Bamboo Specification—An Early Draft

November 8, 2019

This specification contains some exercises. Please try to solve them at least mentally. Your solutions are welcome at `https://gitter.im/bbo-dev/Lobby`. TODO: use url

Chapter 1

Preliminaries

1.1 What Qualifies a Bamboo Implementation

. . . A Bamboo compiler can refuse to compile certain valid programs.

1.2 Notations

$v \in A$ says v is an element of a set A.

Chapter 2

Syntax

2.1 Keywords

The following sequences of characters are *keywords*.

- `true`
- `false`
- `this`
- `now`
- `not`
- `contract`
- `default`
- `case`
- `abort`
- `uint8`
- `uint256`
- `bytes32`
- `address`
- `bool`
- `if`
- `else`
- `then`

- `become`
- `return`
- `deploy`
- `with`
- `reentrance`
- `selfdestruct`
- `block`
- `void`
- `event`
- `log`
- `indexed`

TODO: Use a maththm like environment for exercises.
Exercise: which of the following are keywords?

1. `True`
2. `true`

2.2 Identifier

An *identifier* is a sequence of characters that matches the following regular expression (but is not a keyword):

```
['a'-'z' 'A'-'Z' '_'] ['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

TODO: define identifier
Exercise: which of the following are identifiers?
TODO: complete

2.3 Syntactic Types

The following sequences of characters are *syntactic types*.

- `void`
- `uint256`
- `bool`
- `uint8`

- `bytes32`
- `address`
- `mapping` T `=>` T' when T and T' are syntactic types
- any identifier except those listed above.

Every syntactic type except `void` is a *non-`void` syntactic type*.
Exercise: which of the following is true?

1. `uint256` is the set of integers at least zero and at most $2^{256} - 1$.
2. `uint256` is the set of 256-bit words.
3. none of the above.

2.4 Expressions

A *case-call-expression* looks like

$$e.c(e_1, \ldots, e_n)\ m$$

with e and e_i $(1 \leq i \leq n)$ being expressions and m a message-info.
A *call-expression* is either a case-call-expression or a default-call-expression.
A *deploy-expression* looks

$$\texttt{deploy}\ i(e_1, \ldots, e_n)\ m$$

with i being an identifier, e_i $(1 \leq i \leq n)$ an expression and m a message-info.
Expressions are inductively defined as follows.

- `true` is an expression.
- `false` is an expression.
- `msg.sender` is an expression.
- `msg.value` is an expression.
- `this` is an expression.
- `now` is an expression.
- An identifier is an expression.
- When e is an expression, (e) is an expression.
- When e is an expression, `not` e is an expression.
- When e is an expression, `balance`(e) is an expression.

- A deploy-expression is an expression.
- A call-expression is an expression.
- AddressExp? What is an AddressExp?
- When e_0 and e_1 are expressions, the following are expressions
  - e_0`&&`e_1
  - e_0 `<` e_1
  - e_0 `>` e_1
  - e_0 `!=` e_1
  - e_0 `==` e_1
  - $e_0[e_1]$
  - $e_0 + e_1$
  - $e_0 - e_1$
  - $e_0 \times e_1$

Exercise: prove that `99a` is not an expression.

2.5 Sentences

A *return sentence* is

$$\texttt{return}e_0\texttt{thenbecome}i(e_1, \ldots, e_n)$$

when i is an identifier and e_i $(0 \leq i \leq n)$ is an expression.

Sentences are inductively defined as follows.

- `abort;` is a sentence.
- a return sentence is a sentence.
- an assignment sentnece is a sentence.
- a variable initialization sentence is a sentence.
- an expression-only sentence is a sentence.
- an if sentence is a sentence.
- an if-then-else sentence is a sentence.
- a logging sentence is a sentence.
- a self-destruct sentence is a sentence.

2.6 Blocks

A *block* is a possibly empty sequence of stentences surrounded by { and }.
TODO: talk about whitespaces, perhaps.

2.7 Cases

A *case* is a case header followed by a block.

2.8 Contract Headers

2.9 Contracts

A *contract* is a contract header followed by a {, some (possibly no) cases and a }.

2.9.1 Contract's Signature

(`auction`, [`address`, `uint256`, `address`, `uint256`]).
Exercise: what is the signature of the following contract? TODO: complete the question

2.9.2 Contract Body

A *contract body* is a possibly empty sequence of cases surrounded by { and }.

2.9.3 Contract Definitions

A *contract definition* is a contract header followed by a contract body.

2.10 Event Declarations

Chapter 3

Semantics

3.1 Notations

TODO: describe $\in$

TODO: describe pi

3.2 States

3.2.1 Values

A *value* is a 256-bit word.

We pick something called $\perp$ (pronounced "bottom") which is not a value. The choice should not affect the meaning of a Bamboo program.

(We might specify a set of values for each syntactic type in the future.)

3.2.2 A Contract's States

A contract has *states*.

When a contract has a signature $(x, [T_1, T_2, \ldots, T_n])$ $(n \geq 0)$, the set of the states of the contract is $\Pi_{i=0}^{n} [\![T_i]\!]$.

3.2.3 A Program's Account States

A program determines a set of *account states*.

3.3 Dynamics

3.3.1 Variable Environment

A *variable environment* is a partial map that takes identifiers and may or may not return a value. When a variable environment σ maps and identifier i to a

value v, we write

$$\sigma(i) = v$$

3.3.2 Current Call

A *current call* $c = (c_\mathrm{s}, c_\mathrm{v}, c_\mathrm{t})$ is a tuple of

- a value c_s called the *sender*,
- a value c_v called the *transferred amount* and
- a value c_t called the *timestamp*.

3.3.3 World Oracle

Call Queries

Create Queries

Balance Queries

3.3.4 Timestamp query

The time stamp query `timestamp?` is something different from any query that appears above. Apart from that, `timestamp?` can be anything, and the behavior of Bamboo programs should not be affected by the concrete choice of `timestamp?` (with some adaptations on world oracles).

3.3.5 Current Account Query

The current account query `this?` is something different from any query that appears above.

3.3.6 Sender Query

The sender query `sender?` is something different from any query that appears above.

3.3.7 Value Query

The value query `value?` is something different from any query that appears above.

World Oracle

Call queries, create queries, TODO: fill in are *oracle queries*.

A *world oracle* is defined coinductively as a function that takes an oracle query and returns a pair of a value and a world oracle.

When a world oracle w takes a query q and returns a pair of a value v and a world oracle w', we write $w(q) = (v, w')$.

TODO: Add a possibility that the world oracle calls into the program again.

3.3.8 Evaluation of an Expression

The evaluation for expressions takes

- an expression,
- a current call,
- a world oracle and
- a variable environment.

It returns

- a value or $\perp$ and
- a world oracle

TODO: say it's inductively defined over the definition of expressions.

Evaluation of Literals

Literals are those keywords whose evaluation is defined below.

- $E_\mathrm{e}\left(\boxed{\texttt{true}}, c, w, \sigma\right) := (1, w)$
- $E_\mathrm{e}\left(\boxed{\texttt{false}}, c, w, \sigma\right) := (0, w)$
- $E_\mathrm{e}\left(\boxed{\texttt{now}}, c, w, \sigma\right) := w(\texttt{timestamp?})$
- $E_\mathrm{e}\left(\boxed{\texttt{this}}, c, w, \sigma\right) := w(\texttt{this?})$
- $E_\mathrm{e}\left(\boxed{\texttt{msg.sender}}, c, w, \sigma\right) := w(\texttt{sender?})$
- $E_\mathrm{e}\left(\boxed{\texttt{msg.value}}, c, w, \sigma\right) := w(\texttt{value?})$

balance(x) sends a balance query on the world oracle.

Evaluation of an Identifier

An identifier i is evaluated as follows:

$$E_\mathrm{e}\left(\boxed{i}, c, w, \sigma\right) := (\sigma(i), w)$$

Note that $\sigma(i)$ can be $\perp$.

Evaluation of a new-expression

TODO: Consider new-expressions with nontrivial continuation later. That requires an interaction b

Evaluation of a call-expression

TODO: Consider call-expressions with nontrivial continuation later. That requires an interaction be

Evaluation of Binary Operators

$$E_{\mathrm{e}}\left(\boxed{e_0 \texttt{\&\&} e_1}, c, w, \sigma\right) := \begin{cases} (v_0, w_0) & \text{if } v_0 = 0 \text{ or } v_0 = \bot \\ (v_1, w_1) & \text{otherwise} \end{cases}$$

where

$$E_{\mathrm{e}}\left(\boxed{e_0}, c, w, \sigma\right) = (v_0, w_0)$$

and

$$E_{\mathrm{e}}\left(\boxed{e_1}, c, w_0, \sigma\right) = (v_1, w_1)$$

$$E_{\mathrm{e}}\left(\boxed{e_0 \texttt{<} e_1}, c, w, \sigma\right) := \begin{cases} (1, w_0) & \text{if } v_0 \neq \bot, v_1 \neq \bot \text{ and } v_0 < v_1 \\ (0, w_0) & \text{if } v_0 \neq \bot, v_1 \neq \bot \text{ and } v_0 \geq v_1 \\ (\bot, w) & \text{if } v_0 = \bot \text{ or } v_1 = \bot \end{cases}$$

where

$$E_{\mathrm{e}}\left(\boxed{e_1}, c, w, \sigma\right) = (v_1, w_1)$$

and

$$E_{\mathrm{e}}\left(\boxed{e_0}, c, w, \sigma\right) = (v_0, w_0)$$

$$E_{\mathrm{e}}\left(\boxed{e_0[e_1]}, c, w, \sigma, M\right) := (w_0, M(v_0, v_1))$$

where

$$E_{\mathrm{e}}\left(\boxed{e_1}, c, w, \sigma, M\right) = (v_1, w_1)$$

and

$$E_{\mathrm{e}}\left(\boxed{e_0}, c, w_1, \sigma, M\right) = (v_0, w_0)$$

3.3.9 Evaluation of a Sentence

The evaluation function for sentences take

- a sentence,
- a current call,
- a variable environment and
- a world oracle

and returns

- a variable environment,
- a world oracle
- and optionally an account state.

TODO: Show two forms of equations: one without an account state, the other with an account state.

Evaluation of an Expression Sentence

$$E_{\mathrm{s}}\left(\boxed{\mathsf{void} = e;}, c, \sigma, w\right) := (\sigma', w')$$

where

$$E_{\mathrm{e}}\left(\boxed{e}, c, \sigma, w\right) = (v, \sigma', w')$$

3.3.10 Evaluation of a Case

The evaluation function of a case takes

- A contract state
- a world oracle
- a case call

and returns

- an account state
- a world oracle

3.3.11 Evaluation of a Contract

The evaluation function of a contract takes

- A contract state
- a world oracle
- a contract call

and returns

- An account state
- a world oracle

3.3.12 Evaluation of a Program

The evaluation function of a program takes

- an account state
- a world oracle
- a program call

and returns

- an account state
- a world oracle

3.4 Account Initialization

3.4.1 Account Deployment Query

3.4.2 Initial Variable Environment

Chapter 4

Connection to EVM

4.1 Bamboo Account State as an EVM Account State

4.2 Queries as EVM instructions