Imperial College London

Making Smart Contracts Safer

Aurel Bílý, Catalin Craciun, Calin Farcas,
Constantin Mueller, Yicheng Luo, Niklas Vangerow

January 10, 2019

Introduction

Problems with Smart Contracts

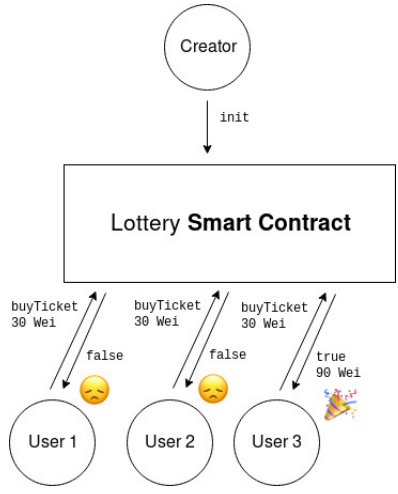
Example Contract - Lottery

```
contract Lottery {
  let ticketPrice: Int
  let oddsThreshold: Int
  var jackpot: Wei = Wei(unsafeRawValue: 0)
}

Lottery :: sender <- (any) {
  public init(ticketPrice: Int, oddsThreshold: Int) {
    self.ticketPrice = ticketPrice
    self.oddsThreshold = oddsThreshold
  }

  @payable
  public mutating func buyTicket(implicit value: Wei) -> Bool {
    if value.getRawValue() != ticketPrice { fatalError() }
    jackpot.transfer(source: &value)
    var success: Bool = false
    if randomNumber() < oddsThreshold {
      send(address: sender, value: &jackpot)
      success = true
    }
    return success
  }

  func randomNumber() -> Int { // ... }
}
```



Problems with Ethereum Smart Contracts

- Smart contracts are **immutable**
- Smart contracts are difficult to program with existing technologies
- Smart contracts **really need to be safe** as they directly handle currency
- Solidity does not provide a good solution as it lacks safety checks and features

$\Rightarrow$ Flint aims to be safer

Issues with Flint

Issues with Flint

- No interaction with other smart contracts due to lack of external calls
- Programmers may ‘lose’ currency as it is modelled as simple integers
- Flint lacked a development ecosystem and unit testing

$\Rightarrow$ Our goal was to improve all these aspects of Flint

Achievements

What are Asset Traits?

```
struct trait Asset { ... }
```

- Flint traits are similar to Java's interfaces
- Captures the commonalities between different assets, like `Wei`

What are Asset Traits?

```
mutating func transfer(source: inout Asset, amount: Int) {
  // It's easy to forget to include this!
  if source.getRawValue() < amount { fatalError() }

  source.setRawValue(value: source.getRawValue() - amount)
  setRawValue(value: getRawValue() + amount)
}
```

Cross-Asset Transfers

Not allowed

```
struct MyStruct: Asset { ... }

// Later ...
let other = Wei(unsafeRawValue: 1)
let value = MyStruct(unsafeRawValue: 1)

other.transfer(source: &value, amount: 1)
// Does not compile
```

Allowed

```
struct MyStruct: Asset { ... }

// Later ...
let other = MyStruct(unsafeRawValue: 1)
let value = MyStruct(unsafeRawValue: 1)

other.transfer(source: &value, amount: 1)
// 'value' is now 0, 'other' is now 200
```

Polymorphic Self

```
// struct trait Asset (interface)
transfer(source: inout Self, amount: Int)

// struct Bitcoin: Asset (implementation)
transfer(source: inout Bitcoin, amount: Int)

// struct Dogecoin: Asset (implementation)
transfer(source: inout Dogecoin, amount: Int)
```

General Improvements

- Unification of event and function declaration syntax
- Function calls parameter passing/ordering
- Testing

Events and Functions

- Unify the declaration syntax
- Implement default parameters for functions

Before

```
event eventA {
    let addr: Address
    let y: Int
}

event eventWithDefault {
    let addr: Address
    let value: Int = 40
}
```

After

```
event eventA(
    addr: Address,
    y: Int
)

event eventWithDefault(
    addr: Address,
    value: Int = 40
)
```

Function Call Parameters

- Improved logic for parameter checking (includes default parameters)
- Enforce labels and Swift-like ordering

Why Test?

- Not all features implemented
- Tests only cover parts of the codebase, and test end-to-end
- No way to tell whether features have regressed
- Less manual testing, more confidence

The Road to Mocking

- Mocking and stubbing like JMock
- Swift has read-only reflection
- Cuckoo was outdated and did not work on Linux

Testing Framework

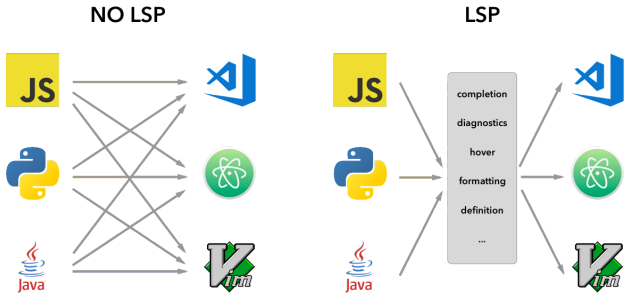
```
// Do not emit a diagnostic when there are
// no undefined functions in a contract
func testTopLevelModule_contractHasNoUndefinedFunctions_noDiagnosticEmitted() {
  // Given
  let f = Fixture()
  let contract = buildDummyContractDeclaration()
  let passContext = buildPassContext { (environment) in
    environment.undefinedFunctions(
        in: equal(to: contract.identifier)
    ).thenReturn([])
  }
  var diagnostics: [Diagnostic] = []

  // When
  _ = f.pass.checkAllContractTraitFunctionsDefined(
      environment: passContext.environment!,
      contractDeclaration: contract,
      diagnostics: &diagnostics
    )

  // Then
  XCTAssertEqual(diagnostics.count, 0)
}
```

- XCTest integrated into Xcode and SPM
- Code requires extensive refactoring to make testing possible
- TDD approach used

Language Server Protocol



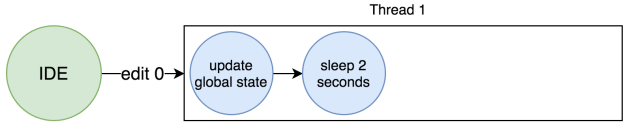
LSP architecture

https://code.visualstudio.com/api/language-extensions/language-server-extension-guide

LSP - Compilation and Auto-saving

- Wire LSP to use the flintc compiler for bringing compilation errors and warnings into the editor
- Initially compiling when the user saves the file
- Implement auto-compilation when the user stops editing the file for 2 seconds
  - LSP not supporting it by default
  - should not interfere with auto-saving
  - 'temporary file' implementation and thread sleeping

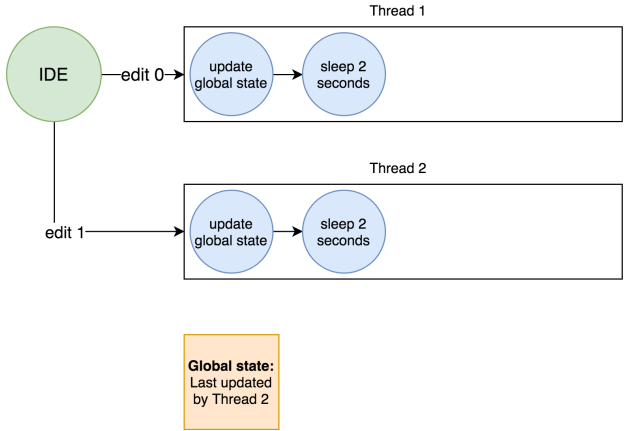
LSP - Auto-Compilation



Global state:
Last updated by Thread 1

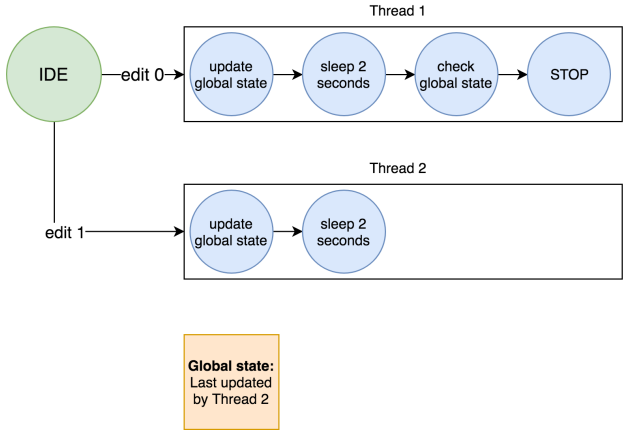
Auto-Compilation Step 1

LSP - Auto-Compilation



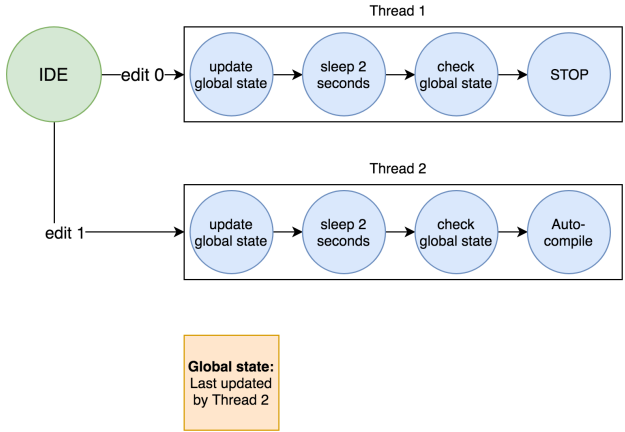
Auto-Compilation Step 2

LSP - Auto-Compilation



Auto-Compilation Step 3

LSP - Auto-Compilation



Auto-Compilation Step 4

External Calls

- Smart contracts can represent microservice-like architectures in Ethereum
- 'Calling' another contract $\Rightarrow$ data, Gas (computation time), and Wei

Problems with External Calls

- Can a contract be trusted?
- Is the contract's ABI (interface) specified correctly?
- Will the call fail completely?
- What code will the contract run?

$\Rightarrow$ Introducing microservices into a network based on antitrust is dangerous; many Ethereum exploits based around external calls

Re-entrancy Attack

```
mutating func withdrawMoney(amount: Int) {
  if balance[caller] < amount {
    fatalError()
  }
  call(value: Wei(amount))! caller.sendMoney()
  balance[caller] -= amount
}
```

Re-entrancy Attack

```
mutating func withdrawMoney(amount: Int) {
  if balance[caller] < amount {
    fatalError()
  }
  call(value: Wei(amount))! caller.sendMoney()
  balance[caller] -= amount
}

// in an attack contract:
@payable func sendMoney(implicit value: Wei) {
  call! bank.withdrawMoney(amount: 100)
}
```

External Calls in Flint

```
external trait Bank {
  @payable func sendMoney(to: string) -> int256
}
// Later ...
do {
  let bank: Bank = Bank(address: 0x...)
  let balance: Int
    = (call(value: Wei(100)) bank.sendMoney(
          to: "Joe" as! string
        )) as! Int
} catch is ExternalCallError {
  // handle gracefully
}
```

External Calls in Flint

- Type-safe (+ runtime value checks)
- Errors revert (`call!`) or have to be handled (`call` in `do ... catch`)
- 'Hyper-parameters' (`value`, `gas`) are separate from function arguments

Code Generation

Wrong abstraction is the root of all evil.

Nested do-catch

```
do {
    call f
    call g
    do {
        call h
    } catch is ExternalCallError {
        // block A
    }
} catch is ExternalCallError {
      // block B
}
```

```
success_f = call f
if (success_f) {
  success_g = call g
  if (success_g) {
    success_h = call h
    if (success_h) {
    } else {
      // block A
    }
  } else {
    // block B
  }
} else {
  // block B
}
```

Refactor Code Generator

- String concatenation $\Rightarrow$ Representation for YUL IR
- New Emitter API for Codegen adequate for complex code generation

Demo

Extensions

Linear Types

- Treating assets as integer values is a source of numerous bugs and exploits not only on the Ethereum network
- Linear types would be integrated into the language to never allow an asset to be destroyed, duplicated, or created from scratch

Linear Types Example

```
@payable
public func deposit(implicit value: Wei) {
  // compilation error: value was not used
}

@payable
public func deposit(implicit value: Wei) {
  self.totalValue.transfer(source: &value)
  // compilation OK
}
```

Modularisation

Allows more complex contracts to be split into multiple files

```
import VersionChecker

contract A {}
A :: (any) {
  public init() {
    VersionChecker.checkVersion(1, 3, 1)
  }
}
```

Package Manager

- A smart contract tracking deployed Flint contracts
- Would allow modules to be imported from addresses in a type-safe manner

Package Manager Example

```
import Wallet

// singleton:
import 0x032161A94B0700B13E321C032FC12586A4B07013 as VersionChecker

contract A {}
A :: (any) {
  public init() {
    VersionChecker.checkVersion(1, 3, 1)
    // instance:
    let myWallet: Wallet = 0x... as Wallet
  }
}
```

Conclusion

Conclusion

- We added features that make Flint more **usable**
- We added features that make Flint more **useful**
- We learned a lot about the **Ethereum ecosystem**, **compiler development** and **software engineering**
- There is huge potential for **further improvements**, in particular as the Ethereum ecosystem evolves

Thank you for your attention