Script Grammar

1 Intdoduction

Script is a constrained stack-based language inspired by Forth.

Script programs (scripts) exist within Bitcoin transactions, which eventually are finalized or irrevocably committed to the global state of the Bitcoin network by their inclusion in a block which exists in the block-chain with the most proof of work expended.

Transactions consist of inputs and outputs, both of which include scripts. A transaction input script *unlocks* bitcoins; a transaction output script *locks* bitcoins. An input references the output of a previous transaction and provides a script which yields σ_0 when run. In order for the input to successfully unlock the bitcoins locked up in the referenced output, said output's script must yield $\sigma_1 = \sigma[V = valid]$ when run with it's state initialized to the state σ_0–yielded by the unlocking/input/spending script.

Script program syntax is defined in terms of byte-commands. Each command is one byte long. Not all bytes are defined as valid commands and many bytes which are defined have syntactic restrictions on the following program bytes which must hold in order for the program to be well-formed. Script defines 173 such commands, each corresponding to a particular byte, also called *words* or *opcodes*. For example, the command that adds 1 to the value on top of the stack, is written `0x8b`. These opcodes have human-readable aliases to make description of the language easier. For example, the opcode `0x8b` is referred to as `OP_1ADD` in the reference implementation's source code and documentation.

Script is intentionally constrainted: There language does not allow for the expression of loops or recursion. This is desirable, guaranteeing that the runtime of a Script program is linear with the size of the program.

Script is defined by the specification provided by the reference implementation. Script is as sequences of bytecodes. In describing its semantics, we will generally not refer to the format of the values, but some commands in the language only have meaning if we take into account the fact that any part of a Script program is a byte vector. For this reason, we define byte vectors as the following, and will reserve `b` and `B` to express a byte and a byte vector, respectively.

Section 2 presents a consistent and exhaustive BNF describing Script's syntax.

Section 3 and Section 4 describe the big and small step operational semantics of these syntactic constructs.

2 Syntax

2.1 Fundamental data types

`b ::= 0x00 | 0x01 | ... | 0xFF`
`B ::= b | b B`
$\mathtt{b}_i$ `::= b`
$\mathtt{B}^n$ `::=` $\mathtt{b}_1\ \mathtt{b}_2\ \ldots\ \mathtt{b}_n$
`varint ::=` B^1 `|` B^3 `|` B^5 `|` B^9

2.2 Transactions

`tx ::= tx_version tx_numin tx_inputs tx_numout tx_outputs tx_locktime`
`tx_version ::=` $\mathtt{B}^4$

tx_numin ::= varint
tx_inputs ::= txin | txin tx_inputs
tx_numout ::= varint
tx_outputs ::= txout | txout tx_outputs
tx_locktime ::= B^4

2.2.1 Transaction Inputs

txin ::= txin_prevhash txin_index txin_scriptlen txin_script txin_seqno
txin_prevhash ::= B^{256}
txin_index ::= B^4
txin_scriptlen ::= varint
txin_script ::= script
txin_seqno ::= B^4

2.2.2 Transaction Outputs

txout ::= txout_value txout_scriptlen txout_script
txout_value ::= B^8
txout_scriptlen ::= varint
txout_script ::= script

2.3 Script

script ::= com | com script
com ::= scom | mcom

2.3.1 Single-Word Commands

scom ::= scom_push | scom_control | scom_stack | scom_splice | scom_bitlogic | scom_numeric | scom_crypto | scom_expansion | scom_template

scom_push ::= OP_0 | OP_FALSE | OP_1NEGATE | ~~OP_RESERVED~~ | OP_1 | OP_TRUE | OP_2 | OP_3 | OP_4 | OP_5 | OP_6 | OP_7 | OP_8 | OP_9 | OP_10 | OP_11 | OP_12 | OP_13 | OP_14 | OP_15 | OP_16

scom_control ::= OP_NOP | ~~OP_VER~~ | ~~OP_VERIF~~ | ~~OP_VERNOTIF~~ | OP_VERIFY | OP_RETURN

scom_stack ::= OP_TOALTSTACK | OP_FROMALTSTACK | OP_2DROP | OP_2DUP | OP_3DUP | OP_2OVER | OP_2ROT | OP_2SWAP | OP_IFDUP | OP_DEPTH | OP_DROP | OP_DUP | OP_NIP | OP_OVER | OP_PICK | OP_ROLL | OP_ROT | OP_SWAP | OP_TUCK

scom_splice ::= ~~OP_CAT~~ | ~~OP_SUBSTR~~ | ~~OP_LEFT~~ | ~~OP_RIGHT~~ | OP_SIZE

scom_bitlogic ::= ~~OP_INVERT~~ | ~~OP_AND~~ | ~~OP_OR~~ | ~~OP_XOR~~ | OP_EQUAL | OP_EQUALVERIFY | ~~OP_RESERVED1~~ | ~~OP_RESERVED2~~

scom_numeric ::= OP_1ADD | OP_1SUB | ~~OP_2MUL~~ | ~~OP_2DIV~~ | OP_NEGATE | OP_ABS | OP_NOT | OP_0NOTEQUAL | OP_ADD | OP_SUB | ~~OP_MUL~~ | ~~OP_DIV~~ | ~~OP_MOD~~ | ~~OP_LSHIFT~~ | ~~OP_RSHIFT~~ | OP_BOOLAND | OP_BOOLOR | OP_NUMEQUAL | OP_NUMEQUALVERIFY | OP_NUMNOTEQUAL | OP_LESSTHAN | OP_GREATERTHAN | OP_LESSTHANOREQUA | OP_GREATERTHANOREQUAL | OP_MIN | OP_MAX | OP_WITHIN

scom_crypto ::= OP_RIPEMD160 | OP_SHA1 | OP_SHA256 | OP_HASH160 | OP_HASH256 | OP_CODESEPARATOR
| OP_CHECKSIG | OP_CHECKSIGVERIFY | OP_CHECKMULTISIG | OP_CHECKMULTISIGVERIFY

scom_expansion ::= ~~OP_NOP1~~ | OP_CHECKLOCKTIMEVERIFY | OP_NOP2 | OP_CHECKSEQUENCEVERIFY | OP_NOP3
| ~~OP_NOP4~~ | ~~OP_NOP5~~ | ~~OP_NOP6~~ | ~~OP_NOP7~~ | ~~OP_NOP8~~ | ~~OP_NOP9~~ | ~~OP_NOP10~~

scom_template ::= ~~OP_SMALLINTEGER~~ | ~~OP_PUBKEYS~~ | ~~OP_PUBKEYHASH~~ | ~~OP_PUBKEY~~ | ~~OP_INVALIDOPCODE~~

2.3.2 Multiple-Word Commands

mcom ::= mcom_if | mcom_notif | mcom_push

mcom_if ::= OP_IF script OP_ENDIF | OP_IF script mcom_else OP_ENDIF
mcom_notif ::= OP_NOTIF script OP_ENDIF | OP_NOTIF script mcom_else OP_ENDIF
mcom_else ::= OP_ELSE script | OP_ELSE script mcom_else

mcom_push ::= OP_PUSHBYTES$_N$ B^N | OP_PUSHDATA1 B^1 B |OP_PUSHDATA2 B^2 B | OP_PUSHDATA4 B^4 B

2.3.3 Disabled Commands

dcom ::= dcom_push | dcom_control | dcom_stack | dcom_splice | dcom_bitlogic | dcom_numeric |
dcom_crypto | dcom_expansion | dcom_template

dcom_push ::= ~~OP_RESERVED~~

scom_control ::= ~~OP_VER~~ | ~~OP_VERIF~~ | ~~OP_VERNOTIF~~

dcom_stack ::=

dcom_splice ::= ~~OP_CAT~~ | ~~OP_SUBSTR~~ | ~~OP_LEFT~~ | ~~OP_RIGHT~~

dcom_bitlogic ::= ~~OP_INVERT~~ | ~~OP_AND~~ | ~~OP_OR~~ | ~~OP_XOR~~ | ~~OP_RESERVED1~~ | ~~OP_RESERVED2~~

dcom_numeric ::= ~~OP_2MUL~~ | ~~OP_2DIV~~ | ~~OP_MUL~~ | ~~OP_DIV~~ | ~~OP_MOD~~ | ~~OP_LSHIFT~~ | ~~OP_RSHIFT~~

dcom_crypto ::=

dcom_expansion ::= ~~OP_NOP1~~ | ~~OP_NOP4~~ | ~~OP_NOP5~~ | ~~OP_NOP6~~ | ~~OP_NOP7~~ | ~~OP_NOP8~~ | ~~OP_NOP9~~ | ~~OP_NOP10~~

dcom_template ::= ~~OP_SMALLINTEGER~~ | ~~OP_PUBKEYS~~ | ~~OP_PUBKEYHASH~~ | ~~OP_PUBKEY~~ | ~~OP_INVALIDOPCODE~~

2.3.4 OP Codes

push value

OP_0 ::= 0x00
OP_FALSE ::= OP_0
OP_PUSHBYTES^N ::= 0x01-0x4b
OP_PUSHDATA1 ::= 0x4c
OP_PUSHDATA2 ::= 0x4d
OP_PUSHDATA4 ::= 0x4e
OP_1NEGATE ::= 0x4f
OP_RESERVED ::= 0x50
OP_1 ::= 0x51
OP_TRUE::=OP_1
OP_2 ::= 0x52
OP_3 ::= 0x53
OP_4 ::= 0x54
OP_5 ::= 0x55
OP_6 ::= 0x56
OP_7 ::= 0x57
OP_8 ::= 0x58
OP_9 ::= 0x59
OP_10 ::= 0x5a
OP_11 ::= 0x5b
OP_12 ::= 0x5c
OP_13 ::= 0x5d
OP_14 ::= 0x5e
OP_15 ::= 0x5f
OP_16 ::= 0x60

control

OP_NOP ::= 0x61
OP_VER ::= 0x62
OP_IF ::= 0x63
OP_NOTIF ::= 0x64
OP_VERIF ::= 0x65
OP_VERNOTIF ::= 0x66
OP_ELSE ::= 0x67
OP_ENDIF ::= 0x68
OP_VERIFY ::= 0x69
OP_RETURN ::= 0x6a

stack ops

OP_TOALTSTACK ::= 0x6b
OP_FROMALTSTACK ::= 0x6c
OP_2DROP ::= 0x6d
OP_2DUP ::= 0x6e
OP_3DUP ::= 0x6f
OP_2OVER ::= 0x70
OP_2ROT ::= 0x71
OP_2SWAP ::= 0x72
OP_IFDUP ::= 0x73
OP_DEPTH ::= 0x74
OP_DROP ::= 0x75
OP_DUP ::= 0x76
OP_NIP ::= 0x77
OP_OVER ::= 0x78
OP_PICK ::= 0x79
OP_ROLL ::= 0x7a
OP_ROT ::= 0x7b
OP_SWAP ::= 0x7c
OP_TUCK ::= 0x7d

splice ops

OP_CAT ::= 0x7e
OP_SUBSTR ::= 0x7f
OP_LEFT ::= 0x80
OP_RIGHT ::= 0x81
OP_SIZE ::= 0x82

bit logic

OP_INVERT ::= 0x83
OP_AND ::= 0x84
OP_OR ::= 0x85
OP_XOR ::= 0x86
OP_EQUAL ::= 0x87
OP_EQUALVERIFY ::= 0x88
OP_RESERVED1 ::= 0x89
OP_RESERVED2 ::= 0x8a

numeric

OP_1ADD ::= 0x8b
OP_1SUB ::= 0x8c
OP_2MUL ::= 0x8d
OP_2DIV ::= 0x8e
OP_NEGATE ::= 0x8f
OP_ABS ::= 0x90
OP_NOT ::= 0x91
OP_0NOTEQUAL ::= 0x92
OP_ADD ::= 0x93
OP_SUB ::= 0x94
OP_MUL ::= 0x95
OP_DIV ::= 0x96
OP_MOD ::= 0x97
OP_LSHIFT ::= 0x98
OP_RSHIFT ::= 0x99
OP_BOOLAND ::= 0x9a
OP_BOOLOR ::= 0x9b
OP_NUMEQUAL ::= 0x9c
OP_NUMEQUALVERIFY ::= 0x9d
OP_NUMNOTEQUAL ::= 0x9e
OP_LESSTHAN ::= 0x9f
OP_GREATERTHAN ::= 0xa0
OP_LESSTHANOREQUAL ::= 0xa1
OP_GREATERTHANOREQUAL ::= 0xa2
OP_MIN ::= 0xa3
OP_MAX ::= 0xa4
OP_WITHIN ::= 0xa5

crypto

OP_RIPEMD160 ::= 0xa6
OP_SHA1 ::= 0xa7
OP_SHA256 ::= 0xa8
OP_HASH160 ::= 0xa9
OP_HASH256 ::= 0xaa
OP_CODESEPARATOR ::= 0xab
OP_CHECKSIG ::= 0xac
OP_CHECKSIGVERIFY ::= 0xad
OP_CHECKMULTISIG ::= 0xae
OP_CHECKMULTISIGVERIFY ::= 0xaf

expansion

OP_NOP1 ::= 0xb0
OP_CHECKLOCKTIMEVERIFY ::=
OP_NOP2 ::= 0xb1
OP_CHECKSEQUENCEVERIFY ::=
OP_NOP3 ::= 0xb2
OP_NOP4 ::= 0xb3
OP_NOP5 ::= 0xb4
OP_NOP6 ::= 0xb5
OP_NOP7 ::= 0xb6
OP_NOP8 ::= 0xb7
OP_NOP9 ::= 0xb8
OP_NOP10 ::= 0xb9

template matching params

OP_SMALLINTEGER ::= 0xfa
OP_PUBKEYS ::= 0xfb
OP_PUBKEYHASH ::= 0xfd
OP_PUBKEY ::= 0xfe
OP_INVALIDOPCODE ::= 0xff

3 Operational Semantics – Big Step Semantics

At any given point of a program's execution, the state is entirely described by the following elements:

- The stack S, whose elements S_i for $1 \leq i \leq |S|$ are indexed starting from the bottom;
- The alt-stack AS, an auxiliary stack indexed in the same way;
- The validity variable V, equal to `true` by default.

Any state in which $V =$ `false` causes the program to immediately terminate with a special exception, handled by the environment in which the program was being run.

3.1 Generic rules

In order to more succinctly define the operational semantics of Script, we define generic rules not tied to any command in the language that express operations common in a stack-based language.

3.1.1 PUSH

$$\frac{|S| = L}{< \texttt{PUSH}\ \ B, \sigma > \Downarrow \sigma[|S| = L+1, S_{|S|} = B]}$$

3.1.2 DROP

$$\frac{\sigma(|S|) = L \quad B \Downarrow n \quad L \geq n}{< \texttt{DROP}\ \ B, \sigma > \Downarrow \sigma[S_i = S_{i+1} \quad \forall i.\ n \leq i < L]}$$

3.1.3 PUSH_ALT

$$\frac{|AS| = L}{< \texttt{PUSH\_ALT}\ \ B, \sigma > \Downarrow \sigma[|AS| = L+1, AS_{|AS|} = B]}$$

3.1.4 DROP_ALT

$$\frac{\sigma(|AS|) = L \quad \sigma(|AS|) \geq B}{< \texttt{DROP\_ALT}\ \ B, \sigma > \Downarrow \sigma[AS_i = AS_{i+1} \quad \forall i.\ B \leq i < L]}$$

3.1.5 TEST

$$\frac{\sigma(|S|) \geq x \quad \sigma(S_x) \Downarrow t \quad t = 0}{< \texttt{TEST}\ \ x, \sigma > \Downarrow FALSE}$$

$$\frac{\sigma(|S|) \geq x \quad \sigma(S_x) \Downarrow t \quad t \neq 0}{< \texttt{TEST}\ \ x, \sigma > \Downarrow TRUE}$$

3.2 Constants

3.2.1 OP_0, OP_FALSE

Push the byte-vector representing 0 onto the stack.

$$\frac{\sigma(|S|) = L \quad B \Downarrow 0}{< \texttt{OP\_0}, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B]}$$

3.2.2 OP_N, *OP_PUSHNBYTES*

Push the next N bytes onto the stack.

$$\frac{\sigma(|S|) = L}{< \texttt{OP\_N}\ b_1 \ldots b_N, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} =< b_1 \ldots b_N >]}$$

3.2.3 OP_PUSHDATA1

The next byte specifies how many bytes to push onto the stack as a byte-vector.

$$\frac{B_1 \Downarrow k \quad \sigma(|S|) = L}{< \texttt{OP\_PUSHDATA1}\ B_1^1\ \ B_2^k, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B_2]}$$

3.2.4 OP_PUSHDATA2

The next two bytes specify how many bytes to push onto the stack as a byte-vector.

$$\frac{B_1 \Downarrow k \quad \sigma(|S|) = L}{< \texttt{OP\_PUSHDATA2}\ B_1^2\ \ B_2^k, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B_2]}$$

3.2.5 OP_PUSHDATA4

The next four bytes specify how many bytes to push onto the stack as a byte-vector.

$$\frac{B_1 \Downarrow k \quad \sigma(|S|) = L}{< \texttt{OP\_PUSHDATA4}\ B_1^4\ \ B_2^k, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B_2]}$$

3.2.6 OP_1NEGATE

Push the byte-vector representing -1 onto the stack.

$$\frac{\sigma(|S|) = L \quad B \Downarrow -1}{< \texttt{OP\_0}, \sigma > \Downarrow \sigma[|S| = l+1, S_{l+1} = B]}$$

3.2.7 OP_1, OP_TRUE

Push the byte-vector representing 1 onto the stack.

$$\frac{\sigma(|S|) = L \quad B \Downarrow 1}{< \texttt{OP\_0}, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B]}$$

3.2.8 OP_2-OP_16, *OP_PUSHN*

Push the byte-vector representing the number specified in the word name onto the stack.

$$\frac{\sigma(|S|) = L \quad B \Downarrow N}{< \texttt{OP\_PUSHN}, \sigma > \Downarrow \sigma[|S| = L+1, S_{L+1} = B]}$$

3.3 Flow Control

3.3.1 OP_NOP

Do nothing.

$$\frac{}{<\texttt{OP\_NOP}, \sigma> \Downarrow \sigma}$$

3.3.2 OP_IF, OP_ELSE, OP_ENDIF

Run C_1 if top of stack is present and evaluates to true. Otherwise run C_2 if stack is present and evaluates to false.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{TRUE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_1, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_IF}\ C_1\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{FALSE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma'}{<\texttt{OP\_IF}\ C_1\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{TRUE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_1, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_IF}\ C_1\ \texttt{OP\_ELSE}\ C_2\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{FALSE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_2, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_IF}\ C_1\ \texttt{OP\_ELSE}\ C_2\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

3.3.3 OP_NOTIF, OP_ELSE, OP_ENDIF

Run C_1 if top of stack is present and evaluates to false. Otherwise run C_2 if stack is present and evaluates to true.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{FALSE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_1, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_NOTIF}\ C_1\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{TRUE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma'}{<\texttt{OP\_NOTIF}\ C_1\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{FALSE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_1, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_NOTIF}\ C_1\ \texttt{OP\_ELSE}\ C_2\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{TRUE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma_1 \quad <C_2, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_NOTIF}\ C_1\ \texttt{OP\_ELSE}\ C_2\ \texttt{OP\_ENDIF}> \Downarrow \sigma'}$$

3.3.4 OP_VERIFY := 0x69

If top of stack is present and evaluates to true then remove top of stock and mark transaction as valid; otherwise mark transaction as invalid.

$$\frac{\sigma(|S|) = L \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{FALSE} \quad \sigma'(V) = \texttt{invalid}}{<\texttt{OP\_VERIFY}, \sigma> \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad <\texttt{TEST}\ L, \sigma> \Downarrow \texttt{TRUE} \quad <\texttt{DROP}\ L, \sigma> \Downarrow \sigma' \quad \sigma'(V) = \texttt{valid}}{<\texttt{OP\_VERIFY}, \sigma> \Downarrow \sigma'}$$

3.3.5 OP_RETURN := 0x6a

Unconditionally mark transaction as invalid.

$$\frac{}{<\texttt{OP\_RETURN}, \sigma> \Downarrow \sigma[V = \texttt{invalid}]}$$

3.4 Stack

3.4.1 OP_TOALTSTACK

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) = x \quad <\texttt{DROP}\ \ L, \sigma> \Downarrow \sigma_1 <\texttt{PUSHALT}\ \ x, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_TOALTSTACK}, \sigma> \Downarrow \sigma}$$

3.4.2 OP_FROMALTSTACK

$$\frac{\sigma(|AS|) = L \quad L > 0 \quad \sigma(AS_L) = x \quad <\texttt{DROPALT}\ \ L, \sigma> \Downarrow \sigma_1 <\texttt{PUSH}\ \ x, \sigma_1> \Downarrow \sigma'}{<\texttt{OP\_FROMALTSTACK}, \sigma> \Downarrow \sigma}$$

3.4.3 OP_IFDUP

If top of stack is non-zero, duplicate the top of the stack.

$$\frac{\sigma(S_{|S|}) \neq 0 \quad <\texttt{PUSH}\ \ S_{|S|} \Downarrow \sigma'>}{<\texttt{OP\_IFDUP}, \sigma> \Downarrow \sigma'}$$

3.4.4 OP_DEPTH

Push the byte-vector representing the depth of the stack onto the stack.

$$\frac{\sigma(|S|) = L \quad <\texttt{PUSH}\ \ L, \sigma> \Downarrow \sigma'}{<\texttt{OP\_DEPTH}, \sigma> \Downarrow \sigma'}$$

3.4.5 OP_DROP

Remove the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{DROP}\ \ L, \sigma> \Downarrow \sigma'}{<\texttt{OP\_DROP}, \sigma> \Downarrow \sigma'}$$

3.4.6 OP_DUP

Duplicate the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad <\texttt{PUSH}\ \ S_L, \sigma> \Downarrow \sigma'}{<\texttt{OP\_DUP}, \sigma> \Downarrow \sigma'}$$

3.4.7 OP_NIP

Remove the byte-vector second from the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad < \texttt{DROP}\ \ L-1, \sigma > \Downarrow \sigma'}{< \texttt{OP\_NIP}, \sigma > \Downarrow \sigma'}$$

3.4.8 OP_OVER

Copy the byte-vector second from the top onto the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad < \texttt{PUSH}\ \ S_{L-1}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_OVER}, \sigma > \Downarrow \sigma'}$$

3.4.9 OP_PICK

Copy the byte-vector n from the top onto the top of the stack, not counting the top element n.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) = n \quad L > n \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ \ S_{L-n-1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_PICK}, \sigma > \Downarrow \sigma'}$$

3.4.10 OP_ROLL

Move the byte-vector n from the top onto the top of the stack, not counting the top element n.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) = n \quad L > n \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ \ S_{L-n-1}, \sigma_1 > \Downarrow \sigma_2 < \texttt{DROP}\ \ L-n-1, \sigma_2 > \Downarrow \sigma'}{< \texttt{OP\_ROLL}, \sigma > \Downarrow \sigma'}$$

3.4.11 OP_ROT

The top three items on the stack are rotated. Equivalently, move the byte-vector third from the top to the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 2 \quad < \texttt{PUSH}\ \ S_{L-2}, \sigma_1 > \Downarrow \sigma_2 < \texttt{DROP}\ \ L-2, \sigma_2 > \Downarrow \sigma'}{< \texttt{OP\_ROT}, \sigma > \Downarrow \sigma'}$$

3.4.12 OP_SWAP

The top two items on the stack are swapped.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad < \texttt{PUSH}\ \ S_{L-1}, \sigma_1 > \Downarrow \sigma_2 < \texttt{DROP}\ \ L-1, \sigma_2 > \Downarrow \sigma'}{< \texttt{OP\_SWAP}, \sigma > \Downarrow \sigma'}$$

3.4.13 OP_TUCK

The item at the top of the stack is copied and inserted before the second-to-top item.

$$\frac{\begin{array}{c}\sigma(|S|) > 1 \quad \sigma(S_{|S|}) = x \quad \sigma(S_{|S|-1}) = y \quad < \texttt{DROP}\ \ |S|, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ |S|, \sigma_1 > \Downarrow \sigma_2 \\ < \texttt{PUSH}\ \ x, \sigma_2 > \Downarrow \sigma_3 \quad < \texttt{PUSH}\ \ y, \sigma_3 > \Downarrow \sigma_4 \quad < \texttt{PUSH}\ \ x, \sigma_4 > \Downarrow \sigma'\end{array}}{< \texttt{OP\_TUCK}, \sigma > \Downarrow \sigma'}$$

3.4.14 OP_2DROP

Removes the top two stack items.

$$\frac{\sigma(|S|) > 1 \quad < \texttt{DROP}\ \ |S|, \sigma >\Downarrow \sigma_1 \quad < \texttt{DROP}\ \ |S|, \sigma_1 >\Downarrow \sigma'}{< \texttt{OP\_2DROP}, \sigma >\Downarrow \sigma'}$$

3.4.15 OP_2DUP

Duplicates the top two stack items.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad < \texttt{PUSH}\ \ S_{L-1}, \sigma >\Downarrow \sigma_1 \quad < \texttt{PUSH}\ \ S_L, \sigma_1 >\Downarrow \sigma'}{< \texttt{OP\_2DUP}, \sigma >\Downarrow \sigma'}$$

3.4.16 OP_3DUP

Duplicates the top three stack items.

$$\frac{\sigma(|S|) = L \quad L > 2 \quad < \texttt{PUSH}\ \ S_{L-2}, \sigma >\Downarrow \sigma_1 \quad < \texttt{PUSH}\ \ S_{L-1}, \sigma_1 >\Downarrow \sigma_2 \quad < \texttt{PUSH}\ \ S_L, \sigma_2 >\Downarrow \sigma'}{< \texttt{OP\_3DUP}, \sigma >\Downarrow \sigma'}$$

3.4.17 OP_2OVER

Copies the pair of items two spaces back in the stack to the front.

$$\frac{\sigma(|S|) = L \quad L > 3 \quad < \texttt{PUSH}\ \ S_{L-3}, \sigma >\Downarrow \sigma_1 \quad < \texttt{PUSH}\ \ S_{L-2}, \sigma_1 >\Downarrow \sigma'}{< \texttt{OP\_2OVER}, \sigma >\Downarrow \sigma'}$$

3.4.18 OP_2ROT

The fifth and sixth items back are moved to the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 5 \quad < \texttt{PUSH}\ \ S_{L-5}, \sigma_2 >\Downarrow \sigma_3 \quad < \texttt{PUSH}\ \ S_{L-4}, \sigma_3 >\Downarrow \sigma' \quad < \texttt{DROP}\ \ L-5, \sigma >\Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L-4, \sigma_1 >\Downarrow \sigma_2}{< \texttt{OP\_2ROT}, \sigma >\Downarrow \sigma'}$$

3.4.19 OP_2SWAP

The first and second items are swapped. The third and fourth items are swapped.

$$\frac{\begin{array}{c}\sigma(|S|) = L \quad L > 3 \quad \sigma(S_L) = x \quad \sigma(S_{L-1}) = y \quad < \texttt{DROP}\ \ L, \sigma >\Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L-1, \sigma_1 >\Downarrow \sigma_2 \\ < \texttt{PUSH}\ \ S_{L-3}, \sigma_2 >\Downarrow \sigma_3 \quad < \texttt{DROP}\ \ L-3, \sigma_3 >\Downarrow \sigma_4 \quad < \texttt{PUSH}\ \ y, \sigma_4 >\Downarrow \sigma_5 \quad < \texttt{PUSH}\ \ x, \sigma_5 >\Downarrow \sigma'\end{array}}{< \texttt{OP\_2SWAP}, \sigma >\Downarrow \sigma'}$$

3.5 Splice

3.5.1 OP_CAT

Disabled

Concatenates two byte strings.

$$\frac{\sigma(|S|) = L \quad < \texttt{PUSH}\ \ S_L S_{L-1}, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L, \sigma_1 > \Downarrow \sigma_2 \quad < \texttt{DROP}\ \ L-1, \sigma_2 > \Downarrow \sigma'}{< \texttt{OP\_CAT}, \sigma > \Downarrow \sigma'}$$

3.5.2 OP_SUBSTR

Disabled
Returns a subtring of the top element, starting from index given by second element, of size given by third element.

$$\frac{\begin{array}{c}\sigma(|S|) = L \quad \sigma(S_L) = b_1 \ldots b_k \quad \sigma(S_{L-1}) = \mathit{start} \quad \sigma(S_{L-2}) = \mathit{size} \quad \mathit{start} < k \quad \mathit{start} + \mathit{size} < k \\ < \texttt{PUSH}\ \ b_{start} \ldots b_{start+size}, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L, \sigma_1 > \Downarrow \sigma_2 \quad < \texttt{DROP}\ \ L-1, \sigma_2 > \Downarrow \sigma_3 \quad < \texttt{DROP}\ \ L-2, \sigma_3 > \Downarrow \sigma'\end{array}}{< \texttt{OP\_SUBSTR}, \sigma > \Downarrow \sigma'}$$

3.5.3 OP_LEFT

Disabled
Returns only character left of a specified point in top element, index given by second element.

$$\frac{\begin{array}{c}\sigma(|S|) = L \quad \sigma(S_L) = b_1 \ldots b_k \quad \sigma(S_{L-1}) = \mathit{idx} \quad \mathit{idx} \leq k \\ < \texttt{PUSH}\ \ b_1 b_{idx-1}, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L, \sigma_1 > \Downarrow \sigma_2 \quad < \texttt{DROP}\ \ L-1, \sigma_2 > \Downarrow \sigma'\end{array}}{< \texttt{OP\_LEFT}, \sigma > \Downarrow \sigma'}$$

3.5.4 OP_RIGHT

Disabled
Returns only character right of a specified point in top element, index given by second element.

$$\frac{\begin{array}{c}\sigma(|S|) = L \quad \sigma(S_L) = b_1 \ldots b_k \quad \sigma(S_{L-1}) = \mathit{idx} \quad \mathit{idx} \leq k \\ < \texttt{PUSH}\ \ b_{idx} \ldots b_k, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L, \sigma_1 > \Downarrow \sigma_2 \quad < \texttt{DROP}\ \ L-1, \sigma_2 > \Downarrow \sigma'\end{array}}{< \texttt{OP\_RIGHT}, \sigma > \Downarrow \sigma'}$$

3.5.5 OP_SIZE

Pushes the number of bytes of the top element of the stack.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) = B \quad < \texttt{PUSH}\ \ |B|, \sigma > \Downarrow \sigma'}{< \texttt{OP\_SIZE}, \sigma > \Downarrow \sigma'}$$

3.6 Bitwise Logic

3.6.1 OP_INVERT

Disabled
Flips all bits of the top stack element.

$$\frac{\sigma(|S|) = L \quad \sigma(S_L) = b_1 \ldots b_k \quad < \texttt{PUSH}\ \ \neg b_1 \neg b_2 \ldots \neg b_k, \sigma > \Downarrow \sigma_1 \quad < \texttt{DROP}\ \ L, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_EQUAL}, \sigma > \Downarrow \sigma'}$$

3.6.2 OP_AND

Disabled
Boolean *and* between bits of first and second stack elements.

$$\frac{\sigma(|S|) = L \quad \sigma(S_L) = B_1 \quad \sigma(S_{L-1}) = B_2 \\ <\texttt{PUSH}\ \ B_1 \ \& \ B_2, \sigma >\Downarrow \sigma_1 \quad <\texttt{DROP}\ \ L, \sigma_1 >\Downarrow \sigma_2 \quad <\texttt{DROP}\ \ L-1, \sigma_2 >\Downarrow \sigma'}{<\texttt{OP\_AND}, \sigma >\Downarrow \sigma'}$$

3.6.3 OP_OR

Disabled
Boolean *and* between bits of first and second stack elements.

$$\frac{\sigma(|S|) = L \quad \sigma(S_L) = B_1 \quad \sigma(S_{L-1}) = B_2 \\ <\texttt{PUSH}\ \ B_1 \mid B_2, \sigma >\Downarrow \sigma_1 \quad <\texttt{DROP}\ \ L, \sigma_1 >\Downarrow \sigma_2 \quad <\texttt{DROP}\ \ L-1, \sigma_2 >\Downarrow \sigma'}{<\texttt{OP\_OR}, \sigma >\Downarrow \sigma'}$$

3.6.4 OP_XOR

Disabled
Boolean *and* between bits of first and second stack elements.

$$\frac{\sigma(|S|) = L \quad \sigma(S_L) = B_1 \quad \sigma(S_{L-1}) = B_2 \\ <\texttt{PUSH}\ \ B_1 \wedge B_2, \sigma >\Downarrow \sigma_1 \quad <\texttt{DROP}\ \ L, \sigma_1 >\Downarrow \sigma_2 \quad <\texttt{DROP}\ \ L-1, \sigma_2 >\Downarrow \sigma'}{<\texttt{OP\_XOR}, \sigma >\Downarrow \sigma'}$$

3.6.5 OP_EQUAL

Returns 1 if the two top elements are equal, 0 otherwise.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) = B_1 \quad \sigma(S_{L-1}) = B_2 \quad B_1 = B_2 \quad <\texttt{PUSH}\ \ 1, \sigma >\Downarrow \sigma'}{<\texttt{OP\_EQUAL}, \sigma >\Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) = B_1 \quad \sigma(S_{L-1}) = B_2 \quad B_1 \neq B_2 \quad <\texttt{PUSH}\ \ 0, \sigma >\Downarrow \sigma'}{<\texttt{OP\_EQUAL}, \sigma >\Downarrow \sigma'}$$

3.6.6 OP_EQUALVERIFY

Description.

$$\frac{<\texttt{OP\_EQUAL}\ \ , \sigma >\Downarrow \sigma_1 <\texttt{OP\_VERIFY}\ \ , \sigma_1 >\Downarrow \sigma'}{<\texttt{OP\_EQUALVERIFY}, \sigma >\Downarrow \sigma'}$$

3.7 Arithmetic

3.7.1 OP_1ADD

Add 1 to the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) \Downarrow x \quad B \Downarrow x+1 \quad < \texttt{DROP}\ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ B, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_2SWAP}, \sigma > \Downarrow \sigma'}$$

3.7.2 OP_1SUB

Subtract 1 from the top of the stack.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) \Downarrow x \quad B \Downarrow x-1 \quad < \texttt{DROP}\ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ B, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_1SUB}, \sigma > \Downarrow \sigma'}$$

3.7.3 OP_ABS

The top of the stack is replaced with the byte-vector representing it's absolute value.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) \Downarrow x \quad B \Downarrow |x| \quad < \texttt{DROP}\ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ B, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_1SUB}, \sigma > \Downarrow \sigma'}$$

3.7.4 OP_NEGATE

The top of the stack is negated.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad \sigma(S_L) \Downarrow x \quad B \Downarrow -x \quad < \texttt{DROP}\ L, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH}\ B, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_1SUB}, \sigma > \Downarrow \sigma'}$$

3.7.5 OP_NOT

If the top of the stack is 0, replace it with the byte vector representing 1. Otherwise, replace it with the byte-vector representing 0.

$$\frac{< \texttt{OP\_IF PUSH OP\_0 OP\_ELSE PUSH OP\_1 OP\_ENDIF}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_NOT}, \sigma > \Downarrow \sigma'}$$

3.7.6 OP_0NOTEQUAL

Push the byte-vector representing 0 onto the stack if the top of stack is 0. Otherwise replace the top of stack with the byte-vector representing 1.

$$\frac{< \texttt{OP\_IF PUSH OP\_1 OP\_ELSE PUSH OP\_0 OP\_ENDIF}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_0NOTEQUAL}, \sigma > \Downarrow \sigma'}$$

3.7.7 OP_ADD

Let the inputs be the top two items of the stack, a and b. Push the byte-vector representing $a + b$ onto the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow x \quad \sigma(S_{L-1}) \Downarrow y \quad B \Downarrow x+y \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH B}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_ADD}, \sigma > \Downarrow \sigma'}$$

3.7.8 OP_SUB

Let the inputs be the top two items of the stack, a and b. Push the byte-vector representing $a - b$ onto the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow x \quad \sigma(S_{L-1}) \Downarrow y \quad B \Downarrow y - x \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH B}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_SUB}, \sigma > \Downarrow \sigma'}$$

3.7.9 OP_BOOLAND

Let the inputs be the top two items of the stack, a and b. If both a and b are not 0, then push 1 onto the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow a \quad \sigma(S_{L-1}) \Downarrow b \quad a \neq 0 \wedge b \neq 0 \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_BOOLAND}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow a \quad \sigma(S_{L-1}) \Downarrow b \quad a = 0 \vee b = 0 \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{PUSH OP\_0}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_BOOLAND}, \sigma > \Downarrow \sigma'}$$

3.7.10 OP_BOOLOR

Let the inputs be the top two items of the stack, a and b. If either a or b are not 0, then push 1 onto the stack.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow a \quad \sigma(S_{L-1}) \Downarrow b \quad a \neq 0 \vee b \neq 0 \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_BOOLOR}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_L) \Downarrow a \quad \sigma(S_{L-1}) \Downarrow b \quad a \neq 0 \wedge b \neq 0 \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_BOOLOR}, \sigma > \Downarrow \sigma'}$$

3.7.11 OP_NUMEQUAL

Let the inputs be the top two items of the stack, a and b. Push 1 onto the stack if $a = b$; push 0 onto the stack otherwise.

$$\frac{< \texttt{OP\_SUB}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_IF PUSH OP\_0 OP\_ELSE PUSH OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_NUMEQUAL}, \sigma > \Downarrow \sigma'}$$

3.7.12 OP_NUMEQUALVERIFY

Let the inputs be the top two items of the stack, a and b. Push 1 onto the stack if $a = b$; push 0 onto the stack otherwise. If 0 is pushed onto the stack, mark the transaction as invalid.

$$\frac{< \texttt{OP\_NUMEQUAL}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_VERIFY}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_NUMEQUALVERIFY}, \sigma > \Downarrow \sigma'}$$

3.7.13 OP_NUMNOTEQUAL

Let the inputs be the top two items of the stack, a and b. Push 1 if the numbers are equal, 0 otherwise.

$$\frac{< \texttt{OP\_SUB}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_IF PUSH OP\_1 OP\_ELSE PUSH OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_NUMNOTEQUAL}, \sigma > \Downarrow \sigma'}$$

3.7.14 OP_LESSTHAN

Let the inputs be the element on the top of the stack(b) and the element second to the top (a). Push 1 onto the stack if a is less than b, otherwise push 0.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \geq b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_LESSTHAN}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a < b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_LESSTHAN}, \sigma > \Downarrow \sigma'}$$

3.7.15 OP_GREATERTHAN

Let the inputs be the element on the top of the stack(b) and the element second to the top (a). Push 1 onto the stack if a is greater than b, otherwise push 0.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \leq b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_GREATERTHAN}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a > b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_GREATERTHAN}, \sigma > \Downarrow \sigma'}$$

3.7.16 OP_LESSTHANOREQUAL

Let the inputs be the element on the top of the stack(b) and the element second to the top (a). Push 1 onto the stack if a is less than or equal to b, otherwise push 0.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a > b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_LESSTHANOREQUAL}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \leq b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_LESSTHANOREQUAL}, \sigma > \Downarrow \sigma'}$$

3.7.17 OP_GREATERTHANOREQUAL

Let the inputs be the element on the top of the stack(b) and the element second to the top (a). Push 1 onto the stack if a is greater than or equal to b, otherwise push 0.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a < b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_0}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_GREATERTHANOREQUAL}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \geq b \quad < \texttt{OP\_2DROP}, \sigma > \Downarrow \sigma_1 \quad < \texttt{OP\_1}, \sigma_1 > \Downarrow \sigma'}{< \texttt{OP\_GREATERTHANOREQUAL}, \sigma > \Downarrow \sigma'}$$

3.7.18 OP_MIN

the larger of the two elements at the top of the stack is removed.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \leq b \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow \sigma'}{< \texttt{OP\_MIN}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad b < a \quad < \texttt{DROP}\ \ L-1, \sigma > \Downarrow \sigma'}{< \texttt{OP\_MIN}, \sigma > \Downarrow \sigma'}$$

3.7.19 OP_MAX

the smaller of the two elements on the top of the stack is removed.

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad a \geq b \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow \sigma'}{< \texttt{OP\_MAX}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 1 \quad \sigma(S_{L-1}) \Downarrow a \quad \sigma(S_L) \Downarrow b \quad b > a \quad < \texttt{DROP}\ \ L-1, \sigma > \Downarrow \sigma'}{< \texttt{OP\_MAX}, \sigma > \Downarrow \sigma'}$$

3.7.20 OP_WITHIN

the input is the 3 elements at the top of the stack x, min, max where max is the top of the stack. push 1 if x is greater than or equal to min and less than max, otherwise push 0.

$$\frac{\sigma(|S|) = L \quad L > 2 \quad \sigma(S_L) \Downarrow max \quad \sigma(S_{L-1}) \Downarrow min \quad \sigma(S_{L-2}) \Downarrow x \quad min \leq x < max \quad < \texttt{OP\_1}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_WITHIN}, \sigma > \Downarrow \sigma'}$$

$$\frac{\sigma(|S|) = L \quad L > 2 \quad \sigma(S_L) \Downarrow max \quad \sigma(S_{L-1}) \Downarrow min \quad \sigma(S_{L-2}) \Downarrow x \quad \neg(min \leq x < max) \quad < \texttt{OP\_1}, \sigma > \Downarrow \sigma'}{< \texttt{OP\_WITHIN}, \sigma > \Downarrow \sigma'}$$

3.8 Crypto

3.8.1 OP_RIPEMD160

The element at the top of the stack is hashed with RIPEMD-160.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad B = \text{RIPEMD-160}(\sigma(S_L)) \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow sigma_1 \quad < \texttt{PUSH}\ \ B, \sigma_1 > \Downarrow sigma'}{< \texttt{OP\_RIPEMD-160}, \sigma > \Downarrow sigma'}$$

3.8.2 OP_SHA1

The element at the top of the stack is hashed with SHA-1.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad B = \text{SHA-1}(\sigma(S_L)) \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow sigma_1 \quad < \texttt{PUSH}\ \ B, \sigma_1 > \Downarrow sigma'}{< \texttt{OP\_SHA1}, \sigma > \Downarrow sigma'}$$

3.8.3 OP_SHA256

The element at the top of the stack is hashed with SHA-256.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad B = \text{SHA-256}(\sigma(S_L)) \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow sigma_1 \quad < \texttt{PUSH}\ \ B, \sigma_1 > \Downarrow sigma'}{< \texttt{OP\_SHA-256}, \sigma > \Downarrow sigma'}$$

3.8.4 OP_HASH160

The element at the top of the stack is hashed with SHA-256 and then with RIPEMD-160.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad B = \text{RIPEMD-160}(\text{SHA-256}(\sigma(S_L))) \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow sigma_1 \quad < \texttt{PUSH}\ \ B, \sigma_1 > \Downarrow sigma'}{< \texttt{OP\_HASH160}, \sigma > \Downarrow sigma'}$$

3.8.5 OP_HASH256

The element at the top of the stack is hashed twice with SHA-256.

$$\frac{\sigma(|S|) = L \quad L > 0 \quad B = \text{SHA-256}(\text{SHA-256}(\sigma(S_L))) \quad < \texttt{DROP}\ \ L, \sigma > \Downarrow sigma_1 \quad < \texttt{PUSH}\ \ B, \sigma_1 > \Downarrow sigma'}{< \texttt{OP\_HASH256}, \sigma > \Downarrow sigma'}$$

3.8.6 OP_CODESEPARATOR

3.8.7 OP_CHECKSIG

3.8.8 OP_CHECKSIGVERIFY

3.8.9 OP_CHECKMULTISIG

3.8.10 OP_CHECKMULTISIGVERIFY

3.9 Locktime

3.9.1 OP_CHECKLOCKTIMEVERIFY (previously OP_NOP2)

3.9.2 OP_CHECKSEQUENCEVERIFY (previously OP_NOP3)

3.10 Disabled / Reserved / Invalid

All disabled, reserved or otherwise invalid opcodes transition to an invalid state.

$$\frac{}{< \texttt{dcom}, \sigma > \Downarrow \sigma[V = invalid]}$$

4 Operational Semantics – Small Step Semantics

4.1 Local reduction rules

[No need to add PUSHDATA commands: these can be described as atomic commands, if we remember that the PUSHDATA opcode is only part of the push commands defined in the syntax - Hugo]

$< \texttt{OP\_NOP com}, \sigma > \quad \longrightarrow \quad < \texttt{com}, \sigma >$

$< \texttt{scom}, \sigma[S = s, V = v] > \quad \longrightarrow \quad < \texttt{OP\_NOP}, \sigma[S = s', V = v'] >$ [1]

$< \texttt{OP\_RETURN com}, \sigma > \quad \longrightarrow \quad < \texttt{OP\_NOP}, \sigma[V = false] >$

$< \texttt{OP\_IF com OP\_ENDIF}, \sigma[S_{|S|} \neq 0] > \quad \longrightarrow \quad < \texttt{com}, \sigma >$

$< \texttt{OP\_IF com OP\_ENDIF}, \sigma[S_{|S|} = 0] > \quad \longrightarrow \quad < \texttt{OP\_NOP}, \sigma >$

$< \texttt{OP\_NOTIF com OP\_ENDIF}, \sigma[S_{|S|} \neq 0] > \quad \longrightarrow \quad < \texttt{OP\_NOP}, \sigma >$

$< \texttt{OP\_NOTIF com OP\_ENDIF}, \sigma[S_{|S|} = 0] > \quad \longrightarrow \quad < \texttt{com}, \sigma >$

$< \texttt{OP\_IF com}_1 \texttt{ OP\_ELSE com}_2 \texttt{ OP\_ENDIF}, \sigma[S_{|S|} \neq 0] > \quad \longrightarrow \quad < \texttt{com}_1, \sigma >$

$< \texttt{OP\_IF com}_1 \texttt{ OP\_ELSE com}_2 \texttt{ OP\_ENDIF}, \sigma[S_{|S|} = 0] > \quad \longrightarrow \quad < \texttt{com}_2, \sigma >$

$< \texttt{OP\_NOTIF com}_1 \texttt{ OP\_ELSE com}_2 \texttt{ OP\_ENDIF}, \sigma[S_{|S|} \neq 0] > \quad \longrightarrow \quad < \texttt{com}_2, \sigma >$

$< \texttt{OP\_NOTIF com}_1 \texttt{ OP\_ELSE com}_2 \texttt{ OP\_ENDIF}, \sigma[S_{|S|} = 0] > \quad \longrightarrow \quad < \texttt{com}_1, \sigma >$

4.2 Global reduction rules

Since programs in Script are evaluated strictly from left to right, there is really only global reduction rule aside from $\bullet$.

$$H ::= \bullet \mid H \ \texttt{com}$$

[1] In reality there is one local reduction rule per single-word command. For each of them the state is changed in the way described in the big-step operational semantics. The important information here is that every single-word command reduces to `OP_NOP` in a single step.