UNIVERSITATEA "ALEXANDRU IOAN CUZA" IAȘI

**FACULTATEA DE INFORMATICĂ IAȘI**

LUCRARE DE LICENȚĂ

# O semantică formală pentru Bitcoin Script

propusă de

**Buțerchi Andreea**

**Sesiunea:** Iulie 2019

Coordonator științific

**Lect. Dr. Arusoaie Andrei**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAȘI
**FACULTATEA DE INFORMATICĂ IAȘI**

# O semantică formală pentru Bitcoin Script

**Buțerchi Andreea**

**Sesiunea:** Iulie 2019

Coordonator științific

**Lect. Dr. Arusoaie Andrei**

**Avizat,**
**Îndrumător Lucrare**
Titlul, Numele și prenumele: **Lect. Dr. Arusoaie Andrei**
Data: **27 iunie 2019** Semnătura: _____

**DECLARAȚIE**
**privind originalitatea conținutului lucrării de licență**

Subsemnatul **Bușterchi M. Andreea** cu domiciliul în **Comuna Călărași, Județul Botoșani** născut la data de **19.03.1997**, identificat prin CNP **2970319070027**, absolvent al Universității „Alexandru Ioan Cuza" din Iași, Facultatea de **Informatică**, specializarea **Informatică**, promoția **2016 - 2019**, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea cu titlul:

**O semantică formală pentru Bitcoin Script**

elaborată sub îndrumarea **Lect. Dr. Arusoaie Andrei**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi,
**27 iunie 2019**

Semnătură student,
**Bușterchi Andreea**

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul **„O semantică formală pentru Bitcoin Script”**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,
**27 iunie 2019**

Semnătură student,
**Bușterchi Andreea**

# Cuprins

# Capitolul 1

# Introducere

În ultimii ani s-a observat o tendință de digitalizare a economiei globale, care a adus cu sine automatizarea și, totodată, eficientizarea proceselor economice. Datorită apetenței pentru progresul tehnologic, societatea actuală a primit cu mare interes noile inovații din domeniul financiar. Un capitol important în digitalizarea sistemelor economice îl constituie introducerea monedelor virtuale și, implicit, a platformelor care suportă operații tranzacționale cu aceste monede.

Bitcoin, prima monedă virtuală descentralizată, a fost introdusă în anul 2009, iar în decursul timpului și-a câștigat statutul de cea mai populară monedă digitală în rândul utilizatorilor. Nodurile rețelei Bitcoin mențin o structură de date publică și imutabilă denumită *blockchain*. Blockchain are rolul de a stoca întregul istoric al transferurilor Bitcoin, cunoscute sub denumirea de *tranzacții* (engl. *transactions*). În momentul în care un utilizator actualizează starea structurii blockchain, restul utilizatorilor din rețea trebuie să valideze noile tranzacții înregistrate. Pentru a realiza această operație, utilizatorii trebuie să verifice dacă condițiile menționate în cadrul scripturilor sau programelor sunt satisfăcute. Scripturile reprezintă liste de instrucțiuni și comenzi, iar ca funcționalitate de bază presupun verificarea corespondenței dintre cheile publice, respectiv cheile private.

Din această perspectivă, Bitcoin mai este considerat și *programmable money*. Limbajul utilizat în redactarea scripturilor sau programelor este Bitcoin Script. Limbajul menționat anterior este considerat un mecanism de securizare a tranzacțiilor în rețeaua Bitcoin. Deși posibilitățile de a scrie programe în Bitcoin Script sunt vaste, doar o serie restrânsă de programe este acceptată în rețeaua Bitcoin. Motivul care a condus la adoptarea acestor măsuri este eliminarea punctelor de vulnerabilitate.

În prezenta lucrare ne propunem analiza unei colecții de programe în Bitcoin Script și demonstrarea unor proprietăți pe baza acestora. Pentru a realiza acest lucru, vom pune la dispoziție o semantică formală care surprinde o serie de operații ale limbajului Bitcoin Script. Contextul în care sunt redate sintaxa, respectiv semantica limbajului este realizat cu ajutorul unui sistem de gestionare al demonstrațiilor formale, Coq. Acesta din urmă oferă o certificare, conform căreia limbajul verificat îndeplinește sau nu anumite proprietăți.

## 1.1 Motivația

Scopul lucrării este de a verifica și ulterior de a valida corectitudinea unui set de programe în Bitcoin Script. Pentru a ne asigura că execuția programelor se va termina cu succes, acestea trebuie să fie corecte din punct de vedere sintactic și semantic. În acest sens, soluția propusă constă în formalizarea limbajului Bitcoin Script în Coq și modelarea unor demonstrații, care au rolul de certifica anumite proprietăți ale programelor. Astfel, având la dispoziție contextul formal redat în Coq, putem identifica programele care sunt în concordanță cu sintaxa, respectiv semantica de bază a limbajului Bitcoin Script.

## 1.2 Contribuțiile lucrării

Contribuția principală a lucrării este o semantică formală în Coq a limbajului Bitcoin Script. Scopul acestei semantici este de a descoperi greșelile din cadrul programelor scrise în Bitcoin Script și de a confirma rezultatele așteptate în urma executării acestora. În cadrul formalizării au fost surprinse următoarele categorii de operații: aritmetice, criptografice și de manipulare a stivei. Pornind de la unitățile de bază menționate anterior, modelarea programelor a fost realizată cu ușurință. Semantica descrisă anterior este îndeajuns de abstractă și permite realizarea de demonstrații pe baza proprietăților programelor. Spre exemplu, o proprietate relevantă pentru programele în Bitcoin Script este existența unei valori nenule în vârful stivei, în cazul execuției cu succes a instrucțiunilor.

Având în vedere aspectele menționate anterior, putem concluziona că modelul propus în cadrul lucrării îndeplinește criteriile necesare demonstrării proprietăților programelor în Bitcoin Script.

## 1.3 Structura lucrării

În secțiunea 2 vom menționa aspecte generale cu privire la tehnologia blockchain, moneda virtuală Bitcoin, dar și limbajul Bitcoin Script. Secțiunea 3 cuprinde o sumarizare a problemei vizate în cadrul lucrării. În secțiunea 4 prezentăm sintaxa și semantica formală a limbajului Bitcoin Script. În secțiunea 5 vom verifica o serie de proprietăți ale unor programe în Bitcoin Script. Secțiunea 6 conține concluziile acestei lucrări.

# Capitolul 2

# Context

## 2.1 Tehnologia blockchain

Tehnologia *blockchain* este deseori amintită în domeniul financiar, fiind o considerată o inovație la nivelul transferurilor cu monede digitale. Blockchain reprezintă o listă imutabilă de înregistrări, denumite blocuri, care sunt relaționate și securizate prin intermediul criptografiei. În acest context, prin menționarea termenilor *bloc* (eng. *block*), respectiv *lanț* (eng. *chain*), facem trimitere la unități de informație digitală, stocate într-o bază de date publică. [3]

La nivel de structură, blocurile stochează o serie de informații necesare pentru a identifica în manieră unică operațiile efectuate. În primul rând, dacă considerăm exemplul utilizării monedelor virtuale, un bloc stochează informații cu privire la data și ora tranzacției, precum și suma transferată. Pe de altă parte, un bloc reține date despre utilizatorii care au inițiat operația. Identitatea reală nu este menționată, ci se folosește o semnătură digitală pentru a atesta faptul că utilizatorul este deținător al sumei transferate. De asemenea, fiecare bloc stochează un id, care este la bază o funcție hash, care are rolul de a-l diferenția de celelalte blocuri.

Pentru a ilustra maniera de lucru în cadrul tehnologiei *blockchain*, vom prezenta etapele care se parcurg până la adăugarea unui nou bloc în listă:

- Inițierea unei tranzacții. Indiferent de scopul tranzacției, aceasta va fi interceptată și ulterior aprobată de nodurile rețelei.
- Verificarea tranzacției. În cadrul tehnologiei blockchain, aprobarea tranzacțiilor este realizată de către nodurile rețelei. În general, rețeaua este compusă dintr-un număr considerabil de computere, care au rolul de a verifica informațiile conținute în cadrul tranzacțiilor: data și ora, suma transferată, precum și utilizatorii care au inițiat operația.
- Adăugarea tranzacției în bloc. După ce a primit acordul de acuratețe din partea nodurilor rețelei, tranzacția poate fi asociată unui bloc, împreună cu informațiile aferente.
- Identificarea blocului. Fiecare bloc primește un id unic prin intermediul căruia se poate distinge de restul blocurilor.

După finalizarea pașilor de mai sus, blocul recent adăugat este vizibil și accesibil tuturor utilizatorilor care aderă rețelei.

Datorită principiilor de funcționare, dar și a inovației aduse, comunitatea tehnologică a găsit în decursul timpului noi potențiale utilizări pentru blockchain. Folosit inițial ca suport pentru tranzacțiile cu monede virtuale, blockchain a fost integrat ulterior în aplicații care vizează: asigurarea socială, serviciile medicale, Internet-of-Things și nu numai. [4]

Proprietățile care au consacrat tehnologia blockchain sunt:

- Descentralizarea
- Transparența
- Imutabilitatea

Blockchain este considerată o aplicație descentralizată, deoarece nu există o autoritate care să gestioneze toate operațiile din cadrul sistemului, această responsabilitate fiind suportată de către nodurile din rețea. Înainte de apariția tehnologiei blockchain, utilizatorii de servicii erau obișnuiți cu sistemele centralizate. Pentru a obține informația solicitată, utilizatorul trebuia să interacționeze cu o entitate care agrega central toate datele din sistem. Un exemplu relevant în acest sens sunt instituțiile bancare. Pentru a realiza o operație bancară de orice tip, utilizatorul trebuie să primească confirmarea din partea băncii, trecând astfel prin mai multe formalități.

Deși sistemele centralizate sunt folosite într-o diversitate de arii, acestea prezintă o serie de vulnerabilități:

- Având statutul de sistem centralizat, datele sunt stocate într-o singură locație. Acest lucru conduce la existența unui singur punct de atac pentru eventualii utilizatori malițioși.
- În cazul impunerii unei reactualizări software, întregul sistem este oprit din cauza operațiilor de mentenanță.
- În eventualitatea unei pierderi majore de date, utilizatorii sunt privați de accesul la informațiile pierdute.

Inovația adusă de tehnologia blockchain constă în eliminarea entității centrale. Astfel, fiecare utilizator din rețea va deține un istoric al tuturor operațiilor și va avea rolul de a confirma noile acțiuni efectuate între două sau mai multe noduri din rețeaua *peer-to-peer*. Considerând exemplul monedelor virtuale, transferul de bunuri digitale se poate realiza de la un nod la altul, de unde și numele de rețea *peer-to-peer*.

O altă caracteristică a aplicației blockchain constă în transparența acțiunilor realizate de către utilizatorii rețelei. Din acest motiv survine întrebarea dacă blockchain este privat sau nu. Fiecare nod din sistem poate accesa istoricul tuturor operațiilor și este implicat în validarea operațiilor ulterioare. Odată aderat la rețea, nodul primește o copie a bazei de date blockchain, care este actualizată de fiecare dată când un nou bloc este adăugat. Deși fiecare copie a bazei de date este identică, faptul că este distribuită tuturor computerelor din rețea, manipularea informațiilor este extrem de dificil de realizat.

Deoarece nu există un punct central de stocare al datelor, orice utilizator care încearcă prin acțiuni malițioase să altereze informațiile, va trebui să modifice simultan toate copiile bazei de date răspândite în rețea. Acest lucru este foarte greu de realizat, luând în considerare numărul foarte mare de noduri din rețea. Cu toate că transparența și accesul la date este asigurat în permanență, identitatea utilizatorilor implicați în realizarea acțiunilor este protejată. În acest sens, identitatea reală a nodurilor este securizată prin procese criptografice avansate. Astfel, fiecărui utilizator îi corespunde o adresă publică, vizibilă tuturor, dar și o adresă privată, prin intermediul căreia poate confirma că este deținătorul de drept al unui bun digital.

În contextul aplicațiilor financiare, toate tranzacțiile pot fi accesate și vizualizate, identitățile persoanelor implicate fiind însă secrete. Având în vedere faptul că identitățile reale ale utilizatorilor sunt protejate, se pune problema securității rețelei blockchain.

Tehnologia blockchain propune o serie de metode de a păstra securitatea și încrederea utilizatorilor. În primul rând, blocurile sunt stocate în manieră liniară și cronologică. Odată ce un nou bloc a fost adăugat, este foarte dificil să alterezi conținuturile acestuia. Acest lucru este asigurat de faptul că fiecare bloc conține propria valoare hash, împreună cu valoarea hash a blocului anterior. Analizând proprietățile funcțiilor hash, putem remarca că orice schimbare produsă în valoarea encriptată, va schimba totalmente valoarea hash. Aplicând acest principiu tehnologiei blockchain, remarcăm că imutabilitatea datelor este îndeplinită. Deși la nivel teoretic coruperea blocurilor este posibilă, puterea computațională necesară depășește cu mult limitele realizării acestui lucru.

În decursul timpului a fost ridicată problema încrederii, și anume de unde ne putem da seama dacă în rețea există sau nu utilizatori malițioși. Acest aspect a fost rezolvat prin propunerea unui sistem de verificare a credibilității nodurilor din rețea denumit *Proof Of Work* [5]. Pentru a introduce un nou bloc, utilizatorul trebuie să rezolve o problemă matematică complexă care necesită o putere computațională ridicată. Având în vedere faptul că nu există o ierarhie prestabilită, ci toți utilizatorii au drepturi și privilegii egale, nodurile corupte trebuie să depună același efort computațional pentru a adăuga noi blocuri în blockchain. Cu toate că sistemul *Proof Of Work* nu împiedică existența acțiunilor corupte, costul de a organiza atacuri asupra rețelei depășește beneficiile care ar putea fi obținute ulterior.

Deoarece datele sunt stocate într-o manieră permanentă, tehnologia blockchain nu permite alterarea sau modificarea informațiilor care circulă în sistem. Imutabilitatea în contextul sistemului blockchain are ca efect imposibilitatea modificării informațiilor deja conținute în sistem. Această proprietate este satisfăcută de utilizarea criptografiei. Fiecare bloc conține suplimentar și valoare encriptată a blocului anterior. Astfel, alterarea unui singur bloc, produce modificarea implicită a tuturor blocurilor din listă. Prin prezentarea acestei proprietăți, rețeaua blockchain este protejată de atacuri sau vulnerabilități.

La nivel de structură, blockchain este menținut de o rețea *peer-to-peer*, reprezentată de o colecție de noduri interconectate. Nodurile sunt computerele personale ale utilizatorilor sau orice device conectat la Internet, care prelucrează un input, operează anumite funcții asupra sa și returnează un output. O rețea *peer-to-peer* își partiționează capacitatea de lucru în mod egal către toate nodurile participante, fiecare având aceleași privilegii în cadrul sistemului. Ideea de bază care se promovează în cadrul rețelei este de a nu exista ierarhii și de a oferi drepturi egale tuturor nodurilor, indiferent de gradul

lor de implicare.

În contextul actual, cele mai cunoscute aplicații, repectiv platforme care au la bază tehnologia blockchain sunt *Bitcoin*, *Ethereum*, dar și *Cardano*.

## 2.2 Bitcoin

Bitcoin reprezintă prima implementare cu succes a unei monede digitale distribuite. Principiile de funcționare ale monedei virtuale Bitcoin au revoluționat industria financiară, deoarece utilizatorii dispun de autonomie monetară. Astfel, fără implicarea instituțiilor bancare în administrarea acțiunilor din cadrul sistemului, nodurile rețelei au sarcina de a monitoriza colectiv tranzacțiile efectuate, dar și de a confirma numărul de Bitcoini puși în circulație. Acest procedeu se numește *consensus* și presupune implicarea tuturor nodurilor în confirmarea fiecărei operații din sistem. Fiind o rețea *peer-to-peer*, tranferul de bunuri digitale se realizează de la un utilizator la altul, fără a depinde de o autoritate centrală. [13]

O tranzacție reprezintă transferul unei valori în Bitcoini de la un nod la altul din rețea. Odată inițiată o tranzacție, aceasta este propagată în întreaga rețea, ulterior fiind confirmată de fiecare nod. O tranzacție colectează sub forma unui nou input, valorile de output ale altor tranzacții efectuate cu succes, iar în momentul primirii confirmării, procedeul se reia pentru următoarele tranzacții care se vor folosi de valoarea transferată anterior. Valoarea de output astfel obținută poate intra în posesia nodului destinație, doar dacă acesta oferă argumentele care deblochează valoarea transferată, în general o semnătură digitală. Pentru a introduce cu succes o tranzacție în rețeaua Bitcoin, utilizatorii trebuie să execute un program denumit *portofel* (eng. *wallet*).

Fiecare portofel digital două chei criptografice unice: o cheie publică, respectiv o cheie privată. Cheia publică reprezintă locația unde sunt depozitate tranzacțiile și ulterior retrase. Chiar dacă un utilizator primește o plată în Bitcoini la adresa sa publică, acesta nu va putea extrage suma destinată lui cu o cheie privată duplicat. Cheia publică a unui utilizator reprezintă o variantă mai scurtă a cheii private, generate printr-un algoritm matematic. Având în vedere complexitatea metodei de generare, este imposibil de realizat procesul invers de criptare pentru a obține cheia privată, pornind de la cheia publică. Din acest motiv, tehnologia blockchain este considerată confidențială.

Un alt element de interes în cadrul tehnologiei blockchain este maniera de înregistrare a datelor. Protocolul blockchain descurajează existența mai multor liste de blocuri în cadrul sistemului, utilizând un proces denumit *consensus*. În prezența mai multor copii diferite ale blockchain, protocolul *consensus* va adopta cea mai lungă listă de blocuri disponibilă. Astfel, nodurile din rețea vor avea acces la o listă de blocuri cu cel mai mare nivel de încredere.

Având la bază tehnologia blockchain, fiecare tranzacție receptată în cadrul rețelei este asociată unui bloc, care este ulterior adăugat într-o secvență de blocuri. Procesul prin care se determină corectitudinea unei tranzacții se numește *Proof Of Work* și presupune rezolvarea unei probleme matematice. Odată primit un număr suficient de

confirmări, o tranzacție este declarată ireversibilă. Pentru a adăuga o nouă tranzacție în cadrul unui bloc, este necesară o putere computațională mare și de resurse hardware suficiente. Nodurile implicate în acest proces se numesc *miners*. Dacă rezolvarea problemei matematice s-a terminat cu succes, nodurile *miners* primesc o recompensă pentru efortul computațional depus, acesta fiind totodată și momentul punerii în circulație de noi Bitcoini.

Un aspect important în păstrarea securității este utilizarea proceselor criptografice. Fiecărui nod din rețea îi sunt asignate o adresă publică, vizibilă tuturor, și o adresă privată prin intermediul căreia demonstrează că este proprietarul unei anumite valori digitale. Tranzacțiile cu moneda digitală Bitcoin sunt însoțite de programe care înglobează adresa destinatarului, suma transferată, precum și alte condiții de validare. Aceste programe sunt redactate în limbajul de programare Bitcoin Script și sunt executate de fiecare nod din rețea, pentru a confirma proprietatea unui anumit utilizator asupra valorii transferate. Forma programelor diferă în funcție de tipul de tranzacție efectuată, dar și de nivelul de securitate dorit.

Statisticile în domeniu menționează o serie de avantaje, dar și dezavantaje ale arhitecturii blockchain. [2]

Avantaje:

- Acuratețea listei de blocuri. Tranzacțiile sunt aprobate de o rețea care conține milioane de calculatoare. Acest fapt reduce erorile umane, rezultatul fiind o mai mare veridicitate a datelor salvate în interiorul blocurilor.
- Reducerea costurilor. În mod normal, prin utilizarea serviciilor care presupun existența unei autorități centrale, există costuri suplimentare pentru mentenața datelor. În cadrul blockchain, elimină verificările unei autorități centrale și implicit, costurile aferente acesteia.
- Tranzacții private. Multe rețele blockchain lucrează ca și baze de date publice, astfel încât, orice persoană cu acces la internet poate vedea lista de tranzacții efectuate în carul rețelei. Chiar dacă utilizatorii pot accesa detaliile privitoare la tranzacții, informațiile personale ale utilizatorilor care au făcut respectivele operații rămân ascunse. Deseori acțiunile sunt considerate a fi anonime, când de fapt, ele sunt confidențiale.
- Tranzacții sigure. Confirmările nodurilor din rețea conferă securitatea vizată.
- Eliminarea autorității centrale. Operațiile nu sunt monitorizate de un singur punct de autoritate, ci toți utilizatorii sunt implicați în acest procesul de validare al tranzacțiilor.
- Securitatea operațiilor. Fiecare operație din cadrul sistemului este securizată prin intermediul unor procese criptografice avansate.
- Reducerea taxelor. Utilizatorii sunt scutiți de comisioanele bancare, deoarece monedele virtuale nu sunt reglementate de nicio autoritate internațională.
- Eliminarea fraudelor. Tranzacțiile cu monede virtuale sunt ireversibile, și rămân înregistrate permanent în sistem.

- Efectuarea instantă a plăților. Timpul de efectuare al unei plăți este scurt, în comparație cu serviciile bancare tradiționale.
- Eliminarea granițelor. Monedele virtuale fac tranzacțiile la nivel global posibile, deoarece nu sunt limitate de schimburile valutare intermediare. [1]

Dezavantaje:

- Costurile tehnologiei. Chiar dacă tehnologia blockchain ne salvează de uzualele comisioane pentru efectuarea tranzacțiilor, există alte surse de costuri. Spre exemplu, sistemul *Proof of work* consumă o cantitate considerabilă de energie pentru a valida tranzacțiile.
- Activități ilegale. Confidențialitatea în rețeaua blockchain protejează pe de o parte identitatea utilizatorilor, iar pe de altă parte, permite existența traficului ilegal de bunuri. Deoarece identitățile reale ale utilizatorilor sunt complet ascunse, nu putem ști cu adevărat cine este persoana din spate și ce fel de bunuri pune spre vânzare.

## 2.3 Ethereum și Cardano

Deși este deseori asociat cu monedele virtuale, tehnologia blockchain a fost integrată cu succes în multe alte aplicații cu manieră de lucru descentralizată. Ethereum pune la dispoziția utilizatorilor posibilități diverse de a crea noi aplicații descentralizate, care vizează nu numai domeniul financiar, ci și cel imobiliar, dar și cel al asigurărilor sociale. Ca și Bitcoin, Ethereum reprezintă o rețea blockchain publică și distribuită. Cu toate acestea, există numeroase diferențe din punct de vedere tehnic între cele două tehnologii. Pe când scopul strict al Bitcoin este de a oferi un sistem electronic de plată, Ethereum se axează pe rularea oricărei aplicații descentralizate.

Spre deosebire de Bitcoin, moneda virtuală utilizată în cadrul rețelei Ethereum are cu totul alt scop. Deși este folosit pe piața publică pentru a achiziționa diverse bunuri, Ether este considerat în același timp „combustibil" pentru rețeaua Ethereum. Dintr-o perspectivă generală, Ethereum este o platformă care permite rularea optimă a *smart contractelor*. Execuția *smart contractelor* pornește automat când sunt satisfăcute anumite condiții și necesită o putere computațională foarte mare. Deoarece resursele computaționale trebuie plătite, aici intervine moneda virtuală Ether. Fiecare persoană care interacționează cu aplicațiile dezvoltate pe platforma Ethereum trebuie să plătească pentru serviciile utilizate. Similar cu maniera de lucru în rețeaua Bitcoin, nodurile *miner* primesc o recompensă în Ether pentru fiecare nou block inclus în blockchain, acesta fiind și momentul punerii în circulație de noi Ether. Așadar, având la bază tehnologia blockchain, fiecare aplicație dezvoltată pe platforma Ethereum este considerată: imutabilă, incoruptibilă și sigură. [6]

Cardano este o platformă tehnologică care are la bază conceptul blockchain și este capabilă de a rula aplicații financiare folosite de organizații sau chiar instituții guvernamentale. Scopul platformei este de a impune un grad mai mare de scalabilitate și securitate prin arhitectura abordată. [7]

Fiecare dintre platformele amintite mai sus presupun execuția unor unități de cod, precum: programe, scripturi, respectiv smart contracte. Acestea din urmă sunt redactate utilizând anumite limbaje de programare, care înglobează funcționalități specifice fiecărei platforme.

## 2.4 Bitcoin Script

Bitcoin Script este limbajul de programare care stă la baza procesării tuturor tranzacțiilor cu moneda virtuală Bitcoin. Este un limbaj simplu, intuitiv și similar ca principiu de funcționare cu Forth. Elementele componente alte limbajului sunt fie *datele*, cheile publice sau semnăturile, fie instrucțiunile denumite OPCODES, care reprezintă funcții aplicabile asupra datelor. Deoarece nu există structuri repetitive, limbajul Bitcoin Script este considerat Turing incomplet.

Executarea programelor este liniară, iar ca structură auxiliară se folosește o stivă de operații. Pentru procesarea operațiilor se folosește notația postfixată. Aceste convenții au fost adoptate pentru a evita vulnerabilitățile și pentru a preveni eventualele intruziuni malițioase. Bitcoin Script conține un set relativ mic de instrucțiuni care cuprinde: structuri decizionale, operații aritmetice, operații de manipulare a stivei, operații pe biți, dar și funcții criptografice. Datorită nivelului ridicat de securitate impus de rețeaua Bitcoin, programele redactate în Bitcoin Script conțin deseori funcții criptografice. Un program redactat în Bitcoin Script reprezintă o listă de instrucțiuni care însoțesc fiecare tranzacție din sistem și conține informații cu privire la destinatarul sumei transferate. Un script tipic pentru un transfer cu moneda virtuală Bitcoin înglobează adresa nodului către care se face tranferul. [8]

Pentru a intra în posesia valorii transferate, destinatarul trebuie să pună la dispoziție următoarele elemente:

- *Cheia publică*. Atunci când va fi aplicată o funcție hash asupra cheii publice, aceasta va trebui să aibă aceeași valoare ca și cea menționată în script.
- *Semnătura digitală*. Pentru a demonstra că este deținătorul cheii private corespunzătoare cheii publice, nodul destinatar trebuie să ofere o semnătură digitală care să ateste acest lucru.

Un script este considerat valid, dacă în urma execuției în vârful stivei se află o valoare nenulă. Această condiție este îndeplinită doar în cazul în care destinatarul furnizează datele de validare corecte.

În acest context este de remarcat faptul că Bitcoin Script are și câteva puncte de slăbiciune. În decursul timpului s-a renunțat la o parte din operații, deoarece ofereau posibilitatea realizării acțiunilor malițioase. Astfel, prin eliminarea operatorilor de multiplicitate, dar și a structurilor repetitive, s-a redus considerabil vulnerabilitatea sistemului. Cu toate acestea, Bitcoin Script deține o serie de proprietăți incontestabile. O proprietate importantă a Bitcoin Script este faptul ca poate fi supus unei analize statice pentru a determina complexitatea operațiilor care urmează a fi executate. Prin calculul inițial al complexității operațiilor, mai cu seamă cele care implică criptografia, se poate determina care programe depășesc sau nu pragul superior de complexitate impus. În cazul în care un program depășește limita impusă va fi considerat invalid. Programele redactate în limbajul Bitcoin Script oferă flexibilitate, deoarece numărul de

parametri poate varia, iar cerințele de validare pot fi diverse. Din nevoia de a tipiza efectuarea tranzacțiilor, în decursul timpului s-au impus anumite modele de programe, care se pliază pe nevoile clientului.

Deoarece s-a dorit evitarea erorilor umane, o parte din operatorii care au existat în versiunile anterioare de Bitcoin Script a fost eliminată. Analizând posibile scenarii, s-a descoperit faptul că unii operatori erau surse recurente de erori, care puteau fi exploatate de utilizatorii malițioși. Așadar, prin verificarea scripturilor și a programelor se pot descoperi eventuale vulnerabilități sau erori. În contextul lucrării, prin furnizarea unei semantici formale, putem defini și ulterior demonstra că orice operație din cadrul limbajului vizat îndeplinește comportamentul așteptat. Prin simularea unor situații specifice, se pot descoperi acțiuni frauduloase, care conduc la destabilizarea sistemului. [9]

# Capitolul 3

# Descrierea problemei

În această secțiune detaliem principalele funcționalități ale limbajului Bitcoin Script și identificăm proprietățile care trebuie surprinse de o semantică formală a acestui limbaj. Prin urmare, vom începe cu o descriere generală a limbajului, sintaxa și semantica informală și câteva particularități ale limbajului Bitcoin Script.

## 3.1 Prezentarea problemei

Anumite operații existente în cadrul unui limbaj de programare pot fi surse exploatabile de vulnerabilitate. Scopul lucrării este de a identifica eventualele probleme care pot apărea la rularea programelor care însoțesc tranzacțiile cu moneda virtuală Bitcoin. În acest sens, vom prelucra un set de programe utilizate în mod curent și vom modela demonstrații formale care să ateste execuția cu succes a acestora.

## 3.2 Soluții existente

Pentru limbajul BitcoinScript nu există momentan semantici formale definite în Coq. Simplicity, utilizat pe platforma Ethereum, este unul din limbajele pentru care există definită o semantică formală în Coq. [12]

# Capitolul 4

# O semantică formală pentru Bitcoin Script

În această secțiune vom prezenta în detaliu o semantică formală pentru limbajul Bitcoin Script în Coq. În linii generale, Coq reprezintă o unealtă software care ghidează utilizatorul în redactarea demonstrațiilor formale. În acest context, Coq permite definierea în manieră matematică a funcționalităților limbajelor de programare, furnizând o serie de pași intermediari, necesari în realizarea cu succes a demonstrațiilor. Așadar, prin selectarea celor mai uzuale operații, dar și prin simularea funcținalităților limbajului, am realizat o semantică formală pentru Bitcoin Script, care ulterior a fost integrată în redactarea de programe care însoțesc tranzacțiile cu moneda virtuală Bitcoin. [10]

## 4.1 Sintaxa

Sintaxa formală este redată cât mai fidel, urmând convențiile de notare prevăzute de sintaxa reală a limbajului Bitcoin Script. Programele în Bitcoin Script sunt formate din instrucțiuni, operații sau comenzi denumite *opcodes* (operational codes). Există mai multe tipuri de *opcodes*, care vizează următoarele categorii: operații aritmetice, structuri decizionale, funcții de manipulare a stivei, dar și operații criptografice. Deoarece execuția programelor implică utilizarea unei strive pentru procesarea instrucțiunilor, operațiile de control al stivei sunt cele mai frecvente. De asemenea, funcțiile criptografice fac parte din setul de operații uzuale. Acest lucru se datorează faptului că Bitcoin Script are la bază procese criptografice avansate, care asigură securitatea, dar și protecția datelor personale ale utilizatorilor.

În cele ce urmează vom face o scurtă introducere în tipurile de instrucțiuni existente în Bitcoin Script și vom specifica o descriere pentru fiecare unitate OP_CODE.[9]

**Constante**

- OP_0, OP_FALSE: valoarea numerică 0 va fi adăugată în stiva de operații.
- OP_1, OP_TRUE: valoarea numerică 1 va fi adăugată în stiva de operații.
- OP_2 - OP_16: valorile numerice cuprinse între 2, respectiv 16 vor fi adăugate în stiva de operații.

**Structuri decizionale**

- OP_IF: dacă valoarea din vârful stivei este nenulă, instrucțiunile de pe ramura *if* vor fi executate.
- OP_ELSE: dacă valoarea din vârful stivei este nulă, instrucțiunile de pe ramura *else* vor fi executate.
- OP_ENDIF: încheie structura decizională.
- OP_VERIFY: marchează o tranzacție ca fiind invalidă, dacă valoarea din vârful stivei este nulă.
- OP_RETURN: marchează o tranzacție ca fiind invalidă.

**Funcții de manipulare a stivei**

- OP_DEPTH: în vârful stivei se va adăuga numărul curent de elemente din stivă.
- OP_DROP: elimină un element din vârful stivei.
- OP_DUP: duplică elementul din vârful stivei.
- OP_SWAP: interschimbă primele două elementele din vârful stivei.

**Operații pe biți**

- OP_EQUAL: verifică dacă două componente sunt egale. Va adăuga valoarea 1 în stivă dacă egalitatea este satisfăcută, 0 în caz contrar.
- OP_EQUALVERIFY: verifică egalitatea dintre două componente, dar mai aplică anterior și operatorul OP_VERIFY.

**Operații aritmetice**

- OP_ADD: adună valorile primelor două elemente din stivă și pune rezultatul în stiva de operații.
- OP_SUB: scade valorile primelor două elemente din stivă și pune rezultatul în stiva de operații.
- OP_NOT: neagă valoarea primită ca și input.
- OP_LESSTHAN: returnează 1 dacă este satisfăcută condiția, 0 în caz contrar.
- OP_GREATERTHAN: returnează 1 dacă este satisfăcută condiția, 0 în caz contrar.
- OP_MIN: returnează elementul cu valoare minimă.
- OP_MAX: returnează elementul cu valoare maximă.

**Funcții criptografice**

- OP_SHA1: aplică funcția hash SHA1.
- OP_SHA256: aplică funcția hash SHA256.
- OP_RIPEMD160: aplică funcția hash RIPEMD160.
- OP_HAS160: aplică funcția hash SHA256, iar apoi RIPEMD160.
- OP_HASH256: aplică funcția hash SHA256.
- OP_CHECKSIG: verifică semnătura digitală primită ca și argument.
- OP_CHECKSIGVERIFY: verifică semnătura digitală primită ca argument, iar ulterior aplică OP_VERIFY.
- OP_CHECKMULTISIG: verifică semnăturile digitale primite ca și argumente.
- OP_CHECKMULTISIGVERIFY: verifică semnăturile digitale primite ca și argumente, iar ulterior aplică OP_VERIFY.

Reprezentarea operațiilor în Coq este similară cu sintaxa reală a limbajului Bitcoin Script. Mai jos prezentăm o definiție inductivă în Coq pentru sintaxa unităților OP_CODE:

```
Inductive OP_CODES : Type :=
  (* Funcții criptografice *)
  | OP_RIPEMD160 : OP_CODES
  | OP_SHA1 : OP_CODES
  | OP_SHA256 : OP_CODES
  | OP_HASH160 : OP_CODES
  | OP_CHECKSIG : OP_CODES
  | OP_CHECKSIGVERIFY : OP_CODES
  | OP_CHECKMULTISIG : OP_CODES
  | OP_CHECKMULTISIGVERIFY : OP_CODES

  (* Operații pe biți *)
  | OP_EQUAL : OP_CODES
  | OP_EQUALVERIFY : OP_CODES
  | OP_VERIFY : OP_CODES
  | OP_RETURN : OP_CODES

  (* Operații de control al stivei *)
  | OP_IFDUP : Z -> OP_CODES
  | OP_DEPTH : OP_CODES
  | OP_DROP : OP_CODES
  | OP_DUP : OP_CODES
  | OP_2DUP : OP_CODES
  | OP_NIP : OP_CODES
```

```
  | OP_OVER : OP_CODES
  | OP_PICK : OP_CODES
  | OP_ROLL : OP_CODES
  | OP_ROT : OP_CODES
  | OP_SWAP : OP_CODES
  | OP_PUSH : Z -> OP_CODES

  (* Operații cu structuri de timp *)
  | OP_CHECKLOCKTIMEVERIFY : Z -> OP_CODES
  | OP_CHECKSEQUENCEVERIFY : Z -> OP_CODES

  (* Structuri decizionale *)
  | OP_IF : OP_CODES
  | OP_NOTIF : OP_CODES
  | OP_ELSE : OP_CODES
  | OP_ENDIF : OP_CODES

  (* Constante *)
  | OP_NUM : Z -> OP_CODES
  | OP_NEGATE : nat -> OP_CODES
  | OP_0 : OP_CODES
  | OP_1 : OP_CODES
  | OP_NOP : OP_CODES

  (* Operații aritmetice *)
  | OP_1ADD : OP_CODES
  | OP_1SUB : OP_CODES
  | OP_ABS : OP_CODES
  | OP_NOT : OP_CODES
  | OP_ADD : OP_CODES
  | OP_SUB : OP_CODES
  | OP_NUMEQUAL : Z -> Z -> OP_CODES
  | OP_NUMEQUALVERIFY : Z -> Z -> OP_CODES
  | OP_NUMNOTEQUAL : Z -> Z -> OP_CODES
  | OP_LESSTHAN : Z -> Z -> OP_CODES
  | OP_GREATERTHAN : Z -> Z -> OP_CODES
  | OP_MIN : Z -> Z -> OP_CODES
  | OP_MAX : Z -> Z -> OP_CODES
  | OP_WITHIN : Z -> Z -> Z -> OP_CODES

  (* Cuvinte rezervate *)
  | OP_PUBKEYHASH : OP_CODES
  | OP_PUB_KEY : OP_CODES
  | OP_INVALIDOPCODE : OP_CODES
  | OP_TIME_TO_ELAPSE : nat -> OP_CODES.
```

Pentru a ilustra asemănarea celor două sintaxe, vom considera un program oarecare redactat atât în Bitcoin Script, cât și în Coq.

```
(* Sintaxa în Bitcoin Script *)
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

(* Sintaxa în Coq *)
[OP_DUP; OP_HASH160; OP_PUSH <pubKeyHash>; OP_EQUALVERIFY; OP_CHECKSIG]
```

După cum se poate observa, la nivel de sintaxă programele prezintă aceleași particularități.

În Coq programele sunt simulate sub formă de liste de operații. Acest lucru este subliniat în următoarea definiție:

```
Definition OP_CODES_List := list OP_CODES.
```

Fiecare OP_CODE este preluat și adăugat în stiva de operații, iar în funcție de valoarea lui semantică, se vor apela o serie de funcții sau proceduri.

De asemenea, stiva de operații este modelată tot cu ajutorul unei liste, asupra căreia se aplică diverse funcții de manipulare a conținuturilor.

```
Definition stack := list Z.
```

## 4.2 Semantica

În această secțiune vom descrie semantica fiecărei unități OP_CODE. Operațiile asupra cărora ne vom centra atenția sunt cele de control al stivei. Deoarece maniera de evaluare a programelor în Bitcoin Script este liniară, se va utiliza ca și structură de date auxiliară o stivă. Astfel, procesarea datelor se va face de la stânga la dreapta, fiecare OP_CODE din cadrul programului fiind adăugat în stivă. Pentru a facilita evaluarea programelor, am considerat o serie de funcții suport:

```
(* top - returnează elementul din vârful stivei *)
Definition top (default : Z) (s : stack) : Z :=
  match s with
  | nil => default
  | h::t => h
  end.

(* pop - extrage elementul din vârful stivei *)
Definition pop (s : stack) : stack :=
  match s with
  | nil => nil
  | h :: t => t
  end.

(* push - adaugă un nou element în vârful stivei *)
Definition push (item : Z) (s : stack) : stack :=
  match s with
  | nil => [item]
  | h :: t => item :: h :: t
  end.
```

De asemenea, fiecare operație descrisă din punct de vedere sintactic va primi o semantică specifică după cum urmează:

```
Definition compute_op (s:stack) (op:OP_CODES) : stack :=
  match op with
  | OP_NOT => if (Z.eqb (top 0 s) 0%Z)
              then (push 1 (pop s))
              else (push 0 (pop s))

  | OP_DUP => push (top 0 s) s
  | OP_2DUP => push (top 0 s)
                    (push (top 0 (pop s)) s)
  | OP_DROP => pop s
  | OP_PUSH n => push n s
  | OP_1 => push 1 s
  | OP_0 => push 0 s
  | OP_NUM n => push n s
  | OP_ADD => push (Zplus (top 0 s) (top 0 (pop s)))
                   (pop (pop s))
  | OP_SUB => push (Zminus (top 0 s) (top 0 (pop s)))
                   (pop (pop s))
  | OP_SHA1 => compute_SHA1 s
  | OP_SHA256 => compute_SHA256 s
  | OP_TIME_TO_ELAPSE n => compute_elapsed_time n s
  | OP_EQUAL => if (Z.eqb (verify_eq (top 0 s)
                                     (top 0 (pop s))))
                                     1%Z
                then (push 1 (pop (pop s)))
                else (push 0 (pop (pop s)))
  | OP_IF => if (Z.eqb (top 0 s) 1%Z) then pop s else s
  | OP_CHECKSIG => if (Z.eqb (verify_eq (top 0 s)
                                (top 0 (pop s))))
                                1%Z
                   then (push 1 (pop (pop s)))
                   else (push 0 (pop (pop s)))
  | _ => s
  end.
```

Metoda COMPUTE_OP atribuie fiecărei unități OP_CODE o semantică și va fi utilizată în momentul evaluării programelor. Deoarece am dorit simplificarea procesului de verificare al programelor, funcțiile criptografice au fost considerate funcții identitate. Spre exemplu, aplicarea unității OP_SHA1 are ca efect returnarea valorii primite ca parametru, după cum sugerează și implementarea:

```
Definition compute_SHA1 (s:stack) : stack :=
  match s with
  | nil => []
  | h :: t => push (top 0 s) (pop s)
  end.
```

Similar au fost implementate și restul metodelor criptografice, inclusiv: OP_SHA256, OP_HASH160, dar și OP_CHECKSIG.

Deoarece Bitcoin Script nu este Turing complet, implicit nu conține instrucțiuni repetitive, singura structură regăsită în implementarea limbajului este structura decizională *if-else*. Execuția instrucțiunii decizionale *if-else* se manifestă diferit în comparație cu alte limbaje de programare. Primul pas este cel de verificare. Dacă în vârful stivei de operații se află valoarea 1, atunci se vor introduce pe rând în stivă instrucțiunile de pe ramura OP_IF - OP_ELSE, respectiv de pe ramura OP_ELSE - OP_ENDIF, în cazul în care în vârful stivei se găsește o valoare nulă. Mai jos poate fi consultată implementarea metodei recursive care validează alegerea ramurei corespunzătoare în funcție de valoarea elementului din vârful stivei:

```
Fixpoint split_if (script:OP_CODES_List)(b:Z):
                  OP_CODES_List*OP_CODES_List :=
  match script with
  | nil => (nil, nil)
  | OP_ENDIF::t1 => (t1, t1)
  | OP_ELSE::t1 => split_if t1 1
  | s::t => if (Z.eqb branch 0%Z)
            then let (a, b) := (split_if t branch) in (s::a, b)
            else let (a, b) := (split_if t branch) in (a, s::b)
  end.
```

Ordinea de evaluare a programelor este de la stânga la dreapta, iar pentru procesarea operațiilor se folosește notația postfixată (eng. *polish notation*). Vom considera un exemplu simplu pentru a ilustra procesul de prelucrare al operațiilor. Să presupunem că dorim să verificăm următorul calcul matematic:

| $3 + 7 = 10$ |
|---|

În Bitcoin Script, expresia anterioară va fi transpusă astfel:

| OP_3 OP_7 OP_ADD OP_10 OP_EQUAL |
|---|

În cele ce urmează vom prezenta pașii de execuție ai programului.

Pasul 1: Valoarea numerică 3 este adăugată în stivă.

| 3 |
|---|

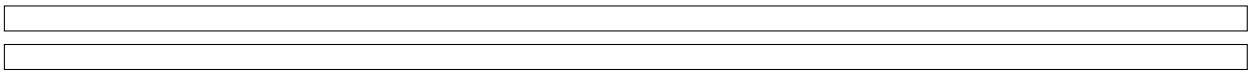
Pasul 2: Valoarea numerică 7 este adăugată în stivă.

| 7 |
|---|
| 3 |

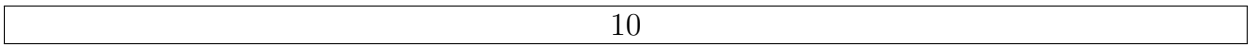
Pasul 3: Operatorul OP_ADD va extrage pe rând primele 2 elemente din vârful stivei, iar după aplicarea operației, rezultatul va fi adăugat în stivă. Valoarea numerică 7 este extrasă din stivă.

| |
|---|
| 3 |

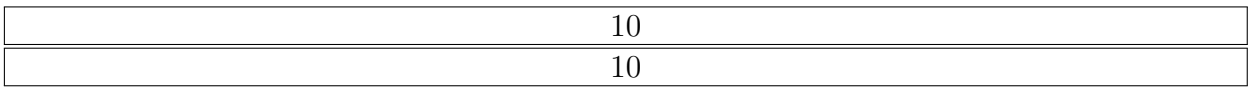
Pasul 4: Valoarea numerică 3 este extrasă din stivă.



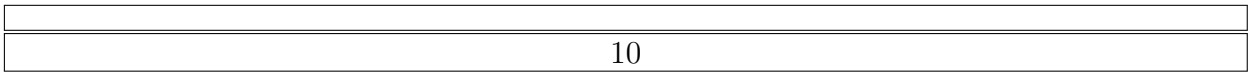
Pasul 5: Rezultatul este adăugat în stivă.



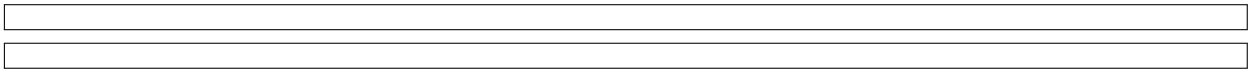

Pasul 6: Valoarea numerică 10 este extrasă din stivă.




Pasul 7: Operatorul OP_EQUAL extrage prima valoare din stivă.




Pasul 8: Operatorul OP_EQUAL extrage următoarea valoare din stivă.



Pasul 9: Operatorul OP_EQUAL verifică egalitatea celor două valori. În caz de egalitate, în stivă va fi adăugat TRUE, în caz contrar, FALSE. În contextul de față, valoarea adăugată este TRUE.


Fiecare program redactat în Bitcoin Script urmărește același tipar de execuție. Pentru a simula acest comportament, am considerat o funcție suport, care prevede câte un caz de execuție pentru fiecare unitate OP_CODE.

```
Fixpoint evaluate_script (s:stack)(script:OP_CODES_List)(n : nat):
                         stack :=
match n with
| O => s
| S m => match script with
        | nil => s
        | OP_IF::t => if (Z.eqb (top 0 s) 1%Z)
                      then let (a,b) := (split_if t 0)
                                        in (evaluate_script
                                            (pop s) a m)
                      else let (a,b) := (split_if t 0)
                                        in (evaluate_script
                                            (pop s) b m)
        | OP_VERIFY::t => if (Z.eqb (top 0 s) 1%Z)
                          then (evaluate_script (pop s) t m)
                          else (evaluate_script s [OP_PUSH 0] m)
```

```
        | OP_CHECKLOCKTIMEVERIFY n :: t => if (Z.leb n (top 0 s))
                                           then (evaluate_script
                                                 s t m)
                                           else (evaluate_script s
                                                 [OP_PUSH 0] m)
        | OP_CHECKSEQUENCEVERIFY n :: t => if (Z.geb n (top 0 s))
                                           then (evaluate_script
                                                s t m)
                                           else (evaluate_script s
                                                 [OP_PUSH 0] m)
        | OP_CHECKMULTISIG :: t =>
          if (compare_lists
              (first_n_elems (Z.to_nat (top 0 s))
                             (pop s))
              (first_n_elems (Z.to_nat (top 0
                             (remove_n_elems (Z.to_nat
                                              (top 0 s))
                                              (pop s))))
              (pop (remove_n_elems (Z.to_nat (top 0 s)) (pop s)))))
          then [1%Z] else [0%Z]
        | h::t => (evaluate_script (compute_op s h) t m)
      end
end.
```

Metoda *evaluate_script* reprezintă o funcție recursivă care preia fiecare OP_CODE din program și calculează rezultatul final, pe care îl adaugă ulterior în stivă. Execuția unui program se termină cu succes în cazul în care în stivă se va găsi o valoare nenulă. În acest sens, funcția *check_stack* va determina dacă un program s-a terminat cu succes sau nu.

```
Definition check_stack (s:stack) : bool :=
  match s with
  | nil => true
  | h::t => if (Z.eqb h 1%Z) then true else false
end.
```

# Capitolul 5

# Demonstrații

Partea experimentală a lucrării constă în demonstrarea unor proprietăți ale unor programe în Bitcoin Script. Vom considera o serie de programe și scripturi comune pentru rețeaua Bitcoin. În restul sectiunii vom specifica proprietati pentru fiecare tip de program, in functie de contextul în care sunt folosite.

## 5.1 Locking script și unlocking script

Mecanismul de verificare al tranzacțiilor are la bază două tipuri de programe:

- Scriptul de blocare (eng. *locking script*)
- Scriptul de deblocare (eng. *unlocking script*)

Scriptul de blocare este plasat pe un output al unei tranzacții și specifică condițiile care trebuie îndeplinite pentru a putea consuma (eng. *spend*) valoarea trasferată. Din punct de vedere practic, noțiunile de *input*, respectiv de *output* sunt relaționate ca sens. Un input pentru o tranzacție reprezintă de fapt un output provenit de la o altă tranzacție. Istoric vorbind, scriptul de blocare mai este denumit și *scriptPubKey*, deoarece acesta conține o cheie publică sau o adresă Bitcoin.

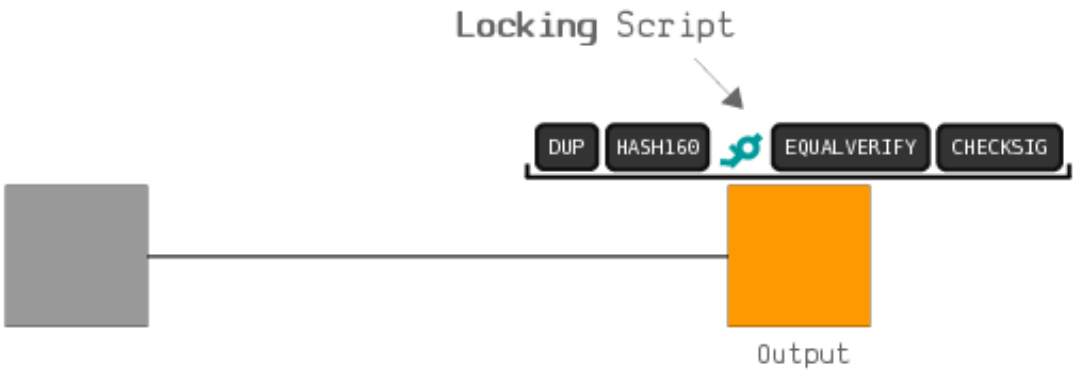



Figura 5.1: Scriptul de blocare[1]

[1] https://learnmeabitcoin.com/glossary/script

Un script de deblocare îndeplinește sau rezolvă condițiile plasate pe un output de către un script de blocare și permite valorii transferate să fie consumată. Scripturile de deblocare sunt denumite și *scriptSig*, deoarece conțin o semnătură digitală. Fiecare client Bitcoin care dorește să valideze tranzacții, trebuie să execute împreună scriptul de blocare, respectiv scriptul de deblocare. Dacă execuția se încheie cu succes, valoarea transferată poate fi comsumată în următoarele tranzacții. Pentru fiecare input al tranzacției, componeta software pentru validare va extrage mai întâi unitățile UTXO (eng. *unspent transaction output*) referențiate de fiecare input în parte. Fiecare unitate UTXO înglobează un script de blocare, care conține toate condițiile necesare pentru a putea intra în posesia valorii transferate. După acest pas, mecanismul de validare va rula împreună scriptul de blocare, respectiv scriptul de deblocare conținut de inputul care dorește să consume unitățile UTXO.

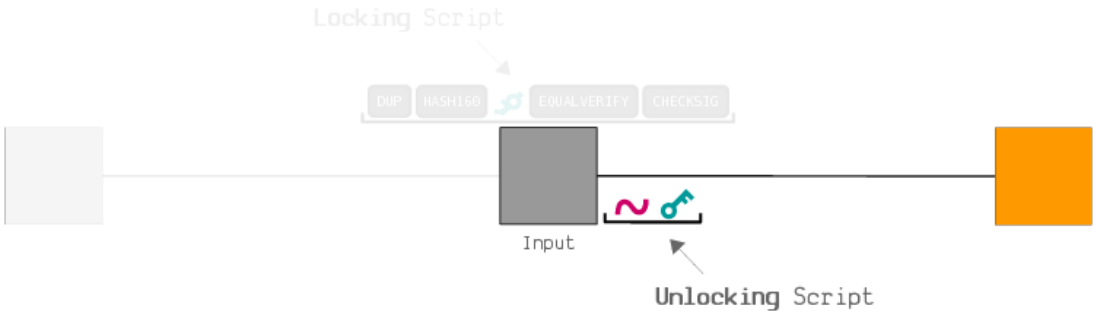



Figura 5.2: Scriptul de deblocare[2]

## 5.2 Ivy

Ca și suport teoretic am utilizat Ivy, un limbaj de nivel înalt care permite scrierea de programe și contracte pentru protocolul Bitcoin. Toate instrucțiunile sunt compilate în Bitcoin Script și au același conținut sintactic și semantic. Pentru a înțelege maniera de funcționare a rețelei Bitcoin, Ivy pune la dispoziția utilizatorilor metode de a simula tranzacții, folosind diverse tipare de contracte. De asemenea, adresele publice, adresele private, semnăturile digitale, precum și alte argumente necesare în redactarea programelor sunt generate automat. Așadar, Ivy oferă posibilitatea de a scrie smart contracte utilizând operații și instrucțiuni suportate de Bitcoin Script. Scopul principal al limbajului Ivy este unul educațional, deoarece se urmărește înțelegerea conceptelor cheie cu privire la rețeua Bitcoin. În acest sens, toate acțiunile utilizatorului sunt coordonate și însoțite de indicații care să faciliteze procesul de învățare. Ca și în rețeaua Bitcoin, există două categorii de acțiuni: crearea de tranzacții, dar și deblocarea tranzacțiilor.

Sintaxa limbajului Ivy conferă flexibilitatea pe care o pune la dispoziție Bitcoin Script, dar mai există și anumite elemente de particularitate, precum: denumirea variabilelor, sintaxa familiară cu celelalte limbaje de programare.

Programele suport puse la dispoziție de limbajul Ivy au fost utilizate pentru formalizarea Bitcoin Script în Coq. Ulterior am identificat și demonstrat proprietățile acestor programe. În cele ce urmează vom prezenta o serie de proprietăți extrase pe baza

[2] `https://learnmeabitcoin.com/glossary/script`

programelor și vom furniza o demonstrație formală, care atestă îndeplinirea acestor proprități într-un anume context. [11]

## 5.3 Modele de tranzacții

Toate aceste programe și scripturi sunt redactate în limbajul Bitcoin Script. Datorită scopului pentru care a fost creat, Bitcoin Script conține o serie restrânsă de operații, întocmai pentru a păstra securitatea acțiunilor la un nivel ridicat. Inexistența structurilor repetitive, indică faptul că programele au o complexitate limitată și și un timp de execuție ușor de prezis. Bitcoin Script nu reprezintă un limbaj de programare uzual, iar limitările la care este supus din punct de vedere al instrucțiunilor, au rolul de a preveni eventualele atacuri asupra sistemului Bitcoin. Din nevoia de a tipiza tranzacțiile, în decursul timpului s-au adoptat o serie de modele standard, care sunt acceptate de majoritatea utilizatorilor din rețeaua Bitcoin. Cele cinci tipuri de programe standard sunt următoarele:

1. P2PKH (Pay-To-Public-Key-Hash)
2. P2PK: (Pay-To-Public-Key)
3. MS: (Multi-Signature)
4. OP_RETURN: (Data-Output)
5. P2SH: (Pay-To-Script-Hash)

**1. Pay-To-Public-Key-Hash**

Majoritatea tranzacțiilor procesate de rețeaua Bitcoin sunt tranzacții P2PKH. Acest tip de tranzacție conține un script de blocare, care atașează secvenței de output un hash al unei chei publice sau, în termeni generali, o adresă Bitcoin. Tranzacțiile care efectuează plăți către o adresă Bitcoin, conțin implicit un script P2PKH. Un output care a fost blocat cu un script P2PKH, poate fi deblocat prin furnizarea unei chei publice, respectiv a unei semnături digitale obținute din cheia privată corespunzătoare.

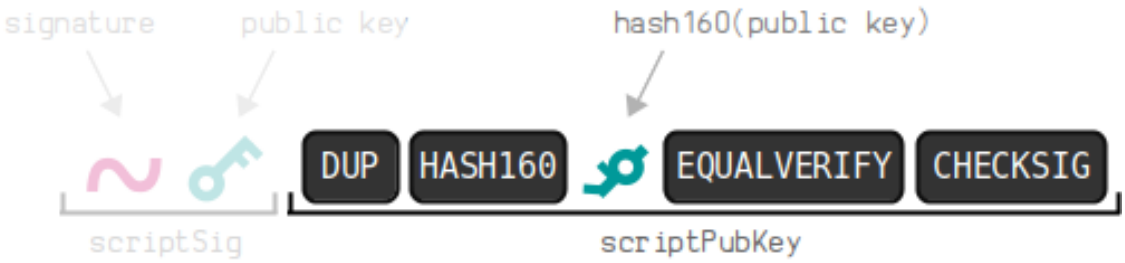



Figura 5.3: Scriptul Pay-To-Public-Key-Hash[3]

[3] https://learnmeabitcoin.com/glossary/p2pkh

Scriptul care va bloca outputul tranzacției va avea următoarea formă:

```
OP_DUP OP_HASH160 <publicKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

Condițiile de deblocare impuse de scriptul anterior pot fi satisfăcute utilizând următorul script de deblocare:

```
<digitalSignature> <publicKey>
```

Pentru a surprinde aceeași funcționalitate și în formalizarea din Coq, punem la dispoziție următoarea implementare:

```
(* Pay-To-Public-Key-Hash *)
(* locking script *)
(* [OP_DUP; OP_SHA256;
    OP_PUSH <publicKeyHash>;
    OP_EQUALVERIFY;
    OP_CHECKSIG] *)
(* unlocking script *)
(* <digitalSignature> <publicKey> *)
Definition P2PKH (hash : Z) := [OP_DUP;
                                OP_SHA256;
                                OP_PUSH hash;
                                OP_EQUALVERIFY;
                                OP_CHECKSIG] .
(* Evaluarea celor două scripturi *)
Compute((check_stack (evaluate_script
                      [<digitalSignature>, <publicKey>]
                      (P2PKH <publicKeyHash>)
                      (length (P2PKH <publicKeyHash>))))).
```

### 2. Pay-To-Public-Key

P2PK este o variantă simplificată a scriptului P2PKH. Pe când P2PKH înglobează un hash al cheii publice, P2PK stochează cheia publică în sine.

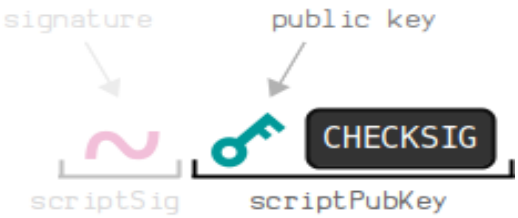



Figura 5.4: Scriptul Pay-To-Public-Key[4]

[4]https://learnmeabitcoin.com/glossary/p2pk

Scriptul care va bloca outputul tranzacției va avea următoarea formă:

```
<publicKey> OP_CHECKSIG
```

Condițiile de deblocare impuse de scriptul anterior pot fi satisfăcute utilizând următorul script de deblocare:

```
<digitalSignature>
```

De asemenea, implementarea în Coq este similară cu sintaxa și semantica reală:

```
(* Pay-To-Public-Key *)
(* locking script - [OP_PUSH <publicKey>; OP_CHECKSIG] *)
(* unlocking script - <digitalSignature> *)
Definition P2PK := [OP_PUSH <publicKey>;
                    OP_CHECKSIG] .
(* Evaluarea celor două scripturi *)
Compute((check_stack (evaluate_script [<digitalSignature>]
                                      P2PK (length P2PK)))).
```

### 3. Multi-Signature

Scripturile *multi-signature* stabilesc un set de condiții care conțin un număr de N chei publice, iar pentru a putea fi deblocat, scriptul de deblocare trebuie să pună la dispoziție cel puțin M semnături digitale, din cele N disponibile. Constanta N reprezintă numărul total de chei publice, iar M reprezintă numărul minim de semnături digitale necesare deblocării.

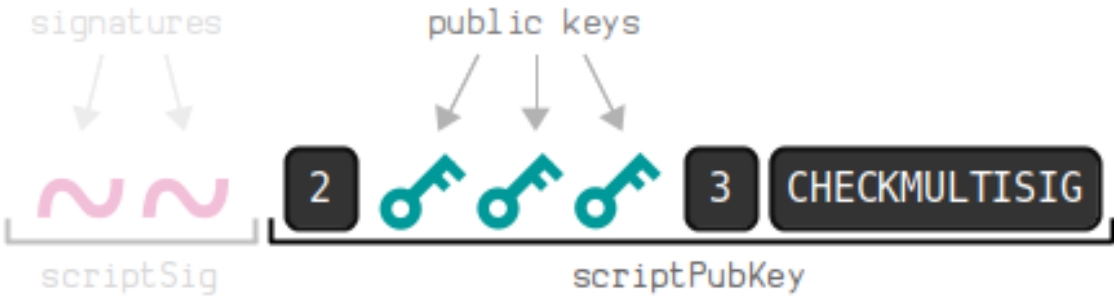



Figura 5.5: Scriptul Multi-Signature[5]

Forma generală a unui script de tip multi-signature este:

```
M <publicKey 1> <publicKey 2> ... <publicKey N> N OP_CHECKMULTISIG
```

[5] https://learnmeabitcoin.com/glossary/p2ms

Forma scriptului de deblocare este:

```
OP_0 <digSig 1> <digSig 2> ... <digSig M>
```

Implementarea în Coq păstrează tiparul menționat mai sus:

```
(* Multi-signature *)
(* locking script *)
(* [OP_PUSH <M>;
    OP_PUSH <publicKeys>; OP_PUSH <N>;
    OP_CHECKMULTISIG] *)
(* unlocking script - <digitalSignatures> *)
(* Evaluarea celor două scripturi *)
Definition MS := [OP_PUSH <M>;
                  OP_PUSH <publicKey 1>; ... OP_PUSH <publicKey N>;
                  OP_PUSH <N>;
                  OP_CHECKMULTISIG] .
Compute(evaluate_script [<digSig 1> ... <digSig M>]
                        MS (length MS)).
```

Pentru a păstra funcționalitatea de bază a programului Multi-Signature, am considerat o serie de metode auxiliare:

```
Fixpoint multiSignature (s:stack) (l:stack) (m : nat): bool :=
  match m with
  | 0 => true
  | S m' => match s with
            | nil => false
            | x :: s' => match l with
                         | nil => false
                         | x' :: l' => if (Z.eqb x x')
                                       then (multiSignature s' l' m')
                                       else false
                         end
             end
  end.


Fixpoint checkElement (s:stack) (x:Z) : bool :=
  match s with
  | nil => false
  | x' :: s' => if (Z.eqb x x')
                then true
                else (checkElement s' x)
  end.

Fixpoint multiSignatureScript (s:stack) (l:stack): bool :=
  match l with
  | nil => true
  | x :: l' => match (checkElement s x) with
               | true => (multiSignatureScript s l')
               | false => false
               end
  end.
```

### 4. OP_RETURN

Utilizarea rețelei blockchain pentru a stoca date care nu au legătură cu plățile a devenit un subiect controversat de-a lungul timpului. Tranzacțiile de acest tip creează unități UTXO care nu pot fi consumate niciodată în cadrul tranzacțiilor viitoare. Deoarece sunt considerate „plăți false", efectul pe care îl au asupra bazei de date UTXO este nefavorabil, conducând la creșterea infinită a dimensiunii acesteia. OP_RETURN se încadrează în această categorie, cu mențiunea că, odată ce este utilizat în cadrul unui script, acesta creează automat un output care nu este consumabil (eng. *provably unspendable output*). Cu toate că nu este inclus în setul de unități UTXO, outputurile de forma OP_RETURN sunt stocate în blockchain, ceea ce determină creșterea în dimensiune a bazei de date blockchain.


Figura 5.6: Scriptul Null Data[6]

Scriptul OP_RETURN are următoarea formă:

```
OP_RETURN <data>
```

Pentru scriptul menționat mai sus nu există asociat niciun script de deblocare. Acesta produce un output nul, deoarece orice sumă în Bitcoin menționată în cadrul scriptului devine irecuperabilă.

### 5. PAY-TO-SCRIPT-HASH

Scriptul standard PAY-TO-SCRIPT-HASH a fost conceput pentru a rezolva anumite probleme practice, în cazul în care un utilizator dorește să folosească un script de blocare mai complex. În cadrul plăților PAY-TO-SCRIPT-HASH, scriptul de blocare este înlocuit cu valoarea lui encriptată. Când o tranzacție încearcă să consume anumite unități UTXO, trebuie să conțină pe lângă scriptul de deblocare și scriptul de blocare original care să se potrivească cu valoarea sa ecriptată furnizată anterior. Scriptul care va înlocui scriptul de blocare inițial se numește script de răscumpărare (eng. redeem script). Acesta va fi prezent și la finalul scriptului de deblocare, pentru a putea verifica că într-adevăr valoarea encriptată furnizată în scriptul de blocare este cea reală. Execuția scriptului de tipul PAY-TO-SCRIPT-HASH se face în două etape. Mai întâi se va aplica funcția hash asupra scriptului de răscumpărare și se va verifica dacă este egală cu valoarea hash furnizată în scriptul de blocare. Dacă prima etapă se finalizează

[6] https://learnmeabitcoin.com/glossary/nulldata

cu succes, atunci scriptul de răscumpărare este deserializat și executat împreună cu scriptul de deblocare, pentru a vedea dacă sunt satisfăcute condițiile impuse inițial.


Figura 5.7: Scriptul Pay-To-Script-Hash[7]

Exemplu de tranzacție fără PAY-TO-SCRIPT-HASH

```
(* Script de blocare *)
2 <publicKey 1> <publicKey 2> <publicKey 3> 3 OP_CHECKMULTISIG

(* Script de deblocare *)
<digitalSignature 1> <digitalSignature 2>
```

Exemplu de tranzacție cu PAY-TO-SCRIPT-HASH

```
(* Script de răscumpărare *)
2 <publicKey 1> <publicKey 2> <publicKey 3> 3 OP_CHECKMULTISIG

(* Script de blocare *)
OP_HASH160 <redeemScriptHash> OP_EQUAL

(* Script de deblocare *)
<digitalSignature 1> <digitalSignature 2> <redeemScript>
```

Principalele proprietăți asupra cărora ne-am centrat atenția sunt:

- Existența unei valori nenule în vârful stivei, în cazul în care execuția programului s-a terminat cu succes.
- Executarea liniară a instrucțiunilor și furnizarea argumentelor în ordinea stabilită.

Pentru a ilustra maniera de verificare a scripturilor și programelor în conformitate cu proprietățile menționate mai sus, punem la dispoziției următoarele demonstrații:

[7] https://learnmeabitcoin.com/glossary/p2sh

```
Theorem crypto_operations : forall s:stack, compute_SHA1 s = s .
Proof.
  intros s.
  destruct s; trivial.
  simpl.
  induction s; simpl; trivial.
Qed.

Theorem exec : forall s, forall x, Z.eqb (top x s) 1%Z = true
                                        -> check_stack s = true.
Proof.
  intros.
  unfold check_stack.
  unfold top in H.
  destruct s. trivial.
  rewrite H. trivial.
Qed.

Lemma LWPKH_Proof : forall s, forall x, forall y,
                    (evaluate_script [y] (LWPKH y)
                     (length (LWPKH y))) = s
                    -> top x s = 1%Z.
Proof.
  intros.
  simpl in H.
  unfold verify_eq in H.
  case_eq (Z.eqb y y); intros; rewrite H0 in H; simpl in *.
  - unfold top. subst; trivial.
  - rewrite Z.eqb_refl in H0. inversion H0.
Qed.

Lemma LU_Proof : forall x, forall y, forall t1, forall t2,
                 (Z.eqb (top x (evaluate_script [y] (LU y t1 t2)
                 (length (LU y t1 t2)))) 1%Z) = true
                 -> (Z.leb t2 t1) = true .
Proof.
  intros.
  simpl in H.
  case_eq (Z.leb t2 t1); intros H'; rewrite H' in *; trivial.
Qed.
```

Fiecare din demonstrațiile menționate mai sus are o anumită particularitate. Generalitatea cu care este tratat fiecare caz de execuție în parte certifică faptul că orice input furnizăm unui program, acesta trebuie să îndeplinească anumite proprietăți pentru ca execuția programului să se încheie cu succes.

# Capitolul 6

# Concluzie

În concluzie, formalizarea limbajului Bitcoin Script s-a realizat cu succes. Fiecare program a fost redat în noua sintaxă descrisă în Coq. Funcționalitățile de bază ale limbajului au fost respectate și aplicate în momentul evaluării programelor. Ca direcție de viitor, ne propunem implementarea operațiilor criptografice, care la nivelul actual sunt considerate funcții identitate. Pe de altă parte, la fel cum există deja în Ivy, avem în vedere realizarea contextului formal care face posibilă compilarea în Bitcoin Script a programelor descrise în Coq.

# Bibliografie


[1] https://www.theglobaltreasurer.com/2018/08/08/the-business-benefits-of-cryptocurrency/

[2] https://www.investopedia.com/terms/b/blockchain.asp

[3] https://blockgeeks.com/guides/what-is-blockchain-technology/

[4] https://blockgeeks.com/guides/blockchain-applications/

[5] https://cointelegraph.com/explained/proof-of-work-explained

[6] https://blockgeeks.com/guides/ethereum/

[7] https://www.cardano.org/en/what-is-cardano/

[8] https://learnmeabitcoin.com/glossary/script

[9] https://en.bitcoin.it/wiki/Script

[10] https://coq.inria.fr/

[11] https://docs.ivy-lang.org/bitcoin/

[12] https://blockstream.com/simplicity.pdf

[13] https://bitcoin.org/bitcoin.pdf