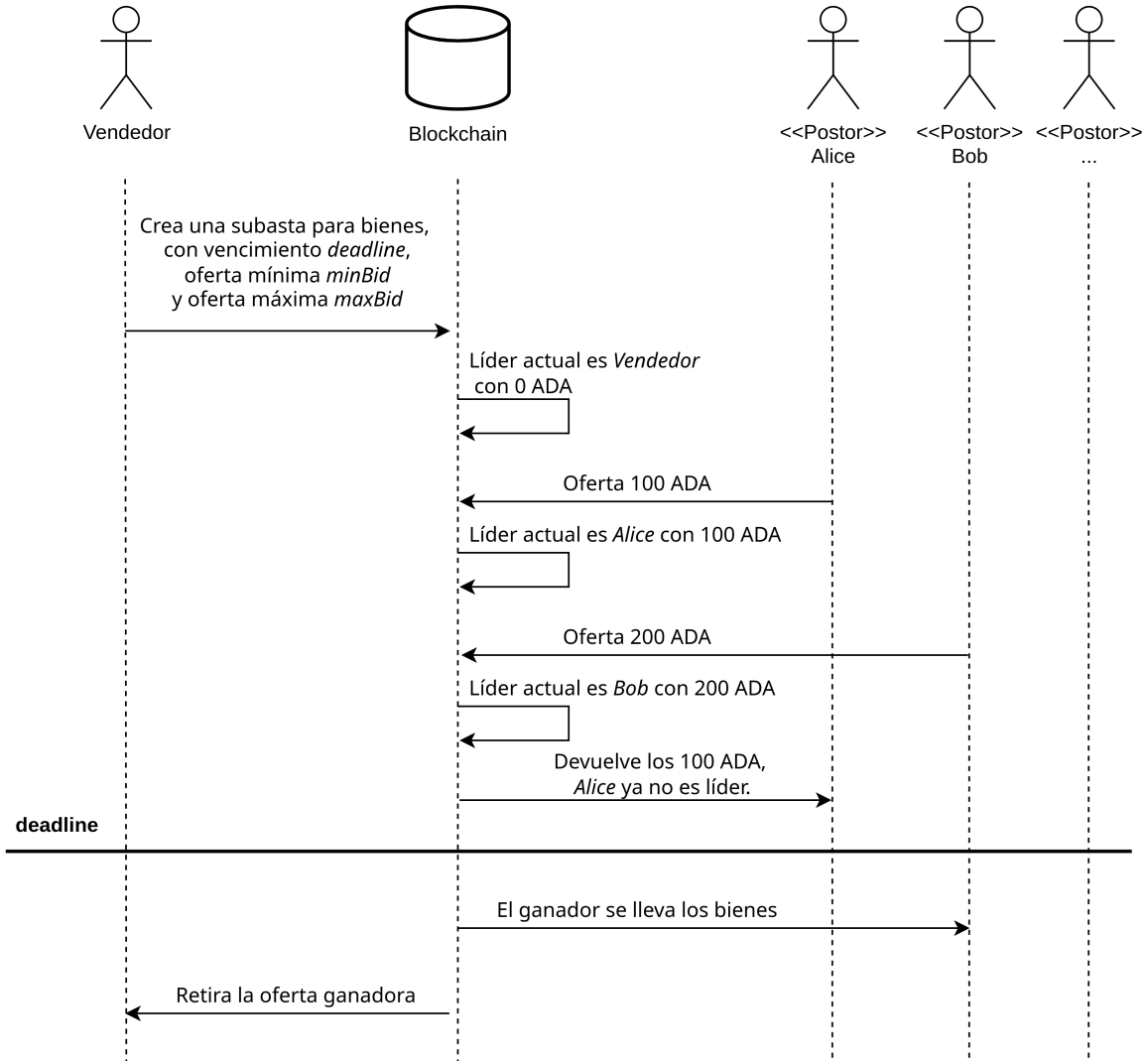
Una subasta o remate (*Auction*) es una venta organizada basada en la competencia directa, es decir, a aquel comprador (postor) que pague la mayor cantidad de dinero o de bienes a cambio del producto. El bien subastado se adjudica al postor que más dinero haya ofrecido por él.

El contrato Auction

Una subasta o remate (*Auction*) es una venta organizada basada en la competencia directa, es decir, a aquel comprador (postor) que pague la mayor cantidad de dinero o de bienes a cambio del producto. El bien subastado se adjudica al postor que más dinero haya ofrecido por él.

En una subasta basada en una *blockchain*, cada postor debe declarar el dinero primero para convertirse en el mejor postor. Los mismos recuperan su dinero cuando se los supera.

Se puede ver en el siguiente diagrama de secuencia, el comportamiento esperado por la subasta:



Probando terminación del contrato *Auction*

Dado que HOL es una lógica de funciones totales, la terminación es un requerimiento fundamental. Isabelle intenta probar la terminación automáticamente cuando se realiza la definición. Pero en algunas circunstancias es posible que falle, y la misma tiene que ser probada manualmente.

Probando terminación del contrato *Auction*

Dado que HOL es una lógica de funciones totales, la terminación es un requerimiento fundamental. Isabelle intenta probar la terminación automáticamente cuando se realiza la definición. Pero en algunas circunstancias es posible que falle, y la misma tiene que ser probada manualmente.

La terminación no solo es una propiedad deseable en los contratos, sino que también le permite a Isabelle utilizar teoremas adicionales, como por ejemplo el de inducción. Esto es de gran importancia a la hora de probar teoremas que puedan utilizar las funciones.

Métrica para la terminación

```
fun evalBoundAuction :: "(contractLoopType + (handleChooseType + handleDepositType
    )) ⇒ nat" where

"evalBoundAuction (Inl (_, ps, qs, _)) =
       2 * length ps + 4 * length qs + 1" |
"evalBoundAuction (Inr (Inl (_, ps, qs, _, p))) =
       2 * length ps + 4 * length qs + (if p ∈ set qs then 0 else 8)" |
"evalBoundAuction (Inr (Inr (_, ps, qs, _, p))) =
       2 * length ps + 4 * length qs + (if p ∈ set ps then 0 else 8)"
```

Invariante para la ausencia de warnings

Las acciones de los usuarios son imprevisibles y pueden generar *warnings* independientes a la semántica. Algunos ejemplos para este contrato podrían ser:

Invariante para la ausencia de warnings

Las acciones de los usuarios son imprevisibles y pueden generar *warnings* independientes a la semántica. Algunos ejemplos para este contrato podrían ser:

- Ofertar un valor fuera del rango esperado

Invariante para la ausencia de warnings

Las acciones de los usuarios son imprevisibles y pueden generar *warnings* independientes a la semántica. Algunos ejemplos para este contrato podrían ser:

- Ofertar un valor fuera del rango esperado
- Ofertar una cantidad negativa

Invariante para la ausencia de warnings

Las acciones de los usuarios son imprevisibles y pueden generar *warnings* independientes a la semántica. Algunos ejemplos para este contrato podrían ser:

- Ofertar un valor fuera del rango esperado
- Ofertar una cantidad negativa
- Dinero insuficiente en la cuenta para la oferta realizada.

Invariante para la ausencia de warnings

Las acciones de los usuarios son imprevisibles y pueden generar *warnings* independientes a la semántica. Algunos ejemplos para este contrato podrían ser:

- Ofertar un valor fuera del rango esperado
- Ofertar una cantidad negativa
- Dinero insuficiente en la cuenta para la oferta realizada.

Es por eso que se formularon una serie de pre-condiciones que deben ser satisfechas para poder asegurar que el contrato no genera *warnings*.

Esta invariante garantiza la presencia de las siguientes propiedades, que generarían la ausencia de *warnings* al aplicar un *input*:

Esta invariante garantiza la presencia de las siguientes propiedades, que generarían la ausencia de *warnings* al aplicar un *input*:

- Que el identificador del `Let` no exista en el momento en el que se crea (si no se producirá un warning de tipo *ReduceShadowing*)

Esta invariante garantiza la presencia de las siguientes propiedades, que generarían la ausencia de *warnings* al aplicar un *input*:

- Que el identificador del `Let` no exista en el momento en el que se crea (si no se producirá un warning de tipo *ReduceShadowing*)
- Que la cantidad en el depósito sea positiva (si no producirá un warning de tipo *NonPositiveDeposit*)

Esta invariante garantiza la presencia de las siguientes propiedades, que generarían la ausencia de *warnings* al aplicar un *input*:

- Que el identificador del `Let` no exista en el momento en el que se crea (si no se producirá un warning de tipo *ReduceShadowing*)
- Que la cantidad en el depósito sea positiva (si no producirá un warning de tipo *NonPositiveDeposit*)
- Que la cantidad en el *Pay* sea positiva y además las cuentas tengan suficiente dinero (si no se producirán warnings de *NonPositivePay* y *PartialPay* respectivamente).

Esta invariante garantiza la presencia de las siguientes propiedades, que generarían la ausencia de *warnings* al aplicar un *input*:

- Que el identificador del `Let` no exista en el momento en el que se crea (si no se producirá un warning de tipo *ReduceShadowing*)
- Que la cantidad en el depósito sea positiva (si no producirá un warning de tipo *NonPositiveDeposit*)
- Que la cantidad en el *Pay* sea positiva y además las cuentas tengan suficiente dinero (si no se producirán warnings de *NonPositivePay* y *PartialPay* respectivamente).
- Que *minBid* sea positivo, sino se producirán *warnings* durante las reducciones.

Definición en Isabelle de la invariante

```
definition invariantHoldsForAuction :: "AuctionTerms ⇒ AuctionWinner ⇒ Party
     list ⇒ Party list ⇒ State ⇒ bool" where

"invariantHoldsForAuction terms m ps qs curState =
       ((∀ x . x ∈ set qs ⟶
                ¬ member (partyToValueId x) (boundValues curState))
     ∧ (∀ x . x ∈ set ps ⟶
                findWithDefault 0 (partyToValueId x) (boundValues curState) > 0)
     ∧ (∀ x y . m = Some (x, y) ⟶
              ((lookup (y, token_ada) (accounts curState) =
                lookup (partyToValueId y) (boundValues curState))
             ∧ (findWithDefault 0 (partyToValueId y) (boundValues curState) > 0)
             ∧ (UseValue (partyToValueId y)) = x))
     ∧ (minBid terms > 0))"
```

Código 11: Invariante para el contrato *Auction*.

Ausencia de warnings en Isabelle

```
lemma auctionIsSafe_computeTransaction :
    "invariantHoldsForAuction terms m ps qs sta ⟹
     computeTransaction tra sta (contractLoop m ps qs terms) =
       TransactionOutput trec ⟹ txOutWarnings trec = []"

  using fixingIntervalPreservesInvariant auctionIsSafe_computeTransactionFixSta
  by (smt (verit, ccfv_SIG) IntervalResult.simps(6) closeIsSafe computeTransaction
       .simps fixInterval.elims)
```

Código 12: Lemma de ausencia de warnings en computeTransaction.

Índice de contenidos

Posibles trabajos futuros relacionados con esta tesis

- Con respecto a la escritura de contratos ACTUS en Cardano, actualmente no se han terminado de escribir todos los contratos especificados por el estándar.

Posibles trabajos futuros relacionados con esta tesis

- Con respecto a la escritura de contratos ACTUS en Cardano, actualmente no se han terminado de escribir todos los contratos especificados por el estándar.
- En cuanto a la verificación de programas, es posible tomar algunos contratos (por ejemplo entre los ejemplos del *Playground*) en los que no se esperan warnings y verificarlos mediante Isabelle. Dichas verificaciones podrán beneficiarse de los lemas probados previamente para el contrato *Auction* y el contrato *Close*.

Posibles trabajos futuros relacionados con esta tesis

- Con respecto a la escritura de contratos ACTUS en Cardano, actualmente no se han terminado de escribir todos los contratos especificados por el estándar.
- En cuanto a la verificación de programas, es posible tomar algunos contratos (por ejemplo entre los ejemplos del *Playground*) en los que no se esperan warnings y verificarlos mediante Isabelle. Dichas verificaciones podrán beneficiarse de los lemas probados previamente para el contrato *Auction* y el contrato *Close*.
- Una última linea de trabajo podría abarcar la traducción automática de algunas sentencias de código fuente Haskell a Isabelle. Parte de esta idea es abarcada en [**?**], aunque manteniendo la traducción manual.

Índice de contenidos

Bibliografía I


Brünjes, L. and Vinogradova, P. (2019).
Plutus: Writing reliable smart contracts.
IOHK.


IOHK (2016).
Marlowe cardano repository.
`https://github.com/input-output-hk/marlowe-cardano`.
Online; accessed 26 May 2022.


Torrini, P., Lueth, C., Maeder, C., and Mossakowski, T. (2007).
Translating haskell to isabelle.
CADE-21, page 14.