Post's Theorem and Blockchain Languages

A Short Course in the Theory of Computation

Russell O'Connor

roconnor@blockstream.com

BPASE, January 26, 2017

MORPHEUS (LOGICIAN):

Tell me Neo, what is the most expressive type of language possible for a blockchain?

NEO (SWE): Turing complete languages. Every computer scientist knows that is the most expressive possible.

MORPHEUS: What if I told you that you could design a language that is just as expressive, yet guarantees termination.

NEO: What are you trying to tell me? That I can execute arbitrary computation?

MORPHEUS: No, Neo. I'm trying to tell you that when the time comes, you won't have to.

Blockchain Programs

- Transactions update a global blockchain state.
- Consensus rules decide which transactions are valid in a given state.
- Miners compete to sequence transactions.

Blockchain Programs

- Transactions update a global blockchain state.
- Consensus rules decide which transactions are valid in a given state.
- Miners compete to sequence transactions.

Blockchain programs let users build *smart contracts* by providing controls for which transactions are valid.

Blockchain Programs

A blockchain program defines a predicate that must be satisfied for a subsequent transaction to be accepted.

$$P : (\text{State}, \text{Transaction}) \to \text{Bool}$$

Blockchain Programs

Predicates can be as expressive as functions when the function's inputs are added to the transaction.

Given $F : (\text{State}, \text{Inputs}) \rightarrow \text{Transaction}$, define P as

$$P : (\text{State}, (\text{Inputs}, \text{Transaction})) \rightarrow \text{Bool}$$
$$P(s, (i, t)) := F(s, i) \stackrel{?}{=} t$$

Theory of Computation

Theory of Computation

We define *computable functions* as partial functions on $\mathbb{N}$ that can be defined by a Turing machine / lambda calculus term / general recursion.

Theory of Computation

We define *computable functions* as partial functions on $\mathbb{N}$ that can be defined by a Turing machine / lambda calculus term / general recursion.

We define *computably enumerable (C.E.) predicates* as predicates on $\mathbb{N}$ that are the domain of some computable function.

The C.E. predicates are the broadest class of predicates we can programmatically define.

This class of predicates has been completely characterized by Emil Post using logic and the *Arithmetic Hierarchy*.

C.E. Predicates and the Arithmetic Hierarchy.

First Order Logic

Atomic Formulas

- Variables: x, y, z, etc.
- Constants: 0, 1
- Arithmetic: $s + t$, $s \times t$
- Equality: $s = t$
- Inequality: $s < t$

First Order Logic

Arithmetic Formulas

- Connectives: $p \wedge q$, $p \vee q$, $p \Rightarrow q$, $\neg p$
- Bound Quantifiers: $\exists x < t.p$, $\forall x < t.p$
- Unbound Quantifiers: $\exists x \in \mathbb{N}.p$, $\forall x \in \mathbb{N}.p$

Arithmetic Hierarchy

A Δ_0 formula is an arithmetic formula with no unbound quantifiers.

$$\forall x < z . \exists y < z . x + y = z$$

Arithmetic Hierarchy

A Σ_1 formula is of the form $\exists x \in \mathbb{N}.p$ where p is a Δ_0 formula.

$$\exists x \in \mathbb{N}.\forall y < z.x + y = z$$

A Π_1 formula is of the form $\forall x \in \mathbb{N}.p$ where p is a Δ_0 formula.

$$\forall x \in \mathbb{N}.\exists y < z.x + y = z$$

Arithmetic Hierarchy

P is a Σ_1 predicate (Π_1 predicate) if it is definable by a Σ_1 formula (Π_1 formula).

Arithmetic Hierarchy

P is a Σ_1 predicate (Π_1 predicate) if it is definable by a Σ_1 formula (Π_1 formula).

Theorem (Post's Theorem)

P is a C.E. predicate if and only if P is a Σ_1 predicate.

Post's Theorem

Every blockchain program is equivalent to some Σ_1 predicate.

$$P : (\text{State}, \text{Transaction}) \rightarrow \text{Bool}$$
$$P(s, t) = \exists x \in \mathbb{N}.Q(x, s, t)$$

where $Q : (\mathbb{N}, \text{State}, \text{Transaction}) \rightarrow \text{Bool}$ is a Δ_0 predicate.

Post's Theorem

Every blockchain program is equivalent to some Σ_1 predicate.

$$P : (\text{State}, \text{Transaction}) \to \text{Bool}$$
$$P(s,t) = \exists x \in \mathbb{N}. Q(x,s,t)$$

where $Q : (\mathbb{N}, \text{State}, \text{Transaction}) \to \text{Bool}$ is a Δ_0 predicate.

All P can do is search for some data x, called a witness, that satisfies Q.

The Key Idea

Rather than searching for the witness, provide it as part of the transaction.

$$P_0 : (\mathsf{State}, (\mathbb{N}, \mathsf{Transaction})) \rightarrow \mathsf{Bool}$$
$$P_0(s, (x, t)) = Q(x, s, t)$$

The Key Idea

Rather than searching for the witness, provide it as part of the transaction.

$$P_0 : (\mathrm{State}, (\mathbb{N}, \mathrm{Transaction})) \to \mathrm{Bool}$$
$$P_0(s, (x, t)) = Q(x, s, t)$$

P_0 is a Δ_0 predicate. Evaluation of Δ_0 predicates always terminates!

Recap

- Turing-complete languages can only define C.E. predicates.
- By Post's Theorem, C.E. predicates are identical to Σ_1 predicates in the Arithmetic Hierarchy.
- Validating a Σ_1 predicate can be reduced to validating a Δ_0 predicate with a witness.
- Evaluating a Δ_0 predicate always terminates.

What is the witness?

Post's theorem does not tell us what data the witness is. There are a number of possible choices.

Here are two extreme examples.

Gas-based witness

1. The witness is the number of steps the Turing machine takes to run. Q is the same program as P but always halts after x steps.

Gas-based witness

1. The witness is the number of steps the Turing machine takes to run. Q is the same program as P but always halts after x steps.

Q is a decidable predicate, but we have not gained much. For example, the best bounds we can have on memory use is

- $x \times MaxAlloc$ if there is a maximum amount of memory that can be allocated per step.
- 2^x if an unbounded amount of data can be copied per step.

Trace-based witness

2. The witness encodes the sequence of all intermediate states occurring during execution. Q is a program that checks that each intermediate state in the sequence follows from the previous one.

Trace-based witness

2. The witness encodes the sequence of all intermediate states occurring during execution. Q is a program that checks that each intermediate state in the sequence follows from the previous one.

Evaluation of Q

- typically linear in the size of the witness
- typically runs in constant space
- can be run in parallel

However, the size of the witness can be very large and contain a lot of redundant data.

Blockchain Language Design

The best solution lies somewhere between these two extremes.

A Turing-complete blockchain language does not look like a Σ_1 formula, even though they are equivalent.

Your next blockchain language will not look like Δ_0 formulas either.

Blockchain Language Design

One possible design is a simple, bounded language that lets users decide how to mix performing computation with validating trace data.

Features include

- Simpler language for blockchain programming
  - Bounded loops only
  - Consensus critical
- Prune unneeded computation
  - Untaken branches
  - Unneccessary searches
- Low computational complexity of evaluation
  - Bounded computational resources
  - Reduced risk of denial of service attacks

Hashtag Post's Theorem

Tweet to all your followers

Blockchain Languages:
Everyone has been talking about Σ_1 when they should be talking about Δ_0. #PostsTheorem #ArithmeticHierarchy