

DRAFT: Clarity contract basics

Part 2

Phillip G. Bradford*

November 28, 2024

Abstract

Clarity is a lisp-like but non-recursive smart-contract language. Furthermore, Clarity restricts iteration.

These notes explore the expressive power of Clarity smart contracts given these restrictions.

1 Basic Clarity

Smart contracts and tokenization are two central contributions of blockchains [3]. Validating and executing smart contracts requires consensus of participants in a blockchain network.

Definition 1 (Smart contract) A smart contract is an automated digital contract that is programatically defined and executed by consensus when certain conditions are met.

Clarity is not recursive and only allows restricted iteration. This restriction was made so that all Clarity contracts are decidable.

A comparison of traditional and smart contracts is given in Table 1.

Kurt Gödel famously showed there are undecidable logical statements. He used logical statements from first-order logic that encode the integers.

Definition 2 (Decidable and undecidable) A first order logical expression E , capturing the integers, is *decidable* iff the validity of E can be mechanically proven.

A first order logical expression E is *undecidable* iff the validity of E cannot be mechanically proven.

*phillip.bradford@uconn.edu, phillip.g.bradford@gmail.com, UNIVERSITY OF CONNECTICUT, SCHOOL OF COMPUTING, STAMFORD, CT USA

Traditional Contract	Smart contract
Fair and transparent	Globally transparent, no trusted 3rd party
Measurable inputs and outputs	Defined with formal language precision
Legally enforceable in one country	Cryptographically enforceable globally
Irrevocable under some conditions	Cryptographically immutable
Accomplishes a goal	Accomplishes and executes a goal

Table 1: Traditional contracts v. Smart contracts

The *Church-Turing thesis* is the hypothesis that all formal notions of computation are captured by Turing machines or, essentially, modern computers. A programming language is *Turing complete* iff it captures all formal notions of computation. Many programming languages are Turing complete. For example: Python, Java, JavaScript, C, C++, Rust, Lisp, are all Turing complete.

Turing complete languages are very expressive. In fact, assuming the Church-Turing thesis, Turing complete languages are as expressive as possible in some sense.

Definition 3 (The Church-Turing thesis) A Turing machine embodies all methods of computation.

Alonzo Church and Alan Turing showed there are problems that are uncomputable. Uncomputable problems cannot be solved by any Turing machine. That is, assuming the Church-Turing thesis, these uncomputable problems cannot be solved by any computer.

This indicates all programs in Turing complete languages cannot be validated in the worst case. Thus, Turing complete smart contract languages must allow contracts that cannot be validated.

Intuitively, uncomputability is an algorithmic view of undecidability. Uncomputability has the same foundations as undecidability. Undecidable questions are often framed as logic statements or statements about integers. Of course, programs are logic statements and may even be viewed as integers. Though, we generally view programs differently. We often view programs with additional features of memory models, implementation details, and execution semantics.

Clarity is not recursive and greatly restricts iteration. Thus, Clarity cannot express uncomputable problems. Moreover, Clarity contracts are decidable.

2 The power of Clarity contracts

Create a new contract `c2`. In `c2.clar` put,

```
(define-public (cube (x uint)) (ok (* x x x)))
```

Running this contract

```
>> (contract-call? .c2 cube u3)
(ok u27)
```

Clarity strongly and statically typed. In particular, all types must be defined in advance.

A language is strongly typed when all operations use compatible types. A language is statically typed when all type definitions are known at the start of a program compilation.

Next is an example of an *error* for passing an `int` where a `uint` is expected:

```
>> (contract-call? .c2 cube 3)
<stdin>:1:26: error : expecting expression of type 'uint',
      found 'int'
(contract-call? .c2 cube 3)
      ^
```

Also, Clarity does not allow the next two functions to be defined in the same contract,

```
(define-public (cube (x uint)) (ok (* x x x)))
(define-public (cube (x int)) (ok (* x x x)))
```

Clarity is *not recursive*. Clarity functions can call themselves. For example, `(square (square u2))`.

```
(define-private (square (x uint)) (* x x))
(define-public (fourth (x uint))
  (ok (square (square x))))
```

This is not general recursive since the number of recursive calls is fixed.

```
>> (fourth u2)
u16
```

Of course,

$$\begin{aligned}(u2)^4 &= u4 \cdot u4 \\ &= u16.\end{aligned}$$

Two return types are: `some` and `none`. Sometimes functions do not return anything, hence `none`.

For example,

```
>> (unwrap-panic (some u10))
u10
```

Also,

```
>> (unwrap-panic none)
<stdin>:1:1: error: attempted to obtain 'ok' value from
      response, but 'ok' type is indeterminate
(unwrap-panic none)
~~~~~
```

3 Satisfiability

Consider a list of clauses in conjunctive normal form

$$B = (x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \wedge \dots \quad (1)$$

Given truth assignments for each x_i , then evaluating B is relatively easy. For example, an expression B with 10^6 or 1 million variables and a total of 10^9 or 1 billion symbols, can be evaluated by a basic modern computer in a few minutes.

However, in the worst-case, searching for truth/false assignments that make such an expression B true appears to require 2^{10^6} Boolean assignments. For each assignment is followed by an evaluation.

Definition 4 (Satisfiability problem) Consider a Boolean expression as given in Equation 1, then finding truth/false assignments that make B true is the *satisfiability problem*.

The 3SAT problem is all satisfiability instances where each clause has at most three literals. A literal is a variable or the NOT of a variable.

It is easy to write a Clarity program to solve an instance of the 3SAT problem. However, in the worst-case, it appears to take an exponentially sized contract to solve such an instance of the 3SAT problem.

In VS Code in <https://platform.hiro.so>, open a terminal.

In VS Code or a native command-line. In the terminal go to the `proj1/contracts` directory.

```
windows> cd proj1\contracts\
```

Create a new contract `c1.clar`. Edit the `c1.clar` file and in the *public function* section, add the code in Listing 1.

Listing 1: Basic functions on Clarity

```
(define-data-var x1 int 0)
(define-data-var x2 int 0)
(define-data-var x3 int 0)
(define-data-var x4 int 0)
(define-data-var x5 int 0)

(define-data-var Nx1 int 1)
(define-data-var Nx2 int 1)
(define-data-var Nx3 int 1)
(define-data-var Nx4 int 1)
(define-data-var Nx5 int 1)
;;
(define-public (eval-me)
  (if (is-eq (three-sat1 (flip (var-get x1)) (flip
    (var-get x2)) (var-get Nx3)) true)
    (ok "yes")
    (err "No")))
;;
(define-private (three-sat1 (x int) (y int) (z int))
  (and (is-eq x 1)
    (is-eq y 1)
    (is-eq z 1)))
;;
(define-private (flip (x int))
  (if (is-eq x 1)
    0
```

1))

4 Notes

David Hilbert and Wilhelm Ackermann gave the Entscheidungsproblem. The Entscheidungsproblem is problem of determining if there are unsolvable problems. The Entscheidungsproblem was itself solved by Alonzo Church and Alan Turing. They showed there are problems that are unsolvable.

5 Exercises

1. A combinatorial logic expression can be expressed as a *DAG (Directed Acyclic Graph)*. So, a combinatorial logic expression has no cycles or loops.

Prove that Clarity can only express combinatorial logic expressions.

2. This SAT expressions here are not evaluated with loops or recursion. So, these

- 3.

References

- [1] Marvin Janssen with contributions of Inow, Mike Cohen and Albert Catama: *Clarity of Mind*, <https://book.clarity-lang.org/> 2023-04-11.
- [2] John H. Conway: *Regular algebra and finite machines*, Dover, 2012.
- [3] Phillip G. Bradford: *Chains that bind us*, Self-published December 2023. Available on Amazon.
- [4] <https://clarity-lang.org/> 2023-12-15.