PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SEMANTICS AND COMPLEXITY OF BITCOIN SCRIPT

THOMAS REISENEGGER BUTRÓN

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisors:
JUAN REUTTER
MARCELO ARENAS

Santiago de Chile, May 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SEMANTICS AND COMPLEXITY OF BITCOIN SCRIPT

THOMAS REISENEGGER BUTRÓN

Members of the Committee:
JUAN REUTTER [illegible]
MARCELO ARENAS [illegible]
DOMAGOJ VRGOČ Domagoj Vrgoč
MIGUEL ROMERO [illegible]
JORGE GIRONÁS [illegible]

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, May 2021

Gratefully to my family and my girlfriend.

ACKNOWLEDGEMENTS

I would like to start by thanking my advisors, Juan Reutter and Marcelo Arenas. Not only did they help me focus on the task at hand and guide me through this process, but they were also always available to answer any questions I had and to discuss any hurdle I came across when developing the results displayed in this work. They gave me the freedom to choose how to approach this undertaking and provided me with the tools I needed to succeed on each step of the way.

I would also like to thank Domagoj Vrgoč. He repeatedly provided insight into the research because of his vast knowledge on the subject and was always happy to answer any questions that arose during the study.

I would like to thank my parents, who put up with me during this process and allowed me to decide what I wanted to do in life. They have supported me this whole time and I am very grateful for that.

Finally, I would like to thank my girlfriend, Nicole, who was always there for me. She was always eager to help in whichever way she could and made the negative aspects bearable. I am very lucky to be able to rely on her.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the increased popularity of Bitcoin, there is a growing need to understand the functionality, security and performance of various mechanisms that comprise its protocol. Bitcoin's scripting language, Script, is one of the main building blocks of Bitcoin transactions. It was purposefully designed to not be Turing complete, such that it would not be possible to program never-ending executions. However, there has not been much research done into studying the properties and limitations of the language. Moreover, there does not exist a formal framework by which to analyze these characteristics. In this work we aim to provide a formal framework to study Script and to analyze certain problems related to the language. Concretely, we formally define the semantics of Script, and study the problem of determining whether a user-defined script is well-formed; that is, whether it can be unlocked, or whether it contains errors that would prevent this from happening. Specifically, we prove that this problem is NP-hard by providing a reduction from integer linear programming and that for the most relevant set of operators, if we establish certain reasonable assumptions on the usage of the language, the problem lies in NP.

Keywords: Bitcoin, Script, static analysis, unlockability.

RESUMEN

Con la creciente popularidad de Bitcoin ha surgido la necesidad de entender las funcionalidades, la seguridad y el rendimiento de los distintos mecanismos que componen su protocolo. El lenguaje de programación asociado a Bitcoin, Script, es uno de los principales componentes de las transacciones de Bitcoin. Este fue diseñado deliberadamente para no ser Turing completo, de forma que no fuera posible crear ejecuciones sin fin. Sin embargo, no existen muchos estudios dedicados a analizar las propiedades y limitaciones del lenguaje. Es más, no existe un marco de referencia formal que permita analizar estas características. En este trabajo buscamos proveer este marco de referencia, que permita estudiar Script y analizar ciertos problemas relacionados al lenguaje. Concretamente, definiremos formalmente la semántica de Script y estudiaremos el problema de determinar si un programa definido por un usuario está bien formado, es decir, si puede ser desbloqueado o presenta errores que impiden que esto ocurra. Específicamente, demostraremos que este problema es NP-duro, proveyendo una reducción desde programación lineal entera, y que, para el conjunto más relevante de operadores, si establecemos ciertas suposiciones razonables sobre el uso del lenguaje, el problema se encuentra en NP.

Palabras Claves: Bitcoin, Script, análisis estático, desbloqueabilidad.

1. INTRODUCTION

Bitcoin (Nakamoto, 2008) is a cryptocurrency system proposed in 2008 by a person or a group of people under the pseudonym Satoshi Nakamoto. Simply put, Bitcoin is a decentralized protocol that relies on a technology called blockchain to maintain a public ledger that stores the information of all of the transactions of Bitcoins, the currency associated with the system. The blockchain ensures that there is consensus between the agents in the network regarding which transactions have been successful and prohibits malicious agents from creating fraudulent transactions by using several cryptographic processes, such as digital signatures and cryptographic hashing functions.

Bitcoin solves the problem of creating a trustworthy system that does not depend on any country or institution, through which individuals are able to transfer digital currency to one another. In doing so, Bitcoin is intended to provide several advantages over a simple system that only supports transactions. The subject of this thesis is one of these features called smart contracts. However, in order to comprehend what smart contracts are, one first needs to understand some of the differences that exist between a common bank transaction and a Bitcoin transaction.

Firstly, in the Bitcoin protocol there does not exist the concept of accounts. If person A wants to transfer X amount of money to person B, instead of having an account with a balance that determines how much money they can transfer, person A must point to one or more transactions of which they are the recipient that add up to at least amount X. Evidently, the system must handle the problem of determining which transaction outputs have been spent and which have not. Instead of inspecting the whole ledger to determine whether certain transaction output has been spent, the nodes in the network keep a record of all of the UTXOs or unspent transaction outputs.

The second main difference between bank and Bitcoin transactions is that the Bitcoin protocol was designed in order to allow for more complex transaction spending requirements. In other words, instead of just indicating a recipient for a transaction, the sender states certain requirements that need to be met by the recipient in order to spend the transferred money. For example, one could wish to forbid the money from being spent before certain date or to require multiple people to agree to spend the money. The tool that is used to establish these requirements is Script, which is a non-Turing-complete scripting language designed specifically for this purpose.

Script was designed to disallow infinite loops from being created, such that the nodes in the network could not be tricked into executing a never-ending program. However, the requirements that can be represented through it can be decently complex. A smart contract is essentially a non-trivial set of requirements.

In practice, the protocol for establishing spending requirements for transaction outputs consists of associating each transaction output with a locking script that corresponds to a sequence of Script operators. Afterwards, when creating a new transaction, in addition to pointing to an unspent transaction output, the sender must provide an unlocking script that fulfills the requirements established through the output's locking script.

Specifically, to determine if the unlocking script is valid, the nodes that receive these transactions append the locking script to the unlocking script, execute the resulting construction and determine whether the execution was successful. An execution is deemed successful if it did not raise any errors and resulted in a structure that represents a boolean value of true.

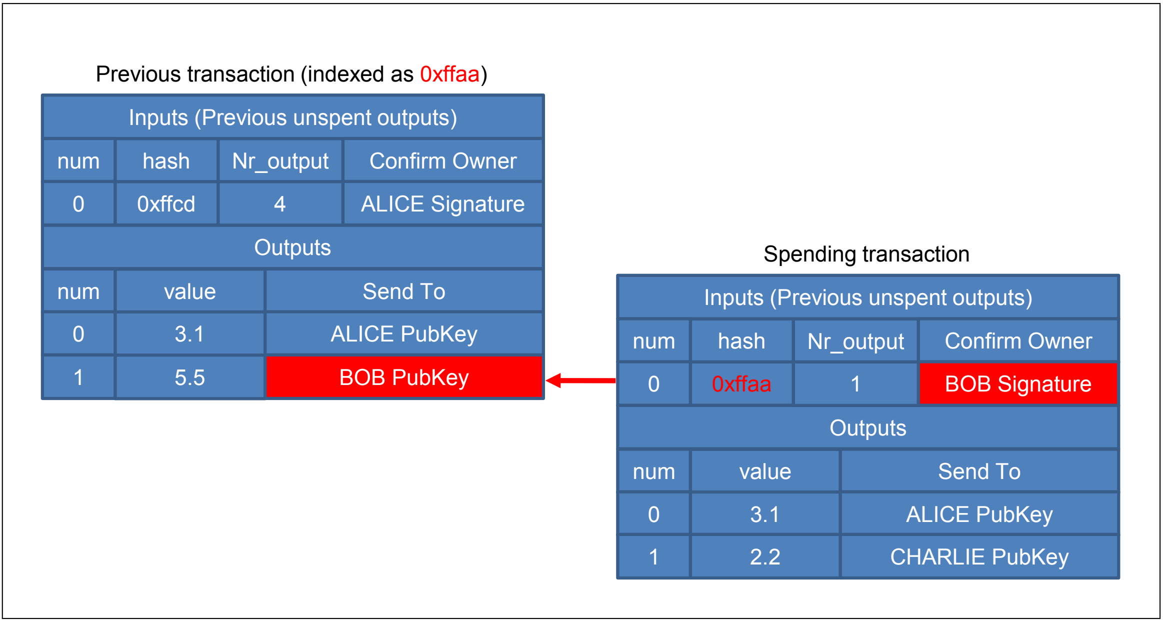
This system provides enough freedom that it is possible to create a locking script for which there does not exist any valid unlocking script. This can be done in purpose and there even is a specific operator that automatically flags the locking script as invalid. In practice this is used to store information in the blockchain, such that there is verifiable proof that said information was available to the sender on certain date. However, locking scripts that can not be unlocked can also be created by mistake.

This causes issues at an individual and a collective level. On the one hand, a person that locked a transaction behind a locking script that can not be unlocked simply loses that money. There is no possible way of accessing funds that have been locked in this manner. On the other hand, these unspent transactions accumulate in the pool of UTXOs, taking up memory on all the nodes that have received it. Given that these outputs can not be spent, this memory can not be freed.

This thesis' purpose is twofold. We first propose a formalization of Script in order to standardize the study into the different problems associated with the language and then analyze the problem of determining whether an output is unspendable by examining its associated locking script. Specifically, we study if it is feasible to construct an efficient algorithm for detecting unspendable transaction outputs that could be deployed both in electronic wallets, that could warn users when they are locking funds behind malformed scripts, and in the nodes of the network, that could free up some of their memory by deleting unspendable transaction outputs from their pool of UTXOs.

2. HOW SCRIPT WORKS

At the core of Bitcoin we have transactions. Simply put, Bitcoin transactions specify which coins are spent and to whom they are transferred. On a technological level, each Bitcoin transaction can have multiple inputs, each of which is an output of a previous transaction. Conceptually, in order for a transaction to be accepted, each input that is used requires a digital signature that corresponds to the public key specified by the transaction where this input was generated. We depict this dependence graphically in Figure 2.1. Additionally, the list of all transactions (grouped into blocks) is kept by a peer-to-peer network "running" Bitcoin, so that we are able to check if the transaction inputs have already been spent. The only transactions that differ from this template are the coinbase transactions in which new "coins" are minted, and that have no inputs. These appear once per block, and only specify who can spend the newly created "coins".



Inputs (Previous unspent outputs)			
num	hash	Nr_output	Confirm Owner
0	0xffcd	4	ALICE Signature

Outputs		
num	value	Send To
0	3.1	ALICE PubKey
1	5.5	BOB PubKey

Inputs (Previous unspent outputs)			
num	hash	Nr_output	Confirm Owner
0	0xffaa	1	BOB Signature

Outputs		
num	value	Send To
0	3.1	ALICE PubKey
1	2.2	CHARLIE PubKey

Figure 2.1. Input to one transaction is the output of a previous transaction. Here Bob confirms with his digital signature that he is the owner of the private key corresponding to the public key used when specifying the recipient of the funds in the previous transaction. Transactions reference each other via their hash (i.e. 0xffaa in this case).

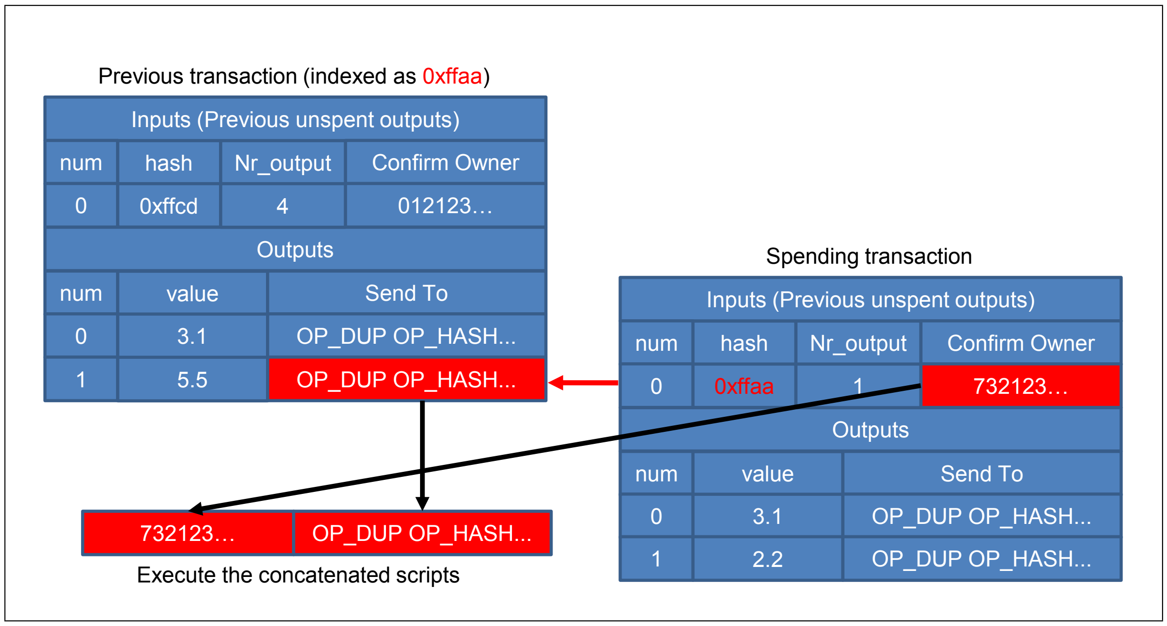


Figure 2.2. Interaction between the locking and unlocking scripts.

In reality, the process of signing a transaction input is somewhat more complicated and depends on Bitcoin's scripting language, Script. More precisely, each transaction output specifies a part of a script written in this language, called the locking script. In order to spend this output, the transaction using it as an input must provide another sequence of Script commands, called the unlocking script, such that the script obtained by concatenating the two executes correctly. Given that stack-based languages operate "in-reverse", the two scripts are also concatenated in this order, namely, the locking script is appended to the unlocking script spending it. We depict this process graphically in Figure 2.2.

When Script was conceived, the process of executing the combination of both scripts was done by literally concatenating them together and executing the resulting Script. However, for safety concerns this procedure has been modified since then, such that the execution of the concatenation is performed by first executing the unlocking script while checking that it was properly constructed, and second executing the locking script with the final state of the execution of the unlocking script as its initial state (*Script implementation: security improvements*, 2010). This distinction is irrelevant in the analysis of the most commonly used locking scripts, however, it will become important in the later sections of this document when laying out proofs about the inner workings of Script.

Script (*Script specification*, 2021) is a simple stack-based language which allows to push elements to a stack, and manipulate its content using basic arithmetic, logical operations, if-else statements, and cryptographic primitives such as hashing and signature verification. Script is designed to be loop-free and is therefore not Turing-complete, which allows it to be more secure, and can be efficiently implemented. Despite of this, Script still allows to express an array of complicated conditions, giving rise to what is known as "smart contracts", which are nothing more than non trivial Script programs that specify how an output of a previous transaction can be unlocked.

In what follows, we briefly recap the main commands of Script, and explain the problems we study in this setting.

At the core of Script is the use of a stack. Simply put, the stack is used in order to store some elements, perform simple operations on them, and later compare them for equality. Instructions of Script can be grouped as follows:

- Data (256 bit numbers), which are pushed onto the stack when encountered.
- Stack operations (push, pop, . . .).
- Logical operations (and, or, . . .).
- Arithmetical operations on numbers.
- Cryptographic primitives (hashing and signature verification).

We show basic Script commands by illustrating how a basic transaction for transferring funds from one address to another works. This is called pay to public-key hash (or P2PKH for short) script, and is one of the simplest meaningful scripts that can be expressed[1]. As stated previously, each input to a transaction has an associated locking script. In the case

[1]We use this for simplicity. Pay to script hash is by far the currently most used type of script, often encapsulating P2PKH.

of P2PKH this locking script has the form

```
OP_DUP OP_HASH160 pubKeyData OP_EQUALVERIFY OP_CHECKSIG
```

To unlock this output, we need to provide a set of Script commands, which, when executed prior to executing the locking script, result in a non empty stack with a nonzero element at the top. A correct unlocking script in this case would be

```
signature pubKeyData
```

Intuitively, the unlocking script provides us the signature `signature` and the public key `pubKeyData` corresponding to this signature, and then the locking script checks its validity. Namely, the locking script will duplicate the top item on the stack (via `OP_DUP`), hash this element (with a combination of ripeMD160 and SHA-256 hash functions), push an item onto the stack, push the public key data onto the stack, check that the provided public key and the one specified in the script match, and finally verify the signature.

This example already shows how locking scripts can specify complex conditions. While it is easy to construct the unlocking script for the locking script above, provided we have the required private key needed to produce the signature, this is not always the case. For instance, the locking script:

```
OP_DUP OP_ADD 7 OP_EQUALVERIFY
```

can never be unlocked since it is asking for a natural number n such that $2n = 7$. This can of course be very problematic if a lot of funds are locked behind such a locking script. A good Bitcoin wallet should prohibit such transactions, or at least warn the user that their outputs will become unspendable due to the locking script condition being proposed. This is known as the unlockability problem, and is the main problem we study in this paper. Formally, the **unlockability** problem for Script can be defined as follows:

PROBLEM:	UNLOCKABILITY OF SCRIPT
INPUT:	A locking script l.
QUESTION:	Is there an unlocking script u such that when l is executed after executing u (starting with an empty stack) we throw no errors and end with a non zero element on top of the stack?

In this paper we study this problem from an efficiency perspective. More precisely, we are asking whether there is an algorithm for answering this question that a wallet could deploy once a locking script has been provided by the user.

3. FORMALIZING SCRIPT

In this section, we develop a formalization for Script that allows us to study the computational complexity of some problems related to the evaluation or unlocking of scripts. Besides, this formalization enables us to fix the notation used throughout the paper.

Given that Script is a stack-based language, we begin with a formal definition of the stacks that are used by this language. We then focus on the operators of Script, defining their semantics in terms of stack operations.

3.1. The stacks in Script

For an arbitrary nonempty set M, we denote the concatenation of two elements $A, B \in M$ as $A \cdot B$, and naturally extend this notion to any finite number of elements. By M^* we denote all finite concatenations of elements of M, including the empty string ε, and with M^+ we denote M^* without ε. A stack over M is any element $A_0 \cdot A_1 \cdot \ldots \cdot A_k \in M^*$. Intuitively, this string over M represents a stack containing A_0 as the top element, A_1 as the element below the top one, etc. Notice that we allow the empty stack, which is denoted by the empty string ε.

Script has two stacks at its disposal: the main stack, denoted by φ_M, and an alternate stack, denoted by φ_A, that can be accessed by a few of the operators. Hence, the stacks of Script shall be denoted as the pair (φ_M, φ_A). To manipulate these stacks, we use functions top and tail, defined as follows: $\mathsf{top} : M^+ \to M$ is used to return the top of the stack, that is $\mathsf{top}(A_0 \cdot A_1 \cdot \ldots \cdot A_k) = A_0$, while $\mathsf{tail} : M^+ \to M^*$ is used to return the stack below the first element, that is, $\mathsf{tail}(A_0 \cdot A_1 \cdot \ldots \cdot A_k) = A_1 \cdot \ldots \cdot A_k$. Notice that the result of tail can be the empty stack ε.

3.2. Script operators

For simplicity, we assume that data items in Script come from the set $\mathbb{Z}$.[1] This is a natural generalization when studying the complexity of the unlockability problem for Script; in fact, it will allow us to establish a tight connection between this problem and integer linear programming (Schrijver, 1998).

Script has a precisely defined set of allowed operations (*Script specification*, 2021), which can be thought of as transforming the two stacks, or giving an error that terminates the execution. We denote the set of Script operators with O. Formally, every Script command f, apart from those used for flow control (see Section 3.2.3), can be understood as a function which takes the main and the alternate stack as its inputs, and transforms them in some way, or produces an error (denoted by $\square$):

$$f : (\mathbb{Z}^* \times \mathbb{Z}^*) \cup \{\square\} \to (\mathbb{Z}^* \times \mathbb{Z}^*) \cup \{\square\}. \tag{3.1}$$

Thus, scripts–as functions–can be composed, which naturally allows us to define the semantics of a sequence of operators. In particular, to handle errors, we impose the restriction that all of Script operators return an error when the input is an error itself, that is, $f(\square) = \square$.

With this notation at hand, we define how each operator $f \in O$ works. We start by introducing in Section 3.2.1 a group of basic operators, and defining how a sequence of them is executed. Then we describe in Section 3.2.2 how the operators associated with cryptographic primitives work. Finally, we introduce in Section 3.2.3 the *flow control* operators and the control stack, which determine when an operator should or should not be executed. A summary of the operators used in this paper, without including the control flow operators, is given in Table 3.1.

[1]In other words, we assume that each binary string encodes an integer.

3.2.1. Basic operators in Script

The most basic operation in Script is pushing data onto the (main) stack, which is achieved using a multitude of different operators (see e.g. the section on "Constants" in (*Script specification*, 2021)). In order to simplify this process, we combine all of these methods of pushing data through the $\mathsf{OP\_PUSH}_C$ operator, which pushes the value C onto the main stack. In terms of our generic description of Script commands (3.1), the semantics of this operation is defined as follows:

$$\mathsf{OP\_PUSH}_C(\varphi_M, \varphi_A) = (C \cdot \varphi_M, \varphi_A).$$

That is, if the operator receives as input a pair of valid stacks φ_M and φ_A, then it puts C on top of φ_M. Moreover, as already mentioned, we assume that $\mathsf{OP\_PUSH}_C(\Box) = \Box$.

Similarly, to pop the top of the stack, we can use $\mathsf{OP\_DROP}$, and to duplicate the top element of the stack, $\mathsf{OP\_DUP}$. Both of these operators require that the main stack φ_M contains at least one element (i.e. $|\varphi_M| \geq 1$), otherwise they return an error. In the case of a nonempty stack, their behaviour is defined as:

$$\begin{aligned}\mathsf{OP\_DROP}(\varphi_M, \varphi_A) &= (\mathsf{tail}(\varphi_M), \varphi_A)\\ \mathsf{OP\_DUP}(\varphi_M, \varphi_A) &= (\mathsf{top}(\varphi_M) \cdot \varphi_M, \varphi_A)\end{aligned}$$

The alternate stack in Bitcoin can be accessed in a very limited number of ways: we can only move the top element from the main stack onto it by means of the operator $\mathsf{OP\_TOALTSTACK}$, and move the top element of the alternate stack onto the main stack by means of the operator $\mathsf{OP\_FROMALTSTACK}$. Formally,

$$\begin{aligned}\mathsf{OP\_TOALTSTACK}(\varphi_M, \varphi_A) &= (\mathsf{tail}(\varphi_M), \mathsf{top}(\varphi_M) \cdot \varphi_A) \qquad \text{if } |\varphi_M| \geq 1\\ \mathsf{OP\_FROMALTSTACK}(\varphi_M, \varphi_A) &= (\mathsf{top}(\varphi_A) \cdot \varphi_M, \mathsf{tail}(\varphi_A)) \qquad \text{if } |\varphi_A| \geq 1\end{aligned}$$

Table 3.1. Semantics of Script commands. We assume that $\varphi_M = A_0{\cdot}A_1{\cdot}\ldots{\cdot}A_k$ whenever $|\varphi_M| > 0$. The condition column states the requirement that needs to be met for each operator not to return an error. Formally, if the condition for operator f is not met by (φ_M, φ_A), then $f(\varphi_M, \varphi_A) = \square$. The function hash corresponds to using SHA-256 and RIPEMD-160 hashing algorithms in succession. The function chksig corresponds to the verification algorithm of the ECDSA protocol for the string comprised of the transaction information, the first input as the public key and the second input as the signature.

Operator	Condition	Semantics
OP_PUSH$_C$	none	OP_PUSH$_C(\varphi_M, \varphi_A) = (C \cdot \varphi_M, \varphi_A)$
OP_DROP	$\lvert\varphi_M\rvert \geq 1$	OP_DROP$(\varphi_M, \varphi_A) = (\mathsf{tail}(\varphi_M), \varphi_A)$
OP_DUP	$\lvert\varphi_M\rvert \geq 1$	OP_DUP$(\varphi_M, \varphi_A) = (\mathsf{top}(\varphi_M) \cdot \varphi_M, \varphi_A)$
OP_VERIFY	$\lvert\varphi_M\rvert \geq 1 \land \mathsf{top}(\varphi_M) \neq 0$	OP_VERIFY$(\varphi_M, \varphi_A) = (\mathsf{tail}(\varphi_M), \varphi_A)$
OP_IFDUP	$\lvert\varphi_M\rvert \geq 1$	OP_IFDUP$(\varphi_M, \varphi_A) = \begin{cases} (\varphi_M, \varphi_A) & \text{if } \mathsf{top}(\varphi_M) = 0 \\ (\mathsf{top}(\varphi_M) \cdot \varphi_M, \varphi_A) & \text{if } \mathsf{top}(\varphi_M) \neq 0 \end{cases}$
OP_NIP	$\lvert\varphi_M\rvert \geq 2$	OP_NIP$(\varphi_M, \varphi_A) = (A_0{\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_OVER	$\lvert\varphi_M\rvert \geq 2$	OP_OVER$(\varphi_M, \varphi_A) = (A_1 \cdot \varphi_M, \varphi_A)$
OP_ROT	$\lvert\varphi_M\rvert \geq 3$	OP_ROT$(\varphi_M, \varphi_A) = (A_2{\cdot}A_0{\cdot}A_1{\cdot}A_3{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_SWAP	$\lvert\varphi_M\rvert \geq 2$	OP_SWAP$(\varphi_M, \varphi_A) = (A_1{\cdot}A_0{\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_TUCK	$\lvert\varphi_M\rvert \geq 2$	OP_TUCK$(\varphi_M, \varphi_A) = (A_0{\cdot}A_1{\cdot}A_0{\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_2DROP	$\lvert\varphi_M\rvert \geq 2$	OP_2DROP$(\varphi_M, \varphi_A) = (\mathsf{tail}(\mathsf{tail}(\varphi_M)), \varphi_A)$
OP_2DUP	$\lvert\varphi_M\rvert \geq 2$	OP_2DUP$(\varphi_M, \varphi_A) = (A_0 \cdot A_1 \cdot \varphi_M, \varphi_A)$
OP_3DUP	$\lvert\varphi_M\rvert \geq 3$	OP_2DUP$(\varphi_M, \varphi_A) = (A_0 \cdot A_1 \cdot A_2 \cdot \varphi_M, \varphi_A)$
OP_2OVER	$\lvert\varphi_M\rvert \geq 4$	OP_2OVER$(\varphi_M, \varphi_A) = (A_2 \cdot A_3 \cdot \varphi_M, \varphi_A)$
OP_2ROT	$\lvert\varphi_M\rvert \geq 6$	OP_2ROT$(\varphi_M, \varphi_A) = (A_4{\cdot}A_5{\cdot}A_0{\cdot}A_1{\cdot}A_2{\cdot}A_3{\cdot}A_6{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_2SWAP	$\lvert\varphi_M\rvert \geq 4$	OP_2SWAP$(\varphi_M, \varphi_A) = (A_2{\cdot}A_3{\cdot}A_0{\cdot}A_1{\cdot}A_4{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_ADD	$\lvert\varphi_M\rvert \geq 2$	OP_ADD$(\varphi_M, \varphi_A) = ((A_0 + A_1) \cdot A_2 \cdots A_k, \varphi_A)$
OP_SUB	$\lvert\varphi_M\rvert \geq 2$	OP_SUB$(\varphi_M, \varphi_A) = ((A_1 - A_0) \cdot A_2 \cdots A_k, \varphi_A)$
OP_EQUAL	$\lvert\varphi_M\rvert \geq 2$	OP_EQUAL$(\varphi_M, \varphi_A) = \begin{cases} (1{\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A) & \text{if } A_0 = A_1 \\ (0{\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A) & \text{if } A_0 \neq A_1 \end{cases}$
OP_EQUALVERIFY	$\lvert\varphi_M\rvert \geq 2 \land A_0 = A_1$	OP_EQUALVERIFY$(\varphi_M, \varphi_A) = (A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_PICK	$\lvert\varphi_M\rvert \geq 1 \land A_0 \geq 0 \land \lvert\varphi_M\rvert \geq A_0 + 2$	OP_PICK$(\varphi_M, \varphi_A) = (A_{A_0+1}{\cdot}A_1{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_ROLL	$\lvert\varphi_M\rvert \geq 1 \land A_0 \geq 0 \land \lvert\varphi_M\rvert \geq A_0 + 2$	OP_ROLL$(\varphi_M, \varphi_A) = (A_{A_0+1}{\cdot}A_1{\cdot}\ldots{\cdot}A_{A_0}{\cdot}A_{A_0+2}{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_DEPTH	none	OP_DEPTH$(\varphi_M, \varphi_A) = (\lvert\varphi_M\rvert \cdot \varphi_M, \varphi_A)$
OP_HASH160	$\lvert\varphi_M\rvert \geq 1$	OP_HASH160$(\varphi_M, \varphi_A) = (\mathsf{hash}(A_0) \cdot \mathsf{tail}(\varphi_M), \varphi_A)$
OP_CHECKSIG	$\lvert\varphi_M\rvert \geq 2$	OP_CHECKSIG$(\varphi_M, \varphi_A) = (\mathsf{chksig}(A_0, A_1){\cdot}A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$
OP_CHECKSIGVERIFY	$\lvert\varphi_M\rvert \geq 2 \land \mathsf{chksig}(A_0, A_1) = 1$	OP_CHECKSIGVERIFY$(\varphi_M, \varphi_A) = (A_2{\cdot}\ldots{\cdot}A_k, \varphi_A)$

In Table 3.1, we provide the list of remaning basic operators and their semantics (except for the last three rows of this table that include the operators defined in the following section).

As Script operators are understood as functions, the semantics of a script $f_1 \cdot f_2 \cdot \ldots \cdot f_n$ consisting of a sequence of operators is defined as the composition of these functions. Moreover, a script is *executed successfully* over a stack φ if upon executing all of its commands with φ as the initial main stack, we are left with a nonempty main stack containing a nonzero element at the top. Formally, a script $f_1 \cdot f_2 \cdot \ldots \cdot f_n$ is executed successfully over a stack φ if $(f_n \circ \ldots \circ f_2 \circ f_1)(\varphi, \varepsilon) = (\varphi_M, \varphi_A)$ with $\varphi_M \neq \varepsilon$ and $\mathsf{top}(\varphi_M) \neq 0$. It is important to notice that the possibility of starting with a nonempty main stack is included because of the way in which the unlocking and the locking script are executed in succession, which does not exactly match the execution of the concatenation of both scripts. More formally, when we have a locking script l, and an unlocking script u, we require that: (i) $u(\varepsilon, \varepsilon) = (\varphi_M^u, \varphi_A)$ (with no errors thrown in between); and (ii) $l(\varphi_M^u, \varepsilon)$ executes successfully.

EXAMPLE 3.1. Consider the script

$$\mathsf{OP\_PUSH}_5 \cdot \mathsf{OP\_PUSH}_{-3} \cdot \mathsf{OP\_ADD}$$

We execute this script starting with empty main and alternate stacks. We first push number 5 onto the main stack, and then push -3 at the top of the main stack. The last operator is OP_ADD, which according to the semantics defined in Table 3.1 generates a main stack containing only the number $2 = -3 + 5$. Hence, this script is executed successfully, since upon its completion, we have a nonempty main stack with a nonzero top element. ■

3.2.2. Operators for executing cryptographic primitives

An important part of Script resides in the execution of cryptographic primitives, since in most of the popular locking scripts these functions are used to verify the identity of the recipient of a transaction. While there are several cryptographic operators in Script, we only consider the most prevalent of them: OP_HASH160, which hashes an input, and OP_CHECKSIG and OP_CHECKSIGVERIFY, which are used to check a digital signature. The analysis for all the other cryptographic primitives is identical to these cases. Let us first describe the primitives hash and chksig underlying these operators.

The operator $\mathsf{hash} : \mathbb{Z} \to \mathbb{Z}$ is a function whose value is the result of hashing the input by using SHA-256 and then RIPEMD-160. Moreover, chksig is defined as follows. In a digital signature protocol, the signature verification function receives as input a public key, a string and a signature. Then such a function determines whether the signature was obtained by executing the signing function over the string and the private key corresponding to the public key. However, the signature verification operators in Script only receive as input a public key and a signature. This is because the purpose of these operators is just to determine if the recipient has access to a certain private key. Therefore, the string that is signed is a predetermined construction that is obtained by executing certain transformations over a combination of the transaction's inputs, outputs and locking scripts. Thus, given that for the purposes of each script the document that is signed is a constant, we will disregard this element in our analysis, and we define $\mathsf{chksig} : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$ as a function that takes only two inputs: a string representing a public key and a string representing a digital signature. The value of $\mathsf{chksig}(n_1, n_2)$ is defined as 1 if n_2 is a valid signature for the document constructed from the transaction (as described previously) and the public key n_1, and the value of $\mathsf{chksig}(n_1, n_2)$ is 0 otherwise. The digital signature protocol that is used to generate and verify signatures is ECSDA with the secp256k1 elliptic curve (*Script specification*, 2021).

Finally, we provide the formal definitions of the hashing and signature checking operators. For the hashing operator, the main stack φ_M is required to contain at least one element (i.e. $|\varphi_M| \geq 1$), whereas both signature checking operators require the main stack to have at least two elements (i.e. $|\varphi_M| \geq 2$). If these conditions are not satisfied, then these operators return an error $\Box$. In the definition, we assume that $\varphi_M = A_0 \cdot A_1 \cdot \ldots \cdot A_k$:

$$\begin{aligned}
\mathsf{OP\_HASH160}(\varphi_M, \varphi_A) &= (\mathsf{hash}(A_0) \cdot \mathsf{tail}(\varphi_M), \varphi_A) \\
\mathsf{OP\_CHECKSIG}(\varphi_M, \varphi_A) &= (\mathsf{chksig}(A_0, A_1) \cdot A_2 \cdot \ldots \cdot A_k, \varphi_A) \\
\mathsf{OP\_CHECKSIGVERIFY}(\varphi_M, \varphi_A) &= \begin{cases} (A_2 \cdot \ldots \cdot A_k, \varphi_A) & \text{if } \mathsf{chksig}(A_0, A_1) = 1 \\ \Box & \text{if } \mathsf{chksig}(A_0, A_1) = 0 \end{cases}
\end{aligned}$$

3.2.3. Operators for flow control

The final piece we need to add are the flow control operators of the form if-then-else. While conceptually simple, formalizing this concept needs an extra piece of notation, since in a block of the form

```
if <some commands> else <some other commands> end_if,
```

we need to determine the correct block of commands to be executed while reading the script from left to right. We achieve this by including an extra stack, called the *control stack*, which is denoted by φ_I. Intuitively, the control stack allows us to decide whether an operator is outside an if-then-else block, in which case it is executed as usual, or whether it belongs to some of the commands within this if-then-else block, in which case we need to make sure that only the operators from the appropriate block are being executed.

The control stack φ_I consist of zeros and ones exclusively, that is, $\varphi_I \in \{0,1\}^*$. A control stack φ_I is said to represent an *execution state* if $\varphi_I \in \{1\}^*$, which indicates that the command we are seeing has to be executed. In this case, either the control stack will consist of just 1s and the command will be within the if-then-else block, or the control stack will be empty, which indicates that we are outside the if-then-else portion of the script. Formally, the semantics of all the commands defined in the previous sections are extended so that these operators are executed only when the control stack is in an execution state. That is, for every Script operator $f \in O$, Equation (3.1) should be replaced by the following:

$$f : (\mathbb{Z}^* \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\square\} \to (\mathbb{Z}^* \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\square\}. \tag{3.2}$$

Hence, each operator takes as input three stacks: the main stack, the alternate stack and the control stack. The semantics of commands from Table 3.1 is then redefined so that there is a third input, φ_I, which is also the third output (φ_I is not changed by the operators in Table 3.1). Besides, the condition column in Table 3.1 is modified to include the fact that φ_I represents an execution state (that is, $\varphi_I \in \{1\}^*$). In particular, for each operator f in Table 3.1, if φ_I is not an execution state, then we have that $f(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M, \varphi_A, \varphi_I)$; namely, the command is not executed. For example, consider again the operation $\mathsf{OP\_PUSH}_C$ with $C \in \mathbb{Z}$. Its semantics, taking now into consideration the control stack, is defined as follows:

$$\mathsf{OP\_PUSH}_C(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (C \cdot \varphi_M, \varphi_A, \varphi_I),$$

whenever φ_I is an execution state. When φ_I is not an execution state, the semantics of this operator is defined as:

$$\mathsf{OP\_PUSH}_C(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (\varphi_M, \varphi_A, \varphi_I).$$

The flow control operators OP_IF, OP_ELSE, OP_ENDIF are the only ones that can modify the control stack. Next we explain how they interact with the main and alternate stacks, and also how they modify the control stack. In essence, these three commands come in tandem, and take the form:

OP_IF `<commands1>` OP_ELSE `<commands2>` OP_ENDIF.

Both `commands1` and `commands2` are sequences of Script commands, which can again contain if-then-else blocks. The objective of the control stack is to signal whether the operators `commands1` or `commands2` are to be executed, depending on whether the top value of the main stack upon reaching the OP_IF is true or false. This is achieved by pushing/popping the appropriate value to/from the control stack when either OP_IF or

OP_ELSE is reached, as to signal which block of commands will be executed. Recall that only a control stack in an execution state allows for a command to be executed, so we will use this property accordingly.

Intuitively, when reaching an OP_IF statement, we will store the truth value of the top of the main stack onto the control stack. If this was true (or nonzero in our notation), we will push 1 onto the control stack, thus making it be in an execution state. Then, upon reaching its corresponding OP_ELSE, we will replace the 1 at the top of the control stack with 0, making it not be in an execution state. This will allow us to skip all the commands until reaching the accompanying OP_ENDIF, which simply pops the top of the control stack. A similar process occurs when the top value of the main stack upon reaching OP_IF is false. Notice that if-then-else statements can be nested. However, in a syntactically correct script this is not an issue, as the control stack is populated and cleared as expected. Formally, the semantics of OP_IF is defined as follows:

$$\mathsf{OP\_IF}(\varphi_M, \varphi_A, \varphi_I) \ = \begin{cases} (\mathsf{tail}(\varphi_M), \varphi_A, 1 \cdot \varphi_I) & \text{if } |\varphi_M| \geq 1 \wedge \mathsf{top}(\varphi_M) \neq 0 \wedge \varphi_I \in \{1\}^* \\ (\mathsf{tail}(\varphi_M), \varphi_A, 0 \cdot \varphi_I) & \text{if } |\varphi_M| \geq 1 \wedge \mathsf{top}(\varphi_M) = 0 \wedge \varphi_I \in \{1\}^* \\ (\varphi_M, \varphi_A, 0 \cdot \varphi_I) & \text{if } |\varphi_I| \geq 1 \wedge \varphi_I \notin \{1\}^* \end{cases}$$

Morover, in any other case, $\mathsf{OP\_IF}(\varphi_M, \varphi_A, \varphi_I) = \Box$. For example, an error is returned if φ_M is an empty stack, as there is no stack element to ascertain the truth value. Thus, the definition of OP_IF states that three outcomes are possible upon reaching this operator, under the appropriate conditions not to produce an error: (1) If the top element of the main stack is different from 0 and we are in an execution state, then 1 is pushed onto the control stack, in order to signal that the `IF` part of the if-then-else block is to be executed. Besides, the main stack is popped. (2) If the top of the main stack is 0, and we are in an execution state, we push 0 onto the control stack (i.e. we do not execute the commands in the `IF` block, but rather in the `ELSE` block), and the main stack is popped.

(3) Finally, if we are not in an execution state, we push the value 0 to the control stack (in order to handle nested if-then-else blocks which should not be executed).

On the other hand, the $\mathsf{OP\_ELSE}$ operator simply has to signal whether the commands that follow it are to be executed or not, which is done by changing the top element of the control stack as follows:

$$\mathsf{OP\_ELSE}(\varphi_M, \varphi_A, \varphi_I) \;=\; \begin{cases} (\varphi_M, \varphi_A, 1\cdot\mathsf{tail}(\varphi_I)) & \text{if } |\varphi_I| \geq 1 \land \mathsf{top}(\varphi_I) = 0 \\ (\varphi_M, \varphi_A, 0\cdot\mathsf{tail}(\varphi_I)) & \text{if } |\varphi_I| \geq 1 \land \mathsf{top}(\varphi_I) = 1 \end{cases}$$

Moreover, if φ_I is empty, then the operator $\mathsf{OP\_ELSE}$ returns an error, which is formalized as $\mathsf{OP\_ELSE}(\varphi_M, \varphi_A, \varphi_I) = \square$. Notice that, same as in the case of $\mathsf{OP\_IF}$, we also push 1 or 0 to the control stack, in order to handle further nesting of if-then-else commands. Finally, each if-then-else block is required to be correctly closed via the $\mathsf{OP\_ENDIF}$ operator. To ensure this, we simply pop the top element of the control stack upon reaching this command:

$$\mathsf{OP\_ENDIF}(\varphi_M, \varphi_A, \varphi_I) \;=\; (\varphi_M, \varphi_A, \mathsf{tail}(\varphi_I))$$

Notice that as for the case of $\mathsf{OP\_ELSE}$, if φ_I is empty, then the operator $\mathsf{OP\_ENDIF}$ returns the error symbol $\square$.

It is important to notice that in adding these flow control operators to Script, we introduce more nuance into the definition of a sucessful execution. More specifically, we now say that a script is *executed successfully* over a stack φ if upon executing all of its operators with φ as our initial main stack, we are left not only with a nonempty main stack which contains a nonzero element at the top, but also with an empty control stack. Formally, a script $f_1\cdot f_2\cdot \ldots \cdot f_n$ is executed successfully over a stack φ if $(f_n \circ \cdots \circ f_2 \circ f_1)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varphi_I)$ with $\varphi_M \neq \varepsilon$, $\mathsf{top}(\varphi_M) \neq 0$ and $\varphi_I = \varepsilon$. Conceptually, this new condition requires flow control blocks to be properly structured in Script. In particular, a script that ends with a nonempty control stack has an unfinished if-then-else block, which indicates that it is not well constructed.

As we have explained previously, when executing a pair of an unlocking and a locking script, the process consists of executing the unlocking script over a trio of empty stacks, and then executing the locking script over the final main stack of the previous execution and a pair of empty stacks. However, if after the first execution we are left with a nonempty control stack signaling unfinished if-then-else blocks, then the locking script is simply given an error and the combined execution ends unsuccessfully (see next section for a formal definition of the unlockability of Script problem). Therefore, when executing a pair of an unlocking and a locking script, both executions have to contain properly structured if-then-else blocks.

EXAMPLE 3.2. To illustrate how flow control operators work, consider the following script:

```
OP_PUSH$_0$
OP_IF
    OP_DUP
OP_ELSE
    OP_PUSH$_3$
    OP_IF
        OP_PUSH$_7$
    OP_ELSE
        OP_DUP
    OP_ENDIF
OP_ENDIF
```

Recall that a script consists of a concatenation of operators, but we have represented this vertically and indented to better illustrate how flow control blocks are nested. When executing this script, value 0 is pushed onto the main stack first (notice that at the beginning the control stack is empty, and we are thus in an execution state), so we have that:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (0, \varepsilon, \varepsilon)$$

Following this, an OP_IF statement is encountered, and the control stack is updated accordingly. In this case, given that we have value 0 on top of the main stack, 0 is pushed onto the control stack, and the main stack is emptied:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (\varepsilon, \varepsilon, 0)$$

Since we are not in an execution state, the OP_DUP command is ignored, and we continue with the OP_ELSE operator. Given that the top of the control stack is equal to 0, we replace this value with 1, signaling that the next block of commands is to be executed:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (\varepsilon, \varepsilon, 1)$$

The operator OP_PUSH$_3$ is then executed, so the value 3 is pushed onto the main stack:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (3, \varepsilon, 1)$$

Afterwards, another OP_IF operator is reached. Since we are in an execution state and value 3 is different from 0, value 3 is popped from the main stack, and 1 is pushed onto the control stack:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (\varepsilon, \varepsilon, 1{\cdot}1)$$

This means that in the next step we push value 7 onto the main stack, when executing the operator OP_PUSH$_7$:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (7, \varepsilon, 1{\cdot}1)$$

The next operator is OP_ELSE, which switches the value 1 on top of the control stack to 0, which in turn means that we are no longer in an execution state:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (7, \varepsilon, 0{\cdot}1)$$

Thus, we need to ignore the following OP_DUP operator, and we need to continue with the OP_ENDIF command. Here the top of the control stack is popped:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (7, \varepsilon, 1)$$

Finally, the last command OP_ENDIF is executed, leaving the control stack empty, and finishing with value 7 on the main stack:

$$(\varphi_M, \varphi_A, \varphi_I) \quad = \quad (7, \varepsilon, \varepsilon)$$

Thus, the script results in a successful execution. ■

4. COMPLEXITY OF SCRIPT

In this section we will focus on analyzing the computational cost of working with Script. In order to draw a complete picture, we start by formally proving the well-known fact that simply running a pair of unlocking and locking script can be done in polynomial time. We then move on to study the unlockability problem. First, in Subsection 4.2 we show that the problem of determining whether a locking script can be successfully unlocked is NP-hard. Next, in Subsection 4.3 we discuss what is the main technical result of this thesis, i.e. the fact that the unlockability problem can be solved in NP. We remark that these results hold both in the case when we have all the necessary private keys and hash pre-images used in the locking script, and when they are not available. We discuss the possible implication of these results further below.

4.1. Evaluating Script

While the main objective of this thesis is studying unlockability of Script, we will start by proving the folklore result saying that any pair of scripts can be evaluated in polynomial time. We do this for two reasons: (i) because this result will be needed to prove that unlockability is solvable in time NP; and (ii) in order to show that our formalization of Script conforms with the intuitive understanding of the language. Formally, the evaluation problem for Script can be defined as follows:

PROBLEM:	EVALUATION OF SCRIPT
INPUT:	A locking script l and an unlocking script u.
QUESTION:	Are the following two executions successful:
	(i) $u(\varepsilon, \varepsilon, \varepsilon) = (\varphi^u_M, \varphi^u_A, \varepsilon)$; and
	(ii) $l(\varphi^u_M, \varepsilon, \varepsilon)$?

As explained in Section 2, the unlocking script u, and the locking script l are executed separately in order to strengthen the security of Script. That is, we first run the unlocking script with a triple of empty stacks. Provided that this execution is successful, the content of the main stack at the end of this execution, denoted φ^u_M, is transferred to the locking script, whose execution starts with an empty alternate stack and an empty control stack. As per the current specification (*Script specification*, 2021), and implementation (*Script implementation: security improvements*, 2010) of Script, the alternate stack content is erased when starting the execution of the locking script. Recall from Section 3 the fact that a successful execution also requires the script to start and finish with an empty control stack in order to validate that flow control commands are properly nested and completed within both locking and unlocking script.

We can now prove the following:

THEOREM 4.1. *The problem* EVALUATION OF SCRIPT *can be solved in* PTIME.

It is straightforward to prove this result. In Subsection 5.2 we will prove a slight generalization of this result from which Theorem 4.1 readily follows.

4.2. Unlockability of Script is hard

In this section we turn to the unlockability problem for Script, and show a lower bound on its complexity, namely, that the problem is NP-hard. We remind the reader that in the unlockability problem, as defined in Section 2, we receive a locking script l as an input, and need to determine whether there exists an unlocking script u, such that l and u, when given as inputs to the EVALUATION OF SCRIPT problem as defined in subsection 4.1 result in a positive answer. In the remainder of this thesis we will say that the pair of scripts u and l executes successfully, whenever this is the case. We can now state the following theorem:

THEOREM 4.2. *The unlockability problem for Script is* NP*-hard.*

PROOF. In order to show this, we present a reduction from integer linear programming, which is one of the classical NP-complete problems (Garey & Johnson, 1979). In other words, for an arbitrary system of equations with integer coefficients and solution, we will show how to construct a script that will be unlockable if and only if the system of equations is solvable.

Let $A\vec{x} = \vec{b}$ be an arbitrary system of equations, with

$$A = \begin{bmatrix} a_{00} & \cdots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{m0} & \cdots & a_{mn} \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_0 \\ \vdots \\ b_m \end{bmatrix}$$

Conceptually, we aim to construct a locking script l that checks that each of the equations are fulfilled sequentially. If there exists a solution $\vec{c} = (c_0, \ldots, c_n)$ for the system, such that $A\vec{c} = \vec{b}$, then our locking script will be unlocked by an unlocking script that constructs a stack φ, with

$$\varphi = c_0 \cdot \ldots \cdot c_n$$

If we assume that the locking script l receives a stack $\varphi = d_0 \cdot d_1 \cdot \ldots \cdot d_p$, the process that it will follow will be:

(i) For each coefficient a_{ij} in the ith row of the matrix construct $a_{ij}d_j$.

(ii) Gradually add these terms into one value.

(iii) Compare the final value with b_i.

(iv) Repeat for the following row.

Now, in practice we will need to be strategic to represent this process in polynomial time as a script. We will start off with an empty script

$$S = \varepsilon$$

We will begin with row 0. For each coefficient a_{0j} in row 0 we will add the following operators to the script

(i) In order to retrieve the correct value in the stack we will add the following operators

$$S := \begin{cases} S \cdot \mathsf{OP\_PUSH}_j \cdot \mathsf{OP\_PICK} & \text{if} \quad j = 0 \\ S \cdot \mathsf{OP\_PUSH}_{j+1} \cdot \mathsf{OP\_PICK} & \text{if} \quad j > 0 \end{cases}$$

This difference stems from the fact that if $j > 0$ we will already have a partial result on top of the stack and will have to retrieve the value from further down in the stack.

(ii) We have to construct $a_{0j}d_j$ next, which we will have to accomplish by adding d_j with itself in a smart way. If we represent $|a_{0j}|$ as a binary number we will read it from right to left and for each bit b add the following operators to the script.

$$S := \begin{cases} S{\cdot}\mathsf{OP\_DUP}{\cdot}\mathsf{OP\_DUP}{\cdot}\mathsf{OP\_ADD} & \text{if} \quad b = 1 \\ S{\cdot}\mathsf{OP\_DUP}{\cdot}\mathsf{OP\_ADD} & \text{if} \quad b = 0 \end{cases}$$

The reasoning behind this is that it would be too inefficient to add d_j $|a_{0j}|$ times with itself. Therefore what we do instead is gradually constructing elements that correspond to d_j multiplied by increasing powers of two. The binary representation of $|a_{0j}|$ shows us the composition of the coefficient in terms of powers of two. For each of this powers 2^y that is present in the composition of $|a_{0j}|$ we leave $2^y d_j$ on the stack. If we add all of these values together we will end up with $|a_{0j}|d_j$, which brings us to the next step.

(iii) We need to add together all of the elements left on top of the stack by the previous step. Let a be the amount of 1s in the binary representation of $|a_{0j}|$, we will add $a - 1$ ‘add’ operators to the script as follows

$$S := S{\cdot}\mathsf{OP\_ADD}$$

If this is the first coefficient in the row we will move on to the next coefficient after this step. However, if we are analyzing any coefficient after the first, we need to add our result with the partial value that was already on top of the stack.

(iv) As previously stated, if the current coefficient is not the first, we need to add our result with the partial value that was already on top of the stack. To do so we will add the following operators to the script

$$S := \begin{cases} S{\cdot}\mathsf{OP\_ADD} & \text{if} \quad a_{0j} \geq 0 \\ S{\cdot}\mathsf{OP\_SUB} & \text{if} \quad a_{0j} < 0 \end{cases}$$

This distinction is made because we have constructed $|a_{0j}|d_j$ and in case the coefficient is negative we must subtract it from the partial result, not add them together.

After all of the coefficients in the first row have been considered, we need to compare the result with b_0 to check if the values in the stack fulfill the first equation. To accomplish this we add the following operators to the script

$$S := S \cdot \mathsf{OP\_PUSH}_{b_0} \cdot \mathsf{OP\_EQUALVERIFY}$$

This process will be repeated for each row, to make sure that all of the equations are fulfilled. From this construction it is pretty evident that S will be unlockable if and only if the system $A, \vec{b}$ is solvable, considering that the script simply adds the terms in each equation and compares the result to the right hand side of the equation.

Now, we just need to make sure that the transformation is polynomial in size. Crucially, there are no complex operations being executed, which means that both time and space complexity will go hand in hand. For this analysis we will assess each of the described steps individually. We will use the terms S_1, S_2, S_3, S_4 to refer to the sections of the script that each step contributes and the function $\|S\|$ to refer to the total size of a script.

(i) The first step will be executed once per coefficient. Given that it adds a push operator for the index of the coefficient and a pick operator, the size of this section's script will be constrained by

$$\|S_1\| \leq p_1(\log_2(m+1)),$$

for some polynomial p_1.

(ii) The second step will be executed once per bit of the coefficient and will add at most three operators to the script each time. Therefore, the size of this section's script will be constrained by

$$\|S_2\| \leq p_2(\log_2(|a_{\max}| + 1)),$$

where p_2 is some polynomial and $a_{\max}$ is the coefficient of maximum absolute value between the matrix and the vector in the system of equations.

(iii) The third step will be executed less than once for each bit in the coefficient and will add one operator to the script each time. Therefore, the size of this section's script will be constrained by

$$\|S_3\| \leq p_3(\log_2(|a_{\max}| + 1)),$$

where p_3 is some polynomial and $a_{\max}$ is the coefficient of maximum absolute value between the matrix and the vector in the system of equations.

(iv) The fourth and final step will be executed once per coefficient and will add one operator to the script. Therefore, the size of this section's script will be constrained by

$$\|S_4\| \leq C_1,$$

for some constant C_1.

This means that for each coefficient, the size of the script that will be constructed will be constrained by

$$\|S_1 \cdot S_2 \cdot S_3 \cdot S_4\| \leq p_c(\log_2(m + 1), \log_2(|a_{\max}| + 1)),$$

where p_c is some polynomial and $a_{\max}$ is the coefficient of maximum absolute value between the matrix and the vector in the system of equations. Now, the script constructed for each equation corresponds to the concatenation of all of the scripts constructed for the coefficients in the equation and a final pair of operators that push an element corresponding to the vector to the stack and checks that the results are equal. Let S_{eq} be the script constructed for an arbitrary equation,

$$\begin{aligned}\|S_{eq}\| &\leq (n+1)p_c(\log_2(m+1), \log_2(|a_{\max}|+1)) + \log_2(|a_{\max}|+1) + C_2\\ &\leq p_{eq}(\log_2(m+1), \log_2(|a_{\max}|+1), n),\end{aligned}$$

where p_{eq} is some polynomial, C_2 is some constant and $a_{\max}$ is the coefficient of maximum absolute value between the matrix and the vector in the system of equations. The complete script just corresponds to the scripts constructed for each equation. Thus,

$$\begin{aligned}\|S\| &\leq mp_{eq}(\log_2(m+1), \log_2(|a_{\max}|+1), n)\\ &\leq p_S(m, n, \log_2(|a_{\max}|+1)),\end{aligned}$$

for some polynomial p_S, which is evidently polynomial in the size of the system of equations. The complete algorithm that performs the described transformation can be found in appendix A. The rigurous formalization helps to explain how the edge cases are handled. For example, it shows how to construct the script when the first coefficient in an equation is negative and how to handle coefficients equal to zero. ■

We would first like to remark that simpler reductions are possible, for instance, from the $3SAT$ problem. We have opted for a reduction from integer linear programming, since we will also use this problem when showing NP-membership of unlockability, thus further illustrating how the two problems are closely linked.

4.3. NP-membership

Although the NP-hardness from the previous subsection might seem to suggest that unlockability is not a practically feasible problem, recent advancements in the area of SAT solvers actually tell us that for scripts of reasonable size, we can solve the problem efficiently, provided that we can also show it belongs to the class NP[1]. That is precisely the objective of this section.

In what is the main technical contribution of our work, we can establish the following:

THEOREM 4.3. *The unlockability problem for Script is in* NP.

The proof for this result will consist in showing that for each unlockable script there exists a certificate that is polynomially-sized compared to the script and that we can evaluate the pair of locking script and certificate in polynomial time. We will also need to prove that a script is unlockable if and only if there exists a certificate for which the evaluation of script and certificate is succesful.

More specifically, we will start by showing why we can not use an unlocking script as our certificate for unlockability. We will move on to propose another structure that will act as our certificate and prove that a script is unlockable if and only if it has a valid certificate. We will then prove that evaluating a locking script over a certificate can be done in polynomial time compared to the size of the pair. Lastly, we will prove that under reasonable restrictions there exists a valid certificate of polynomial size compared to the script for each unlockable locking script.

[1]This is, of course, under the assumption that we have at our disposal the required private keys, and the hash pre-images. We are not actually suggesting that breaking public key cryptography is feasible, although in the sense of complexity analysis it makes no difference. Our claim is that, should we know the required cryptographic data, the analysis of the validity of the locking script itself can be done with a SAT solver.

5. PROVING NP-MEMBERSHIP

Proving that the problem of unlockability is in NP would ideally just consist of showing that script evaluation is in PTIME and that any unlockable locking script has at least one valid solution of polynomial size. If this was the case, one possible non-deterministic algorithm that solves the problem would consist of guessing a script and checking whether it was capable of unlocking our locking script. However, it is pretty easy to construct an unlockable locking script for which any valid solution is exponential in the size of the script. Take the following example

$$S = \mathsf{OP\_PUSH}_x \cdot \mathsf{OP\_PICK} \cdot \mathsf{OP\_PUSH}_1 \tag{5.1}$$

If $x \geq 0$ it is evident that for a script to be able to unlock S it just needs to construct a stack with at least $x + 1$ elements. Considering that the size of this script is $\|S\| = \lceil \log_2(x + 1) \rceil + C$ for some constant $C \in \mathbb{Z}$, any stack with x or more elements will be exponential in the size of S. In addition, given that each operator pushes at most 3 elements to the stack, to construct a stack of exponential size compared to $\|S\|$ we would need a script with an exponential amount of operators.

5.1. An alternative representation

We propose an alternative representation of stacks that allows us to describe only the relevant elements in the stack (i.e. the elements that are utilized by any operator in the script). We add an additional symbol $\perp$ that represents anonymous elements. These are non-explicit elements whose values are not relevant for the execution of the analyzed locking script. In addition, we will group anonymous elements in blocks and only detail how many elements are in each of these blocks. We will denote the set containing the integers and the anonymous element as $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$. The following is a stack described in this manner.

$$\varphi = \perp^{h_0} \cdot A_0 \cdot \perp^{h_1} \cdot A_1 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}$$

In this example $A_0, \ldots, A_{k-1} \in \mathbb{Z}$ and $h_0, \ldots, h_k \in \mathbb{N}$. Note that for consistency we demand the existence of an anonymous block between each pair of integers, even if its size is 0. Let us come back to the example in equation 5.1. With this new representation we can describe a valid unlocking stack $\varphi_M \in \mathbb{Z}_\perp^*$ much more succintly, as

$$\varphi_M = \perp^x \cdot 1 \cdot \perp^0$$

Next we extend the definitions of the functions over stacks $\mathsf{top}, \mathsf{tail}$ to work over this representation of stacks with anonymous elements, in the natural way. Namely, let $\varphi = \perp^{h_0} \cdot A_0 \cdot \perp^{h_1} \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$, then

$$\mathsf{top}_\perp(\varphi) = \begin{cases} \perp & \text{if} \quad h_0 \geq 1 \\ A_0 & \text{if} \quad h_0 = 0 \end{cases}$$

$$\mathsf{tail}_\perp(\varphi) = \begin{cases} \perp^{h_0 - 1} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} & \text{if} \quad h_0 \geq 1 \\ \perp^{h_1} \cdot A_1 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} & \text{if} \quad h_0 = 0 \end{cases}$$

As can be easily seen, we introduce the notation $\mathsf{top}_\perp, \mathsf{tail}_\perp$ to explicitly state if the functions are being applied to a stack with natural or compressed notation.

By utilizing the concept of anonymous elements we can prove that unlockability is in NP as described at the beginning of this section. The process consists of the following three steps:

(i) Prove that a locking script is unlockable if and only if there exists a valid unlocking stack with anonymous elements for it,

(ii) show that evaluating the execution of a script with a starting stack with anonymous elements is in PTIME and

(iii) prove that any unlockable locking script has at least one valid unlocking stack with anonymous elements of polynomial size in this representation.

The first step requires just a simple analysis and the second step is a slight generalization of Theorem 4.1. However, the third step is the most technical. For our bounds we rely on bounds for the solution of integer linear programming (Papadimitriou, 1981), but in order to invoke this result we need to develop several intermediate results.

While Script operators remain the same, their semantics must be extended to account for stacks with anonymous elements. This is straightforward in most cases, but in some cases we need to impose certain conditions on the stacks to make sure that the operators do not manipulate anonymous elements. If the stacks do not fulfill these requirements, then they always evaluate to the error $\square$. The treatment is analogous for all script operators, so we defer the formal semantics to appendix B.

As an example, let us consider $\mathsf{OP\_ADD}$. We originally defined this operator's value as $\mathsf{OP\_ADD}(\varphi_M, \varphi_A) = ((A_0 + A_1) \cdot A_2 \cdots A_k, \varphi_A)$, where $\varphi_M = A_0 \cdot \ldots \cdot A_k$. We also required the main stack to have at least two elements for the operator to execute successfully because otherwise we could not add the top elements together.

Now, we will incorporate the condition that the two first elements need to be integers. Evidently, if one or both of these elements was anonymous, the result of adding them together would be uncertain. Formally, let $\varphi_A \in \mathbb{Z}^*$ and $\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$, the conditions that we impose for the operator to not output an error are

$$|\varphi_M| \geq 2$$
$$h_0 = 0$$
$$h_1 = 0$$

Additionally, we slightly modified the definition of the operator to show how it works over the compact representation of stacks with anonymous elements. This is shown in the following equation

$$\mathsf{OP\_ADD}(\varphi_M, \varphi_A) = (\perp^0 \cdot (A_0 + A_1) \cdot \perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

Before moving on to the aforementioned proofs we will define several auxiliary functions that will be helpful in formalizing certain properties over the execution of scripts. These functions will represent the maximum element in a pair of stacks, the size of the biggest block of anonymous elements in a stack, the amount of integer elements in a stack and the maximum element pushed to the stack by a script, and will be denoted by the following names:

$$\mathsf{maxelem} : \mathbb{Z}_\perp^* \times \mathbb{Z}^* \to \mathbb{N}$$
$$\mathsf{maxgap} : \mathbb{Z}_\perp^* \to \mathbb{N}$$
$$\mathsf{elemnr} : \mathbb{Z}_\perp^* \to \mathbb{N}$$
$$\mathsf{maxpush} : O^* \to \mathbb{N}$$

Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}^*$ and $\psi = B_0 \cdot \ldots \cdot B_\ell \in \mathbb{Z}^*$. We define the values of these functions as the following expressions.

$$\begin{aligned}
\textsf{maxelem}(\varphi, \psi) &= \max_{A \in \{A_0, \ldots, A_{k-1}\} \cup \{B_0, \ldots, B_\ell\}} |A| \\
\textsf{maxgap}(\varphi) &= \max\{h_0, \ldots, h_k\} \\
\textsf{elemnr}(\varphi) &= k \\
\textsf{maxpush}(S) &= \max\{|C| \mid \textsf{OP\_PUSH}_C \in \{f_0, \ldots, f_n\}\}
\end{aligned}$$

As we can see, for both maxelem and maxpush there is an edge case for which they are not defined. Let $S' = g_0 \cdot \ldots \cdot g_m \in O^*$, such that $\{g_0, \ldots, g_m\} \cap \{\textsf{OP\_PUSH}_C \mid C \in \mathbb{Z}\} = \emptyset$, we define their values in these edge cases as

$$\begin{aligned}
\textsf{maxelem}(\perp^0, \varepsilon) &= 0 \\
\textsf{maxpush}(S') &= 0
\end{aligned}$$

Next, we will perform the first step of our process for proving that unlockability is in NP. Specifically, we will prove that a locking script is unlockable if and only if there exists a valid unlocking stack with anonymous elements for it. This sentiment is expressed in the following Lemma.

LEMMA 5.1. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an arbitrary script. S is unlockable if and only if there exists a stack $\varphi \in \mathbb{Z}_\perp^*$, such that*

$$(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varepsilon),$$

where

$$\begin{aligned}
|\varphi_M| &\geq 1 \\
\textit{top}_\perp(\varphi_M) &\notin \{0, \perp\}
\end{aligned}$$

The proof for this Lemma is explained in appendix C. Considering this equivalence, from this point onwards we will exclusively consider unlocking stacks instead of unlocking scripts. With this new development in mind we can reformulate the problem of script unlockability as follows.

PROBLEM:	STACK UNLOCKABILITY OF SCRIPT
INPUT:	A locking script l.
QUESTION:	Is there an unlocking stack φ such that the following execution is successful: $l(\varphi, \varepsilon, \varepsilon)$?

As previously stated, in what follows we will prove that this problem is in NP.

5.2. Script evaluation

This section focuses on proving that determining whether the evaluation of an unlocking script with a starting stack with anonymous elements is successful is in PTIME. Formally,

PROBLEM:	STACK EVALUATION OF SCRIPT
INPUT:	A locking script l and an unlocking stack φ.
QUESTION:	Is the execution following execution successful: $l(\varphi, \varepsilon, \varepsilon)$?

In order to set an upper bound on the complexity of this problem we will begin by constraining the value of the auxiliary functions introduced in section 5.1. Firstly, we will constrain these values for individual operators.

LEMMA 5.2. *Let* $f \in O$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ *and* $\varphi_I \in \{0,1\}^*$, *with* $f(\varphi_M, \varphi_A, \varphi_I) = (\varphi'_M, \varphi'_A, \varphi'_I)$. *Then,*

$$\mathit{maxgap}(\varphi'_M) \leq 2 \cdot \mathit{maxgap}(\varphi_M)$$
$$\mathit{elemnr}(\varphi'_M) \leq \mathit{elemnr}(\varphi_M) + 3$$

Additionally, let $f \notin \{\mathit{OP\_DEPTH}, \mathit{OP\_HASH160}\} \cup \{\mathit{OP\_PUSH}_C \mid C \in \mathbb{Z}\}$. *Then we have*

$$\mathit{maxelem}(\varphi'_M, \varphi'_A) \leq 2 \cdot \mathit{maxelem}(\varphi_M, \varphi_A) + 1$$

PROOF. These bounds are proved case by case in appendix D. ■

Based on these results we will also set upper bounds on the values of these functions over the execution of complete scripts. These bounds must take into account the *size* of the stacks, but in order to talk about the size of stacks, we need to settle on the way these stacks are encoded. For our complexity results we assume that they are always represented as arrays of elements: the main stack is an array of alternating sizes of anonymous blocks and integer elements and the alt stack is an array of integer elements. Then, assuming the elements in the stacks and the size of the blocks of anonymous elements are represented in binary notation and that we use one extra bit to represent the sign of each number, we then define the size $||\varphi||$ of a stack φ as follows:

Let $\varphi = \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot h^k \in \mathbb{Z}_\bot^*$. Then the size $\|\varphi\|$ used to represent φ is as follows:

$$\|\varphi\| = \sum_{i=0}^{k-1} \Big(\log_2(h_i + 1) + \log_2(|A_i| + 1) + C_1 \Big) + \log_2(h_k + 1) + C_2,$$

where C_1 and C_2 are constants that do not depend on φ[1]. Note that we can also derive the following bounds directly from the definition.

$$\log_2(\mathsf{maxgap}(\varphi) + 1) \leq \|\varphi\| \tag{5.2}$$

$$\log_2(\mathsf{maxelem}(\varphi, \varepsilon) + 1) \leq \|\varphi\| \tag{5.3}$$

$$2\mathsf{elemnr}(\varphi) + 1 \leq \|\varphi\| \tag{5.4}$$

These bounds will help us relate several results that make use of the auxiliary functions with the size of the representation of the stack. This is because we will be interested in establishing a relation between the runtime of executing a script with an initial stack and the sizes of the inputs.

As we previously stated, we will now set upper bounds on the value of the auxiliary functions defined in section 5.1. We will start by proving that the value of maxgap over the execution of a script $S = f_0 \cdot \ldots \cdot f_n$ with an initial stack φ is polynomial in 2^n and $\mathsf{maxgap}(\varphi)$.

LEMMA 5.3. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi_M \in \mathbb{Z}_\bot^*$. The biggest block of anonymous elements that can appear in the main stack after any partial execution $(f_i \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ is bounded by*

$$2^{n+1} \textit{maxgap}(\varphi_M)$$

[1]The constants depend on lower level details depending on specifics of the encoding used, but are not important in our analysis, since they do not depend on the stacks.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^i, \varphi_A^i, \varphi_I^i)$ for all $i \in \{1, \ldots, n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$. Formally, what we want to prove is that

$$\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{maxgap}(\varphi_M^i)\} \leq 2^{n+1}\mathsf{maxgap}(\varphi_M)$$

In order to do this, we will prove by induction that for all $i \in \{0, \ldots, n+1\}$

$$\mathsf{maxgap}(\varphi_M^i) \leq 2^i\mathsf{maxgap}(\varphi_M)$$

Base case.

$$\begin{aligned}\mathsf{maxgap}(\varphi_M^0) &= \mathsf{maxgap}(\varphi_M) \\ &= 2^0\mathsf{maxgap}(\varphi_M)\end{aligned}$$

Inductive step. Let's assume that for an arbitrary $i \in \{0, \ldots, n\}$

$$\mathsf{maxgap}(\varphi_M^i) \leq 2^i\mathsf{maxgap}(\varphi_M)$$

Then,

$$\begin{aligned}\mathsf{maxgap}(\varphi_M^{i+1}) &\leq 2\mathsf{maxgap}(\varphi_M^i) \\ &\leq 2 \cdot 2^i\mathsf{maxgap}(\varphi_M) \\ &= 2^{i+1}\mathsf{maxgap}(\varphi_M)\end{aligned}$$

Moreover, for all $i \in \{0, \ldots, n+1\}$

$$\begin{aligned}\mathsf{maxgap}(\varphi_M^i) &\leq 2^i\mathsf{maxgap}(\varphi_M)\\ &\leq 2^{n+1}\mathsf{maxgap}(\varphi_M)\end{aligned}$$

Thus, we have that

$$\max_{i\in\{0,\ldots,n+1\}}\{\mathsf{maxgap}(\varphi_M^i)\} \leq 2^{n+1}\mathsf{maxgap}(\varphi_M)$$

■

Next, we will constrain the value of elemnr. The value of this function over the execution of a script with an initial stack will be polynomial in the amount of operators in the script and the amount of elements in the initial stack as stated in the following Lemma.

LEMMA 5.4. *Let $S = f_0\cdot\ldots\cdot f_n \in O^*$ and $\varphi_M \in \mathbb{Z}_\perp^*$. The amount of integer elements that can appear in the main stack after any partial execution $(f_i \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ is bounded by*

$$\mathsf{elemnr}(\varphi_M) + 3(n+1)$$

PROOF. Let $S = f_0\cdot\ldots\cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^i, \varphi_A^i, \varphi_I^i)$ for all $i \in \{1,\ldots,n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$. Formally, what we want to prove is that

$$\max_{i\in\{0,\ldots,n+1\}}\{\mathsf{elemnr}(\varphi_M^i)\} \leq \mathsf{elemnr}(\varphi_M) + 3(n+1)$$

In order to do this, we will prove by induction that for all $i \in \{0,\ldots,n+1\}$

$$\mathsf{elemnr}(\varphi_M^i) \leq \mathsf{elemnr}(\varphi_M) + 3i$$

Base case.

$$\begin{aligned}\mathsf{elemnr}(\varphi_M^0) &= \mathsf{elemnr}(\varphi_M)\\ &= \mathsf{elemnr}(\varphi_M) + 3 \cdot 0\end{aligned}$$

Inductive step. Let's assume that for an arbitrary $i \in \{0, \ldots, n\}$

$$\mathsf{elemnr}(\varphi_M^i) \leq \mathsf{elemnr}(\varphi_M) + 3i$$

Then,

$$\begin{aligned}\mathsf{elemnr}(\varphi_M^{i+1}) &\leq \mathsf{elemnr}(\varphi_M^i) + 3\\ &\leq \mathsf{elemnr}(\varphi_M) + 3i + 3\\ &= \mathsf{elemnr}(\varphi_M) + 3(i+1)\end{aligned}$$

Moreover, for all $i \in \{0, \ldots, n+1\}$

$$\begin{aligned}\mathsf{elemnr}(\varphi_M^i) &\leq \mathsf{elemnr}(\varphi_M) + 3i\\ &\leq \mathsf{elemnr}(\varphi_M) + 3(n+1)\end{aligned}$$

Thus, we have that

$$\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^i)\} \leq \mathsf{elemnr}(\varphi_M) + 3(n+1)$$

■

By using Lemmas 5.3 and 5.4 we will also be able to set an upper bound on the depth of the stack over an execution of a script with an initial stack. Specifically, if we execute a script $S = f_0 \cdot \ldots \cdot f_n$ over an initial stack, the size of the stack throughout the execution is constrained by a polynomial in 2^n, the biggest gap in the initial stack and the number of integer elements in the initial stack.

LEMMA 5.5. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi_M \in \mathbb{Z}_\perp^*$. The depth of the main stack after any partial execution $(f_i \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ is bounded by*

$$\textit{elemnr}(\varphi_M) + 3(n+1) + 2^{n+1}\textit{maxgap}(\varphi_M)(\textit{elemnr}(\varphi_M) + 3(n+1) + 1)$$

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^i, \varphi_A^i, \varphi_I^i)$ for all $i \in \{1, \ldots, n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$. Formally, what we want to prove is that

$$\begin{aligned}&\max_{i \in \{0,\ldots,n+1\}} \{|\varphi_M^i|\} \leq \\ &\qquad \mathsf{elemnr}(\varphi_M) + 3(n+1) + 2^{n+1}\mathsf{maxgap}(\varphi_M)(\mathsf{elemnr}(\varphi_M) + 3(n+1) + 1)\end{aligned}$$

Now, for an arbitrary step $i \in \{0, \ldots, n+1\}$, we have that

$$\begin{aligned}|\varphi_M^i| &\leq \mathsf{elemnr}(\varphi_M^i) + \mathsf{maxgap}(\varphi_M^i)(\mathsf{elemnr}(\varphi_M^i) + 1) \\ &\leq \max_{j \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^j) + \mathsf{maxgap}(\varphi_M^j)(\mathsf{elemnr}(\varphi_M^j) + 1)\} \\ &\leq \max_{j \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^j)\} + \\ &\qquad \max_{j \in \{0,\ldots,n+1\}} \{\mathsf{maxgap}(\varphi_M^j)\} \left(\max_{j \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^j)\} + 1 \right) \\ &\leq \mathsf{elemnr}(\varphi_M) + 3(n+1) + \\ &\qquad 2^{n+1}\mathsf{maxgap}(\varphi_M)(\mathsf{elemnr}(\varphi_M) + 3(n+1) + 1)\end{aligned}$$

Note that some of the transformations are performed by applying the results in lemmas 5.3 and 5.4. ■

By utilizing the result of Lemma 5.5 we will be able to set an upper bound on maxelem over the execution of a script. Specifically, we will show that if we execute a script $S = f_0 \cdot \ldots \cdot f_n$ with an initial stack φ, the element with a maximum absolute value in the pair of stacks can be constrained by a polynomial over 2^n, $2^{\|\varphi\|}$ and $\mathsf{maxpush}(S)$. For this purpose, we will use $\mathsf{maxhash} = \max_{C \in \mathbb{Z}}\{|\mathsf{hash}(C)|\}$ to denote the maximum absolute value of any hash. This value is clearly a constant.

LEMMA 5.6. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi_M \in \mathbb{Z}^*_\perp$. The biggest element that can appear in either stack after any partial execution $(f_i \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ is bounded by*

$$p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big),$$

for some fixed polynomial p_{maxelem}, independent of S and φ_M.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}^*_\perp$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi^i_M, \varphi^i_A, \varphi^i_I)$ for all $i \in \{1, \ldots, n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi^0_M, \varphi^0_A, \varphi^0_I)$. Formally, we wish to prove that

$$\max_{i \in \{0,\ldots,n+1\}}\{\mathsf{maxelem}(\varphi^i_M, \varphi^i_A)\} \leq p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big),$$

for some polynomial p_{maxelem}. In order to do this, we will begin by simplifying the result of Lemma 5.5 to make it easier to work with. As we know from equations (5.2) and (5.4),

$$\log_2(\mathsf{maxgap}(\varphi_M) + 1) \leq \|\varphi_M\|$$
$$2\mathsf{elemnr}(\varphi_M) + 1 \leq \|\varphi_M\|$$

Therefore, we have that

$$\begin{aligned}\max_{i\in\{0,\dots,n+1\}}\{|\varphi^i_M|\} &\leq \mathsf{elemnr}(\varphi_M)+3(n+1)+\\ &\qquad 2^{n+1}\mathsf{maxgap}(\varphi_M)(\mathsf{elemnr}(\varphi_M)+3(n+1)+1)\\ &\leq \|\varphi_M\|+3(n+1)+\\ &\qquad 2^{n+1}2^{\|\varphi_M\|}(\|\varphi_M\|+3(n+1)+1)\\ &\leq p_{depth}\Big(2^n, 2^{\|\varphi_M\|}\Big),\end{aligned}$$

for some polynomial p_{depth}. With this established, we will move on to prove by induction that for all $i \in \{0,\dots,n+1\}$

$$\begin{aligned}&\mathsf{maxelem}(\varphi^i_M,\varphi^i_A) \leq\\ &\qquad 2^i\max\{\mathsf{maxelem}(\varphi_M,\varphi_A),\mathsf{maxpush}(S),\mathsf{maxhash},p_{depth}(2^n,2^{\|\varphi_M\|})\}+i\end{aligned}$$

Base case.

$$\begin{aligned}\mathsf{maxelem}(\varphi^0_M,\varphi^0_A) &= \mathsf{maxelem}(\varphi_M,\varphi_A)\\ &\leq \max\{\mathsf{maxelem}(\varphi_M,\varphi_A),\mathsf{maxpush}(S),\\ &\qquad \mathsf{maxhash},p_{depth}(2^n,2^{\|\varphi_M\|})\}\\ &= 2^0\max\{\mathsf{maxelem}(\varphi_M,\varphi_A),\mathsf{maxpush}(S),\\ &\qquad \mathsf{maxhash},p_{depth}(2^n,2^{\|\varphi_M\|})\}+0\end{aligned}$$

Inductive step. Let's assume that for an arbitrary $i \in \{0,\dots,n\}$

$$\begin{aligned}&\mathsf{maxelem}(\varphi^i_M,\varphi^i_A) \leq\\ &\qquad 2^i\max\{\mathsf{maxelem}(\varphi_M,\varphi_A),\mathsf{maxpush}(S),\mathsf{maxhash},p_{depth}(2^n,2^{\|\varphi_M\|})\}+i\end{aligned}$$

Given that the element with maximum absolute value behaves differently depending on the operator that is executed on step i, we will carry out the analysis differentiating between the different possible cases.

- Case 1: $f_i = \mathsf{OP\_PUSH}_C$

$$\begin{aligned}
\mathsf{maxelem}(\varphi_M^{i+1}, \varphi_A^{i+1}) &\leq \max\{\mathsf{maxelem}(\varphi_M^i, \varphi_A^i), |C|\} \\
&\leq \max\{2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i, |C|\} \\
&= 2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i \\
&\leq 2^{i+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i + 1
\end{aligned}$$

- Case 2: $f_i = \mathsf{OP\_HASH160}$

$$\begin{aligned}
\mathsf{maxelem}(\varphi_M^{i+1}, \varphi_A^{i+1}) &\leq \max\{\mathsf{maxelem}(\varphi_M^i, \varphi_A^i), \mathsf{maxhash}\} \\
&\leq \max\{2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i, \mathsf{maxhash}\} \\
&= 2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i \\
&\leq 2^{i+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i + 1
\end{aligned}$$

- Case 3: $f_i = \mathsf{OP\_DEPTH}$

By applying the simplified result of Lemma 5.5,

$$\begin{aligned}
\mathsf{maxelem}(\varphi_M^{i+1}, \varphi_A^{i+1}) &\leq \max\{\mathsf{maxelem}(\varphi_M^i, \varphi_A^i), |\varphi_M^i|\} \\
&\leq \max\{2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i, |\varphi_M^i|\} \\
&= 2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i \\
&\leq 2^{i+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i + 1
\end{aligned}$$

- Case 4: $f_i \notin \{\mathsf{OP\_HASH160}, \mathsf{OP\_DEPTH}\} \cup \{\mathsf{OP\_PUSH}_C \mid C \in \mathbb{Z}\}$

$$\begin{aligned}
\mathsf{maxelem}(\varphi_M^{i+1}, \varphi_A^{i+1}) &\leq 2 \cdot \mathsf{maxelem}(\varphi_M^i, \varphi_A^i) + 1 \\
&\leq 2 \cdot (2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i) + 1 \\
&= 2^{i+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i + 1
\end{aligned}$$

Moreover, for all $i \in \{0, \ldots, n+1\}$

$$\begin{aligned}
\mathsf{maxelem}(\varphi_M^i, \varphi_A^i) &\leq 2^i \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + i \\
&\leq 2^{n+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + n + 1
\end{aligned}$$

Thus, given that we know from equation (5.3) that $\log_2(\mathsf{maxelem}(\varphi_M, \varepsilon) + 1) \leq \|\varphi_M\|$, for some polynomial p_{maxelem},

$$\begin{aligned}
\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{maxelem}(\varphi_M^i, \varphi_A^i)\} &\leq 2^{n+1} \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + n + 1 \\
&\leq 2^{n+1} \max\{2^{\|\varphi_M\|}, \mathsf{maxpush}(S), \\
&\qquad \mathsf{maxhash}, p_{depth}(2^n, 2^{\|\varphi_M\|})\} + n + 1 \\
&\leq p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big)
\end{aligned}$$

As a point of notice, the definition of p_{maxelem} is not dependent on the analyzed instance. In other words, the same polynomial is used as a bound for any execution, only interchanging its variables. ■

The bounds found through Lemmas 5.3, 5.4, 5.5 and 5.6 will become useful, insofar as they prove that the sizes of the main stack and the biggest element in both stacks throughout the execution are polynomial in the size of the inputs. This sentiment is expressed in the following lemma.

LEMMA 5.7. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi_M \in \mathbb{Z}_\perp^*$. The size of the representation of the main stack after any partial execution $(f_i \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ is bounded by*

$$p_{size}(n, \|\varphi_M\|, \log_2(\textsf{\textit{maxpush}}(S) + 1)),$$

for some fixed polynomial p_{size}, independent of S and φ_M.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^i, \varphi_A^i, \varphi_I^i)$ for all $i \in \{1, \ldots, n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$. Formally, we want to prove that

$$\max_{i \in \{0,\ldots,n+1\}} \{\|\varphi_M^i\|\} \leq p_{size}(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S)+1)),$$

for some polynomial p_{size}. In order to do this, we will begin by simplifying the results of Lemmas 5.3 and 5.4 to make them easier to work with. As we know from equations (5.2) and (5.4),

$$\begin{aligned}\log_2(\mathsf{maxgap}(\varphi_M)+1) &\leq \|\varphi_M\| \\ 2\mathsf{elemnr}(\varphi_M)+1 &\leq \|\varphi_M\|\end{aligned}$$

Therefore, we have that

$$\begin{aligned}\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{maxgap}(\varphi_M^i)\} &\leq 2^{n+1}\mathsf{maxgap}(\varphi_M) \\ &\leq 2^{n+1}2^{\|\varphi_M\|} \\ &\leq p_{\mathsf{maxgap}}\left(2^n, 2^{\|\varphi_M\|}\right)\end{aligned}$$

for some fixed polynomial p_{maxgap}, independent of S and φ_M. In addition, we have

$$\begin{aligned}\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^i)\} &\leq \mathsf{elemnr}(\varphi_M) + 3(n+1) \\ &\leq \|\varphi_M\| + 3(n+1) \\ &\leq p_{\mathsf{elemnr}}(\|\varphi_M\|, n),\end{aligned}$$

for some fixed polynomial p_{elemnr}, independent of S and φ_M. Now, let $i \in \{0,\ldots,n+1\}$ be an arbitrary step, $\varphi_M = \bot^{h_0}\cdot A_0\cdot \ldots \cdot A_{k-1}\cdot\bot^{h_k}$ and $\varphi_M^i = \bot^{q_0}\cdot B_0\cdot \ldots \cdot B_{r-1}\cdot\bot^{q_r}$. We know that

$$\|\varphi_M^i\| = \sum_{j=0}^{r-1} \Big(\log_2(q_j+1) + \log_2(|B_j|+1) + C_1 \Big) + \log_2(q_r+1) + C_2,$$

for some constants C_1 and C_2. We also know from the simplification of Lemma 5.3 that for all $j \in \{0, \dots, r\}$,

$$q_j \leq p_{\mathsf{maxgap}}\Big(2^n, 2^{\|\varphi_M\|}\Big)$$

Thus,

$$\begin{aligned}\log_2(q_j+1) &\leq \log_2\left(p_{\mathsf{maxgap}}\Big(2^n, 2^{\|\varphi_M\|}\Big)+1\right)\\ &\leq p_1(n, \|\varphi_M\|),\end{aligned}$$

for some polynomial p_1. From Lemma 5.6 we know that for all $j \in \{0, \dots, r-1\}$,

$$|B_j| \leq p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big)$$

Thus,

$$\begin{aligned}\log_2(|B_j|+1) &\leq \log_2\left(p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big)+1\right)\\ &\leq p_2\Big(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S)+1)\Big),\end{aligned}$$

for some polynomial p_2. In addition, from the simplification of Lemma 5.4 we know that

$$r \leq p_{\mathsf{elemnr}}(\|\varphi_M\|, n)$$

If we combine these results, we have that

$$\begin{aligned}
\|\varphi_M^i\| &\leq \sum_{j=0}^{r-1} \Big(p_1(n, \|\varphi_M\|) + p_2\Big(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)\Big) + C_1\Big) \\
&\qquad + p_1(n, \|\varphi_M\|) + C_2 \\
&= r\Big(p_1(n, \|\varphi_M\|) + p_2\Big(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)\Big) + C_1\Big) \\
&\qquad + p_1(n, \|\varphi_M\|) + C_2 \\
&\leq p_{\mathsf{elemnr}}(\|\varphi_M\|, n)\Big(p_1(n, \|\varphi_M\|) + p_2\Big(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)\Big) \\
&\qquad + C_1\Big) + p_1(n, \|\varphi_M\|) + C_2 \\
&\leq p_{size}(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)),
\end{aligned}$$

for some polynomial p_{size}. It is important to note that this polynomial is independent of the analyzed script and initial stack. This means that the same polynomial will be a valid upper bound for the size of the representation of the stack throughout the execution of any pair of script and stack, only needing to interchange the variables of the polynomial. ■

Now we can move on to prove an upper bound on the execution time of script evaluation. We will consider a naïve algorithm that receives a script $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and a stack with anonymous elements $\varphi \in \mathbb{Z}_\perp^*$, executes each operator in succession over the stack and determines whether the execution did not raise any errors and finished with a nonzero and non-anonymous element on top of the stack. We will use T to denote the execution time of the algorithm that executes a series of operators over either a trio of stacks or an error, with

$$T : O^* \times ((\mathbb{Z}_\perp^* \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\Box\}) \to \mathbb{N}$$

LEMMA 5.8. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi \in \mathbb{Z}_\perp^*$,*

$$T(S, \varphi, \varepsilon, \varepsilon) \leq p_T(n, \|\varphi\|, \log_2(\textit{maxpush}(S) + 1))$$
$$T(S, \Box) \leq p_T(n, \|\varphi\|, \log_2(\textit{maxpush}(S) + 1)),$$

for some fixed polynomial p_T, independent of S and φ.

PROOF. By analizing the definition of the operators in Script provided in appendix B we can ascertain that all of them can be executed in polynomial time in the size of the representation of the main stack, the depth of the control stack, the size of the top of the alt stack and, in the case of $\mathsf{OP\_PUSH}_C$, in the size of the pushed element. We include a more detailed explanation for this observation along with some illustrative examples in appendix E.

Formally, let $f \in O - \{\mathsf{OP\_PUSH}_C \mid C \in \mathbb{Z}\}$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, we have that

$$T(f, \varphi_M, \varphi_A, \varphi_I) \leq p_1(\|\varphi_M\|, |\varphi_I|, \log_2(\mathsf{top}(\varphi_A) + 1)),$$

for some polynomial p_1. Now, let $f = \mathsf{OP\_PUSH}_C$, we have that

$$T(f, \varphi_M, \varphi_A, \varphi_I) \leq p_2(\|\varphi_M\|, |\varphi_I|, \log_2(|\mathsf{top}(\varphi_A)| + 1), \log_2(|C| + 1)),$$

for some polynomial p_2.

Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A = \varepsilon \in \mathbb{Z}^*$ and $\varphi_I = \varepsilon \in \{0,1\}^*$. Additionally, let $(f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^i, \varphi_A^i, \varphi_I^i)$ for all $i \in \{1, \ldots, n+1\}$ and $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$. It is clear that

$$T(S, \varphi_M, \varphi_A, \varphi_I) \leq \sum_{i=0}^{n} T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i)$$

Now, let $i \in \{0, \ldots, n\}$ be an arbitrary step. If $f_i = \mathsf{OP\_PUSH}_C$ for some $C \in \mathbb{Z}$, then

$$T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \leq p_2(\|\varphi_M^i\|, |\varphi_I^i|, \log_2(|\mathsf{top}(\varphi_A^i)| + 1), \log_2(|C| + 1)).$$

Otherwise, we have that

$$T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \leq p_1(\|\varphi_M^i\|, |\varphi_I^i|, \log_2(|\mathsf{top}(\varphi_A^i)| + 1)).$$

Thus, there must exist some polynomial p_3, such that

$$T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \leq p_3(\|\varphi_M^i\|, |\varphi_I^i|, \log_2(|\mathsf{top}(\varphi_A^i)| + 1), \log_2(\mathsf{maxpush}(S) + 1)).$$

This bound will be helpful, but is not readily usable, as it does not relate the execution time of an operator directly to the sizes of the script and the initial stack. Thus, we will perform two simplifications based on the previous Lemmas. Specifically, using the results in Lemma 5.6 we know that

$$|\mathsf{top}(\varphi_A^i)| \leq p_{\mathsf{maxelem}}\Big(2^n, 2^{\|\varphi_M\|}, \mathsf{maxpush}(S)\Big)$$

Thus, there must exist a polynomial p_4, such that

$$\log_2(|\mathsf{top}(\varphi_A^i)| + 1) \leq p_4(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1))$$

Additionally, from Lemma 5.7 we learned that

$$\|\varphi_M^i\| \leq p_{size}(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1))$$

Finally, we have not talked extensively about the way in which the control stack behaves through the execution of a script. However, it is evident that the only operator that can push elements to this stack ($\mathsf{OP\_IF}$) can do so only one at a time. Thus, we can set an upper bound on the depth of the control stack as follows

$$|\varphi_I^i| \leq n + 1$$

There must then exist a polynomial p_5, such that

$$T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \leq p_5(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1))$$

If we combine these results we can easily see that there must exist a polynomial p_T, such that

$$\begin{aligned} T(S, \varphi_M, \varepsilon, \varepsilon) &\leq \sum_{i=0}^{n} T(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \\ &\leq (n+1)p_5(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)) \\ &\leq p_T(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S) + 1)) \end{aligned}$$

It is important to note that this polynomial is independent of the analyzed script and initial stack. This means that the same polynomial will be a valid upper bound for the execution time of the algorithm with any pair of script and stack, only needing to interchange the variables of the polynomial. It is also evident that if for any step $i \in \{0, \ldots, n\}$ the result of applying the i-th operator over the trio of stacks is an error

$$f_i(\varphi_M^i, \varphi_A^i, \varphi_I^i) = \Box,$$

then the execution from this point forward will just consist of carrying over the error and will therefore be even shorter than it would have been when executing all of the operators. Similarly, if we start off with an error as our input, we will not need to execute any operator in the script. In other words,

$$\begin{aligned} T(S, \Box) &\leq C \\ &\leq p_T(n, \|\varphi_M\|, \log_2(\textsf{maxpush}(S) + 1)), \end{aligned}$$

for some constant C. ■

Now, if we consider the complete execution of the evaluation algorithm, it not only needs to execute the operators over the initial stack, but it also must assess if the final stacks are valid. We will use T_{comp} to denote the execution time of the complete algorithm that executes a series of operators over either a trio of stacks or an error and determines whether the finishing stacks fulfill the conditions for a successful execution, with

$$T_{comp} : O^* \times ((\mathbb{Z}_\perp^* \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\Box\}) \to \mathbb{N} \tag{5.5}$$

We will use the following Lemma to set an upper bound on the value of this function.

LEMMA 5.9. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi \in \mathbb{Z}_\perp^*$,*

$$T_{comp}(S, \varphi, \varepsilon, \varepsilon) \leq p_{comp}(n, \|\varphi\|, \log_2(\textsf{\textit{maxpush}}(S) + 1))$$
$$T_{comp}(S, \Box) \leq p_{comp}(n, \|\varphi\|, \log_2(\textsf{\textit{maxpush}}(S) + 1)),$$

for some fixed polynomial p_{comp}, independent of S and φ.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ and $\varphi \in \mathbb{Z}_\perp^*$.

- Case 1: $(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = \Box$
  Given that the execution raises an error, we do not need to perform additional operations to determine whether the execution is successful. Therefore,

$$T_{comp}(S, \varphi, \varepsilon, \varepsilon) = T(S, \varphi, \varepsilon, \varepsilon)$$
$$\leq p_T(n, \|\varphi\|, \log_2(\textsf{maxpush}(S) + 1))$$

- Case 2: $(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varphi_I)$
  In this case, after executing the operators over the initial stack, we must also compare the top of the final main stack with 0 to determine if it represents a boolean value of true and check if the final control stack is empty. These operations are clearly polynomial in the size of the final main stack. We will use p to represent this polynomial, such that

$$T_{comp}(S, \varphi, \varepsilon, \varepsilon) = T(S, \varphi, \varepsilon, \varepsilon) + p(\|\varphi_M\|)$$

  However, Lemma 5.7 states that

$$\|\varphi_M\| \leq p_{size}(n, \|\varphi\|, \log_2(\textsf{maxpush}(S) + 1))$$

and Lemma 5.8, that

$$T(S, \varphi, \varepsilon, \varepsilon) \leq p_T(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1))$$

Thus, for some pair of polynomials p_2 and p_3,

$$\begin{aligned}
T_{comp}(S, \varphi, \varepsilon, \varepsilon) &= T(S, \varphi, \varepsilon, \varepsilon) + p(\|\varphi_M\|) \\
&\leq p_T(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1)) + \\
&\quad\; p_2(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1)) \\
&\leq p_3(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1))
\end{aligned}$$

Now that we have successfully set a polynomial upper bound for both cases, trivially there must exist a polynomial p_{comp}, such that

$$T_{comp}(S, \varphi, \varepsilon, \varepsilon) \leq p_{comp}(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1))$$

Additionally, the case where the execution begins with an error is analogous to the first analyzed case. In this situation we do not need to execute any operator in the script and we also do not need to perform any additional operations after the script execution. Thus,

$$\begin{aligned}
T_{comp}(S, \Box) &= T(S, \Box) \\
&\leq p_T(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1)) \\
&\leq p_{comp}(n, \|\varphi\|, \log_2(\mathsf{maxpush}(S) + 1))
\end{aligned}$$

■

As we can see, Lemma 5.9 ultimately proves that the problem of Script evaluation presented at the beginning of this section is in PTIME. This can be concluded by noticing that the representation of a script is clearly linear in the amount of operators it has and the maximum element that it pushes to the stack.

Lemma 5.9 also allows us to easily prove that the simpler problem of script evaluation presented in section 4.1 is in PTIME. This extension of the provided result is discussed in appendix F.

5.3. Limiting the number of elements accessed by a script

With the introduction of anonymous elements we can start to think about the amount of elements in an unlocking stack that are actually necessary for the execution of a certain script. Our objective is to compress a stack as far as possible. In other words, when considering a stack that is able to unlock certain script, we want to anonymize every possible integer element, such that the resulting stack is still able to unlock the script.

Ideally, identifying these relevant elements would just consist of taking every operator in a given script and marking the elements that it accesses. However, this process is not as straightforward as it seems because the elements can be moved around during the execution, causing them to be in a different position when they get used than the one in which they started. Additionally, some operators' executions depend on the values of the stack elements and would therefore demand to be analyzed in runtime. Thus, we propose determining which elements are relevant in an execution by establishing which of them can not be anonymized for the execution to end successfully.

In order to capture this notion we define a new function accessed which represents the amount of elements in a stack that cannot be anonymized in order for the execution of a certain operator over said stack to not result in an error. As such, we describe the structure of accessed as

$$\mathsf{accessed} : O \times \mathbb{Z}_{\perp}^{*} \times \mathbb{Z}^{*} \times \{0,1\}^{*} \to \mathbb{N}$$

In order to define the value of accessed, let $f \in O$, $\varphi_M = \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k} \in \mathbb{Z}_\bot^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, such that $f(\varphi_M, \varphi_A, \varphi_I) \neq \Box$. We start by defining a subset $H \subseteq \{0, \ldots, k-1\}$ that contains the indices of integer elements in the main stack that can not be anonymized for the execution to be successful. Formally, for any element $i \in H$,

$$f(\bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i+1+h_{i+1}} \cdot A_{i+1} \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A, \varphi_I) = \Box$$

Similarly, for any element $i \in \{0, \ldots, k-1\} - H$,

$$f(\bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i+1+h_{i+1}} \cdot A_{i+1} \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A, \varphi_I) \neq \Box$$

In other words, H represents the set of integer elements of φ_M, such that if we anonymize any integer in the set, the execution will output an error, and if we anonymize any integer outside of the set, the execution will still be successful. Having defined H we can establish the value of accessed simply as

$$\textsf{accessed}(f, \varphi_M, \varphi_A, \varphi_I) = |H|.$$

Based on this definition we can move on to establish an upper bound on the value of accessed, which is presented in the following Lemma.

LEMMA 5.10. *Let $f \in O$, $\varphi_M \in \mathbb{Z}_\bot^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, such that $f(\varphi_M, \varphi_A, \varphi_I) \neq \Box$. Then,*

$$\textsf{accessed}(f, \varphi_M, \varphi_A, \varphi_I) \leq 3$$

PROOF. We prove this case by case in appendix D. ■

Next, we will expand the domain of the $\mathsf{accessed}$ function to encompass whole scripts instead of individual operators. This will allow us to capture the notion of relevant stack elements for the execution of a script. The domain of the function will be extended naturally as follows

$$\mathsf{accessed} : O^* \times \mathbb{Z}_\perp^* \times \mathbb{Z}^* \times \{0,1\}^* \to \mathbb{N}$$

Now, let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, such that $(f_n \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) \neq \Box$. Once again, we define a subset $H \subseteq \{0, \ldots, k-1\}$ that contains the indices of integer elements in the main stack that can not be anonymized for the execution to be successful. Formally, for any element $i \in H$,

$$(f_n \circ \ldots \circ f_0)(\perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \perp^{h_i+1+h_{i+1}} \cdot A_{i+1} \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A, \varphi_I) = \Box$$

Similarly, for any element $i \in \{0, \ldots, k-1\} - H$,

$$(f_n \circ \ldots \circ f_0)(\perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \perp^{h_i+1+h_{i+1}} \cdot A_{i+1} \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A, \varphi_I) \neq \Box$$

Just as in the previous case, H represents the set of integer elements of φ_M, such that if we anonymize any integer in the set, the execution will output an error, whereas if we anonymize any integer outside of the set, the execution will still be successful. Again, we can establish the value of $\mathsf{accessed}$ simply as

$$\mathsf{accessed}(S, \varphi_M, \varphi_A, \varphi_I) = |H|.$$

Once again we want to establish an upper bound on the value of $\mathsf{accessed}$. However, before doing so we need to enunciate an intermediate result that will aid us in proving said bound. Namely, that because of the way in which the operators are defined, anonymous elements are neither created nor deleted.

Consequently, each anonymous element can be traced from start to finish of the execution in the main stack. In addition, if we replace any anonymous element by an integer, it will remain untouched in the stack through the execution. We formalize this concept through the following Lemma.

LEMMA 5.11. *Let $f \in O$, $\varphi_M = \alpha_0 \cdot \ldots \cdot \alpha_m, \varphi'_M = \beta_0 \cdot \ldots \cdot \beta_n \in \mathbb{Z}^*_\perp$, $\varphi_A, \varphi'_A \in \mathbb{Z}^*$ and $\varphi_I, \varphi'_I \in \{0,1\}^*$, such that*

$$f(\varphi_M, \varphi_A, \varphi_I) = (\varphi'_M, \varphi'_A, \varphi'_I)$$

For every subindex $p \in \{0, \ldots, n\}$ with $\beta_p = \perp$ there exists a subindex $q \in \{0, \ldots, m\}$ with $\alpha_q = \perp$, such that for all $\gamma \in \mathbb{Z}$, the execution of the operator will fulfill

$$f(\alpha_0 \cdot \ldots \cdot \alpha_{q-1} \cdot \gamma \cdot \alpha_{q+1} \cdot \ldots \cdot \alpha_m, \varphi_A, \varphi_I) = (\beta_0 \cdot \ldots \cdot \beta_{p-1} \cdot \gamma \cdot \beta_{p+1} \cdot \ldots \cdot \beta_n, \varphi'_A, \varphi'_I)$$

PROOF. This result is evident from the definition of the operators in appendix B. ■

Now we will define an auxiliary function that will also help us prove the bound on $\mathsf{accessed}$. This function will be denoted by deanon and what it does is receive a stack with anonymous elements and a subset of the indices of the integer elements of said stack, and outputs the stack obtained by anonymizing every integer that is not contained in the set. Formally, let $\psi = \perp^{p_0} \cdot q_0 \cdot \ldots \cdot q_{r-1} \cdot \perp^{p_r} \in \mathbb{Z}^*_\perp$ and $J = \{i_0, \ldots, i_s\} \subseteq \{0, \ldots, r-1\}$, where $i_0 < \ldots < i_s$, we define

$$\mathsf{deanon}(\psi, J) = \perp^{p_0 + \ldots + p_{i_0} + i_0} \cdot q_{i_0} \cdot \perp^{p_{i_0+1} + \ldots + p_{i_1} + i_1 - i_0 - 1} \cdot \ldots \cdot q_{i_s} \cdot \perp^{p_{i_s+1} + \ldots + q_r + r - i_s - 1}$$

By utilizing the result enunciated in Lemma 5.11 and the auxiliary function deanon we can move on to establish an upper bound on the value of the accessed function over a script instead of individual operators. This bound will consist of the sum of the values of the accessed function over each of the individual operators in the script.

Intuitively, given that anonymous elements are not created nor deleted in the execution of the script, if the execution of one of the operators in the script results in an error after anonymizing a certain amount of integer elements of the stack, we should not need to deanonymize more elements than the value of the accessed function for said operator for the execution to be successful. Evidently, if we start off with a completely anonymous stack and apply this argument for each of the operators we arrive at the aformentioned bound.

LEMMA 5.12. *Let* $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ *and* $\varphi_I \in \{0,1\}^*$*, such that* $(f_n \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi'_M, \varphi'_A, \varphi'_I)$*. If we define* $(\varphi_M, \varphi_A, \varphi_I) = (\varphi^0_M, \varphi^0_A, \varphi^0_I)$ *and* $(f_{\ell-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi^\ell_M, \varphi^\ell_A, \varphi^\ell_I)$ *for all* $\ell \in \{1, \ldots, n+1\}$*, we have that*

$$\textit{accessed}(S, \varphi_M, \varphi_A, \varphi_I) \leq \sum_{i=0}^{n} \textit{accessed}(f_i, \varphi^i_M, \varphi^i_A, \varphi^i_I)$$

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, such that $(f_n \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi'_M, \varphi'_A, \varphi'_I)$. We will say that $(\varphi_M, \varphi_A, \varphi_I) = (\varphi^0_M, \varphi^0_A, \varphi^0_I)$ and that $(f_{\ell-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi^\ell_M, \varphi^\ell_A, \varphi^\ell_I)$ for all $\ell \in \{1, \ldots, n+1\}$. We will prove by induction that for all $j \in \{0, \ldots, n\}$

$$\text{accessed}(f_j \circ \ldots \circ f_0, \varphi_M, \varphi_A, \varphi_I) \leq \sum_{i=0}^{j} \text{accessed}(f_i, \varphi^i_M, \varphi^i_A, \varphi^i_I)$$

Base Case. It is evident that

$$\begin{aligned}\mathsf{accessed}(f_0, \varphi_M, \varphi_A, \varphi_I) &\leq \sum_{i=0}^{0} \mathsf{accessed}(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \\ &= \mathsf{accessed}(f_0, \varphi_M, \varphi_A, \varphi_I)\end{aligned}$$

Inductive Step. We will assume that for some $j \in \{0, \dots, n-1\}$

$$\mathsf{accessed}(f_j \circ \ldots \circ f_0, \varphi_M, \varphi_A, \varphi_I) \leq \sum_{i=0}^{j} \mathsf{accessed}(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i)$$

Now, we will define $\varphi_M^{j+1} = \bot^{\eta_0} \cdot \alpha_0 \cdot \ldots \cdot \alpha_{\kappa-1} \cdot \bot^{\eta_\kappa}$ and $H \subseteq \{0, \dots, \kappa - 1\}$ with $|H| = \mathsf{accessed}(f_{j+1}, \varphi_M^{j+1}, \varphi_A^{j+1}, \varphi_I^{j+1})$, such that

$$f_{j+1}(\mathsf{deanon}(\varphi_M^{j+1}, H), \varphi_A^{j+1}, \varphi_I^{j+1}) \neq \Box$$

Additionally, let $\varphi_M = \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}$, $J \subseteq \{0, \dots, k-1\}$, with $|J| = \mathsf{accessed}(f_j \circ \ldots \circ f_0, \varphi_M, \varphi_A, \varphi_I)$, and $K \subseteq \{0, \dots, \kappa - 1\}$, such that

$$(f_j \circ \ldots \circ f_0)(\mathsf{deanon}(\varphi_M, J), \varphi_A, \varphi_I) = (\mathsf{deanon}(\varphi_M^{j+1}, K), \varphi_A^{j+1}, \varphi_I^{j+1})$$

Let $t \in H - K$ be an arbitrary subindex. From Lemma 5.11 we know that there must exist a subindex $v \in \{0, \dots, k-1\} - J$, such that

$$(f_j \circ \ldots \circ f_0)(\mathsf{deanon}(\varphi_M, J \cup \{v\}), \varphi_A, \varphi_I) = (\mathsf{deanon}(\varphi_M^{j+1}, K \cup \{t\}), \varphi_A^{j+1}, \varphi_I^{j+1})$$

Note that we can repeat this procedure for every subindex in $H - K$. Intuitively, if we execute the first $j + 1$ operators over a stack with only the required integer elements for these operators to work, any element that is required in order to execute operator f_{j+1} and is anonymous after the described partial execution can be traced back to an integer element that was anonymized in the initial stack.

As such, we can deanonymize each of these again in order for the execution of f_{j+1} to not output an error. This means that there will exist a set $L \subseteq \{0, \ldots, k-1\} - J$, where $|L| = |H - K|$, such that

$$(f_j \circ \ldots \circ f_0)(\mathsf{deanon}(\varphi_M, J \cup L), \varphi_A, \varphi_I) = (\mathsf{deanon}(\varphi_M^{j+1}, K \cup H), \varphi_A^{j+1}, \varphi_I^{j+1})$$

Consequently,

$$(f_{j+1} \circ \ldots \circ f_0)(\mathsf{deanon}(\varphi_M, J \cup L), \varphi_A, \varphi_I) \neq \Box$$

This is because after the partial execution of $f_0 \cdot \ldots \cdot f_i$ over $\mathsf{deanon}(\varphi_M, J \cup L)$, the main stack will have the elements that correspond to H deanonymized. As such, from the definition of H, we know that the execution of f_{i+1} will be successful. Furthermore, we can see that

$$\mathsf{elemnr}(\mathsf{deanon}(\varphi_M, J \cup L)) = |J| + |H - K|,$$

from which we can conclude that

$$\begin{aligned}
\textsf{accessed}(f_{j+1} \circ \ldots \circ f_0, \varphi_M, \varphi_A, \varphi_I) &\leq |J| + |H - K| \\
&\leq |J| + |H| \\
&= \textsf{accessed}(f_j \circ \ldots \circ f_0, \varphi_M, \varphi_A, \varphi_I) \\
&\quad + \textsf{accessed}(f_{j+1}, \varphi_M^{j+1}, \varphi_A^{j+1}, \varphi_I^{j+1}) \\
&\leq \sum_{i=0}^{j} \textsf{accessed}(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \\
&\quad + \textsf{accessed}(f_{j+1}, \varphi_M^{j+1}, \varphi_A^{j+1}, \varphi_I^{j+1}) \\
&= \sum_{i=0}^{j+1} \textsf{accessed}(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i)
\end{aligned}$$

■

By putting together Lemmas 5.10 and 5.12 we can establish a more concrete bound on the value of accessed over a script. This will help us prove that for any unlockable script there exists a stack that is polynomial in its size and is able to unlock it.

COROLLARY 5.12.1. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$. For all $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$ such that $(f_n \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) \neq \Box$. Then,*

$$\textsf{accessed}(S, \varphi_M, \varphi_A, \varphi_I) \leq 3n + 3$$

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$ such that $(f_n \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) \neq \Box$. For convenience we will say that $(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^0, \varphi_A^0, \varphi_I^0)$ and that $(f_{\ell-1} \circ \ldots \circ f_0)(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M^\ell, \varphi_A^\ell, \varphi_I^\ell)$ for all $\ell \in \{1, \ldots, n+1\}$.

$$\begin{aligned}\mathsf{accessed}(S, \varphi_M, \varphi_A, \varphi_I) &\leq \sum_{i=0}^{n} \mathsf{accessed}(f_i, \varphi_M^i, \varphi_A^i, \varphi_I^i) \\ &\leq \sum_{i=0}^{n} 3 \\ &= 3n + 3\end{aligned}$$

■

Finally, we can use the result of Corollary 5.12.1 to prove the main result of this section, which is that the maximum amount of accessed elements is linear in the amount of operators in the script. Moreover, this proves that any unlockable script has at least one valid unlocking stack that only has a linear amount of integer elements. This concept is formalized in the following Lemma.

LEMMA 5.13. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an unlockable script. There exists a stack $\varphi_M \in \mathbb{Z}_\perp^*$, such that $\mathsf{elemnr}(\varphi_M) \leq 3n + 4$ and*

$$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi_M^F, \varphi_A^F, \varepsilon),$$

where $|\varphi_M^F| \geq 1$ and $\mathsf{top}_\perp(\varphi_M^F) \notin \{\perp, 0\}$.

PROOF. To prove this Lemma we will reutilize the deanon auxiliary function used in the proof for Lemma 5.12. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an unlockable script. There must exist a stack $\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$, such that

$$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi_M^F, \varphi_A^F, \varepsilon),$$

where $|\varphi_M^F| \geq 1$ and $\mathsf{top}_\perp(\varphi_M^F) \notin \{\perp, 0\}$. From Corollary 5.12.1 we know that $\mathsf{accessed}(S, \varphi_M, \varepsilon, \varepsilon) \leq 3n+3$. This means that there exists a set of subindices $J \subseteq \{0, \ldots, k-1\}$, where $|J| \leq 3n+3$, such that

$$(f_n \circ \ldots \circ f_0)(\mathsf{deanon}(\varphi_M, J), \varepsilon, \varepsilon) = (\psi_M^F, \psi_A^F, \varepsilon)$$

Case 1: $\mathsf{top}_\perp(\psi_M^F) \neq \perp$

If this is the case, the way in which operators over anonymous elements are constructed ensures that

$$\mathsf{top}_\perp(\psi_M^F) = \mathsf{top}_\perp(\varphi_M^F) \notin \{\perp, 0\}$$

Consequently, ψ_M is a valid unlocking stack for S. Moreover,

$$\begin{aligned} \mathsf{elemnr}(\psi_M) &= |J| \\ &\leq 3n+3 \\ &< 3n+4 \end{aligned}$$

Case 2: $\mathsf{top}_\perp(\psi_M^F) = \perp$

We will use both representations of stacks for convenience, depending on the context. Let

$$\begin{aligned} \mathsf{deanon}(\varphi_M, J) &= B_0 \cdot \ldots \cdot B_\ell \\ \psi_M^F &= C_0 \cdot \ldots \cdot C_m \end{aligned}$$

We know that $C_0 = \bot$. From Lemma 5.11 it is clear that there must exist a subindex $i \in \{0, \ldots, k-1\} - J$, such that if we define $p = \sum_{j=0}^{i}(h_j + 1) - 1$, then

$$B_p = \bot$$

and

$$(f_n \circ \ldots \circ f_0)(B_0 \cdot \ldots \cdot B_{p-1} \cdot A_i \cdot B_{p+1} \cdot \ldots \cdot B_\ell, \varepsilon, \varepsilon) = (A_i \cdot C_1 \cdot \ldots \cdot C_m, \psi_A^F, \varepsilon)$$

It is clear that $A_i = \mathsf{top}_\bot(\varphi_M^F)$. Therefore, we can assert that

$$\mathsf{top}(A_i \cdot C_1 \cdot \ldots \cdot C_m) = \mathsf{top}_\bot(\varphi_M^F) \notin \{\bot, 0\}$$

Consequently, $B_0 \cdot \ldots \cdot B_{p-1} \cdot A_i \cdot B_{p+1} \cdot \ldots \cdot B_\ell = \mathsf{deanon}(\varphi_M, J \cup \{i\})$ is a valid unlocking stack for S. Furthermore,

$$\begin{aligned}\mathsf{elemnr}(\mathsf{deanon}(\varphi_M, J \cup \{i\})) &= |J \cup \{i\}| \\ &\le |J| + 1 \\ &\le 3n + 4\end{aligned}$$

■

5.4. Limiting the size of stack elements

In this section we aim to prove that for any unlockable locking script there exists a stack that is able to unlock it that only has elements of polynomial size in the size of the script. However, to prove this we will restrict the usage of the cryptographic functions. Specifically, we will assume that the cryptographic operators $\textsf{OP\_HASH160}$, $\textsf{OP\_CHECKSIG}$ and $\textsf{OP\_CHECKSIGVERIFY}$ only work over relatively small stack elements. Concretely, we will add another condition to the execution of these operators.

Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$, $\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$ and $\varphi_A \in \mathbb{Z}^*$, for $\textsf{OP\_HASH160}$ to be executed over (φ_M, φ_A), we will require that the stacks fulfill $A_0 \leq p_{crypto}(2^n, \textsf{maxpush}(S))$ for some polynomial p_{crypto}. Similarly, in the cases of $\textsf{OP\_CHECKSIG}$ and $\textsf{OP\_CHECKSIGVERIFY}$ we will require that they fulfill $A_0 \leq p_{crypto}(2^n, \textsf{maxpush}(S))$ and that $A_1 \leq p_{crypto}(2^n, \textsf{maxpush}(S))$. These conditions are additions to the ones established in appendix B.

The reason for adding these restrictions is that we speculate that without them the problem would climb significantly in the complexity hierarchy, possibly becoming undecidable. To further illustrate this point we turn to the following example of a system of equations easily reproduceable by utilizing script

$$x = y + 1$$

$$\textsf{hash}(x) = \textsf{hash}(y)$$

This sort of problems in which we combine arithmetic and hash functions might be solvable but there is no guarantee that the size of their solutions is bounded by any polynomial. Therefore, although it might be possible to determine whether they have a solution without constructing it, it is not likely the case and it would probably depend on the construction of the utilized hash functions.

Furthermore, we propose that all of the common uses for scripts abide by these restrictions. In particular, scripts tend to use $\textsf{OP\_HASH160}$ for hashing public keys, whereas they make use of $\textsf{OP\_CHECKSIG}$ and $\textsf{OP\_CHECKSIGVERIFY}$ exclusively to determine whether a signature is valid for certain public key. Both cryptographic keys and signatures are of constant size, which means that requiring the inputs for these operators to be of polynomial size wouldn't hinder the utilization of common scripts.

Systems of linear equations. Our upper bound relies on a classic bound established for integer solutions of systems of linear equations developed by Papadimitriou (Papadimitriou, 1981). Hence, to invoke this result we need a correspondence between finding unlocking scripts and solving linear equations. We were not able to find a simple, direct reduction between these two problems. Instead, we proceed as follows. Assume that a given locking script has an arbitrary (possibly big) solution, in the form of an unlocking stack. With this solution at hand, we construct a system of equations that, in a sense, expresses the essential properties that the stack must fulfill when executing the locking script over the unlocking stack. Afterwards, we will show that the constructed system of equations must have a polynomially-sized solution, which in turn ensures the existence of a solution for the script which is also polynomially-sized.

It is important to note that this strategy only allows us to prove that if the script is unlockable, it must have a polynomially-sized solution. The system of equations that we will construct through the proof does not fully represent the locking script, in that there might exist solutions to the script that do not translate into solutions to the system of equations. Now we can enunciate the main result that we will prove in this section.

THEOREM 5.14. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an unlockable script. In addition, let $\varphi_M \in \mathbb{Z}_\perp^*$, such that*

$$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi_M^F, \varphi_A^F, \varepsilon),$$

where φ_M^F is a valid final main stack. There exists a system of equations $\Phi\vec{x} = \vec{b}$ that has at least one solution with

$$\vec{x} = (\bar{x}, \bar{g}, \bar{v}, \bar{w})^T,$$

where $\bar{x}$ and $\bar{g}$ have k and $k+1$ elements, respectively, with $k = \textbf{\textsf{elemnr}}(\varphi_M)$, and where $\bar{v}$ and $\bar{w}$ both have $n+2$ elements each, constrained by $\bar{g}, \bar{v} \geq 0$, such that

$$|a_{\max}| \leq p_{coef}(2^n, \textbf{\textsf{elemnr}}(\varphi_M), \textbf{\textsf{maxpush}}(S)), \tag{5.6}$$

where $a_{\max}$ is the biggest element between Φ and $\vec{b}$ and p_{coef} is some polynomial, and from any valid solution to the system

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

where $\bar{C} = (C_0, \ldots, C_{k-1})$ and $\bar{E} = (E_0, \ldots, E_k)$, we can construct a valid unlocking stack ψ_M for S as follows

$$\psi_M = \perp^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \perp^{E_k}$$

PROOF. We will prove this by constructing the system of equations that fulfills the aforementioned properties. Conceptually, we want to take a valid solution to the locking script and express what allows it to unlock the script through a system of equations. To accomplish this we will go through the execution of the script over the existing solution and capture, for each operator, the conditions that the stack needs to fulfill for the execution not to end in an error, and for any other stack to take the same execution branch as the existing solution.

For this purpose we will utilize an auxiliary pair of stacks with variables. We will start by establishing a pair of starting variable stacks and then detailing how they change based on the operators in the script. Additionally, we will establish how the system of equations is constructed based on these operators and show a basic solution for it.

Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an unlockable script. In addition, let $\varphi_M \in \mathbb{Z}_\perp^*$, such that

$$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi_M^F, \varphi_A^F, \varepsilon),$$

where φ_M^F is a valid final main stack. For convenience we will define $(\varphi_M^i, \varphi_A^i, \varphi_I^i) = (f_{i-1} \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon)$ for each $i \in \{1, \ldots, n+1\}$ and $(\varphi_M^0, \varphi_A^0, \varphi_I^0) = (\varphi_M, \varepsilon, \varepsilon)$. Let

$$\vec{x} = (\bar{x}, \bar{g}, \bar{v}, \bar{w})^T,$$

be the vector of variables for our eventual system of equations. Our initial system of equations will be

$$\Phi_0 \vec{x} = \vec{b}_0,$$

where

$$\begin{aligned} \Phi_0 &= [\quad] \\ \vec{b}_0 &= [\quad] \end{aligned}$$

are a matrix and a vector, both with 0 rows. Let $\bar{x} = (x_0, \ldots, x_{k-1})$ and $\bar{g} = (g_0, \ldots, g_k)$, with $k = \mathsf{elemnr}(\varphi_M)$. The starting variable stacks will be

$$\begin{aligned} \chi_M^0 &= \bot^{g_0} \cdot x_0 \cdot \ldots \cdot x_{k-1} \cdot \bot^{g_k} \\ \chi_A^0 &= \varepsilon \end{aligned}$$

Now, let $i \in \{1, \ldots, n\}$ be an arbitrary number with

$$\chi_M^i = \perp^{d_0} \cdot z_0 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}$$
$$\chi_A^i = p_0 \cdot \ldots \cdot p_{\ell'}$$
$$\varphi_M^i = \perp^{P_0} \cdot M_0 \cdot \ldots \cdot M_{k'-1} \cdot \perp^{P_{k'}}$$
$$\varphi_A^i = N_0 \cdot \ldots \cdot N_{\ell'}$$

We will establish the value of $(\chi_M^{i+1}, \chi_A^{i+1})$ based on (χ_M^i, χ_A^i) and f_i. It's important to note that in case φ_I^i doesn't represent an execution state, then

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\chi_M^i, \chi_A^i)$$

We will also detail the equations that are added to the system depending on the value of the variable stacks and the operator in each step. We will use the notation

$$C = \begin{bmatrix} A \\ B \end{bmatrix}$$

to denote the vertical concatenation of matrices. Let Ω_i and $\vec{a}_i$ be the matrix and the vector that represent the equations we would add to the system on step i. We define

$$\Phi_i = \begin{bmatrix} \Phi_{i-1} \\ \Omega_i \end{bmatrix} \qquad \vec{b}_i = \begin{bmatrix} \vec{b}_{i-1} \\ \vec{a}_i \end{bmatrix}$$

as the matrix and the vector that represent the whole system up until step i (i.e. in each step we add equations to the system). Once again, if φ_I^i doesn't represent an execution state, then

$$\Omega_i = [\quad]$$
$$\vec{a}_i = [\quad]$$

Next, we will establish the value of $\chi_M^{i+1}, \chi_A^{i+1}, \Omega_{i+1}, \vec{a}_{i+1}$ based on the values of $\chi_M^i, \chi_A^i, \varphi_M^i, \varphi_A^i$. However, the additional equations will be detailed as equations and not in matrix form for convenience. Additionally, we establish an upper bound on the coefficients of the added equations by operator to prove the upper bound on the size of the coefficients of our system. We use $a_{\max}$ to refer to the biggest element between Ω_i and $\vec{a}_i$.

To construct a basic solution for the system we will just need to assign $x_i = A_i$ for all $i \in \{0, \ldots, k-1\}$ and $g_i = h_i$ for all $i \in \{0, \ldots, k\}$ and calculate the necessary values for the rest of the variables based on these. It will be easy to see that for each new equation there will be at most one unassigned variable, which will make the construction of the solution trivial. Also, the equations that only consist of already assigned variables will be satisfied automatically because otherwise the execution over the unlocking stack would end in an error, which is evidently not the case.

Firstly, we will show the way in which the pair of variable stacks will be modified depending on the i-th operator in the locking script and the equations that get added to the system. We will also gradually prove that the coefficients in those equations fulfill the bound described in equation (5.6).

The complete examination that describes these three points of analysis for each of the operators is displayed in appendix G. However, in what follows we will show the analysis for some of the operators, as a way to exemplify the process. As a reminder, the bound that we will be proving for the coefficients in the system of equations, which was introduced in equation (5.6) is

$$|a_{\max}| \leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)).$$

We will start by analyzing the case where $f_i = \mathsf{OP\_PUSH}_C$, as it is the simplest. The modification in the variable stacks will be determined by

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \chi_M^i, \chi_A^i) \tag{5.7}$$

The equations that will be added to the system in this case are the following:

$$\begin{aligned} w_{null} &= 0 \\ w_i &= C \end{aligned}$$

In this case it is easy to see that the maximum coefficient present in these new equations is

$$\begin{aligned} |a_{\max}| &\leq \max\{1, |C|\} \\ &\leq \mathsf{maxpush}(S) + 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

for some polynomial p_{coef}. Next, we analyze the case where $f_i = \mathsf{OP\_IFDUP}$, which is more complex. The modification in the variable stacks will be determined by

$$(\chi_M^{i+1}, \chi_A^{i+1}) = \begin{cases} (\chi_M^i, \chi_A^i) & \text{if } \mathsf{top}(\varphi_M^i) = 0 \\ (\perp^{w_{null}} \cdot z_0 \cdot \chi_M^i, \chi_A^i) & \text{if } \mathsf{top}(\varphi_M^i) \neq 0 \end{cases} \tag{5.8}$$

The basic equations that will be added to the system in this case are the following:

$$d_0 = 0$$
$$w_{null} = 0$$

However, in this case we will add additional equations depending on the value of $\mathsf{top}_\perp(\varphi_M^i)$. Specifically,

(i) If $\mathsf{top}(\varphi_M^i) > 0$, then we will add the equation $z_0 = v_i + 1$
(ii) If $\mathsf{top}(\varphi_M^i) < 0$, then we will add the equation $z_0 = -v_i - 1$
(iii) If $\mathsf{top}(\varphi_M^i) = 0$, then we will add the equation $z_0 = 0$

In any case we can see that the maximum coefficient in these new equations is bounded by

$$\begin{aligned} |a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

for some polynomial p_{coef}. Now, we aim to prove that the sequence of variable stacks accurately represents the evolution that a pair of stacks constructed from a solution to the system of equations would go through when subjected to the execution of the script. This will be proved through the next Claim, but we need to define some elements before introducing the formalization of this notion. Let $i \in \{0, \ldots, n+1\}$ be an arbitrary step and $\vec{c}$ a solution to the system of equations on step i, such that

$$\Phi_i \vec{c} = \vec{b}_i,$$

where

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

with $\bar{C} = (C_0, \ldots, C_{k-1})$ and $\bar{E} = (E_0, \ldots, E_k)$. We define

$$(\psi_M^0, \psi_A^0, \psi_I^0) = (\perp^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \perp^{E_k}, \varepsilon, \varepsilon)$$

and for all $i \in \{1, \ldots, n+1\}$

$$(\psi_M^i, \psi_A^i, \psi_I^i) = (f_{i-1} \circ \ldots \circ f_0)(\psi_M^0, \psi_A^0, \psi_I^0)$$

CLAIM 5.14.1. *For all $i \in \{0, \ldots, n+1\}$, if we assign $\vec{x} = \vec{c}$, then*

$$(\chi_M^i, \chi_A^i, \varphi_I^i) = (\psi_M^i, \psi_A^i, \psi_I^i)$$

PROOF OF CLAIM. We will prove this statement by induction.

Base Case. Let $\vec{c}$ be an instance for the variable vector $\vec{x}$, with

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T$$
$$\bar{C} = (C_0, \ldots, C_{k-1})$$
$$\bar{E} = (E_0, \ldots, E_k),$$

from which we define the pair of stacks

$$\psi_M^0 = \perp^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \perp^{E_k}$$
$$\psi_A^0 = \varepsilon,$$

such that

$$\Phi_0 \vec{c} = \vec{b}_0$$

It is clear that if we assign $\vec{x} = \vec{c}$, then

$$\begin{aligned}(\chi_M^0, \chi_A^0, \varphi_I^0) &= (\bot^{g_0} \cdot x_0 \cdot \ldots \cdot x_{k-1} \cdot \bot^{g_k}, \varepsilon, \varepsilon) \\ &= (\bot^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \bot^{E_k}, \varepsilon, \varepsilon) \\ &= (\psi_M^0, \psi_A^0, \psi_I^0)\end{aligned}$$

Inductive Step. Let $i \in \{0, \ldots, n\}$ be an arbitrary step. We will assume that the following holds for all solutions to the system of equations on step i. Let $\vec{c}$ be a solution to the system on step i, such that $\Phi_i \vec{c} = \vec{b}_i$, where

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

with $\bar{C} = (C_0, \ldots, C_{k-1})$ and $\bar{E} = (E_0, \ldots, E_k)$. In addition, let $(\psi_M^0, \psi_A^0, \psi_I^0) = (\bot^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \bot^{E_k}, \varepsilon, \varepsilon)$ and for all $i \in \{1, \ldots, n+1\}$

$$(\psi_M^i, \psi_A^i, \psi_I^i) = (f_{i-1} \circ \ldots \circ f_0)(\psi_M^0, \psi_A^0, \psi_I^0)$$

Then, if we assign $\vec{x} = \vec{c}$, we will assume that

$$(\chi_M^i, \chi_A^i, \varphi_I^i) = (\psi_M^i, \psi_A^i, \psi_I^i)$$

We will prove that the same property is fulfilled for step $i + 1$. It is important to note that if φ_I does not represent an execution state, then the variable stacks, the integer stacks and the system of equations are not modified. Therefore, the equivalence of the main and the alt stacks will be evident. If $f_i \notin \{\text{OP\_IF}, \text{OP\_ELSE}, \text{OP\_ENDIF}\}$, the control stacks will also be evidently equal.

Furthermore, given that

$$\Phi_{i+1} = \begin{bmatrix} \Phi_i \\ \\ \Omega_{i+1} \end{bmatrix}$$

$$\vec{b}_{i+1} = \begin{bmatrix} \vec{b}_i \\ \\ \vec{a}_{i+1} \end{bmatrix}$$

If an arbitrary vector $\vec{d}$ fulfills

$$\Phi_{i+1}\vec{d} = \vec{b}_{i+1},$$

then it also fulfills

$$\Phi_i\vec{d} = \vec{b}_i$$

Let

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

where the partial vectors are expanded as $\bar{C} = (C_0, \ldots, C_{k-1})$, $\bar{E} = (E_0, \ldots, E_k)$, $\bar{F} = (F_0, \ldots, F_{n+1})$ and $\bar{G} = (G_0, \ldots, G_n, G_{null})$, be a solution to the system of equations on step $i + 1$, such that

$$\Phi_{i+1}\vec{c} = \vec{b}_{i+1}$$

For convenience we will say that

$$(\psi_M^0, \psi_A^0, \psi_I^0) = (\bot^{E_0}\cdot C_0\cdot \ldots \cdot C_{k-1}\cdot \bot^{E_k}, \varepsilon, \varepsilon)$$

and that for all $j \in \{1, \ldots, i+1\}$,

$$(\psi_M^j, \psi_A^j, \psi_I^j) = (f_{j-1} \circ \ldots \circ f_0)(\psi_M^0, \psi_A^0, \psi_I^0)$$

Now, let

$$\begin{aligned}
\chi_M^i &= \bot^{d_0}\cdot z_0\cdot \ldots \cdot z_{k'-1}\cdot \bot^{d_{k'}} \\
\chi_A^i &= p_0\cdot \ldots \cdot p_{\ell'} \\
\psi_M^i &= \bot^{L_0}\cdot J_0\cdot \ldots \cdot J_{k'-1}\cdot \bot^{L_{k'}} \\
\psi_A^i &= K_0\cdot \ldots \cdot K_{\ell'} \\
\varphi_M^i &= \bot^{P_0}\cdot M_0\cdot \ldots \cdot M_{k'-1}\cdot \bot^{P_{k'}} \\
\varphi_A^i &= N_0\cdot \ldots \cdot N_{\ell'}
\end{aligned}$$

First we will analyze what happens with the control stacks in case φ_I^i and ψ_I^i do not represent execution states and $f_i \in \{\mathsf{OP\_IF}, \mathsf{OP\_ELSE}, \mathsf{OP\_ENDIF}\}$.

- Let $f_i = \mathsf{OP\_IF}$. Then,

$$\begin{aligned}
\varphi_I^{i+1} &= 0\cdot \varphi_I^i \\
&= 0\cdot \psi_I^i \\
&= \psi_I^{i+1}
\end{aligned}$$

- Let $f_i = \textsf{OP\_ELSE}$. If $\textsf{top}(\varphi_I^i) = 0$, then

$$\begin{aligned}\varphi_I^{i+1} &= 1{\cdot}\textsf{tail}(\varphi_I^i)\\ &= 1{\cdot}\textsf{tail}(\psi_I^i)\\ &= \psi_I^{i+1}\end{aligned}$$

  If $\textsf{top}(\varphi_I^i) = 1$, then

$$\begin{aligned}\varphi_I^{i+1} &= 0{\cdot}\textsf{tail}(\varphi_I^i)\\ &= 0{\cdot}\textsf{tail}(\psi_I^i)\\ &= \psi_I^{i+1}\end{aligned}$$

- Let $f_i = \textsf{OP\_ENDIF}$. Then,

$$\begin{aligned}\varphi_I^{i+1} &= \textsf{tail}(\varphi_I^i)\\ &= \textsf{tail}(\psi_I^i)\\ &= \psi_I^{i+1}\end{aligned}$$

Now we will analyze what happens with the trio of stacks in case φ_I^i and ψ_I^i do represent execution states. We will describe the same two cases as before ($\textsf{OP\_PUSH}_C$ and $\textsf{OP\_IFDUP}$) but the full analysis of all the operators will be detailed in appendix G. Let $f_i = \textsf{OP\_PUSH}_C$. We know from the construction of the system of equations that $G_{null} = 0$ and that $G_i = C$. Therefore, if we assign $\vec{x} = \vec{c}$, we have that

$$
\begin{aligned}
\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \chi_M^i \\
&= \perp^{G_{null}} \cdot G_i \cdot \psi_M^i \\
&= \perp^0 \cdot C \cdot \psi_M^i \\
&= \psi_M^{i+1} \\
\chi_A^{i+1} &= \chi_A^i \\
&= \psi_A^i \\
&= \psi_A^{i+1} \\
\varphi_I^{i+1} &= \varphi_I^i \\
&= \psi_I^i \\
&= \psi_I^{i+1}
\end{aligned}
$$

Similarly, let $f_i = \textsf{OP\_IFDUP}$. If $\textsf{top}_\perp(\varphi_M^i) > 0$, then

$$
\begin{aligned}
J_0 &= F_i + 1 \\
&\geq 1 \\
&> 0
\end{aligned}
$$

We also know that $L_0 = 0$ and that $G_{null} = 0$. Thus, if we assign $\vec{x} = \vec{c}$, we have that

$$
\begin{aligned}
\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_0 \cdot \chi_M^i \\
&= \perp^{G_{null}} \cdot J_0 \cdot \psi_M^i \\
&= \perp^0 \cdot J_0 \cdot \psi_M^i \\
&= \psi_M^{i+1}
\end{aligned}
$$

Conversely, if $\textsf{top}_\perp(\varphi_M^i) < 0$, then

$$\begin{aligned} J_0 &= -F_i - 1 \\ &\leq -1 \\ &< 0 \end{aligned}$$

Once again, we know that $L_0 = 0$ and that $G_{null} = 0$. Thus, if we assign $\vec{x} = \vec{c}$, we have that

$$\begin{aligned} \chi_M^{i+1} &= \bot^{w_{null}} \cdot z_0 \cdot \chi_M^i \\ &= \bot^{G_{null}} \cdot J_0 \cdot \psi_M^i \\ &= \bot^0 \cdot J_0 \cdot \psi_M^i \\ &= \psi_M^{i+1} \end{aligned}$$

Finally, if $\mathsf{top}_\bot(\varphi_M^i) = 0$, then

$$J_0 = 0$$

Once more, we know that $L_0 = 0$ and that $G_{null} = 0$. Thus, if we assign $\vec{x} = \vec{c}$, we have that

$$\begin{aligned} \chi_M^{i+1} &= \chi_M^i \\ &= \psi_M^i \\ &= \psi_M^{i+1} \end{aligned}$$

In any case the value of the alt variable stack and the control stacks does not change. Consequently,

$$\begin{aligned}
\chi_A^{i+1} &= \chi_A^i \\
&= \psi_A^i \\
&= \psi_A^{i+1} \\
\varphi_I^{i+1} &= \varphi_I^i \\
&= \psi_I^i \\
&= \psi_I^{i+1}
\end{aligned}$$

As previously stated the full analysis for the complete set of operators is detailed in appendix G. ■

From Claim 5.14.1 we can conclude that there exists a system of equations $\Phi_{n+1}, \vec{b}_{n+1}$, such that for any arbitrary solution

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T$$

with $\bar{C} = (C_0, \ldots, C_{k-1})$ and $\bar{E} = (E_0, \ldots, E_k)$, such that

$$\Phi_{n+1}\vec{c} = \vec{b}_{n+1}$$

If we assign

$$\begin{aligned}
(\psi_M^{n+1}, \psi_A^{n+1}, \psi_I^{n+1}) &= (f_n \circ \ldots \circ f_0)(\bot^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \bot^{E_k}, \varepsilon, \varepsilon) \\
\vec{x} &= \vec{c},
\end{aligned}$$

then

$$(\chi_M^{n+1}, \chi_A^{n+1}, \varepsilon) = (\psi_M^{n+1}, \psi_A^{n+1}, \psi_I^{n+1})$$

This means that any solution to the system of equations can result in a stack that follows the same execution branch of the original valid unlocking stack. However, we also need to ensure that our final stack is valid, which translates concretely into making sure that the top of the final stack is a nonzero integer. In order to accomplish this, we will need to incorporate a few additional equations to the system. We will assign

$$\chi_M^{n+1} = \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}$$
$$\chi_A^{n+1} = p_0 \cdot \ldots \cdot p_{\ell'}$$

Based on these values we will add a final set of equations to the system represented by $\Omega_{n+2}, \vec{a}_{n+2}$. Once again we will detail these as equations, as opposed to a matrix and a vector, for convenience, and we will establish an upper bound on the biggest element between Ω_{n+2} and $\vec{a}_{n+2}$. We use $a_{\max}$ to refer to this element. The basic equation that we will add is

$$d_0 = 0$$

However, if $\mathsf{top}_\bot(\varphi_M^{n+1}) > 0$, then we also add

$$z_0 = v_{n+1} + 1$$

Conversely, if $\mathsf{top}_\bot(\varphi_M^{n+1}) < 0$, then we add

$$z_0 = -v_{n+1} - 1$$

We can easily note that in all of these equations the maximum coefficient fulfills

$$\begin{aligned}|a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)),\end{aligned}$$

for some polynomial p_{coef}. Let's note that if $\mathsf{top}_\perp(\varphi_M^{n+1}) > 0$, then

$$\begin{aligned}z_0 &= v_i + 1 \\ &\geq 1 \\ &> 0\end{aligned}$$

and that if $\mathsf{top}_\perp(\varphi_M^{n+1}) < 0$, then

$$\begin{aligned}z_0 &= -v_i - 1 \\ &\leq -1 \\ &< 0\end{aligned}$$

Let us now finish by showing how this relates to the enunciated result. Let

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T$$

with $\bar{C} = (C_0, \ldots, C_{k-1})$ and $\bar{E} = (E_0, \ldots, E_k)$ be a solution to the final system, represented by

$$\Phi_{n+2} = \begin{bmatrix} \Phi_{n+1} \\ \Omega_{n+2} \end{bmatrix} \qquad \vec{b}_{n+2} = \begin{bmatrix} \vec{b}_{n+1} \\ \vec{a}_{n+2} \end{bmatrix}$$

In addition, let

$$\begin{aligned}(\psi_M^{n+1}, \psi_A^{n+1}, \varepsilon) &= (f_n \circ \ldots \circ f_0)(\perp^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \perp^{E_k}, \varepsilon, \varepsilon) \\ &= (\perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}}, K_0 \cdot \ldots \cdot K_{\ell'}, \varepsilon)\end{aligned}$$

We know from the previous analysis that if we assign $\vec{x} = \vec{c}$, we will have that

$$(\chi_M^{n+1}, \chi_A^{n+1}) = (\psi_M^{n+1}, \psi_A^{n+1})$$

Given that $\vec{c}$ fulfills the last equations we added, we can assert that $L_0 = 0$ and that $J_0 \neq 0$. Thus,

$$\psi_M^{n+1} = \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}}$$

is a valid final stack. ■

EXAMPLE 5.15. To better illustrate how the construction of the aforementioned system of equations works, we will show a simple example of a script for which we have a basic solution. Let

$$\begin{aligned} S = {} & \mathsf{OP\_2DUP} \cdot \mathsf{OP\_PICK} \cdot \mathsf{OP\_ADD} \cdot \mathsf{OP\_ROT} \cdot \\ & \qquad \mathsf{OP\_PICK} \cdot \mathsf{OP\_ROT} \cdot \mathsf{OP\_ADD} \cdot \mathsf{OP\_EQUAL} \end{aligned}$$

First, we will show that it is an unlockable script by presenting an unlocking stack and showing that its execution is valid. Let

$$\varphi_M^0 = \perp^0 \cdot 5 \cdot \perp^0 \cdot 8 \cdot \perp^2 \cdot 6 \cdot \perp^3 \cdot 9 \cdot \perp^0$$

Given that no operator in the script modifies the alt stack, it will not change throughout the execution of the script. Likewise, no operator in the script modifies the control stack, so it will not change either. Let

$$(\varphi_M^i, \varepsilon, \varepsilon) = (f_{i-1} \circ \ldots \circ f_0)(\varphi_M^0, \varepsilon, \varepsilon)$$

for all $i \in \{1, \ldots, n+1\}$, we will show the value of the main stack on each step of the execution.

$$\begin{aligned}
\varphi_M^0 &= \bot^0{\cdot}5{\cdot}\bot^0{\cdot}8{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^1 &= \bot^0{\cdot}5{\cdot}\bot^0{\cdot}8{\cdot}\bot^0{\cdot}5{\cdot}\bot^0{\cdot}8{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^2 &= \bot^0{\cdot}6{\cdot}\bot^0{\cdot}8{\cdot}\bot^0{\cdot}5{\cdot}\bot^0{\cdot}8{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^3 &= \bot^0{\cdot}14{\cdot}\bot^0{\cdot}5{\cdot}\bot^0{\cdot}8{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^4 &= \bot^0{\cdot}8{\cdot}\bot^0{\cdot}14{\cdot}\bot^0{\cdot}5{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^5 &= \bot^0{\cdot}9{\cdot}\bot^0{\cdot}14{\cdot}\bot^0{\cdot}5{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^6 &= \bot^0{\cdot}5{\cdot}\bot^0{\cdot}9{\cdot}\bot^0{\cdot}14{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^7 &= \bot^0{\cdot}14{\cdot}\bot^0{\cdot}14{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0 \\
\varphi_M^8 &= \bot^0{\cdot}1{\cdot}\bot^2{\cdot}6{\cdot}\bot^3{\cdot}9{\cdot}\bot^0
\end{aligned}$$

As we can see, $\mathsf{top}_\bot(\varphi_M^8) \notin \{0, \bot\}$ and therefore it is a valid final main stack. Now, based on the execution over this stack we will construct the aforementioned sequence of variable stacks and its corresponding system of equations. It is important to note that we will only show each equation once (i.e. any repeated equations will be left out because they do not add additional conditions). Once again, we will disregard the variable alt stack, for it will not change.

- *Initial stack.*

$$\chi_M^0 = \bot^{g_0}{\cdot}x_0{\cdot}\bot^{g_1}{\cdot}x_1{\cdot}\bot^{g_2}{\cdot}x_2{\cdot}\bot^{g_3}{\cdot}x_3{\cdot}\bot^{g_4}$$

- $f_0 =$ OP_2DUP

  Variable stack.

$$\chi_M^1 = \bot^{w_{null}} \cdot x_0 \cdot \bot^{w_{null}} \cdot x_1 \cdot \bot^{g_0} \cdot x_0 \cdot \bot^{g_1} \cdot x_1 \cdot \bot^{g_2} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$g_0 = 0$$
$$g_1 = 0$$
$$w_{null} = 0$$

- $f_1 =$ OP_PICK

  Variable stack.

$$\chi_M^2 = \bot^{w_{null}} \cdot x_2 \cdot \bot^{w_{null}} \cdot x_1 \cdot \bot^{g_0} \cdot x_0 \cdot \bot^{g_1} \cdot x_1 \cdot \bot^{g_2} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$x_0 - g_0 - g_1 - g_2 - w_{null} = 3$$

- $f_2 =$ OP_ADD

  Variable stack.

$$\chi_M^3 = \bot^{w_{null}} \cdot w_2 \cdot \bot^{g_0} \cdot x_0 \cdot \bot^{g_1} \cdot x_1 \cdot \bot^{g_2} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$x_1 + x_2 - w_2 = 0$$

- $f_3 =$ OP_ROT

  Variable stack.

$$\chi_M^4 = \bot^{w_{null}} \cdot x_1 \cdot \bot^{w_{null}} \cdot w_2 \cdot \bot^{g_0} \cdot x_0 \cdot \bot^{v_3} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

Equations.

$$g_0 + g_1 + w_{null} = 0$$

$$g_1 + g_2 - v_3 = 0$$

- $f_4 =$ OP_PICK

  Variable stack.

$$\chi_M^5 = \bot^{w_{null}} \cdot x_3 \cdot \bot^{w_{null}} \cdot w_2 \cdot \bot^{g_0} \cdot x_0 \cdot \bot^{v_3} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$x_1 - g_0 - g_3 - v_3 - w_{null} = 3$$

- $f_5 =$ OP_ROT

  Variable stack.

$$\chi_M^6 = \bot^{w_{null}} \cdot x_0 \cdot \bot^{w_{null}} \cdot x_3 \bot^{w_{null}} \cdot w_2 \cdot \bot^{v_5} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$g_0 + 2w_{null} = 0$$

$$g_0 + v_3 - v_5 = 0$$

- $f_6 =$ OP_ADD

  Variable stack.

$$\chi_M^7 = \bot^{w_{null}} \cdot w_6 \cdot \bot^{w_{null}} \cdot w_2 \cdot \bot^{v_5} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

Equations.

$$x_0 + x_3 - w_6 = 0$$

- $f_7 =$ OP_EQUAL

  Variable stack.

$$\chi_M^8 = \bot^{w_{null}} \cdot w_7 \cdot \bot^{v_5} \cdot x_2 \cdot \bot^{g_3} \cdot x_3 \cdot \bot^{g_4}$$

  Equations.

$$w_2 - w_6 = 0$$

$$w_7 = 1$$

- *Final equations.*

$$v_8 - w_7 = -1$$

Now, let's analyze the execution of the script by using the following solution to the system of equations:

$x_0 = 10$	$x_1 = 15$	$x_2 = 2$	$x_3 = 7$
$g_0 = 0$	$g_1 = 0$	$g_2 = 7$	$g_3 = 5$
$g_4 = 0$	$v_0 = 0$	$v_1 = 0$	$v_2 = 0$
$v_3 = 7$	$v_4 = 0$	$v_5 = 7$	$v_6 = 0$
$v_7 = 0$	$v_8 = 0$	$w_0 = 0$	$w_1 = 0$
$w_2 = 17$	$w_3 = 0$	$w_4 = 0$	$w_5 = 0$
$w_6 = 17$	$w_7 = 1$	$w_{null} = 0$	

Let

$$\psi_M^0 = \bot^0 \cdot 10 \cdot \bot^0 \cdot 15 \cdot \bot^7 \cdot 2 \cdot \bot^5 \cdot 7 \cdot \bot^0$$

Once again, given that no operator in the script modifies the alt stack, it will not change throughout the execution of the script. Likewise, no operator in the script modifies the control stack, so it will not change either. Let

$$(\psi_M^i, \varepsilon, \varepsilon) = (f_{i-1} \circ \ldots \circ f_0)(\psi_M^0, \varepsilon, \varepsilon)$$

for all $i \in \{1, \ldots, n+1\}$, we will show the value of the main stack on each step of the execution.

$$\begin{aligned}
\psi_M^0 &= \perp^0 \cdot 10 \cdot \perp^0 \cdot 15 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^1 &= \perp^0 \cdot 10 \cdot \perp^0 \cdot 15 \cdot \perp^0 \cdot 10 \cdot \perp^0 \cdot 15 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^2 &= \perp^0 \cdot 2 \cdot \perp^0 \cdot 15 \cdot \perp^0 \cdot 10 \cdot \perp^0 \cdot 15 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^3 &= \perp^0 \cdot 17 \cdot \perp^0 \cdot 10 \cdot \perp^0 \cdot 15 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^4 &= \perp^0 \cdot 15 \cdot \perp^0 \cdot 17 \cdot \perp^0 \cdot 10 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^5 &= \perp^0 \cdot 7 \cdot \perp^0 \cdot 17 \cdot \perp^0 \cdot 10 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^6 &= \perp^0 \cdot 10 \cdot \perp^0 \cdot 7 \cdot \perp^0 \cdot 17 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^7 &= \perp^0 \cdot 17 \cdot \perp^0 \cdot 17 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0 \\
\psi_M^8 &= \perp^0 \cdot 1 \cdot \perp^7 \cdot 2 \cdot \perp^5 \cdot 7 \cdot \perp^0
\end{aligned}$$

As we can see, $\mathsf{top}_\perp(\psi_M^8) \notin \{0, \perp\}$ and therefore it is a valid final main stack. Thus, the proposed solution to the system of equations results in a pair of valid unlocking stacks for S. ■

Polynomial solution for the system. Now that we know that for each unlockable script we can construct a solvable system of equations from whose solutions we can construct valid stacks to unlock the script, we aim to use the classic results for integer linear programming to set an upper bound on the smallest solutions to these systems of equations. However, the fact that we only restrict the domains of some of the used variables prohibits us from using these results directly. Consequently, we will need to prove that these results are still applicable in this situation.

LEMMA 5.16. *Let $A = [a_{ij}]_{m\times n}$ and $\vec{b} = [b_i]_{m\times 1}$, with $a_{ij}, b_i \in \mathbb{Z}$ for all $(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,n\}$. Let us consider the system of equations defined by A and $\vec{b}$,*

$$A\vec{x} = \vec{b}, \quad \vec{x} \textit{ integer}$$

Let $\vec{c} = (c_1,\ldots,c_n)^T$ be a valid solution for said system. There exists a vector $\vec{d} = (d_1,\ldots,d_n)^T$, with $|d_j| \le (an)^{p(m)}$, where p is a polynomial and a is the maximum element in the system of equations, $a = \max_{(i,j)\in\{1,\ldots,m\}\times\{1,\ldots,n\}}\{|a_{ij}|, |b_i|\}$, such that $A\vec{d} = \vec{b}$ and for all $j \in \{1,\ldots,n\}$

$$d_j \ge 0 \quad \textit{if} \quad c_j \ge 0$$

$$d_j \le 0 \quad \textit{if} \quad c_j < 0$$

PROOF. Let $A = [a_{ij}]_{m\times n}$ and $\vec{b} = [b_i]_{m\times 1}$ be an arbitrary system of equations, with $a_{ij}, b_j \in \mathbb{Z}$ for all $(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,n\}$. In addition, let $\vec{c} = (c_1,\ldots,c_n)^T$ be a valid solution for the system

$$A\vec{x} = \vec{b}, \quad \vec{x} \text{ integer}$$

Let $\{j_1,\ldots,j_k\} \subseteq \{1,\ldots,n\}$ with $j_i < j_{i+1}$ for each $i \in \{1,\ldots,k-1\}$, be the set of subindices for which $c_{j_i} < 0$ for each $i \in \{1,\ldots,k\}$ and $c_j \ge 0$ for each $j \in \{1,\ldots,n\} - \{j_1,\ldots,j_k\}$. We can now analyze what would happen if we modified A to create A_{-j}, such that

$$A_{-i} = \begin{bmatrix} a_{11} & \cdots & a_{1(j-1)} & -a_{1j} & a_{1(j+1)} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(j-1)} & -a_{ij} & a_{i(j+1)} & \cdots & a_{in} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m(j-1)} & -a_{mj} & a_{m(j+1)} & \cdots & a_{mn} \end{bmatrix}$$

It is clear that $\vec{c}_{-j}$ will be a valid solution for the new system $A_{-j}\vec{x} = \vec{b}$, where

$$\vec{c}_{-j} = \begin{bmatrix} c_1 \\ \vdots \\ c_{j-1} \\ -c_j \\ c_{j+1} \\ \vdots \\ c_n \end{bmatrix}$$

This is because if we conduct the multiplication, we have

$$\begin{aligned}
A_{-j}\vec{c}_{-j} &= \begin{bmatrix} a_{11}c_1 + \cdots + (-a_{1j})(-c_j) + \cdots + a_{1n}c_n \\ \vdots \\ a_{m1}c_1 + \cdots + (-a_{mj})(-c_j) + \cdots + a_{mn}c_n \end{bmatrix} \\
&= \begin{bmatrix} a_{11}c_1 + \cdots + a_{1j}c_j + \cdots + a_{1n}c_n \\ \vdots \\ a_{m1}c_1 + \cdots + a_{mj}c_j + \cdots + a_{mn}c_n \end{bmatrix} \\
&= A\vec{c} \\
&= \vec{b}
\end{aligned}$$

If we now carry out the transformation for each of the subindices that we singled out previously, we will end up with the system

$$A_{-(j_1,\ldots,j_k)}\vec{x} = \vec{b} \tag{5.9}$$

For this system we know that $\vec{c}_{-(j_1,\ldots,j_k)}$ will be a valid solution. Given that we performed the modification for each negative element, we can describe this vector as

$$\vec{c}_{-(j_1,\ldots,j_k)} = \begin{bmatrix} |c_1| \\ \vdots \\ |c_n| \end{bmatrix}$$

This means that the system of equations

$$A_{-(j_1,\ldots,j_k)}\vec{x} = \vec{b}, \quad \vec{x} \geq 0, \text{ integer}$$

has at least one solution. Consequently, from (Papadimitriou, 1981) we can conclude that there must exist a second vector $\vec{d} = (d_1, \ldots, d_n)^T$, with $\vec{d} \geq 0$ and $|d_i| \leq (an)^{p(m)}$ for all $i \in \{1, \ldots, n\}$, where p is a polynomial and $a = \max_{(i,j)\in\{1,\ldots,m\}\times\{1,\ldots,n\}}\{|a_{ij}|, |b_i|\}$, such that $A_{-(j_1,\ldots,j_k)}\vec{d} = \vec{b}$. By inverting the previous transformation to the system we can find a vector that fulfills the properties described in the lemma. Specifically, if we construct $\vec{d}_{-(j_1,\ldots,j_k)}$ as

$$\vec{d}_{-(j_1,\ldots,j_k)} = \begin{bmatrix} d_1 \\ \vdots \\ d_{j_1-1} \\ -d_{j_1} \\ d_{j_1+1} \\ \vdots \\ d_{j_k-1} \\ -d_{j_k} \\ d_{j_k+1} \\ \vdots \\ d_n \end{bmatrix}$$

we can prove that this vector satisfies the requirements. First of all, it is evident that

$$
\begin{aligned}
A\vec{d}_{-(j_1,\ldots,j_k)} &= A_{-(j_1,\ldots,j_k)}\vec{d} \\
&= \vec{b}
\end{aligned}
$$

Moreover, as we know, the only elements that are negative in $\vec{c}$ are the ones indicated by the subindices $\{j_1, \ldots, j_k\}$. If we denote $\vec{d}_{-(j_1,\ldots,j_k)} = (d'_1, \ldots, d'_n)$, given that $\vec{d} \geq 0$, it is evident that for all $j \in \{1, \ldots, n\}$

$$
d'_j \geq 0 \quad \text{if} \quad c_j \geq 0
$$

$$
d'_j \leq 0 \quad \text{if} \quad c_j < 0
$$

which means that this new vector satisfies the sign condition. Lastly, it is clear that for all $j \in \{1, \ldots, n\}$

$$
|d'_j| = |d_j| \leq (an)^{p(m)}
$$

■

5.5. Putting everything together

In this section we will combine the previous results in order to set an upper bound on the size of the smallest solution for an arbitrary unlockable script.

THEOREM 5.17. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an arbitrary unlockable script with the stricter definitions for the cryptographic operators OP_HASH160, OP_CHECKSIG, OP_CHECKSIGVERIFY presented at the beginning of section 5.4. There exists a stack $\varphi \in \mathbb{Z}^*_\perp$, such that*

$$
(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varepsilon),
$$

where

$$|\varphi_M| \geq 1$$

$$\textbf{\textit{top}}_\perp(\varphi_M) \notin \{0, \perp\}$$

$$\|\varphi\| \leq p_{fin}(n, \log_2(\textbf{\textit{maxpush}}(S) + 1)),$$

for some fixed polynomial p_{fin}, independent of S.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an arbitrary unlockable script with the stricter definitions for the three cryptographic operators OP_HASH160, OP_CHECKSIG and OP_CHECKSIGVERIFY. Let $\varphi \in \mathbb{Z}_\perp^*$, such that

$$(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varepsilon),$$

where

$$|\varphi_M| \geq 1$$

$$\textsf{top}_\perp(\varphi_M) \notin \{0, \perp\}$$

From Lemma 5.13 we know that there must exist a stack φ_2, with $\textsf{elemnr}(\varphi_2) \leq 3n + 4$, such that

$$(f_n \circ \ldots \circ f_0)(\varphi_2, \varepsilon, \varepsilon) = (\varphi_M^2, \varphi_A^2, \varepsilon),$$

where

$$|\varphi_M^2| \geq 1$$

$$\textsf{top}_\perp(\varphi_M^2) \notin \{0, \perp\}$$

Let $\varphi_2 = \bot^{h_0}{\cdot}A_0{\cdot}\dots{\cdot}A_{k-1}{\cdot}\bot^{h_k}$. From Theorem 5.14 we know that there must exist a solvable system of equations $\Phi\vec{x} = \vec{b}$, with

$$\vec{x} = (\bar{x}, \bar{g}, \bar{v}, \bar{w})^T,$$

where $\bar{x}$ and $\bar{g}$ have k and $k+1$ elements, respectively, and where $\bar{v}$ and $\bar{w}$ both have $n+2$ elements each, constrained by $\bar{g}, \bar{v} \geq 0$, such that

$$|a_{\max}| \leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_2), \mathsf{maxpush}(S)),$$

where $a_{\max}$ is the element of maximum absolute value between Φ and $\vec{b}$ and p_{coef} is some polynomial, and from any arbitrary solution we can construct a valid solution.

Let Φ be a matrix of dimensions $m \times \eta$. From Lemma 5.16 we can gather that there must exist a solution to the system

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

such that for every element C in $\bar{C}$, E in $\bar{E}$, F in $\bar{F}$ and G in $\bar{G}$,

$$|C|, |E|, |F|, |G| \leq (|a_{\max}|\eta)^{p(m)}$$

for some polynomial p. Let $\bar{C} = (C_0, \dots, C_{k-1})$ and $\bar{E} = (E_0, \dots, E_k)$. We will define

$$\varphi_3 = \bot^{E_0}{\cdot}C_0{\cdot}\dots{\cdot}C_{k-1}{\cdot}\bot^{E_k}.$$

Let us now bound the size of φ_3. It is clear that

$$(f_n \circ \ldots \circ f_0)(\varphi_3, \varepsilon, \varepsilon) = (\varphi^3_M, \varphi^3_A, \varepsilon),$$

where

$$|\varphi^3_M| \geq 1$$
$$\mathsf{top}_\perp(\varphi^3_M) \notin \{0, \perp\}$$

In addition, we can set an upper bound on the size of this stack as follows:

$$\|\varphi_3\| \leq (2k+1)\log_2((|a_{\max}|\eta)^{p(m)} + 1)$$

Nevertheless, this bound on φ_3 is not readily usable, as it does not directly relate the size of the stack with the size of the script. Thus, we will need to simplify this expression by utilizing certain upper bounds on the different terms in it. Namely,

$$k = \mathsf{elemnr}(\varphi_2) \leq 3n + 4$$
$$|a_{\max}| \leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_2), \mathsf{maxpush}(S))$$
$$\eta \leq 2k + 1 + 2(n+2) \leq 8n + 13$$
$$m \leq 6(n+1) + 2$$

Most of these transformations stem from the Lemmas referenced above. However, we also added an upper bound $m \leq 6(n+1)+2$ on the maximum amount of equations in the system, which is obtained from the construction of the system of equations. Specifically, for each operator in the script there are at most 6 equations added to the system and at the end there are 2 additional equations that are incorporated. Now, we can establish a more useful bound on the size of φ_3.

$$\begin{aligned}
\|\varphi_3\| &\leq (2k+1)\log_2\big((|a_{\max}|\eta)^{p(m)}+1\big)\\
&\leq (6n+9)\log_2\Big(\big(p_{coef}(2^n,\mathsf{elemnr}(\varphi_2),\mathsf{maxpush}(S))(8n+13)\big)^{p_1(n)}+1\Big)\\
&\leq (6n+9)\log_2\Big(p_2(2^n,\mathsf{maxpush}(S))^{p_1(n)}+1\Big)\\
&\leq (6n+9)p_3\big(n,\log_2(\mathsf{maxpush}(S)+1)\big)\\
&\leq p_{fin}\big(n,\log_2(\mathsf{maxpush}(S)+1)\big),
\end{aligned}$$

for some polynomials p_1, p_2, p_3, p_{fin}. ■

6. CONCLUSIONS

In this work we established a mathematical formalization for Script, allowing us to establish a standardized way of studying the language. Currently, Script is almost exclusively used for establishing the most basic unlocking conditions. One of the main reasons for this is that the nodes in the network tend to favor standard locking scripts because they guarantee that their executions will be short and efficient. Our work provides a more extensive understanding of the language, promoting further developement of efficient algorithms for dealing with general scripts. We expect that this developement will help and promote users to write non-trivial spending conditions, ultimately making the usage of non-standard scripts a widespread practice.

We have also defined the problem of script unlockability. Being able to solve this problem efficiently would help prevent users from locking funds in unspendable transactions and free up memory space from the nodes' executions by discarding unspendable transactions. We also proved that the problem of unlockability, when setting harsher restrictions for the use of cryptographic operators, is NP complete.

This means that there does not exist an efficient algorithm that can determine for all possible locking scripts which are unlockable. However, SAT solvers have advanced to the point that it is feasible to determine whether a formula is satisfiable for reasonable inputs. Therefore, moving forward we plan to search for a transformation that would convert a script into a formula that is satisfiable if and only if the script is unlockable. This will allow us to use a SAT solver to determine script unlockability. Such a result will set the foundation for developing an algorithm to be implemented in electronic wallets and in the nodes of the Bitcoin network for the purposes detailed previously.

In the future, we would also like to study different subsections of Script in search of tractable fragments, for which unlockability could be efficiently determined. We would also like to determine the complexity of Script in case we considered the unrestricted versions of the cryptographic commands. It would be interesting to determine how much the problem would scale in the complexity hierarchy under these circumstances.

REFERENCES

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of np-completeness*. W. H. Freeman.

Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system* (Tech. Rep.). Manubot.

Papadimitriou, C. H. (1981, October). On the complexity of integer programming. *Journal of the Association for Computing Machinery*, *28*(4), 765-768. Retrieved from https://lara.epfl.ch/w/_media/papadimitriou81complexityintegerprogramming.pdf

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Script implementation: security improvements. (2010, July). https://github.com/bitcoin/bitcoin/commit/6ff5f718b6a67797b2b3bab8905d607ad216ee21.

Script specification. (2021, February). https://en.bitcoin.it/wiki/Script.

APPENDIX

A. INTEGER LINEAR PROGRAMMING SCRIPT UNLOCKABILITY REDUCTION

The following is an algorithm that constructs a script S from a system of equations $A, \vec{b}$, such that S is unlockable if and only if $A, \vec{b}$ is solvable. We assume that each equation has at least one nonzero coefficient. This algorithm serves as proof that script unlockability is NP hard. For an explanation as to how the algorithm works, see section 4.2.

```
procedure ILPToScript($A_{(n+1)\times(n+1)}$, $\vec{b}_{(n+1)\times 1}$)
    $S := \varepsilon$
    for $i = 0..n$ do
        $first :=$ True
        for $j = 0..n$ do
            $a := |a_{ij}|$
            if $a > 0$ then
                $powers := 0$
                if $first$ then
                    $S := S\cdot$OP_PUSH$_j\cdot$OP_PICK
                else
                    $S := S\cdot$OP_PUSH$_{j+1}\cdot$OP_PICK
                end if
                while $a > 0$ do
                    if $a \mod 2 = 1$ then
                        $S := S\cdot$OP_DUP$\cdot$OP_DUP$\cdot$OP_ADD
                        $powers++$
                    else
                        $S := S\cdot$OP_DUP$\cdot$OP_ADD
                    end if
                $a >> 1$
            end while
            $S := S$·OP_DROP
            for $k = 2..powers$ do
                $S := S$·OP_ADD
            end for
            if $first$ then
                if $a_{ij} < 0$ then
                    $S := S$·OP_PUSH$_0$·OP_SWAP·OP_SUB
                end if
                $first :=$ False
            else
                if $a_{ij} \geq 0$ then
                    $S := S$·OP_ADD
                else
                    $S := S$·OP_SUB
                end if
            end if
        end if
    end for
    $S := S$·OP_PUSH$_{b_j}$·OP_EQUALVERIFY
  end for
  $S := S$·OP_PUSH$_1$
  return $S$
end procedure
```

B. OPERATOR SEMANTICS WITH ANONYMOUS ELEMENTS

In this section we will formalize the definition of the Script operators over a stack with anonymous elements. For each operator we will describe the conditions that need to be fulfilled for the operator to execute succesfully. In other words, let $f \in O$, $\varphi_M \in \mathbb{Z}_\perp^*$ and $\varphi_A \in \mathbb{Z}^*$. If (φ_M, φ_A) does not fulfill the conditions associated with f, then

$$f(\varphi_M, \varphi_A) = \square$$

It is also important to note that we will not include the control stack in the analysis of the basic operators (i.e. the operators that do not correspond to flow control). Let $f \in O - \{\mathsf{OP\_IF}, \mathsf{OP\_ELSE}, \mathsf{OP\_ENDIF}\}$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$. If φ_I does not represent an execution state, then

$$f(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M, \varphi_A, \varphi_I)$$

Therefore, we will assume that the control stack represents an execution state. Now, let

$$\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$$
$$\varphi_A = B_0 \cdot \ldots \cdot B_\ell \in \mathbb{Z}^*$$

- $\mathsf{OP\_PUSH}_C$

  Conditions

  None

  Value

  $$\mathsf{OP\_PUSH}_C(\varphi_M, \varphi_A) = (\perp^0 \cdot C \cdot \varphi_M, \varphi_A)$$

- OP_VERIFY

  Conditions

$$|\varphi_M| \geq 1$$
$$\mathsf{top}_\perp(\varphi_M) \notin \{0, \perp\}$$

  Value

$$\mathsf{OP\_VERIFY}(\varphi_M, \varphi_A) = (\mathsf{tail}_\perp(\varphi_M), \varphi_A)$$

- OP_TOALTSTACK

  Conditions

$$|\varphi_M| \geq 1$$
$$\mathsf{top}_\perp(\varphi_M) \neq \perp$$

  Value

$$\mathsf{OP\_TOALTSTACK}(\varphi_M, \varphi_A) = (\mathsf{tail}_\perp(\varphi_M), \mathsf{top}_\perp(\varphi_M) \cdot \varphi_A)$$

- OP_FROMALTSTACK

  Conditions

$$|\varphi_A| \geq 1$$

  Value

$$\mathsf{OP\_FROMALTSTACK}_\perp(\varphi_M, \varphi_A) = (\perp^0 \cdot \mathsf{top}(\varphi_A) \cdot \varphi_M, \mathsf{tail}(\varphi_A))$$

- OP_IFDUP

  Conditions

$$|\varphi_M| \geq 1$$
$$\mathsf{top}_\perp(\varphi_M) \neq \perp$$

Value

$$\textsf{OP\_IFDUP}(\varphi_M, \varphi_A) = \begin{cases} (\varphi_M, \varphi_A) & \text{if} \quad \textsf{top}_\perp(\varphi_M) = 0 \\ (\perp^0 \cdot \textsf{top}_\perp(\varphi_M) \cdot \varphi_M, \varphi_A) & \text{if} \quad \textsf{top}_\perp(\varphi_M) \neq 0 \end{cases}$$

- OP_DROP

  Conditions

$$|\varphi_M| \geq 1$$
$$\textsf{top}_\perp(\varphi_M) \neq \perp$$

  Value

$$\textsf{OP\_DROP}(\varphi_M, \varphi_A) = (\textsf{tail}_\perp(\varphi_M), \varphi_A)$$

- OP_DUP

  Conditions

$$|\varphi_M| \geq 1$$
$$\textsf{top}_\perp(\varphi_M) \neq \perp$$

  Value

$$\textsf{OP\_DUP}(\varphi_M, \varphi_A) = (\perp^0 \cdot \textsf{top}_\perp(\varphi_M) \cdot \varphi_M, \varphi_A)$$

- OP_NIP

  Conditions

$$|\varphi_M| \geq 2$$
$$\exists i \in \{0,1\}. \sum_{j=0}^{i} (h_j + 1) = 2$$

  Value

  Let $i \in \{0,1\}$, such that $\sum_{j=0}^{i}(h_j+1) = 2$,

$$\textsf{OP\_NIP}(\varphi_M, \varphi_A) = (\perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \perp^{h_i + h_{i+1}} \cdot A_{i+1} \cdot \ldots A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

- OP_OVER

  Conditions

$$|\varphi_M| \geq 2$$
$$\exists i \in \{0,1\}. \sum_{j=0}^{i}(h_j + 1) = 2$$

  Value

  Let $i \in \{0,1\}$, such that $\sum_{j=0}^{i}(h_j+1) = 2$,

$$\mathsf{OP\_OVER}(\varphi_M, \varphi_A) = (\bot^0 \cdot A_i \cdot \varphi_M, \varphi_A)$$

- OP_ROT

  Conditions

$$|\varphi_M| \geq 3$$
$$\exists i \in \{0,1,2\}. \sum_{j=0}^{i}(h_j + 1) = 3$$

  Value

  Let $i \in \{0,1,2\}$, such that $\sum_{j=0}^{i}(h_j+1) = 3$,

$$\mathsf{OP\_ROT}(\varphi_M, \varphi_A) = (\bot^0 \cdot A_i \cdot \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i + h_{i+1}} \cdot A_{i+1} \cdot \ldots A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_SWAP

  Conditions

$$|\varphi_M| \geq 2$$
$$\exists i \in \{0,1\}. \sum_{j=0}^{i}(h_j + 1) = 2$$

  Value

  Let $i \in \{0,1\}$, such that $\sum_{j=0}^{i}(h_j+1) = 2$,

$$\mathsf{OP\_SWAP}(\varphi_M, \varphi_A) = (\bot^0 \cdot A_i \cdot \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i + h_{i+1}} \cdot A_{i+1} \cdot \ldots A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_TUCK

  Conditions

$$|\varphi_M| \geq 2$$
$$h_0 = 0$$
$$h_1 = 0$$

  Value

$$\text{OP\_TUCK}(\varphi_M, \varphi_A) = (\perp^0 \cdot A_0 \cdot \perp^0 \cdot A_1 \cdot \perp^0 \cdot A_0 \cdot \perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

- OP_2DROP

  Conditions

$$|\varphi_M| \geq 2$$
$$h_0 = 0$$
$$h_1 = 0$$

  Value

$$\text{OP\_2DROP}(\varphi_M, \varphi_A) = (\perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

- OP_2DUP

  Conditions

$$|\varphi_M| \geq 2$$
$$h_0 = 0$$
$$h_1 = 0$$

  Value

$$\text{OP\_2DUP}(\varphi_M, \varphi_A) = (\perp^0 \cdot A_0 \cdot \perp^0 \cdot A_1 \cdot \varphi_M, \varphi_A)$$

- OP_3DUP

  Conditions

$$|\varphi_M| \geq 3$$
$$h_0 = 0$$
$$h_1 = 0$$
$$h_2 = 0$$

  Value

$$\text{OP\_3DUP}(\varphi_M, \varphi_A) = (\perp^0 \cdot A_0 \cdot \perp^0 \cdot A_1 \cdot \perp^0 \cdot A_2 \cdot \varphi_M, \varphi_A)$$

- OP_2OVER

  Conditions

$$|\varphi_M| \geq 4$$
$$\exists i \in \{0, 1, 2\}. \sum_{j=0}^{i} (h_j + 1) = 3 \wedge h_{i+1} = 0$$

  Value

  Let $i \in \{0, 1, 2\}$, such that $\sum_{j=0}^{i} (h_j + 1) = 3$,

$$\text{OP\_2OVER}(\varphi_M, \varphi_A) = (\perp^0 \cdot A_i \cdot \perp^0 \cdot A_{i+1} \cdot \varphi_M, \varphi_A)$$

- OP_2ROT

  Conditions

$$|\varphi_M| \geq 6$$
$$\exists i \in \{0, 1, 2, 3, 4\}. \sum_{j=0}^{i} (h_j + 1) = 5 \wedge h_{i+1} = 0$$

  Value

Let $i \in \{0,1,2,3,4\}$, such that $\sum_{j=0}^{i}(h_j+1)=5$,

$$\mathsf{OP\_2ROT}(\varphi_M,\varphi_A) = $$
$$(\bot^0 \cdot A_i \cdot \bot^0 \cdot A_{i+1} \cdot \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i+h_{i+2}} \cdot A_{i+2} \cdot \ldots A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_2SWAP

  Conditions

  $$|\varphi_M| \geq 4$$
  $$\exists i \in \{0,1,2\}. \sum_{j=0}^{i}(h_j+1)=3 \wedge h_{i+1}=0$$

  Value

  Let $i \in \{0,1,2\}$, such that $\sum_{j=0}^{i}(h_j+1)=3$,

$$\mathsf{OP\_2SWAP}(\varphi_M,\varphi_A) = $$
$$(\bot^0 \cdot A_i \cdot \bot^0 \cdot A_{i+1} \cdot \bot^{h_0} \cdot A_0 \cdot \ldots \cdot A_{i-1} \cdot \bot^{h_i+h_{i+2}} \cdot A_{i+2} \cdot \ldots A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_PICK

  Conditions

  $$|\varphi_M| \geq 1$$
  $$\mathsf{top}_\bot(\varphi_M) \neq \bot$$
  $$0 \leq \mathsf{top}_\bot(\varphi_M) \leq |\mathsf{tail}_\bot(\varphi_M)| - 1$$
  $$\exists i \in \{1,\ldots,k-1\}.\mathsf{top}_\bot(\varphi_M) = i - 1 + \sum_{j=1}^{i} h_j$$

  Value

  Let $i \in \{0,\ldots,k-1\}$, such that $\mathsf{top}_\bot(\varphi_M) = i-1+\sum_{j=1}^{i} h_j$,

  $$\mathsf{OP\_PICK}(\varphi_M,\varphi_A) = (\bot^0 \cdot A_i \cdot \bot^{h_1} \cdot A_1 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_ROLL

Conditions

$$|\varphi_M| \geq 1$$

$$\mathsf{top}_\perp(\varphi_M) \neq \perp$$

$$0 \leq \mathsf{top}_\perp(\varphi_M) \leq |\mathsf{tail}_\perp(\varphi_M)| - 1$$

$$\exists i \in \{1, \ldots, k-1\}.\mathsf{top}_\perp(\varphi_M) = i - 1 + \sum_{j=1}^{i} h_j$$

Value

Let $i \in \{0, \ldots, k-1\}$, such that $\mathsf{top}_\perp(\varphi_M) = i - 1 + \sum_{j=1}^{i} h_j$,

$$\mathsf{OP\_ROLL}(\varphi_M, \varphi_A) = (\perp^0 \cdot A_i \cdot \perp^{h_1} \cdot \ldots \cdot A_{i-1} \cdot \perp^{h_i + h_{i+1}} \cdot A_{i+1} \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

- OP_EQUAL

  Conditions

$$|\varphi_M| \geq 2$$

$$h_0 = 0$$

$$h_1 = 0$$

Value

$$\mathsf{OP\_EQUAL}(\varphi_M, \varphi_A) = \begin{cases} (\perp^0 \cdot 1 \cdot \perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A) & \text{if} \quad A_0 = A_1 \\ (\perp^0 \cdot 0 \cdot \perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A) & \text{if} \quad A_0 \neq A_1 \end{cases}$$

- OP_EQUALVERIFY

Conditions

$$|\varphi_M| \geq 2$$

$$h_0 = 0$$

$$h_1 = 0$$

$$A_0 = A_1$$

Value

$$\textsf{OP\_EQUALVERIFY}(\varphi_M, \varphi_A) = (\bot^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_ADD

  Conditions

$$|\varphi_M| \geq 2$$

$$h_0 = 0$$

$$h_1 = 0$$

  Value

$$\textsf{OP\_ADD}(\varphi_M, \varphi_A) = (\bot^0 \cdot (A_0 + A_1) \cdot \bot^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_SUB

  Conditions

$$|\varphi_M| \geq 2$$

$$h_0 = 0$$

$$h_1 = 0$$

  Value

$$\textsf{OP\_SUB}(\varphi_M, \varphi_A) = (\bot^0 \cdot (A_1 - A_0) \cdot \bot^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_DEPTH

  Conditions

  None

  Value

$$\mathsf{OP\_DEPTH}(\varphi_M, \varphi_A) = (\bot^0 \cdot |\varphi_M| \cdot \varphi_M, \varphi_A)$$

- OP_HASH160

  Conditions

$$|\varphi_M| \geq 1$$
$$h_0 = 0$$

  Value

  As a reminder, the function hash corresponds to applying the SHA-256 and RIPEMD-160 hashing algorithms in succession over the input.

$$\mathsf{OP\_HASH160}(\varphi_M, \varphi_A) = (\bot^0 \cdot \mathsf{hash}(A_0) \cdot \bot^{h_1} \cdot A_1 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_CHECKSIG

  Conditions

$$|\varphi_M| \geq 2$$
$$h_0 = 0$$
$$h_1 = 0$$

  Value

  As a reminder, the function chksig is defined as 1 if its second input is a valid signature for the transaction and the public key that is supplied as its first input, and 0 otherwise.

$$\mathsf{OP\_CHECKSIG}(\varphi_M, \varphi_A) = (\bot^0 \cdot \mathsf{chksig}(A_0, A_1) \cdot \bot^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \bot^{h_k}, \varphi_A)$$

- OP_CHECKSIGVERIFY

  Conditions

$$|\varphi_M| \geq 2$$

$$h_0 = 0$$

$$h_1 = 0$$

$$\mathsf{chksig}(A_0, A_1) = 1$$

  Value

  As a reminder, the function chksig is defined as 1 if its second input is a valid signature for the transaction and the public key that is supplied as its first input, and 0 otherwise.

$$\mathsf{OP\_CHECKSIGVERIFY}(\varphi_M, \varphi_A) = (\perp^{h_2} \cdot A_2 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k}, \varphi_A)$$

Now, we will define the flow control operators. It is evident that to accomplish this we will need to consider the control stack, as opposed to the previous case. Once again, we will describe the conditions that need to be fulfilled for an operator to execute succesfully. In other words, let $f \in \{\mathsf{OP\_IF}, \mathsf{OP\_ELSE}, \mathsf{OP\_ENDIF}\}$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$. If $(\varphi_M, \varphi_A, \varphi_I)$ does not fulfill the conditions associated with f, then

$$f(\varphi_M, \varphi_A, \varphi_I) = \Box$$

Now, let

$$\varphi_M = \perp^{h_0} \cdot A_0 \cdot \ldots \cdot A_{k-1} \cdot \perp^{h_k} \in \mathbb{Z}_\perp^*$$

$$\varphi_A = B_0 \cdot \ldots \cdot B_\ell \in \mathbb{Z}^*$$

$$\varphi_I = I_0 \cdot \ldots \cdot I_p \in \{0,1\}^*$$

- OP_IF

  Conditions

  None

  Value

$$\mathsf{OP\_IF}(\varphi_M, \varphi_A, \varphi_I) =$$

$$\begin{cases} \square & \text{if} \quad \big(|\varphi_M| = 0 \lor (|\varphi_M| \geq 1 \land \\ & \qquad \mathsf{top}_\perp(\varphi_M) = \perp)\big) \land \varphi_I \in \{1\}^* \\ (\mathsf{tail}_\perp(\varphi_M), \varphi_A, 1 \cdot \varphi_I) & \text{if} \quad |\varphi_M| \geq 1 \land \mathsf{top}_\perp(\varphi_M) \notin \{0, \perp\} \land \\ & \qquad \varphi_I \in \{1\}^* \\ (\mathsf{tail}_\perp(\varphi_M), \varphi_A, 0 \cdot \varphi_I) & \text{if} \quad |\varphi_M| \geq 1 \land \mathsf{top}_\perp(\varphi_M) = 0 \land \\ & \qquad \varphi_I \in \{1\}^* \\ (\varphi_M, \varphi_A, 0 \cdot \varphi_I) & \text{if} \quad \varphi_I \notin \{1\}^* \end{cases}$$

- OP_ELSE

  Conditions

$$|\varphi_I| \geq 1$$

  Value

$$\mathsf{OP\_ELSE}(\varphi_M, \varphi_A, \varphi_I) = \begin{cases} (\varphi_M, \varphi_A, 1 \cdot \mathsf{tail}(\varphi_I)) & \text{if} \quad \mathsf{top}(\varphi_I) = 0 \\ (\varphi_M, \varphi_A, 0 \cdot \mathsf{tail}(\varphi_I)) & \text{if} \quad \mathsf{top}(\varphi_I) = 1 \end{cases}$$

- OP_ENDIF

  Conditions

$$|\varphi_I| \geq 1$$

Value

$$\mathsf{OP\_ENDIF}(\varphi_M, \varphi_A, \varphi_I) = (\varphi_M, \varphi_A, \mathsf{tail}(\varphi_I))$$

C. EQUIVALENCE OF UNLOCKING SCRIPT AND UNLOCKING STACK

This section is dedicated towards proving Lemma 5.1 shown in section 5.1.

PROOF. We will prove both directions of the assertion. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$.

- ($\rightarrow$) We will first assume that S is unlockable. This means that there exists a script $S_U = g_0 \cdot \ldots \cdot g_m \in O^*$, such that

  $$(g_m \circ \ldots \cdot g_0)(\varepsilon, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varepsilon),$$

  where φ_M fulfills

  $$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi'_M, \varphi'_A, \varepsilon),$$

  with

  $$|\varphi'_M| \geq 1$$
  $$\mathsf{top}(\varphi'_M) \neq 0$$

  Thus, if we represent φ_M as a stack with anonymous elements ($\varphi_M \in \mathbb{Z}^*_\perp$), it still satisfies the conditions.

  $$(f_n \circ \ldots \circ f_0)(\varphi_M, \varepsilon, \varepsilon) = (\varphi'_M, \varphi'_A, \varepsilon),$$

  with

  $$|\varphi'_M| \geq 1$$
  $$\mathsf{top}_\perp(\varphi'_M) \notin \{0, \perp\}$$

- ($\leftarrow$) We will now assume that there exists a stack $\varphi \in \mathbb{Z}^*_\perp$, such that

$$(f_n \circ \ldots \circ f_0)(\varphi, \varepsilon, \varepsilon) = (\varphi_M, \varphi_A, \varepsilon),$$

where

$$|\varphi'_M| \geq 1$$

$$\mathsf{top}_{\perp}(\varphi'_M) \notin \{0, \perp\}$$

We can now deanonymize every anonymous element in φ by transforming it into a 0. Thus, let $\varphi = A_0 \cdot A_1 \cdot \ldots \cdot A_k \in \mathbb{Z}_{\perp}^*$ be the stack in normal representation, we can construct $\psi = B_0 \cdot \ldots \cdot B_k \in \mathbb{Z}^*$ that fulfills

$$B_i = \begin{cases} A_i & \text{if} \quad A_i \in \mathbb{Z} \\ 0 & \text{if} \quad A_i \notin \mathbb{Z} \end{cases}$$

It is evident that ψ fulfills

$$(f_n \circ \ldots \circ f_0)(\psi, \varepsilon, \varepsilon) = (\psi_M, \psi_A, \varepsilon),$$

where

$$|\psi_M| \geq 1$$

$$\mathsf{top}(\psi_M) \neq 0$$

Now, it is easy to construct a valid unlocking script S_U in the following manner

$$S_U = \mathsf{OP\_PUSH}_{B_k} \cdot \mathsf{OP\_PUSH}_{B_{k-1}} \cdot \ldots \cdot \mathsf{OP\_PUSH}_{B_0}$$

If we execute S_U over $(\varepsilon, \varepsilon, \varepsilon)$,

$$(\mathsf{OP\_PUSH}_{B_0} \circ \ldots \circ \mathsf{OP\_PUSH}_{B_k})(\varepsilon, \varepsilon, \varepsilon) = (\psi, \varepsilon, \varepsilon)$$

Thus, it is evident that S_U unlocks S.

■

D. CONSTRAINING OPERATOR PROPERTIES

In this section we will prove several properties regarding the execution of a single operator over a trio of stacks. Specifically, we will constrain the value of several functions over these elements. Let $f \in O$, $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, such that $(\varphi'_M, \varphi'_A, \varphi'_I) = f(\varphi_M, \varphi_A, \varphi_I)$, the constrains for each individual function are the following

$$\begin{aligned}
&\mathsf{accessed}(f, \varphi_M, \varphi_A, \varphi_I) && \leq 3 \\
&\mathsf{maxelem}(\varphi'_M, \varphi'_A) && \leq 2\mathsf{maxelem}(\varphi_M, \varphi_A) + 1 \\
&\mathsf{maxgap}(\varphi'_M) && \leq 2\mathsf{maxgap}(\varphi_M) \\
&\mathsf{elemnr}(\varphi'_M) && \leq \mathsf{elemnr}(\varphi_M) + 3
\end{aligned}$$

All of these constrains are universal except for the one regarding $\mathsf{maxelem}$, that does not apply for $f \in \{\mathsf{OP\_HASH160}, \mathsf{OP\_DEPTH}\} \cup \{\mathsf{OP\_PUSH}_C \mid C \in \mathbb{Z}\}$. To prove these constrains we will analyze how each operator works case by case.

- $f = \mathsf{OP\_PUSH}_C$

$$\begin{aligned}
&\mathsf{accessed}(\mathsf{OP\_PUSH}_C, \varphi_M, \varphi_A, \varphi_I) && = 0 \\
&\mathsf{maxelem}(\varphi'_M, \varphi'_A) && \leq \max\{\mathsf{maxelem}(\varphi_M, \varphi_A), |C|\} \\
&\mathsf{maxgap}(\varphi'_M) && = \mathsf{maxgap}(\varphi_M) \\
&\mathsf{elemnr}(\varphi'_M) && \leq \mathsf{elemnr}(\varphi_M) + 1
\end{aligned}$$

- $f = \text{OP\_VERIFY}$

$$\begin{aligned}
\mathsf{accessed}(\text{OP\_VERIFY}, \varphi_M, \varphi_A, \varphi_I) &\leq 1\\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \mathsf{maxelem}(\varphi_M, \varphi_A)\\
\mathsf{maxgap}(\varphi'_M) &\leq \mathsf{maxgap}(\varphi_M)\\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_TOALTSTACK}$

$$\begin{aligned}
\mathsf{accessed}(\text{OP\_TOALTSTACK}, \varphi_M, \varphi_A, \varphi_I) &\leq 1\\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &= \mathsf{maxelem}(\varphi_M, \varphi_A)\\
\mathsf{maxgap}(\varphi'_M) &\leq \mathsf{maxgap}(\varphi_M)\\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_FROMALTSTACK}$

$$\begin{aligned}
\mathsf{accessed}(\text{OP\_FROMALTSTACK}, \varphi_M, \varphi_A, \varphi_I) &= 0\\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &= \mathsf{maxelem}(\varphi_M, \varphi_A)\\
\mathsf{maxgap}(\varphi'_M) &= \mathsf{maxgap}(\varphi_M)\\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M) + 1
\end{aligned}$$

- $f = \text{OP\_IFDUP}$

$$\begin{aligned}
\mathsf{accessed}(\text{OP\_IFDUP}, \varphi_M, \varphi_A, \varphi_I) &\leq 1\\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &= \mathsf{maxelem}(\varphi_M, \varphi_A)\\
\mathsf{maxgap}(\varphi'_M) &= \mathsf{maxgap}(\varphi_M)\\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M) + 1
\end{aligned}$$

- $f =$ OP_DROP

accessed(OP_DROP, $\varphi_M, \varphi_A, \varphi_I)$	≤ 1
maxelem($\varphi'_M, \varphi'_A)$	$\leq$ maxelem($\varphi_M, \varphi_A)$
maxgap($\varphi'_M)$	$\leq$ maxgap($\varphi_M)$
elemnr($\varphi'_M)$	$\leq$ elemnr($\varphi_M)$

- $f =$ OP_DUP

accessed(OP_DUP, $\varphi_M, \varphi_A, \varphi_I)$	≤ 1
maxelem($\varphi'_M, \varphi'_A)$	$=$ maxelem($\varphi_M, \varphi_A)$
maxgap($\varphi'_M)$	$=$ maxgap($\varphi_M)$
elemnr($\varphi'_M)$	$\leq$ elemnr($\varphi_M) + 1$

- $f =$ OP_NIP

accessed(OP_NIP, $\varphi_M, \varphi_A, \varphi_I)$	≤ 1
maxelem($\varphi'_M, \varphi'_A)$	$\leq$ maxelem($\varphi_M, \varphi_A)$
maxgap($\varphi'_M)$	≤ 2maxgap($\varphi_M)$
elemnr($\varphi'_M)$	$\leq$ elemnr($\varphi_M)$

- $f =$ OP_OVER

accessed(OP_OVER, $\varphi_M, \varphi_A, \varphi_I)$	≤ 1
maxelem($\varphi'_M, \varphi'_A)$	$=$ maxelem($\varphi_M, \varphi_A)$
maxgap($\varphi'_M)$	$=$ maxgap($\varphi_M)$
elemnr($\varphi'_M)$	$\leq$ elemnr($\varphi_M) + 1$

- $f = \text{OP\_ROT}$

$$\begin{aligned}
\text{accessed}(\text{OP\_ROT}, \varphi_M, \varphi_A, \varphi_I) &\leq 1 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq 2\text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_SWAP}$

$$\begin{aligned}
\text{accessed}(\text{OP\_SWAP}, \varphi_M, \varphi_A, \varphi_I) &\leq 1 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq 2\text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_TUCK}$

$$\begin{aligned}
\text{accessed}(\text{OP\_TUCK}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M) + 1
\end{aligned}$$

- $f = \text{OP\_2DROP}$

$$\begin{aligned}
\text{accessed}(\text{OP\_2DROP}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &\leq \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_2DUP}$

$$\begin{aligned}
\text{accessed}(\text{OP\_2DUP}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &= \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M) + 2
\end{aligned}$$

- $f = \text{OP\_3DUP}$

$$\begin{aligned}
\text{accessed}(\text{OP\_3DUP}, \varphi_M, \varphi_A, \varphi_I) &\leq 3 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &= \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M) + 3
\end{aligned}$$

- $f = \text{OP\_2OVER}$

$$\begin{aligned}
\text{accessed}(\text{OP\_2OVER}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &= \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M) + 2
\end{aligned}$$

- $f = \text{OP\_2ROT}$

$$\begin{aligned}
\text{accessed}(\text{OP\_2ROT}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq 2\text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_2SWAP}$

$$\begin{aligned}
\text{accessed}(\text{OP\_2SWAP}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq 2\text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_EQUAL}$

$$\begin{aligned}
\text{accessed}(\text{OP\_EQUAL}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &\leq \text{maxelem}(\varphi_M, \varphi_A) + 1 \\
\text{maxgap}(\varphi'_M) &\leq \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_EQUALVERIFY}$

$$\begin{aligned}
\text{accessed}(\text{OP\_EQUALVERIFY}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &\leq \text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_ADD}$

$$\begin{aligned}
\text{accessed}(\text{OP\_ADD}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\text{maxelem}(\varphi'_M, \varphi'_A) &\leq 2\text{maxelem}(\varphi_M, \varphi_A) \\
\text{maxgap}(\varphi'_M) &\leq \text{maxgap}(\varphi_M) \\
\text{elemnr}(\varphi'_M) &\leq \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \mathsf{OP\_SUB}$

$$\begin{aligned}
\mathsf{accessed}(\mathsf{OP\_SUB}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &\leq 2\mathsf{maxelem}(\varphi_M, \varphi_A) \\
\mathsf{maxgap}(\varphi'_M) &\leq \mathsf{maxgap}(\varphi_M) \\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \mathsf{OP\_ROLL}$

$$\begin{aligned}
\mathsf{accessed}(\mathsf{OP\_ROLL}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \mathsf{maxelem}(\varphi_M, \varphi_A) \\
\mathsf{maxgap}(\varphi'_M) &\leq 2\mathsf{maxgap}(\varphi_M) \\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \mathsf{OP\_PICK}$

$$\begin{aligned}
\mathsf{accessed}(\mathsf{OP\_PICK}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \mathsf{maxelem}(\varphi_M, \varphi_A) \\
\mathsf{maxgap}(\varphi'_M) &\leq \mathsf{maxgap}(\varphi_M) \\
\mathsf{elemnr}(\varphi'_M) &= \mathsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \mathsf{OP\_DEPTH}$

$$\begin{aligned}
\mathsf{accessed}(\mathsf{OP\_DEPTH}, \varphi_M, \varphi_A, \varphi_I) &= 0 \\
\mathsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \max\{|\varphi_M|, \mathsf{maxelem}(\varphi_M, \varphi_A)\} \\
\mathsf{maxgap}(\varphi'_M) &= \mathsf{maxgap}(\varphi_M) \\
\mathsf{elemnr}(\varphi'_M) &\leq \mathsf{elemnr}(\varphi_M) + 1
\end{aligned}$$

- $f = \text{OP\_HASH160}$

$$\begin{aligned}
\textsf{accessed}(\textsf{OP\_HASH160}, \varphi_M, \varphi_A, \varphi_I) &\leq 1 \\
\textsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \max\{\textsf{maxhash}, \textsf{maxelem}(\varphi_M, \varphi_A)\} \\
\textsf{maxgap}(\varphi'_M) &= \textsf{maxgap}(\varphi_M) \\
\textsf{elemnr}(\varphi'_M) &= \textsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_CHECKSIG}$

$$\begin{aligned}
\textsf{accessed}(\textsf{OP\_CHECKSIG}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\textsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \textsf{maxelem}(\varphi_M, \varphi_A) + 1 \\
\textsf{maxgap}(\varphi'_M) &= \textsf{maxgap}(\varphi_M) \\
\textsf{elemnr}(\varphi'_M) &\leq \textsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_CHECKSIGVERIFY}$

$$\begin{aligned}
\textsf{accessed}(\textsf{OP\_CHECKSIGVERIFY}, \varphi_M, \varphi_A, \varphi_I) &\leq 2 \\
\textsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \textsf{maxelem}(\varphi_M, \varphi_A) \\
\textsf{maxgap}(\varphi'_M) &= \textsf{maxgap}(\varphi_M) \\
\textsf{elemnr}(\varphi'_M) &\leq \textsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_IF}$

$$\begin{aligned}
\textsf{accessed}(\textsf{OP\_IF}, \varphi_M, \varphi_A, \varphi_I) &\leq 1 \\
\textsf{maxelem}(\varphi'_M, \varphi'_A) &\leq \textsf{maxelem}(\varphi_M, \varphi_A) \\
\textsf{maxgap}(\varphi'_M) &= \textsf{maxgap}(\varphi_M) \\
\textsf{elemnr}(\varphi'_M) &\leq \textsf{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_ELSE}$

$$\begin{aligned}
\text{accessed}(\text{OP\_ELSE}, \varphi_M, \varphi_A, \varphi_I) &= 0\\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A)\\
\text{maxgap}(\varphi'_M) &= \text{maxgap}(\varphi_M)\\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

- $f = \text{OP\_ENDIF}$

$$\begin{aligned}
\text{accessed}(\text{OP\_ENDIF}, \varphi_M, \varphi_A, \varphi_I) &= 0\\
\text{maxelem}(\varphi'_M, \varphi'_A) &= \text{maxelem}(\varphi_M, \varphi_A)\\
\text{maxgap}(\varphi'_M) &= \text{maxgap}(\varphi_M)\\
\text{elemnr}(\varphi'_M) &= \text{elemnr}(\varphi_M)
\end{aligned}$$

E. EXECUTION TIME OF OPERATORS

We devote this section to explaining why the execution time of each individual operator is polynomial in the various parameters of the trio of stacks over which it is executed and the pushed element if the operator is $\mathsf{OP\_PUSH}_C$. This result is used in the proof for Lemma 5.8. It is easy to see from the definition of the operators in appendix B that the operations required to execute any individual operator can be categorized in a few different groups:

(i) Basic arithmetic operations (addition, subtraction, comparison)
(ii) Pushing stack elements
(iii) Reading, reorganizing, duplicating and dropping stack elements
(iv) Assessing execution status
(v) Pushing elements to or from the alt stack
(vi) Basic cryptographic functions (hash, signature verification)

In addition, each operator can only execute a constant amount of operations in any of these groups. Each of these operations can be executed in polynomial time in the size of the main and the control stacks, in the size of the top of the alt stack and in the case of $\mathsf{OP\_PUSH}_C$ in the size of the pushed element. Both of these statements in conjunction mean that each operator can be executed in polynomial time in the size of the main and the control stacks, in the size of the top of the alt stack and in the case of $\mathsf{OP\_PUSH}_C$ in the size of the pushed element.

We will not analyze each operator individually to prove this, but this is clear when examining the previously referenced operator definitions. Remember that the function T corresponds to the execution time of the algorithm that executes a sequence of operators over a trio of stacks. As a way to exemplify this notion, we will go over the execution of 3 different operators:

- $\mathsf{OP\_PUSH}_C$

When executing an $\mathsf{OP\_PUSH}_C$ operator we must first determine whether the current control stack is in an execution state. Checking this condition will take a linear time in the size of the control stack. Afterwards, the operator will either do nothing or push one element to the stack. In either case, let $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, it is clear that

$$T(\mathsf{OP\_PUSH}_C, \varphi_M, \varphi_A, \varphi_I) \leq p(|\varphi_I|, \log_2(|C| + 1)),$$

for some polynomial p.

- OP_FROMALTSTACK

  When executing an OP_FROMALTSTACK operator we must first determine if the current control stack is in an execution state. Checking this condition will take a linear time in the size of the control stack. Afterwards, the operator will either do nothing or move one element from the alt to the main stack. In either case, let $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, it is clear that

$$T(\mathsf{OP\_FROMALTSTACK}, \varphi_M, \varphi_A, \varphi_I) \leq p(|\varphi_I|, \log_2(\mathsf{top}(\varphi_A) + 1)),$$

  for some polynomial p.

- OP_IF

  Once again, when executing an OP_IF operator we must first determine whether the current control stack is in an execution state. As we have previously stated, checking this condition will take a linear time in the size of the control stack. Afterwards, the operator will either just push a 0 to the control stack or compare the first element in the main stack with a 0 to check whether it is true or false, in which cases we will drop the first element in the main stack and push a 1 or a 0 to the control stack, respectively. If the control stack represents an execution state and the main stack does not contain integer elements, then we will just return an error. In any case, let $\varphi_M \in \mathbb{Z}_\perp^*$, $\varphi_A \in \mathbb{Z}^*$ and $\varphi_I \in \{0,1\}^*$, it is clear that

$$T(\mathsf{OP\_IF}, \varphi_M, \varphi_A, \varphi_I) \leq p(|\varphi_I|, \|\varphi_M\|),$$

for some polynomial p.

F. STANDARD SCRIPT EVALUATION

This section is dedicated to extending the result presented in Lemma 5.9 to the standard script evaluation problem presented in section 4.1. As a reminder, this problem consisted in determining whether the execution of a pair of scripts is successful. As a point of notice, this process consists of executing the unlocking script over a trio of empty stacks, determining whether the control stack is empty after this first execution, and if it is, executing the locking script over the final main stack and determining whether the final main stack is valid and the final control stack is empty.

It is evident that the second part of the described process is identical to the algorithm associated with the T_{comp} function described in equation (5.5). However, in order to bound the execution time of the first part of the process, we will need to define a function T_{scomp} that represents the execution time of the algorithm that executes a script over a trio of stacks and afterwards determines whether the final control stack is empty.

$$T_{scomp} : O^* \times \mathbb{Z}_\perp^* \times \mathbb{Z}^* \times \{0,1\}^* \to \mathbb{N}$$

The difference between this algorithm and the one associated with T_{comp} is that the former does not need to check whether the top of the final main stack is true. We will first show that this function has a polynomial upper bound.

LEMMA F.1. *Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$,*

$$T_{scomp}(S, \varepsilon, \varepsilon, \varepsilon) \leq p_{scomp}(n, \log_2(\textit{\textbf{maxpush}}(S) + 1)),$$

for some fixed polynomial p_{scomp}, independent of S.

PROOF. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$. From Lemma 5.9 we know that

$$\begin{aligned} T_{scomp}(S,\varepsilon,\varepsilon,\varepsilon) &\leq T_{comp}(S,\varepsilon,\varepsilon,\varepsilon) \\ &\leq p_{comp}(n, \|\varepsilon\|, \log_2(\textsf{maxpush}(S)+1)) \\ &\leq p_{scomp}(n, \log_2(\textsf{maxpush}(S)+1), \end{aligned}$$

for some polynomial p_{scomp}. ■

Now we will prove that the evaluation of a pair of scripts is in PTIME. For this we will recognize that we can construct an algorithm for evaluating a pair of an unlocking script and a locking script by combining the algorithm associated with T_{scomp} over the unlocking script and a trio of empty stacks and the algorithm associated with T_{comp} over the final stack of the previous execution and two empty stacks. We will note that in case after the execution of the first script the control stack is not empty, we will execute the second script over $\square$.

LEMMA F.2. *Let $S_L = f_0\cdot \ldots \cdot f_n \in O^*$ and $S_U = g_0\cdot \ldots \cdot g_m \in O^*$, such that*

$$(g_m \circ \ldots \circ g_0)(\varepsilon,\varepsilon,\varepsilon) = \psi,$$

where $\psi \in (\mathbb{Z}^*_\perp \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\square\}$. Then,

$$T_{scomp}(S_U,\varepsilon,\varepsilon,\varepsilon) + T_{comp}(S_L,\psi) \leq \\ p_{emp}(n, m, \log_2(\textsf{\textit{maxpush}}(S_L)+1), \log_2(\textsf{\textit{maxpush}}(S_U)+1))$$

for some fixed polynomial p_{emp}, independent of S_L and S_U.

PROOF. Let $S_L = f_0\cdot \ldots \cdot f_n \in O^*$ and $S_U = g_0\cdot \ldots \cdot g_m \in O^*$, such that

$$(g_m \circ \ldots \circ g_0)(\varepsilon,\varepsilon,\varepsilon) = \psi,$$

where $\psi \in (\mathbb{Z}_\perp^* \times \mathbb{Z}^* \times \{0,1\}^*) \cup \{\Box\}$. From Lemma F.1 we know that

$$T_{scomp}(S_U, \varepsilon, \varepsilon, \varepsilon) \leq p_{scomp}(n, \log_2(\mathsf{maxpush}(S) + 1)),$$

for some polynomial p_{scomp}. Now, we need to examine two different cases for the second half of the execution:

- Case 1: $\psi = (\varphi_M, \varphi_A, \varepsilon)$
  From Lemma 5.7 we know that

  $$\|\varphi_M\| \leq p_{size}(m, \|\varepsilon\|, \log_2(\mathsf{maxpush}(S_U) + 1))$$

  Thus, by utilizing Lemma 5.16,

  $$\begin{aligned} T_{comp}(S_L, \varphi_M, \varepsilon, \varepsilon) &\leq p_{comp}(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S_L) + 1)) \\ &\leq p_{emp}\big(n, m, \log_2(\mathsf{maxpush}(S_L) + 1), \\ &\qquad \log_2(\mathsf{maxpush}(S_U) + 1)\big), \end{aligned}$$

  for some polynomial p_{emp}.
- Case 2: $\psi = (\varphi_M, \varphi_A, \varphi_I)$, with $\varphi_I \neq \varepsilon$, or $\psi = \Box$
  Similarly to case 1,

  $$\begin{aligned} T_{comp}(S_L, \Box) &\leq p_{comp}(n, \|\varphi_M\|, \log_2(\mathsf{maxpush}(S_L) + 1)) \\ &\leq p_{emp}\big(n, m, \log_2(\mathsf{maxpush}(S_L) + 1), \\ &\qquad \log_2(\mathsf{maxpush}(S_U) + 1)\big), \end{aligned}$$

  for some polynomial p_{emp}.

■

G. CONSTRUCTION OF THE SYSTEM OF EQUATIONS TO PROVE POLYNOMIALITY

In this section we will show how to construct the system of equations to prove that there must exist a solution with elements of polynomial size. We will also prove that the coefficients in the system are constrained and that the variable stacks used in the proof for theorem 5.14 represent the execution of a script over a stack constructed from a solution to the system of equations.

Construction of the system of equations. Firstly, we show how to construct the system of equations. Simultaneously, we will show that the new equations fulfill the constrain enunciated in the proof for theorem 5.14 for the coefficients in the system.

Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be the analyzed script. We denote the stacks as $\chi_M^i = \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}$, $\chi_A^i = p_0 \cdot \ldots \cdot p_{\ell'}$, $\varphi_M^i = \bot^{P_0} \cdot M_0 \cdot \ldots \cdot M_{k'-1} \cdot \bot^{P_{k'}}$ and finally $\varphi_A^i = N_0 \cdot \ldots \cdot N_{\ell'}$. We will also use $a_{\max}$ to denote the maximum element in the newly added equations, disregarding its sign.

It is important to note that in order to be able to constrain the value of the coefficients we will use a sideproduct obtained through the proof of lemma 5.4. Namely, we found that

$$\begin{aligned}\max_{i \in \{0,\ldots,n+1\}} \{\mathsf{elemnr}(\varphi_M^i)\} &\leq \mathsf{elemnr}(\varphi_M) + 3(n+1) \\ &\leq p_{aux}(n, \mathsf{elemnr}(\varphi_M)),\end{aligned}$$

for some polynomial p_{aux}. As a reminder, any coefficient a in the system of equations must fulfill the following condition

$$|a| \leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)),$$

for some polynomial p_{coef}. We will also note that in case φ_I^i does not represent an execution state, then we will add no equations to the system on step i and

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\chi_M^i, \chi_A^i),$$

in which case we will evidently not need to prove the constrain for any coefficients. We will use p_{coef} to denote some polynomial that is able to serve as an upper bound for all of the coefficients in the system of equations. It will be made clear how all of the coefficients can be constrained polynomially and p_{coef} will be a polynomial that acts as an upper bound for all of the coefficients.

- $f_i = \mathsf{OP\_PUSH}_C$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \chi_M^i, \chi_A^i)$$

  Equations

$$w_{null} = 0$$
$$w_i = C$$

  Coefficients

$$\begin{aligned} |a_{\max}| &\leq \max\{1, C\} \\ &\leq \mathsf{maxpush}(S) + 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_VERIFY}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

Equations

$$d_0 = 0$$
$$z_0 = v_i + 1 \quad \text{if } \mathsf{top}_\perp(\varphi_M^i) > 0$$
$$z_0 = -v_i - 1 \quad \text{if } \mathsf{top}_\perp(\varphi_M^i) < 0$$

Coefficients

$$\begin{aligned} |a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i =$ OP_TOALTSTACK

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, z_0 \cdot \chi_A^i)$$

  Equations

  $$d_0 = 0$$

  Coefficients

  $$\begin{aligned} |a_{\max}| &\leq 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i =$ OP_FROMALTSTACK

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (p_0 \cdot \chi_M^i, p_1 \cdot \ldots \cdot p_{\ell'})$$

  Equations

  None

  Coefficients

  Not applicable

- $f_i = \mathsf{OP\_IFDUP}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = \begin{cases} (\chi_M^i, \chi_A^i) & \text{if} \quad \mathsf{top}(\varphi_M^i) = 0 \\ (\bot^{w_{null}} \cdot z_0 \cdot \chi_M^i, \chi_A^i) & \text{if} \quad \mathsf{top}(\varphi_M^i) \neq 0 \end{cases}$$

  Equations

$$\begin{aligned} d_0 &= 0 \\ w_{null} &= 0 \\ z_0 &= v_i + 1 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) > 0 \\ z_0 &= -v_i - 1 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) < 0 \\ z_0 &= 0 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) = 0 \end{aligned}$$

  Coefficients

$$\begin{aligned} |a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_DROP}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

$$d_0 = 0$$

  Coefficients

$$\begin{aligned} |a_{\max}| &\leq 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_DUP}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_0 \cdot \chi_M^i, \chi_A^i)$$

  Equations

$$d_0 = 0$$
$$w_{null} = 0$$

  Coefficients

$$\begin{aligned}|a_{\max}| &\leq 1\\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_NIP}$

  Variable Stacks

  Let $m \in \{0, 1\}$, such that $\sum_{j=0}^{m}(P_j + 1) = 2$

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

$$d_0 + \ldots + d_m + m + 1 = 2$$
$$d_m + d_{m+1} = v_i$$

  Coefficients

$$\begin{aligned}|a_{\max}| &\leq 3\\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_OVER}$

  Variable Stacks

Let $m \in \{0,1\}$, such that $\sum_{j=0}^{m}(P_j+1)=2$

$$(\chi_M^{i+1},\chi_A^{i+1}) = (\bot^{w_{null}}\cdot z_m\cdot\chi_M^i,\chi_A^i)$$

Equations

$$d_0+\ldots+d_m+m+1=2$$
$$w_{null}=0$$

Coefficients

$$\begin{aligned}|a_{\max}| &\leq 2\\ &\leq p_{coef}(2^n,\mathsf{elemnr}(\varphi_M),\mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_ROT}$

  Variable Stacks

  Let $m \in \{0,1,2\}$, such that $\sum_{j=0}^{m}(P_j+1)=3$

$$(\chi_M^{i+1},\chi_A^{i+1}) = (\bot^{w_{null}}\cdot z_m\cdot\bot^{d_0}\cdot z_0\cdot\ldots\cdot z_{m-1}\cdot\bot^{v_i}\cdot z_{m+1}\cdot\ldots\cdot z_{k'-1}\cdot\bot^{d_{k'}},\chi_A^i)$$

  Equations

$$d_0+\ldots+d_m+m+1=3$$
$$d_m+d_{m+1}=v_i$$
$$w_{null}=0$$

  Coefficients

$$\begin{aligned}|a_{\max}| &\leq 3\\ &\leq p_{coef}(2^n,\mathsf{elemnr}(\varphi_M),\mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_SWAP}$

  Variable Stacks

Let $m \in \{0,1\}$, such that $\sum_{j=0}^{m}(P_j+1)=2$

$$(\chi_M^{i+1},\chi_A^{i+1}) = (\bot^{w_{null}}\cdot z_m\cdot \bot^{d_0}\cdot z_0\cdot \ldots \cdot z_{m-1}\cdot \bot^{v_i}\cdot z_{m+1}\cdot \ldots \cdot z_{k'-1}\cdot \bot^{d_{k'}},\chi_A^i)$$

Equations

$$d_0+\ldots+d_m+m+1=2$$
$$d_m+d_{m+1}=v_i$$
$$w_{null}=0$$

Coefficients

$$|a_{\max}| \le 3$$
$$\le p_{coef}(2^n,\mathsf{elemnr}(\varphi_M),\mathsf{maxpush}(S))$$

- $f_i = \mathsf{OP\_TUCK}$

  Variable Stacks

$$(\chi_M^{i+1},\chi_A^{i+1}) = (\bot^{w_{null}}\cdot z_0\cdot \bot^{w_{null}}\cdot z_1\cdot \bot^{w_{null}}\cdot z_0\cdot \bot^{d_2}\cdot z_2\cdot \ldots \cdot z_{k'-1}\cdot \bot^{d_{k'}},\chi_A^i)$$

Equations

$$d_0=0$$
$$d_1=0$$
$$w_{null}=0$$

Coefficients

$$|a_{\max}| \le 1$$
$$\le p_{coef}(2^n,\mathsf{elemnr}(\varphi_M),\mathsf{maxpush}(S))$$

- $f_i = \mathsf{OP\_2DROP}$

Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

Equations

$$d_0 = 0$$

$$d_1 = 0$$

Coefficients

$$\begin{aligned} |a_{\max}| &\leq 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_2DUP}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_0 \cdot \bot^{w_{null}} \cdot z_1 \cdot \chi_M^i, \chi_A^i)$$

  Equations

  $$d_0 = 0$$

  $$d_1 = 0$$

  $$w_{null} = 0$$

  Coefficients

  $$\begin{aligned} |a_{\max}| &\leq 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_3DUP}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_0 \cdot \bot^{w_{null}} \cdot z_1 \cdot \bot^{w_{null}} \cdot z_2 \cdot \chi_M^i, \chi_A^i)$$

Equations

$$\begin{aligned} d_0 &= 0 \\ d_1 &= 0 \\ d_2 &= 0 \\ w_{null} &= 0 \end{aligned}$$

Coefficients

$$\begin{aligned} |a_{\max}| &\leq 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_2OVER}$

  Variable Stacks

  Let $m \in \{0,1,2\}$, such that $\sum_{j=0}^{m}(P_j + 1) = 3$

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_m \cdot \bot^{w_{null}} \cdot z_{m+1} \cdot \chi_M^i, \chi_A^i)$$

  Equations

  $$\begin{aligned} d_0 + \ldots + d_m + m + 1 &= 3 \\ d_{m+1} &= 0 \\ w_{null} &= 0 \end{aligned}$$

  Coefficients

  $$\begin{aligned} |a_{\max}| &\leq 3 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_2ROT}$

  Variable Stacks

Let $m \in \{0, 1, 2, 3, 4\}$, such that $\sum_{j=0}^{m}(P_j + 1) = 5$

$$(\chi_M^{i+1}, \chi_A^{i+1}) =$$
$$(\bot^{w_{null}} \cdot z_m \cdot \bot^{w_{null}} \cdot z_{m+1} \cdot \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+2} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

Equations

$$d_0 + \ldots + d_m + m + 1 = 5$$
$$d_{m+1} = 0$$
$$d_m + d_{m+2} = v_i$$
$$w_{null} = 0$$

Coefficients

$$|a_{\max}| \leq 5$$
$$\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))$$

- $f_i = \mathsf{OP\_2SWAP}$

  Variable Stacks

  Let $m \in \{0, 1, 2\}$, such that $\sum_{j=0}^{m}(P_j + 1) = 3$

$$(\chi_M^{i+1}, \chi_A^{i+1}) =$$
$$(\bot^{w_{null}} \cdot z_m \cdot \bot^{w_{null}} \cdot z_{m+1} \cdot \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+2} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

Equations

$$d_0 + \ldots + d_m + m + 1 = 3$$
$$d_{m+1} = 0$$
$$d_m + d_{m+2} = v_i$$
$$w_{null} = 0$$

Coefficients

$$|a_{\max}| \leq 3$$
$$\leq p_{coef}(2^n, \textsf{elemnr}(\varphi_M), \textsf{maxpush}(S))$$

- $f_i = \textsf{OP\_PICK}$

  Variable Stacks

  Let $m \in \{0, \ldots, k' - 1\}$, such that $\sum_{j=1}^{m}(P_j + 1) - 1 = M_0$

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_m \cdot \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

  $$d_0 = 0$$
  $$d_1 + \ldots + d_m + m - 1 = z_0$$
  $$w_{null} = 0$$

  Coefficients

  $$|a_{\max}| \leq \textsf{elemnr}(\varphi_M^i)$$
  $$\leq p_{aux}(n, \textsf{elemnr}(\varphi_M))$$
  $$\leq p_{coef}(2^n, \textsf{elemnr}(\varphi_M), \textsf{maxpush}(S))$$

- $f_i = \textsf{OP\_ROLL}$

  Variable Stacks

  Let $m \in \{0, \ldots, k' - 1\}$, such that $\sum_{j=1}^{m}(P_j + 1) - 1 = M_0$

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot z_m \cdot \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

Equations

$$\begin{aligned} d_0 &= 0 \\ d_1 + \ldots + d_m + m - 1 &= z_0 \\ d_m + d_{m+1} &= v_i \\ w_{null} &= 0 \end{aligned}$$

Coefficients

$$\begin{aligned} |a_{\max}| &\leq \max\{\mathsf{elemnr}(\varphi_M^i), 3\} \\ &\leq p_{aux}(n, \mathsf{elemnr}(\varphi_M)) + 3 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i = \mathsf{OP\_EQUAL}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

  Equations

  $$\begin{aligned} d_0 &= 0 \\ d_1 &= 0 \\ z_1 + v_i + 1 &= z_0 \quad \text{if } M_0 > M_1 \\ z_1 - v_i - 1 &= z_0 \quad \text{if } M_0 < M_1 \\ z_1 &= z_0 \quad \text{if } M_0 = M_1 \\ w_i &= 0 \quad \text{if } M_0 > M_1 \\ w_i &= 0 \quad \text{if } M_0 < M_1 \\ w_i &= 1 \quad \text{if } M_0 = M_1 \\ w_{null} &= 0 \end{aligned}$$

Coefficients

$$\begin{aligned}|a_{\max}| &\leq 3 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i =$ OP_EQUALVERIFY

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

  $$d_0 = 0$$

  $$d_1 = 0$$

  $$z_0 = z_1$$

  Coefficients

  $$\begin{aligned}|a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i =$ OP_ADD

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{w_{null}} \cdot w_i \cdot \bot^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

  $$d_0 = 0$$

  $$d_1 = 0$$

  $$z_0 + z_1 = w_i$$

  $$w_{null} = 0$$

Coefficients

$$\begin{aligned}|a_{\max}| &\leq 3\\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_SUB}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

  Equations

$$d_0 = 0$$

$$d_1 = 0$$

$$z_1 - z_0 = w_i$$

$$w_{null} = 0$$

  Coefficients

$$\begin{aligned}|a_{\max}| &\leq 3\\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i = \mathsf{OP\_DEPTH}$

  Variable Stacks

$$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot v_i \cdot \chi_M^i, \chi_A^i)$$

  Equations

$$\begin{aligned}d_0 + \ldots + d_{k'} + k' - 1 &= v_i\\ w_{null} &= 0\end{aligned}$$

Coefficients

$$\begin{aligned}|a_{\max}| &\leq \mathsf{elemnr}(\varphi_M^i) + 2 \\ &\leq p_{aux}(n, \mathsf{elemnr}(\varphi_M)) + 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i =$ OP_IF

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}}, \chi_A^i)$$

  Equations

  $$\begin{aligned}d_0 &= 0 \\ v_i + 1 &= z_0 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) > 0 \\ -v_i - 1 &= z_0 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) < 0 \\ z_0 &= 0 \quad \text{if } \mathsf{top}_\bot(\varphi_M^i) = 0\end{aligned}$$

  Coefficients

  $$\begin{aligned}|a_{\max}| &\leq 2 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

- $f_i =$ OP_ELSE

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\chi_M^i, \chi_A^i)$$

  Equations

  None

  Coefficients

  Not applicable

- $f_i = \textsf{OP\_ENDIF}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\chi_M^i, \chi_A^i)$$

  Equations

  None

  Coefficients

  Not applicable

- $f_i = \textsf{OP\_HASH160}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

  Equations

  $$\begin{aligned} d_0 &= 0 \\ z_0 &= M_0 \\ w_i &= \textsf{hash}(M_0) \\ w_{null} &= 0 \end{aligned}$$

  Coefficients

  $$\begin{aligned} |a_{\max}| &\leq \max\{1, \textsf{maxhash}, p_{crypto}(2^n, \textsf{maxpush}(S))\} \\ &\leq p_{crypto}(2^n, \textsf{maxpush}(S)) + \textsf{maxhash} + 1 \\ &\leq p_{coef}(2^n, \textsf{elemnr}(\varphi_M), \textsf{maxpush}(S)) \end{aligned}$$

- $f_i = \textsf{OP\_CHECKSIG}$

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

Equations

$$\begin{aligned} d_0 &= 0 \\ d_1 &= 0 \\ z_0 &= M_0 \\ z_1 &= M_1 \\ w_i &= \mathsf{chksig}(M_0, M_1) \\ w_{null} &= 0 \end{aligned}$$

Coefficients

$$\begin{aligned} |a_{\max}| &\leq \max\{1, p_{crypto}(2^n, \mathsf{maxpush}(S))\} \\ &\leq p_{crypto}(2^n, \mathsf{maxpush}(S)) + 1 \\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S)) \end{aligned}$$

- $f_i =$ OP_CHECKSIGVERIFY

  Variable Stacks

  $$(\chi_M^{i+1}, \chi_A^{i+1}) = (\perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}}, \chi_A^i)$$

  Equations

  $$\begin{aligned} d_0 &= 0 \\ d_1 &= 0 \\ z_0 &= M_0 \\ z_1 &= M_1 \end{aligned}$$

Coefficients

$$\begin{aligned}|a_{\max}| &\leq \max\{1, p_{crypto}(2^n, \mathsf{maxpush}(S))\}\\ &\leq p_{crypto}(2^n, \mathsf{maxpush}(S)) + 1\\ &\leq p_{coef}(2^n, \mathsf{elemnr}(\varphi_M), \mathsf{maxpush}(S))\end{aligned}$$

Stack equivalence. Now, we will prove that if we construct a stack from a solution to the system of equations constructed from a script as described in the previous section, the execution of the script over this stack will be represented correctly by the sequence of variable stacks. This section serves as part of the inductive step in the proof by induction of Claim 5.14.1. The case in which the control stacks do not represent an execution state is described in said proof. Therefore, we will just conduct the analysis for the opposite case.

Firstly, we will analyze the execution of the basic operators (i.e. the operators not associated with flow control). These do not modify the control stack, which is why it will not be considered in this part of the analysis. For conciseness, we will also exclude the alt stack operators from the initial analysis, for these are the only operators which modify the alt stack. Thus, we will not consider the alt stack in the analysis of the basic operators either. Let $S = f_0 \cdot \ldots \cdot f_n \in O^*$ be an arbitrary script and

$$\vec{c} = (\bar{C}, \bar{E}, \bar{F}, \bar{G})^T,$$

where the partial vectors are expanded as $\bar{C} = (C_0, \ldots, C_{k-1})$, $\bar{E} = (E_0, \ldots, E_k)$, $\bar{F} = (F_0, \ldots, F_{n+1})$ and $\bar{G} = (G_0, \ldots, G_n, G_{null})$, be a solution to the system of equations corresponding to S on step $i + 1$, such that

$$\Phi_{i+1}\vec{c} = \vec{b}_{i+1}$$

For convenience we will say that

$$(\psi_M^0, \psi_A^0, \psi_I^0) = (\perp^{E_0} \cdot C_0 \cdot \ldots \cdot C_{k-1} \cdot \perp^{E_k}, \varepsilon, \varepsilon)$$

and that for all $j \in \{1, \ldots, i+1\}$,

$$(\psi_M^j, \psi_A^j, \psi_I^j) = (f_{j-1} \circ \ldots \circ f_0)(\psi_M^0, \psi_A^0, \psi_I^0)$$

Now, let

$$\begin{aligned}
\chi_M^i &= \perp^{d_0} \cdot z_0 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\
\chi_A^i &= p_0 \cdot \ldots \cdot p_{\ell'} \\
\psi_M^i &= \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
\psi_A^i &= K_0 \cdot \ldots \cdot K_{\ell'} \\
\varphi_M^i &= \perp^{P_0} \cdot M_0 \cdot \ldots \cdot M_{k'-1} \cdot \perp^{P_{k'}} \\
\varphi_A^i &= N_0 \cdot \ldots \cdot N_{\ell'}
\end{aligned}$$

We want to show that if we assign $\vec{x} = \vec{c}$ and then assume that $(\chi_M^i, \chi_A^i, \varphi_I^i) = (\psi_M^i, \psi_A^i, \psi_I^i)$, then

$$(\chi_M^{i+1}, \chi_A^{i+1}, \varphi_I^{i+1}) = (\psi_M^{i+1}, \psi_A^{i+1}, \psi_I^{i+1})$$

- $f_i = \mathsf{OP\_PUSH}_C$

Observations

$$G_{null} = 0$$

$$G_i = C$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{w_{null}} \cdot w_i \cdot \chi_M^i \\ &= \bot^{G_{null}} \cdot G_i \cdot \psi_M^i \\ &= \bot^0 \cdot C \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i = \mathsf{OP\_VERIFY}$
  - Case 1: $\mathsf{top}_\bot(\varphi_M^i) > 0$

    Observations

    $$J_0 = F_i + 1 \geq 1 > 0$$

    $$L_0 = 0$$

    Stacks values

    $$\begin{aligned}\chi_M^{i+1} &= \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\ &= \bot^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

  - Case 2: $\mathsf{top}_\bot(\varphi_M^i) < 0$

    Observations

    $$J_0 = -F_i - 1 \leq -1 < 0$$

    $$L_0 = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\ &= \bot^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_IFDUP
  - Case 1: $\mathsf{top}_\bot(\varphi_M^i) > 0$

    Observations

    $$J_0 = F_i + 1 \geq 1 > 0$$

    $$L_0 = 0$$

    $$G_{null} = 0$$

    Stacks values

    $$\begin{aligned}\chi_M^{i+1} &= \bot^{w_{null}} \cdot z_0 \cdot \chi_M^i \\ &= \bot^{G_{null}} \cdot J_0 \cdot \psi_M^i \\ &= \bot^0 \cdot J_0 \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

  - Case 2: $\mathsf{top}_\bot(\varphi_M^i) < 0$

    Observations

    $$J_0 = -F_i - 1 \leq -1 < 0$$

    $$L_0 = 0$$

    $$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_0 \cdot \chi_M^i \\ &= \perp^{G_{null}} \cdot J_0 \cdot \psi_M^i \\ &= \perp^0 \cdot J_0 \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

– Case 3: $\mathsf{top}_\perp(\varphi_M^i) = 0$

Observations

$$J_0 = 0$$

$$L_0 = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \chi_M^i \\ &= \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i = \mathsf{OP\_DROP}$

  Observations

$$L_0 = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i = \mathsf{OP\_DUP}$

Observations

$$L_0 = 0$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{w_{null}} \cdot z_0 \cdot \chi_M^i \\ &= \bot^{G_{null}} \cdot J_0 \cdot \psi_M^i \\ &= \bot^0 \cdot J_0 \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_NIP

  Let $m \in \{0, 1\}$ be such that $\sum_{j=0}^{m}(P_j + 1) = 2$

  Observations

$$L_0 + \ldots + L_m + m + 1 = 2$$

$$F_i = L_m + L_{m+1}$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\ &= \bot^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \bot^{F_i} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \bot^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \bot^{L_m + L_{m+1}} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_OVER

  Let $m \in \{0, 1\}$ be such that $\sum_{j=0}^{m}(P_j + 1) = 2$

Observations

$$L_0 + \ldots + L_m + m + 1 = 2$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \chi_M^i \\ &= \perp^{G_{null}} \cdot J_m \cdot \psi_M^i \\ &= \perp^0 \cdot J_m \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_ROT

  Let $m \in \{0,1,2\}$ be such that $\sum_{j=0}^{m}(P_j + 1) = 3$

  Observations

$$L_0 + \ldots + L_m + m + 1 = 3$$

$$F_i = L_m + L_{m+1}$$

$$G_{null} = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \perp^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{F_i} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^0 \cdot J_m \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{L_m + L_{m+1}} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_SWAP

  Let $m \in \{0,1\}$ be such that $\sum_{j=0}^{m}(P_j + 1) = 2$

Observations

$$L_0 + \ldots + L_m + m + 1 = 2$$

$$F_i = L_m + L_{m+1}$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \perp^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{F_i} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^0 \cdot J_m \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{L_m + L_{m+1}} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_TUCK

  Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$G_{null} = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_0 \cdot \perp^{w_{null}} \cdot z_1 \cdot \perp^{w_{null}} \cdot z_0 \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_0 \cdot \perp^{G_{null}} \cdot J_1 \cdot \perp^{G_{null}} \cdot J_0 \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^0 \cdot J_0 \cdot \perp^0 \cdot J_1 \cdot \perp^0 \cdot J_0 \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_2DROP

Observations

$$L_0 = 0$$

$$L_1 = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_2DUP

  Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_0 \cdot \perp^{w_{null}} \cdot z_1 \cdot \chi_M^i \\ &= \perp^{G_{null}} \cdot J_0 \cdot \perp^{G_{null}} \cdot J_1 \cdot \psi_M^i \\ &= \perp^0 \cdot J_0 \cdot \perp^0 \cdot J_1 \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_3DUP

Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$L_2 = 0$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_0 \cdot \perp^{w_{null}} \cdot z_1 \cdot \perp^{w_{null}} \cdot z_2 \cdot \chi_M^i \\ &= \perp^{G_{null}} \cdot J_0 \cdot \perp^{G_{null}} \cdot J_1 \cdot \perp^{G_{null}} \cdot J_2 \cdot \psi_M^i \\ &= \perp^0 \cdot J_0 \cdot \perp^0 \cdot J_1 \cdot \perp^0 \cdot J_2 \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_2OVER

  Let $m \in \{0, 1, 2\}$ be such that $\sum_{j=0}^{m}(P_j + 1) = 3$

  Observations

$$L_0 + \ldots + L_m + m + 1 = 3$$

$$L_{m+1} = 0$$

$$G_{null} = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{w_{null}} \cdot z_{m+1} \cdot \chi_M^i \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{G_{null}} \cdot J_{m+1} \cdot \psi_M^i \\ &= \perp^0 \cdot J_m \cdot \perp^0 \cdot J_{m+1} \cdot \psi_M^i \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_2ROT

Let $m \in \{0,1,2,3,4\}$ be such that $\sum_{j=0}^{m}(P_j+1) = 5$

Observations

$$L_0 + \ldots + L_m + m + 1 = 5$$

$$L_{m+1} = 0$$

$$F_i = L_m + L_{m+2}$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}
\chi_M^{i+1} &= \bot^{w_{null}} \cdot z_m \cdot \bot^{w_{null}} \cdot z_{m+1} \cdot \bot^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \bot^{v_i} \cdot z_{m+2} \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\
&= \bot^{G_{null}} \cdot J_m \cdot \bot^{G_{null}} \cdot J_{m+1} \cdot \bot^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \bot^{F_i} \cdot J_{m+2} \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\
&= \bot^{0} \cdot J_m \cdot \bot^{0} \cdot J_{m+1} \cdot \bot^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \bot^{L_m + L_{m+2}} \cdot J_{m+2} \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\
&= \psi_M^{i+1}
\end{aligned}$$

- $f_i =$ OP_2SWAP

  Let $m \in \{0,1,2\}$ be such that $\sum_{j=0}^{m}(P_j+1) = 3$

  Observations

$$L_0 + \ldots + L_m + m + 1 = 3$$

$$L_{m+1} = 0$$

$$F_i = L_m + L_{m+2}$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{w_{null}} \cdot z_{m+1} \cdot \perp^{d_0} \cdot z_0 \cdot \ldots \cdot z_{m-1} \cdot \perp^{v_i} \cdot z_{m+2} \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{G_{null}} \cdot J_{m+1} \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{F_i} \cdot J_{m+2} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot J_m \cdot \perp^{0} \cdot J_{m+1} \cdot \perp^{L_0} \cdot J_0 \cdot \ldots \cdot J_{m-1} \cdot \perp^{L_m + L_{m+2}} \cdot J_{m+2} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_PICK

  Let $m \in \{0, \ldots, k'-1\}$ be such that $\sum_{j=1}^{m}(P_j + 1) - 1 = M_0$

  Observations

$$L_0 = 0$$

$$L_1 + \ldots + L_m + m - 1 = J_0$$

$$G_{null} = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot J_m \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_ROLL

  Let $m \in \{0, \ldots, k'-1\}$ be such that $\sum_{j=1}^{m}(P_j + 1) - 1 = M_0$

  Observations

$$L_0 = 0$$

$$L_1 + \ldots + L_m + m - 1 = J_0$$

$$F_i = L_m + L_{m+1}$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot z_m \cdot \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{m-1} \cdot \perp^{v_i} \cdot z_{m+1} \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot J_m \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{m-1} \cdot \perp^{F_i} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot J_m \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{m-1} \cdot \perp^{L_m + L_{m+1}} \cdot J_{m+1} \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_EQUAL
  - Case 1: $M_0 > M_1$

    Observations

$$J_0 = J_1 + F_i + 1 \geq J_1 + 1 > J_1$$

$$L_0 = 0$$

$$L_1 = 0$$

$$G_i = 0$$

$$G_{null} = 0$$

    Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot 0 \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

  - Case 2: $M_0 < M_1$

Observations

$$J_0 = J_1 - F_i - 1 \leq J_1 - 1 < J_1$$

$$L_0 = 0$$

$$L_1 = 0$$

$$G_i = 0$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^0 \cdot 0 \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

– Case 3: $M_0 = M_1$

Observations

$$J_0 = J_1$$

$$L_0 = 0$$

$$L_1 = 0$$

$$G_i = 1$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{w_{null}} \cdot w_i \cdot \bot^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\ &= \bot^{G_{null}} \cdot G_i \cdot \bot^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \bot^{0} \cdot 1 \cdot \bot^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_EQUALVERIFY

  Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$J_0 = J_1$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \bot^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{d_{k'}} \\ &= \bot^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_ADD

  Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$G_i = J_0 + J_1$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot (J_0 + J_1) \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_SUB

  Observations

$$L_0 = 0$$
$$L_1 = 0$$
$$G_i = J_1 - J_0$$
$$G_{null} = 0$$

  Stacks values

$$\begin{aligned}\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \perp^{0} \cdot (J_1 - J_0) \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1}\end{aligned}$$

- $f_i =$ OP_HASH160

  Observations

$$L_0 = 0$$
$$J_0 = M_0$$
$$G_i = \mathsf{hash}(M_0)$$
$$G_{null} = 0$$

Stacks values

$$\begin{aligned}
\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\
&= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \perp^{0} \cdot \mathsf{hash}(M_0) \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \perp^{0} \cdot \mathsf{hash}(J_0) \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \psi_M^{i+1}
\end{aligned}$$

- $f_i =$ OP_CHECKSIG

Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$J_0 = M_0$$

$$J_1 = M_1$$

$$G_i = \mathsf{chksig}(M_0, M_1)$$

$$G_{null} = 0$$

Stacks values

$$\begin{aligned}
\chi_M^{i+1} &= \perp^{w_{null}} \cdot w_i \cdot \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\
&= \perp^{G_{null}} \cdot G_i \cdot \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \perp^{0} \cdot \mathsf{chksig}(M_0, M_1) \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \perp^{0} \cdot \mathsf{chksig}(J_0, J_1) \cdot \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\
&= \psi_M^{i+1}
\end{aligned}$$

- $f_i =$ OP_CHECKSIGVERIFY

Observations

$$L_0 = 0$$

$$L_1 = 0$$

$$\left.\begin{aligned} J_0 &= M_0 \\ J_1 &= M_1 \end{aligned}\right\} \mathsf{chksig}(J_0, J_1) = 1$$

Stacks values

$$\begin{aligned} \chi_M^{i+1} &= \perp^{d_2} \cdot z_2 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{L_2} \cdot J_2 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1} \end{aligned}$$

Now we will move on to the analysis of the alt stack operators (i.e. OP_TOALTSTACK and OP_FROMALTSTACK). Once again, these operators are unable to modify the control stack. Thus, we will continue to ignore this stack in this part of the analysis.

- $f_i =$ OP_TOALTSTACK

  Observations

$$L_0 = 0$$

  Stacks values

$$\begin{aligned} \chi_M^{i+1} &= \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{d_{k'}} \\ &= \perp^{L_1} \cdot J_1 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \\ &= \psi_M^{i+1} \end{aligned} \qquad \begin{aligned} \chi_A^{i+1} &= z_0 \cdot \chi_A^i \\ &= J_0 \cdot \psi_A^i \\ &= \psi_A^{i+1} \end{aligned}$$

- $f_i =$ OP_FROMALTSTACK

  Observations

None

Stacks values

$$\chi_M^{i+1} = p_0 \cdot \chi_M^i \qquad \chi_A^{i+1} = p_1 \cdot \ldots \cdot p_{\ell'}$$
$$= K_0 \cdot \psi_M^i \qquad = K_1 \cdot \ldots \cdot K_{\ell'}$$
$$= \psi_M^{i+1} \qquad = \psi_A^{i+1}$$

Finally, we show the analysis of the flow control operators (i.e. OP_IF, OP_ELSE and OP_ENDIF). These operators are unable to modify the alt stack. Thus, we will ignore this stack in this part of the analysis. It is important to note that the case in which the control stack after step i does not represent an execution state is analyzed in the proof for Claim 5.14.1. Therefore, in this section we only analyze the opposite case.

- $f_i = \mathsf{OP\_IF}$
  - Case 1: $\mathsf{top}_\perp(\varphi_M^i) > 0$

    Observations

    $$J_0 = F_i + 1 \geq 1 > 0$$
    $$L_0 = 0$$

    Stacks values

    $$\chi_M^{i+1} = \perp^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \perp^{h_{k'}} \qquad \varphi_I^{i+1} = 1 \cdot \varphi_I^i$$
    $$= \perp^{L_1} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \perp^{L_{k'}} \qquad = 1 \cdot \psi_I^i$$
    $$= \psi_M^{i+1} \qquad = \psi_I^{i+1}$$

  - Case 2: $\mathsf{top}_\perp(\varphi_M^i) < 0$

    Observations

    $$J_0 = -F_i - 1 \leq -1 < 0$$
    $$L_0 = 0$$

Stacks values

$$\chi_M^{i+1} = \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{h_{k'}} \qquad \varphi_I^{i+1} = 1 \cdot \varphi_I^i$$
$$= \bot^{L_1} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \qquad = 1 \cdot \psi_I^i$$
$$= \psi_M^{i+1} \qquad = \psi_I^{i+1}$$

- Case 3: $\mathsf{top}_\bot(\varphi_M^i) = 0$

  Observations

$$J_0 = 0$$
$$L_0 = 0$$

  Stacks values

$$\chi_M^{i+1} = \bot^{d_1} \cdot z_1 \cdot \ldots \cdot z_{k'-1} \cdot \bot^{h_{k'}} \qquad \varphi_I^{i+1} = 0 \cdot \varphi_I^i$$
$$= \bot^{L_1} \cdot J_0 \cdot \ldots \cdot J_{k'-1} \cdot \bot^{L_{k'}} \qquad = 0 \cdot \psi_I^i$$
$$= \psi_M^{i+1} \qquad = \psi_I^{i+1}$$

- $f_i = \mathsf{OP\_ELSE}$
  - Case 1: $\mathsf{top}(\varphi_I^i) = 0$

    Observations

    None

    Stacks values

$$\chi_M^{i+1} = \chi_M^i \qquad \varphi_I^{i+1} = 1 \cdot \mathsf{tail}(\varphi_I^i)$$
$$= \psi_M^i \qquad = 1 \cdot \mathsf{tail}(\psi_I^i)$$
$$= \psi_M^{i+1} \qquad = \psi_I^{i+1}$$

  - Case 2: $\mathsf{top}(\varphi_I^i) = 1$

Observations

None

Stacks values

$$\begin{aligned} \chi_M^{i+1} &= \chi_M^i & \varphi_I^{i+1} &= 0 \cdot \mathsf{tail}(\varphi_I^i) \\ &= \psi_M^i & &= 0 \cdot \mathsf{tail}(\psi_I^i) \\ &= \psi_M^{i+1} & &= \psi_I^{i+1} \end{aligned}$$

- $f_i = \mathsf{OP\_ENDIF}$

  Observations

  None

  Stacks values

$$\begin{aligned} \chi_M^{i+1} &= \chi_M^i & \varphi_I^{i+1} &= \mathsf{tail}(\varphi_I^i) \\ &= \psi_M^i & &= \mathsf{tail}(\psi_I^i) \\ &= \psi_M^{i+1} & &= \psi_I^{i+1} \end{aligned}$$