Building SmartWeave Contracts with Clarity

Arto Bendiken

An Overview

SmartWeave: Arweave's Smart Contract Protocol

- Enables client-side, computation-heavy dapps on top of the Arweave network
- In a traditional smart contract system (such as Ethereum), every validator node must execute and validate every transaction
  - Results in a severe computation and transaction processing bottleneck
- SmartWeave instead pushes contract execution to users of the smart contract
  - Frees network validators from contract state management and validation
  - Altogether eliminates the need for 'gas' to pay for contract execution
- Read more at Introducing SmartWeave: Building Smart Contracts with Arweave and With Arweave's 'Lazy' Approach to Smart Contracts, Its Version of Web3 Does More
- Find the GitHub repository at https://github.com/ArweaveTeam/SmartWeave

Clarity: A Safe, Decidable Smart Contract Language

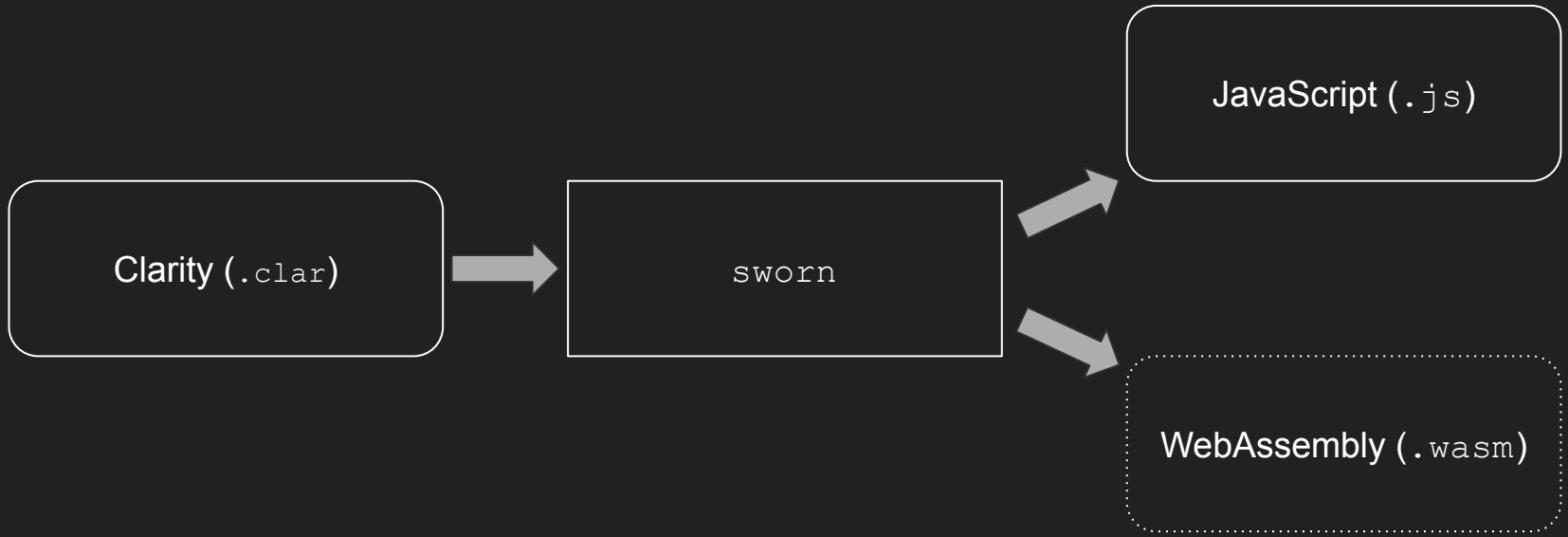
- A new smart contract language that is safe (doesn't contain footguns) and decidable (won't let you get into an infinite loop)
- Syntactically and semantically inspired by Lisp (familiar to fans of Clojure!)
  - For machines: it is trivial to generate, parse, and analyze Clarity code, from anywhere
  - For humans: don't fear the parentheses, they do fade away after a little use
- Originally developed by Hiro (formerly known as Blockstack) and Algorand
- Read more at Introducing Clarity, a Language for Predictable Smart Contracts and Bringing 'Clarity' to 8 Dangerous Smart Contract Vulnerabilities
- Find the GitHub repositories at https://github.com/clarity-lang

Sworn: A Compiler for Clarity on SmartWeave

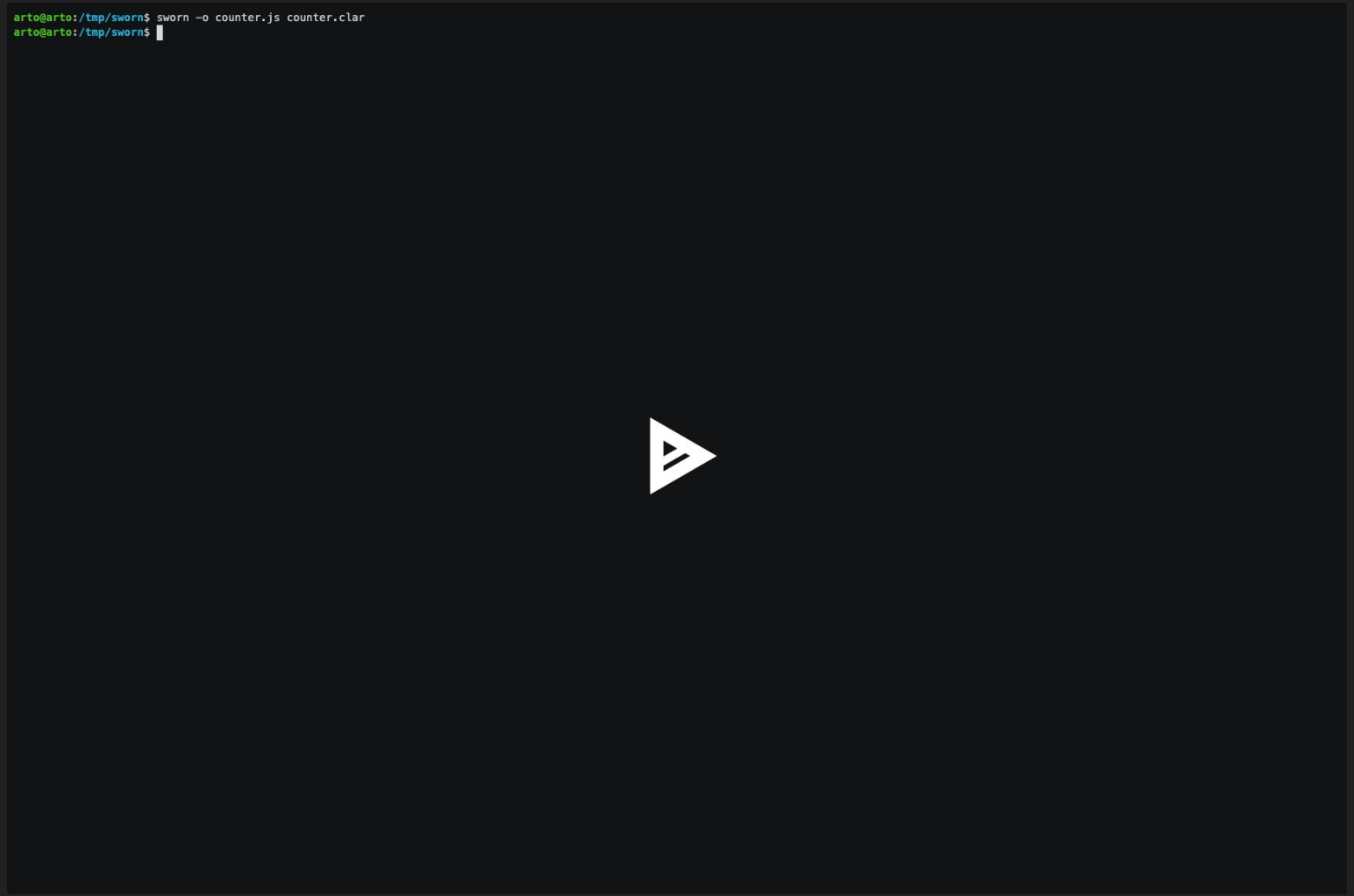
- Compiles Clarity smart contracts into SmartWeave contracts that run on the Arweave blockchain
  - The `sworn` program parses and compiles `.clar` files
  - The output is an equivalent SmartWeave program in the form of JavaScript code
  - Also includes experimental WebAssembly output, but JavaScript is recommended since the generated JS contracts are perfectly human readable and thus feasible to audit
- The generated SmartWeave code requires Clarity.js, which implements the necessary runtime support for Clarity's standard library
- Read more at Weaving Clarity: Safe Smart Contracts for SmartWeave
- Find the compiler's GitHub repository at https://github.com/weavery/sworn and the Clarity.js runtime's at https://github.com/weavery/clarity.js

SmartWeave Compilers

Sworn Outputs



The Demo


arto@arto:/tmp/sworn$ sworn -o counter.js counter.clar
arto@arto:/tmp/sworn$

The Rationale

“The history of smart contracts is really the history of smart contract bugs.”

— Aaron Blankstein

Anti-Footgun Matrix

	Solidity	JavaScript	TypeScript	Clarity
Decidability	✗	✗	✗	✓
Strong typing	✓	✗	✓	✓
Safe arithmetic	○	✗	✗	✓
Null safety	○	✗	✓	✓
Error checking	✗	✗	✗	✓

The Language

Clarity: A **Decidable** Language

- Intentionally Turing *incomplete*, avoiding Turing complexity
  - It is **not** possible to write an infinite loop in a Clarity program
  - Each and every Clarity program **will** halt, guaranteed
- You can know, with certainty, from the code itself what the program will do
  - It is possible to analyze Clarity code for runtime cost and data usage
- Enables the complete static analysis of the entire call graph
  - For auditability, the set of reachable code can be efficiently determined
- The type checker can eliminate whole classes of bugs
  - Unintended casts, reentrancy bugs, reads of uninitialized values, etc.

Clarity: A **Safe** Language

- Strong static typing to the rescue
  - The type system does **not** have a universal supertype
  - The language does **not** support sequences that have dynamic length
  - The length of a sequence (string, buffer, or list) is a part of its static type
- Safe arithmetic only
  - No silent overflow, underflow, or truncation permitted
- Null safety isn't optional
  - No null type nor value! Replaced by an `optional` type, as in other modern languages
  - *"I call it my billion-dollar mistake."* — Tony Hoare
- Error checking is serious business
  - No unchecked return values nor silently swallowed errors
- Omits a long list of Solidity footguns (anti-features)
  - No reentrancy, no untyped inputs, no default functions, etc.

```
(define-data-var counter int 0)

(define-read-only (get-counter)
  (ok (var-get counter)))

(define-public (increment)
  (begin
    (var-set counter (+ (var-get counter) 1))
    (ok (var-get counter))))

(define-public (decrement)
  (begin
    (var-set counter (- (var-get counter) 1))
    (ok (var-get counter))))
```

The Compiler

```
(define-data-var counter int 0)

(define-read-only (get-counter)
  (ok (var-get counter)))

(define-public (increment)
  (begin
    (var-set counter (+ (var-get counter) 1))
    (ok (var-get counter))))

(define-public (decrement)
  (begin
    (var-set counter (- (var-get counter) 1))
    (ok (var-get counter))))
```

```
clarity.requireVersion("0.1")

function getCounter(state) {
  return clarity.ok(state.counter);
}

function increment(state) {
  state.counter = clarity.add(state.counter, 1);
  return {state, result: clarity.ok(state.counter)};
}

function decrement(state) {
  state.counter = clarity.sub(state.counter, 1);
  return {state, result: clarity.ok(state.counter)};
}

export function handle(state, action) {
  const input = action.input;
  if (input.function === 'getCounter') {
    return {result: getCounter(state)};
  }
  if (input.function === 'increment') {
    return increment(state);
  }
  if (input.function === 'decrement') {
    return decrement(state);
  }
  return {state};
}
```

FAQ: Why does basic math require function calls?

- To support Clarity's language semantics of 128-bit integers and safe arithmetic that traps on numeric overflow and underflow, arithmetic operations need runtime support
- Thus, in the general case, an operation such as `(* a b)` must be compiled to `clarity.mul(a, b)` instead of the trivial but ultimately incorrect `a * b`.
- If the compiler can prove overflow or underflow will not occur in a particular context, it can elide the function call and output ordinary JavaScript arithmetic

Clarity	TypeScript	JavaScript
`bool`	`boolean`	`boolean`
`(buff N)`	`Uint8Array`	`Uint8Array`
`err`	`Err<T>`	`Err`
`Int, uint`	`number or bigint`	`number or BigInt`
`(list N T)`	`Array<T>`	`Array`
`(optional T)`	`T or null`	`T or null`
`principal`	`String`	`String`
`(response T E)`	`T or Err<E>`	`T or Err`
`(string-ascii N)`	`String`	`String`
`(string-utf8 N)`	`String`	`String`
`(tuple …)`	`Map<String, any>`	`Map`

The Future

Sworn 1.1, 2.0 — And Beyond

- The Sworn 1.1 release is coming soon, and focuses on user experience
  - More static analysis for Clarity input, rejecting more invalid programs
  - Significantly improved error messages from the compiler
- Many exciting things on the wishlist for an eventual Sworn 2.0
  - Prototyping is already going on for ingesting (a subset of) Solidity contracts via the Solidity project's Yul intermediate language
- Much work remains on Clarity.js and SmartWeave integration
  - Arweave-specific Clarity functions for easily building profit-sharing tokens and communities!
  - Contributors most welcome: TypeScript developers needed
- Clarity contracts can soon already be used on three blockchains: Arweave, Ethereum, and Stacks
  - Some expressions of interest from other blockchains as well

An Industry Standard?

ethereum

Stacks

arweave.org

Thank you!

Find me at:

https://ar.to