Chain

Ivy: A Declarative Predicate Language for Smart Contracts

Dan Robinson
Product Architect, Chain

JANUARY 2017

Chain

Introduction: Two Blockchain Models

	Mutable State (Accounts and Contracts)	Immutable State (UTXOs)
Examples	Ethereum	Bitcoin, Chain, Corda
State	Stored in accounts and “contract” objects	Stored in “unspent transaction outputs” (UTXOs)
Transactions	Trigger messages between accounts and contracts	Consume and create new UTXOS
Programs	Associated with contract objects Alter state and send messages	Associated with UTXOs Either succeed (authorizing spending of the UTXO) or fail

Chain

Smart contract development in Ethereum

```
PUSH1 0x60  PUSH1 0x40  MSTORE PUSH1 0x0  CALLDATALOAD PUSH29
0x100000000000000000000000000000000000000000000000000000000  SWAP1
DIV DUP1 PUSH4 0x2e6e504a  EQ PUSH2 0x5a  JUMPI DUP1 PUSH4 0x3ccfd60b
EQ PUSH2 0x69  JUMPI DUP1 PUSH4 0xeedcf50a  EQ PUSH2 0x78  JUMPI DUP1
PUSH4 0xfdf97cb2  EQ PUSH2 0xb1  JUMPI PUSH2 0x58  JUMP JUMPDEST STOP
JUMPDEST PUSH2 0x67  PUSH1 0x4  DUP1 POP POP PUSH2 0xea  JUMP
JUMPDEST STOP JUMPDEST PUSH2 0x76  PUSH1 0x4  DUP1 POP POP PUSH2
0x277  JUMP JUMPDEST STOP JUMPDEST PUSH2 0x85  PUSH1 0x4  DUP1 POP
POP PUSH2 0x424  JUMP JUMPDEST PUSH1 0x40  MLOAD DUP1 DUP3 PUSH20
0xffffffffffffffffffffffffffffffffffffffff  AND DUP2 MSTORE PUSH1 0x20  ADD SWAP2 POP
POP PUSH1 0x40  MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST
PUSH2 0xbe  PUSH1 0x4  DUP1 POP POP PUSH2 0x43c  JUMP JUMPDEST
PUSH1 0x40  MLOAD DUP1 DUP3 PUSH20 0xffffffffffffffffffffffffffffffffffffffff  AND DUP2
MSTORE PUSH1 0x20  ADD SWAP2 POP POP PUSH1 0x40  MLOAD DUP1
SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x0  PUSH1 0x0  SWAP1
SLOAD SWAP1 PUSH2 0x100  EXP SWAP1 DIV PUSH20 0xffffffffffffffffffffffffffffffffffffffff
AND PUSH20 0xffffffffffffffffffffffffffffffffffffffff  AND PUSH1 0x0  PUSH20
0xbb9bc244d798123fde783fcc1c72d3bb8c189413  PUSH20
0xffffffffffffffffffffffffffffffffffffffff  AND PUSH4 0x18160ddd  PUSH1 0x40  MLOAD DUP2
PUSH29 0x100000000000000000000000000000000000000000000000000000000
MUL DUP2 MSTORE PUSH1 0x4  ADD DUP1 SWAP1 POP PUSH1 0x20  PUSH1
0x40  MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0  DUP8 PUSH2 0x61da  GAS
SUB CALL ISZERO PUSH2 0x2  JUMPI POP POP POP PUSH1 0x40  MLOAD
DUP1 MLOAD SWAP1 PUSH1 0x20  ADD POP PUSH20
0xbb9bc244d798123fde783fcc1c72d3bb8c189413  PUSH20
0xffffffffffffffffffffffffffffffffffffffff  AND PUSH4 0x70a08231  ADDRESS PUSH1 0x40
```

EVM Assembly

```solidity
contract DAO {
    function balanceOf(address addr) returns(uint);

    function transferFrom(address from, address to, uint balance) returns(bool);
    uint public totalSupply;
}

contract WithdrawDAO {
    DAO constant public mainDAO = DAO(0xbb9bc244d798123fde783fcc1c72d3bb8c189413);
    address public trustee = 0xda4a4626d3e16e094de3225a751aab7128e96526;

    function withdraw() {
        uint balance = mainDAO.balanceOf(msg.sender);

        if (!mainDAO.transferFrom(msg.sender, this, balance) || !msg.sender.send(balance))
            throw;
    }

    function trusteeWithdraw() {
        trustee.send((this.balance + mainDAO.balanceOf(this)) - mainDAO.totalSupply());
    }
}
```

Solidity

Chain

Smart contract development in Bitcoin

```
HASH160 DUP <R-HASH> EQUAL
SWAP <Commit-Revocation-Hash> EQUAL ADD
IF
    <Alice's pubkey>
ELSE
    "2015/10/20 10:33" CHECKLOCKTIMEVERIFY
    "24h" CHECKSEQUENCEVERIFY
    2DROP
    <Bob's pubkey>
ENDIF
CHECKSIG
```

Bitcoin Script

???

Chain

Smart contract development in Bitcoin

```
HASH160 DUP <R-HASH> EQUAL
SWAP <Commit-Revocation-Hash> EQUAL ADD
IF
    <Alice's pubkey>
ELSE
    "2015/10/20 10:33" CHECKLOCKTIMEVERIFY
    "24h" CHECKSEQUENCEVERIFY
    2DROP
    <Bob's pubkey>
ENDIF
CHECKSIG
```

Bitcoin Script

```
1  program hashLock(rHash, revokeHash, deadline, delay,
2                   alicePubKey, bobPubKey) {
3    path alice(preimage, sig) {
4      let hash = hash160(preimage)
5      verify (hash == rHash) || (hash == revokeHash)
6      verify checkSig(alicePubKey, sig)
7    }
8    path bob(sig) {
9      verify checkLockTime(deadline)
10     verify checkSequence(delay)
11     verify checkSig(bobPubKey, sig)
12   }
13 }
```

Ivy

Ivy

- Predicate language (Δ_0) — programs either succeed or fail
- Programs guard **value** in a **UTXO**
- Designed for writing programs for the **Chain VM**
- A (limited) dialect compiles to **Bitcoin Script**

Ivy — Benefits

- Abstracts away stack manipulation (`OP_DUP`, `OP_SWAP`) in favor of named variables
- Avoids unfamiliar postfix notation (`1 1 OP_ADD 2 OP_EQUAL OP_VERIFY`)
- Easier to write, read, and teach

Ivy Examples

PayToPubKey

```
program p2pk(pubKey: PublicKey) {
  path spend(sig: Signature) {
    verify checkSig(pubKey, sig)
  }
}
```

Arguments (“ScriptSig”)

```
<sig>
```

Program (“ScriptPubKey”)

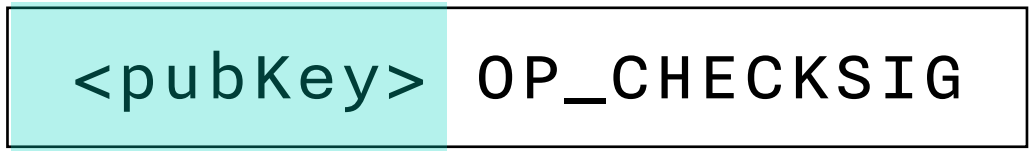
```
<pubKey> OP_CHECKSIG
```

PayToPubKey

```
program p2pk(pubKey: PublicKey) {
  path spend(sig: Signature) {
    verify checkSig(pubKey, sig)
  }
}
```

Arguments (“ScriptSig”)

Program (“ScriptPubKey”)



PayToPubKey

```
program p2pk(pubKey: PublicKey) {
  path spend(sig: Signature) {
    verify checkSig(pubKey, sig)
  }
}
```

Arguments (“ScriptSig”)

```
<sig>
```

Program (“ScriptPubKey”)

```
<pubKey> OP_CHECKSIG
```

PayToPubKey

```
program p2pk(pubKey: PublicKey) {
  path spend(sig: Signature) {
    verify checkSig(pubKey, sig)
  }
}
```

Arguments (“ScriptSig”)

Program (“ScriptPubKey”)

PayToPubKeyHash

```
program p2pkh(pkHash: Hash) {
  path spend(pubKey: PublicKey, sig: Signature) {
    verify hash160(pubKey) == pkHash
    verify checkSig(pubKey, sig)
  }
}
```

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pkHash> OP_EQUALVERIFY OP_CHECKSIG
```

PayToPubKeyHash

```
program p2pkh(pkHash: Hash) {
  path spend(pubKey: PublicKey, sig: Signature) {
    verify hash160(pubKey) == pkHash
    verify checkSig(pubKey, sig)
  }
}
```

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pkHash> OP_EQUALVERIFY OP_CHECKSIG
```

PayToPubKeyHash

```
program p2pkh(pkHash: Hash) {
  path spend(pubKey: PublicKey, sig: Signature) {
    verify hash160(pubKey) == pkHash
    verify checkSig(pubKey, sig)
  }
}
```

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pkHash> OP_EQUALVERIFY OP_CHECKSIG
```

PayToPubKeyHash

```
program p2pkh(pkHash: Hash) {
  path spend(pubKey: PublicKey, sig: Signature) {
    verify hash160(pubKey) == pkHash
    verify checkSig(pubKey, sig)
  }
}
```

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pkHash> OP_EQUALVERIFY OP_CHECKSIG
```

PayToPubKeyHash

```
program p2pkh(pkHash: Hash) {
  path spend(pubKey: PublicKey, sig: Signature) {
    verify hash160(pubKey) == pkHash
    verify checkSig(pubKey, sig)
  }
}
```

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pkHash> OP_EQUALVERIFY OP_CHECKSIG
```

Escrow with timeout

```
program Escrow(sender: PublicKey, recipient: PublicKey,
               agent: PublicKey, timeout: Time) {
  path approve(sig1: Signature, sig2: Signature) {
    verify checkMultiSig(3, sender, recipient, agent,
                         2, sig1, sig2)
  }
  path cancel(sig: Signature) {
    verify checkLockTime(timeout)
    verify checkSig(recipient, sig)
  }
}
```

Escrow with timeout

```
program Escrow(sender: PublicKey, recipient: PublicKey,
               agent: PublicKey, timeout: Time) {
  path approve(sig1: Signature, sig2: Signature) {
    verify checkMultiSig(3, sender, recipient, agent,
                         2, sig1, sig2)
  }
  path cancel(sig: Signature) {
    verify checkLockTime(timeout)
    verify checkSig(recipient, sig)
  }
}
```

Escrow with timeout

```
program Escrow(sender: PublicKey, recipient: PublicKey,
               agent: PublicKey, timeout: Time) {
  path approve(sig1: Signature, sig2: Signature) {
    verify checkMultiSig(3, sender, recipient, agent,
                         2, sig1, sig2)
  }
  path cancel(sig: Signature) {
    verify checkLockTime(timeout)
    verify checkSig(recipient, sig)
  }
}
```

Chain

Covenants

```
<0x0100000001>
OP_SWAP OP_SIZE 36 OP_NUMEQUALVERIFY OP_CAT
<0x00> OP_CAT
OP_SWAP OP_SIZE 32 OP_NUMEQUALVERIFY OP_CAT
<0x00005f> OP_CAT
2 OP_PICK OP_SIZE 95 OP_NUMEQUALVERIFY OP_CAT
<0xfffffff0100> OP_CAT
OP_SWAP OP_SIZE 32 OP_NUMEQUALVERIFY OP_CAT
<0x0000> OP_HASH256 OP_CAT
<0x17a914> OP_CAT
OP_SWAP OP_HASH160 OP_CAT
<0x8700000000001000000> OP_CAT
OP_SHA256
1 OP_DUP OP_CAT OP_DUP OP_CAT OP_DUP OP_CAT OP_DUP OP_CAT OP_DUP OP_CAT OP_DUP OP_CAT
OP_DUP
2 OP_ROLL 3 OP_PICK
OP_CHECKSIGFROMSTACKVERIFY
1 OP_CAT OP_SWAP
OP_CHECKSIG
```

Covenants

```
program QuineCovenant(fixedSignature: Signature) {
  path spend(outPoint: String, valueIn: String, valueOut: String,
             script: String, recoveredPubKey: PublicKey) {
    verify size(outPoint) == 36
    verify size(valueIn) == 32
    verify size(script) == 95
    verify size(valueOut) == 32
    let tx = (0x0100000001 ++ outPoint ++ 0x00 ++ valueIn ++ 0x00005f ++
              script ++ 0xffffffff0100 ++ valueOut ++ hash256(0x0000) ++
              0x17a914 ++ hash160(script) ++ 0x8700000000001000000)
    let txSigData = hash256(tx)
    verify checkSigFromStack(recoveredPubKey, txSigData, fixedSignature)
    verify checkSig(recoveredPubKey, fixedSignature ++ 0x01)
  }
}
```

Offer

```
program Offer(seller: PublicKey, price: Number, currency: Asset) {
  path lift() {
    verify tx.hasOutput(price, currency, Account(sellerKey))
  }
  path cancel(signature: Signature) {
    verify checkSig(sellerKey, signature)
  }
}
```

Offer

```
program Offer(seller: PublicKey, price: Number, currency: Asset) {
  path lift() {
    verify tx.hasOutput(price, currency, Account(sellerKey))
  }
  path cancel(signature: Signature) {
    verify checkSig(sellerKey, signature)
  }
}
```

Offer

```
program Offer(seller: PublicKey, price: Number, currency: Asset) {
  path lift() {
    verify tx.hasOutput(price, currency, Account(sellerKey))
  }
  path cancel(signature: Signature) {
    verify checkSig(sellerKey, signature)
  }
}
```

Offer

```
program Offer(seller: PublicKey, price: Number, currency: Asset) {
  path lift() {
    verify tx.hasOutput(price, currency, Account(sellerKey))
  }
  path cancel(signature: Signature) {
    verify checkSig(sellerKey, signature)
  }
}
```

Potential applications and compilation targets

- **Chain VM**
- **Bitcoin Script and extensions** (Lightning Network, lotteries, merkelized scripts, Elements Alpha, covenants…)
- **Crypto-Conditions**
- **zk-SNARKs**

Learn more

CHAIN CORE

Get the open-source Chain Core Developer Edition at **chain.com**.

CHAIN PROTOCOL

Protocol whitepaper, VM specification, and Ivy tutorial at **chain.com/docs**.

GET IN TOUCH

Slack: **slack.chain.com**

Twitter: **@chain**

Support forums: **support.chain.com**

Chain

IC3 Open House: February 23 in SF

WHAT & WHERE	Initiative for CryptoCurrencies and Contracts (IC3) Winter Retreat Hosted by Chain
WHO CAN APPLY	Employees of companies interested in joining IC3 Students interested in pursuing graduate studies at IC3
AGENDA & REGISTRATION	**initc3.org/events.html**

Chain

Dan Robinson
@danrobinson

Oleg Andreev
@oleganza

Tony Arcieri
@bascule