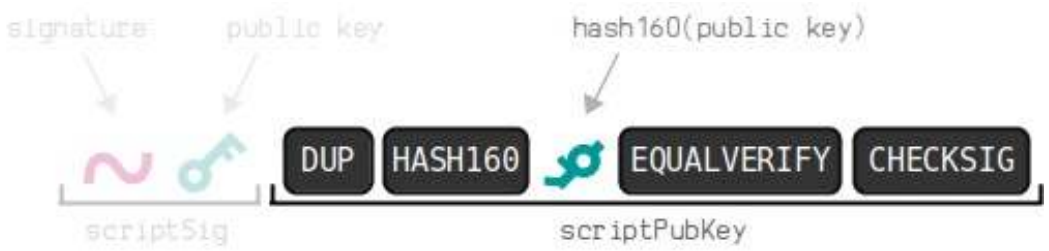
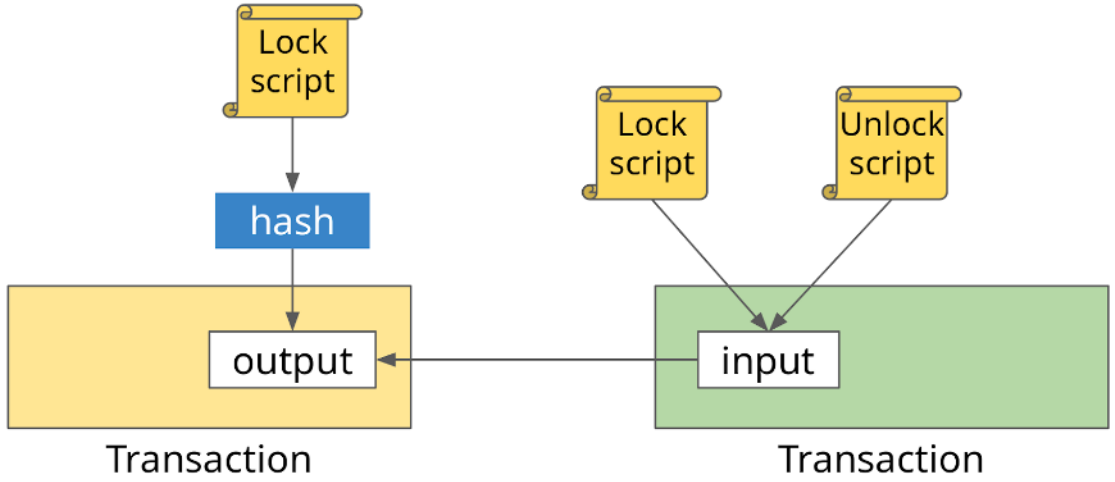
CashScript

Bitcoin Script






```rust
fn script(&self) -> Script {
    use crate::script::OpCodeType::*;
    Script::new(vec![
        Op::Code(Op2Dup),
        Op::Code(OpDup),
        Op::Code(OpHash160),
        Op::Push(self.address.bytes().to_vec()),
        Op::Code(OpEqualVerify),
        Op::Code(OpSwap),
        Op::Push(vec![0x41]),
        Op::Code(OpCat),
        Op::Code(OpSwap),
        Op::Code(OpCheckSigVerify),
        Op::Code(OpRot),
        Op::Code(OpSha256),
        Op::Code(OpSwap),
        Op::Code(OpCheckDataSig),
    ])
}
```

Contracts on BCH

- Fundamentally very different from ETH
- Adoption of contracts on BCH
  - SLP DEX
  - Mecenas & Last Will
  - Local.bitcoin.com & jeton-lib

CashScript

- Focus on accessibility
  - Without Bitcoin Script knowledge
- Focus on integration
  - Within web applications
- Focus on familiarity
  - For Solidity developers

```
contract TransferWithTimeout(
    pubkey sender,
    pubkey recipient,
    int timeout
) {
    function transfer(sig recipientSig) {
        require(checkSig(recipientSig, recipient));
    }

    function timeout(sig senderSig) {
        require(checkSig(senderSig, sender));
        require(tx.time >= timeout);
    }
}
```

JavaScript SDK

```
// Compile the P2PKH Cash Contract
const P2PKH = Contract.fromCashFile(path.join(__dirname, 'p2pkh.cash'), network);

// Instantiate a new P2PKH contract with constructor arguments: { pkh: alicePkh }
const instance = P2PKH.new(alicePkh);

// Get contract balance & output address + balance
const contractBalance = await instance.getBalance();
console.log('contract address:', instance.address);
console.log('contract balance:', contractBalance);

// Call the spend function with alice's signature + pk
// And use it to send 0. 000 100 00 BCH back to the contract's address
const tx = await instance.functions.spend(alicePk, new Sig(alice, 0x01))
  .send(instance.address, 10000);
console.log('transaction details:', tx);
```

Artifacts

```
interface Artifact {
  contractName: string // Contract name
  constructorInputs: AbiInput[] // Arguments required to instantiate a contract
  abi: AbiFunction[] // functions that can be called
  bytecode: string // Compiled Script without constructor parameters added (in ASM format)
  source: string // Source code of the CashScript contract
  networks: {
    // Dictionary per network (testnet / mainnet)
    [network: string]: {
      // Dictionary of contract addresses with the corresponding compiled script (in ASM format)
      [address: string]: string
    }
  }
  compiler: {
    name: string // Compiler used to compile this contract
    version: string // Compiler version used to compile this contract
  }
  updatedAt: string // Last datetime this artifact was updated (in ISO format)
}

interface AbiInput {
  name: string // Input name
  type: string // Input type (see language documentation)
}

interface AbiFunction {
  name: string // Function name
  inputs: AbiInput[] // Function inputs / parameters
}
```


```
{
  "contractName": "P2PKH",
  "constructorInputs": [
    {
      "name": "pkh",
      "type": "bytes20"
    }
  ],
  "abi": [
    {
      "name": "spend",
      "inputs": [
        {
          "name": "pk",
          "type": "pubkey"
        },
        {
          "name": "s",
          "type": "sig"
        }
      ]
    }
  ],
  "bytecode": "OP_1 OP_PICK OP_HASH160 OP_1 OP_PICK OP_EQUAL OP_VERIFY OP_2 OP_
  "source": "contract P2PKH(bytes20 pkh) {\n    // Require pk to match stored p
  "networks": {
    "testnet": {
      "bchtest:ppzllwzk775qk86zfskzyzae7pa9h4dvzcfezpsdkl": "ffd7616dccb66109ac
    }
  },
  "compiler": {
    "name": "cashc",
    "version": "v0.1.0-beta.1"
  },
  "updatedAt": "2019-06-11T08:06:29.010Z"
}
```

CashScript vs Bitcoin Script

- Less errors
- Faster development
- Easier integration
- Easier debugging through meeping

Limitations in CashScript

- Inefficient compiled code
  - 5x as large as Script
- Not all functionality is enabled
  - Limited multisig support
  - Lack of bitwise and some other opcodes
- Covenants are still hard

The future of CashScript

- Compiler optimisation
- Improved support for covenants
  - ‘tx.value’, ‘tx.outputs’ -> automatically passed by SDK
- User-defined structs
  - Better Oracle message support
- Full debugger support

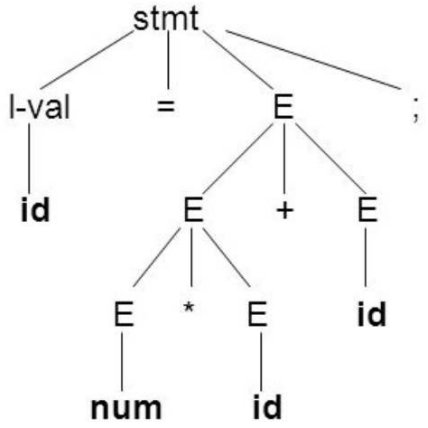
Limitations in Bitcoin Script

- No way to push deep into the stack / reverse OP_ROLL
- Redeem script size limit of 520 bytes

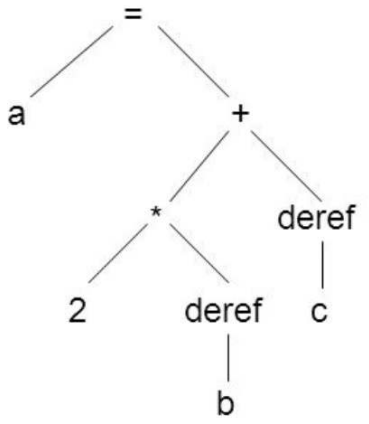
Compiler



- Implementation in TypeScript
- ANTLR4 + antlr4ts to generate lexer/parser
- Parse tree to AST conversion
- Visitor pattern for AST traversal



Parse Tree



Syntax Tree

Compiler Optimisations

- Merging OP_VERIFY
- Using OP_ROLL instead of OP_PICK
- Hardcoding stack operations
- Removing final OP_VERIFY
- Improving stack ordering
- Simplification