Systems and Methods for Implementing Deterministic Finite Automata (DFA) via a Blockchain

Craig S. Wright

Abstract We present a novel technology for the establishment and (discretionary) automatic execution of (financial) contracts based on the realisation of the commitments of the different parties, and other clauses and provisions, as a non-deterministic finite automaton (NFA) embodied in a computational and record-keeping structure on the (Bitcoin) blockchain. In particular, the process provides methods for constructing non-deterministic finite-state automata in Bitcoin script. The "best" method produces a one-to-one relation between the definition and the state table of the automaton.

Keywords Automata · DFA · Computation · Bitcoin · Blockchain

1 Introduction

The automation of (financial) contracts has been a topic of continued academic research (see, e.g., [1] and references therein) and practical interest since the realisation that an electronic version of the essence of such contracts can be better defined (e.g. avoiding ambiguities and interpretations of current legalese as well as potentially costly and long litigations) and executable and enforceable by computers, and consequently cheaper and more reliable.

Among the different approaches which have been proposed in the literature (see Chap. 1 of [1] for a short review), it has been shown that a deterministic finite automata (DFA), also known as deterministic finite-state machines, have a rich enough structure to represent a wide range (if not all) of imaginable financial agreements, and other kinds of contracts [2, 3].

A DFA is a mathematical model of computation conceived as an abstract machine that can be in one of a finite set of states and can change from one state to another (transition) when a triggering event or condition occurs. Its computational capabilities are more than those of combinational logic but less than those of a stack machine.

C. S. Wright (✉)
CNAM, Paris, France
e-mail: c.wright@nchain.com

Going beyond existing literature, we develop a technological innovation, which facilitates the incarnation of a DFA in a practical way on the existing infrastructure of the (Bitcoin) blockchain. The states of the machine are defined and recorded on the permanent ledger which is the blockchain, and blockchain transactions work as agents changing the machine from one state to another. When the DFA defines a contract, the innovation provides with a mechanism for the automatic execution and enforcement of the commitments of the different parties, and other clauses and provisions.

Key Elements. The blockchain-based DFA has the following key features: legalise and litigation-free by construction; executable and enforceable automatically by computers; and permanent record of contracts, their execution, and outcomes.

And it offers the following benefits inherent to the Bitcoin blockchain: inherently secure by design (the Bitcoin protocol requires no trusted parties); distributed (so avoids a large single point of failure and is not vulnerable to attack); easy to manage and maintain (the Bitcoin network is straightforward to use); inexpensive (just a small transaction fee is usually expected under the Bitcoin protocol); global and can be used at any time by anyone with access to the Internet; transparent—once data has been written to the blockchain, anyone can see it; immutable—once data has been written to the blockchain, no one can change it; privacy is maintained, as no personally identifying information is involved.

Section Organisation. The contribution is broken up into two main sections: a functional specification, offering a high-level outline, and explanation of the matter and nature of the proposed solution; and a technical specification, outlining the technical possibilities and novelties involving the innovation, ending with an example.

2 Functional Specification

At a high level of abstraction, the system we are proposing consists of the DFA itself and a closely related system of agents (Botnet) which write the transactions and submit them to the blockchain. We will concentrate here on the description of the DFA mechanism, and possible realisations of the transactions, leaving the specification of the system of agents for parallel companion work [4].

An illustration of the complete system can be found in Fig. 1. There the Botnet[1] interacts with the world (humans or other computers) to receive instructions, e.g. which contract to create and execute, as indicated by the connection lines. The specification of the contract itself can be provided in any format, e.g. xBRL [5], and stored in a secure and decentralised manner, for example, in a distributed hash table (DHT) on the torrent network. From the specification of the contract, a Botnet agent

[1]We will refer generally to the Botnet, often without specifying, which agent actually carry out the actions described, this could be one of the lower-level bots, a bot manager (Botman) or any other appropriate entity as specified in [4].

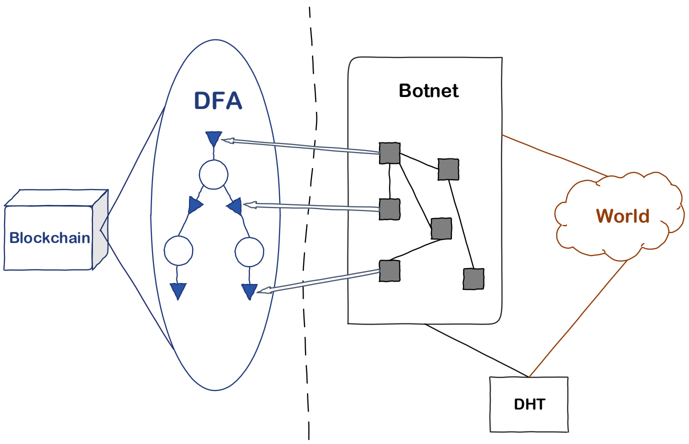

Fig. 1 Diagram describing the blockchain-based DFA and Botnet systems

constructs the DFA, which is subsequently incarnated on the blockchain by Botnet agents.

The DFA itself is specified as a finite set {S, I, t, s0, F}, where S stands for the (finite) set of possible states in which the contract/DFA can be; I is a (finite) set of inputs (also known as the alphabet), which in our context means any event or condition which can occur in relation to the contract, e.g. a payment is made, the maturity of the instrument is reached, a counterparty defaults, etc., in our mechanism, these input signals are received/produced by Botnet agents, which then determine the next state of the system (possibly the same one).

The third component of a DFA is a transition function t: $S \times I \rightarrow S$. Deterministic refers to the uniqueness of the decision: given a state and an input there is only one new state (possibly the same one); thus, given an initial state (s0) and a history of inputs the outcome of the calculation (contract) is unique, one among the set of all possible final outcomes ($F \subseteq S$). Once all these elements have been established, the DFA is completely defined by a transition table, specifying the future states for all possible current states and input signals. The states of the DFA are themselves associated with unspent transaction outputs (UTXO) on the blockchain. Note that the Bitcoin network continuously tracks all available UTXO. The mechanism by which the DFA moves from one state to another is incarnated in our proposal by Bitcoin transactions; effectively they spend the UTXO associated with one state (an input of the transaction) and create the UTXO associated with the next state (an output). This constitutes a key inventive element of our proposal.

To illustrate these ideas, we are going to consider a discount (zero-coupon) bond, which is a simple debt instrument usually bought at a price (normally at a discount with respect to its face value), then held for some time, until its principal is returned at maturity. The possible states we will consider are S = {s0, f0, f1}, indicating, respectively, the holding state (s0), the normal conclusion of the contract (if it follows the happy path) or happy ending (f0), and a state (f1) in which things go wrong, e.g.

Table 1 Transition matrix for a DFA representing a zero-coupon bond

t	r	d	e
s_0	f_0	f_1	f_1
f_0	–	–	–
f_1	–	–	–

litigation; the final states of the system are thus F = {f0, f1}. The alphabet we will consider is I = {r, d, e}, indicating, respectively, repayment of the principal at (or before) expiration (r), default of the issuer at (or before) expiration (d), and expiration of the contract without repayment (e). The transition matrix for this simple contract is presented in Table 1. Note that for the final states represent the completion of the contract, thus no further states need to be specified from them (currently noted as '–' in the transition table, although those lines could be omitted).

Figure 2 represents the embodiment of the zero-coupon bond DFA on the (Bitcoin) blockchain. The states are represented by circles and the Bitcoin transactions which move the machine from one state to the other by the blue triangles. Note that the inputs received by the Botnet agents are omitted in this diagram, however, in each state one or other transition should occur according to these inputs, which is reflected in the diagram by the construction of one or other Bitcoin transaction (e.g. t0 or t1 in state s0); no transactions are required for transitions which do not change the state, thus they have been omitted. In addition to the transitional transactions of the DFA (ti), an initial origination transaction (o) and transactions corresponding to the completion of the contract (ci) are considered.

A last point to discuss before turning to the technical specification of the invention is the flow of funds in the transactions (originations, transitions and completions). An important observation is that because of the finite nature of the DFA, and of (financial) contracts, they would be completed after a number of transitions. This

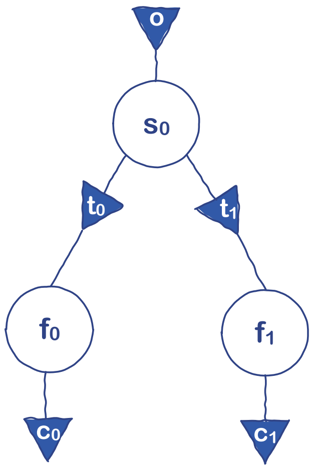

Fig. 2 Diagram describing a blockchain-based DFA

necessarily implies (assuming some finite fees for the Botnet agents involved and the Bitcoin miners) that the maximum costs of the establishment and execution of the contract is bound and can be determined in advance, e.g. at the point of establishment of the DFA. It is given by the total amount of funds required to execute the contract following the longest imaginable path. This, of course, excludes the possibility of infinite loops in the execution, note, however, that this is not relevant for current (financial) contracts; even contracts such as perpetuities are bound to be completed at some point in future, despite their name.

The particular distribution of the fees, how much each agent receives for their work, although of obvious practical significance is not fundamental to the invention described here. Continuing with our example of a streamlined zero-coupon bond, we will arbitrarily assume a fee of 3 mBitcoin for the origination transaction (o), a transition fee (ti transaction) of 1 mBitcoin and a 2 mBitcoin fee for the completion transaction (ci); note that these fees are automatically included in the transactions themselves. Together with 3 mBitcoin of total mining fees for the 3 transactions, this results in a total maximum cost of 9 mBitcoin. In our example, the length of all paths is equal, thus the final cost of the contract will certainly coincide with its maximum cost. Since this need not be the case in general, in order to completely illustrate a general flow of funds, we will assume that the funds provided for the execution of the contract are 10 mBitcoin, and that 1 mBitcoin is returned to the same source of funds after completion (this takes the place of eventual unused funds at completion, i.e. if maximum and used funds were to differ).

We will assume that the funds for the establishment and execution of the contract (10 mBitcoin) are initially provided by some funding source referred to as the originator (as mentioned, in our example, this source will also receive 1 mBitcoin unused funds after completion). In principle one could also include additional inputs and outputs of funds in the transactions, e.g. the price paid for the zero-coupon bond, the repayment of the principal, or any other imaginable transfer of funds. Although this may be of practical interest, at this point it would only help to obscure the essential elements of blockchain-based DFA processing. For the sake of clarity, we have decided not to include such details in the examples below; note, however, that the structure is completely general and such possibilities are not excluded.

Once the functional specification of the technology has been established, the remaining components of the system ought to be specified at a technical level, in particular, the inner working and the flow of information and funds of the Bitcoin transactions which constitute a fundamental innovation of our proposal.

3 Technical Specification

There are several options here, depending to a large extent on the particular configuration of the Botnet system [4], e.g. whether the particular agents involved are known in advance or not. We will explicitly consider two options here. If participation on the contract is kept fairly open, so that a variety of agents can participate, a structure based on the standard transaction of the type transaction puzzle [6] is feasible. Conversely, if the transactions for each state can be assigned in advance to a particular agent (or group of agents), a pay-to-script-hash (P2SH) [7] transaction can be used. Of course, one can imagine a number of possibilities in between, e.g. a group of agents which change in time, a hierarchy of agents, private information (keys) could be securely interchanged just before each transaction is written, etc. The possibilities are many and, as mentioned, depend to a large extent on the particular configuration of the Botnet to be discussed elsewhere [4]; we will only specify explicitly the two options mentioned.

Note, however, that which particular type of transaction is used, as specifically who provides/receives the funds in the transactions (the issuer, the purchaser, the payee, etc.), is ultimately not fundamental to the invention described in this paper. The essence of the proposal is that at any point in time the state of the contract is well defined within the blockchain, and that we describe a mechanism (whatever its detailed concrete realisation) which generates the contract, executes it on the blockchain, and enforces the appropriate outcome according to the sequence of events that occurs. Note that, because the whole mechanism is embodied on the blockchain, it automatically provides a permanent immutable record of the history and outcome of the contract, among many other advantages.

3.1 Transaction-Puzzle-Based

One of the configurations of the Botnet contemplated in [4] regards the system as an open network of computers in which anyone with some Internet-connected processing power may join, provide some processing power to the system (e.g. a provider of the establishment and execution of the blockchain-based DFA contracts) and get rewarded for their resources (see [4] for details). Thus, we are compelled to assume that it is impossible to know in advance which particular agent will submit the transaction to the blockchain, i.e. it is not possible to use any specific information on the agent (e.g. its public keys); however, a transaction-puzzle type is still feasible. The general locking/unlocking mechanism of this type of transactions is [6]:

$$\textbf{Locking Script} : \text{OP\_HASH256}\ \ \text{<state}\, s_i\ \text{puzzle} > \text{OP\_EQUAL}$$

$$\textbf{Unlocking Script} :\ \ \text{<puzzle}\, s_i\ \text{solution} >$$

where <state s_i puzzle> = HASH(<state s_i puzzle solution>) and the puzzle solution itself can include any desired information, including a code for the contract, the label of the state, and any other desired information, for example, an extra bit of *salt* (for added security [8, 9]):

$$\text{<state}\, s_i\ \text{solution} > = \text{HASH(<contract code;}\ \ \text{state}\, s_i;\ \ \text{other data;}\ \ \text{salt >)}$$

The sequence of actions is as follows. First, the Botnet system (the Botman or another agent, as specified in [4]) creates the DFA structure, stores the transition table externally (e.g. in a DHT), create the puzzles for each possible state of the DFA and distributes them securely to the agents which are allowed to participate on the execution of the contract. There are a number of possibilities for this flow of information but, as before, since it is not fundamental to the process, we will omit such details here.

Next, a Botnet agent (the same or a different one [4]), creates the origination transaction as specified in Table 2. At this stage, the contract is incarnated as a structure on the blockchain and is in its first state s_0, i.e. there is an UTXO associated with the state s_0 of the particular contract on the blockchain.

As can be deduced by studying the transaction, the original funding for the contract is received (from the originator in the example) by a P2PKH-type transaction, o*utput 0* (puzzle-based) can be unlocked by any Botnet agent in possession of the puzzle solution, and *output 1* (P2PKH) pays the required fee to the Botnet agent which has succeeded in placing the transaction on the blockchain.

Successive transitions *on the execution of the contract* are carried out by Botnet agents in a similar fashion as exemplified in Table 3. They need to get the puzzle

Table 2 Example of an origination (to state s0) transaction (o). The funds allocated to the contract have been assumed to be 10 mBitcoin, the Botnet fee 3 mBitcoin, and the outgoing funds for further processing of the contract 6 mBitcoin (1 mBitcoin mining fee is implicit)

Transaction identifier			`origination` *o*
Version number			`<version number>`
Number of inputs			`1`
Input	Previous transaction	Hash	`<10 mBitcoin from originator>`
		Output index	`00`
	Length of signature script		`<unlocking script length>`
	Signature script		`<originator signature>` `<originator public key>`
	Sequence number		`<sequence number>`
Number of outputs			`2`
Output 0	Value		`600000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_HASH256 <state` s_0 `puzzle>` `OP_EQUAL`
Output 1	Value		`300000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_DUP OP_HASH160 <Botnet agent public key hash>` `OP_EQUALVERIFY OP_CHECKSIG`
Locktime			`0`

Table 3 Example of a transition (from state $s0$ to state ff) transaction (tf).. The incoming funds have been assumed to be 6 mBitcoin, the Botnet fee 1 mBitcoin, and the outgoing funds for further processing of the contract 4 mBitcoin (1 mBitcoin mining fee is implicit)

Transaction identifier		`transition` t_f	
Version number		`<version number>`	
Number of inputs		`1`	
Input	Previous transaction	`Hash`	`<6 mBitcoin from origination` o`>`
		`Output index`	`00`
	Length of signature script	`<unlocking script length>`	
	Signature script	`<state` s_0 `puzzle solution>`	
	Sequence number	`<sequence number>`	
Number of outputs		`2`	
Output 0	Value	`400000`	
	Length of public key script	`<locking script length>`	
	Public key script	`OP_HASH256 <state` f_f `puzzle> OP_EQUAL`	
Output 1	Value	`100000`	
	Length of public key script	`<locking script length>`	
	Public key script	`OP_DUP OP_HASH160 <Botnet agent public key hash> OP_EQUALVERIFY OP_CHECKSIG`	
Locktime		`0`	

solution corresponding to the current state (s_0), interact with the world (or some other computer on the Botnet) in order to receive the appropriate input, read the transition table (or just the part of them corresponding to the current state) and get the puzzle corresponding to the appropriate next state (f_f). They can then submit the transaction to the blockchain, if they succeed in placing it, they will get their fee and the DFA will be in the state f_f.

In order to conclude the technical description of the mechanism, we need to define the structure of the last kind of possible transactions, those completing the execution of the contract (c_f). This is shown in Table 4, where the input part follows the same transaction puzzle logic as before, while *output 0* pays the *unused funds* back to the originator (1 mBitcoin, as discussed above), and *output* 1 pays the fee to the Botnet agent (Table 5).

Table 4 Example of a completion (from state ff.) transaction (cf). The incoming funds have been assumed to be 4 mBitcoin, 1 mBitcoin unused funds are returned to the originator, and the Botnet fee is 2 mBitcoin (a 1 mBitcoin mining fee is implicit)

Transaction identifier			completion c_f
Version number			<version number>
Number of inputs			1
Input	Previous transaction	Hash	<4 mBitcoin from transition t_f>
		Output index	00
	Length of signature script		<unlocking script length>
	Signature script		<state f_f puzzle solution>
	Sequence number		<sequence number>
Number of outputs			2
Output 0	Value		100000
	Length of public key script		<locking script length>
	Public key script		OP_DUP OP_HASH160 <originator public key hash> OP_EQUALVERIFY OP_CHECKSIG
Output 1	Value		200000
	Length of public key script		<locking script length>
	Public key script		OP_DUP OP_HASH160 <Botnet agent public key hash> OP_EQUALVERIFY OP_CHECKSIG
Locktime			0

3.2 P2SH Based

Instead of being open to a large number of a priori unknown participants, another possible configuration of the Botnet could be that of a limited number of acknowledged computers. In this case, transactions of the type P2SH [7] might be more appropriate since they can be configured to include the public keys of the agents, thereby providing an extra layer of security. In the case of a single acknowledged agent, a feasible locking/unlocking mechanism for the transactions is:

Locking Script: OP_HASH160 <state s_i redeem script hash> OP_EQUAL

Unlocking Script: OP_0 <agent signature> <state s_i redeem script>

Redeem Script: OP_1 <state s_i metadata> <agent public key> OP_2 OP_CHECKMULTISIG

Note that this can be trivially extended to a larger number of acknowledged agents by including their signature as well. As before, the metadata of state a state s_i can include any desired information, for example:

<state s_i metadata> = HASH (<contract code; state s_i; other data>)

Table 5 Example analogous to that in Table 2 but using the P2SH transaction type for the DFA states

Transaction identifier			`origination` o
Version number			`<version number>`
Number of inputs			`1`
Input	Previous transaction	Hash	`<10 mBitcoin from originator>`
		Output index	`00`
	Length of signature script		`<unlocking script length>`
	Signature script		`<originator signature> <originator public key>`
	Sequence number		`<sequence number>`
Number of outputs			`2`
Output 0	Value		`600000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_HASH160 <state` s_0 `redeem script hash> OP_EQUAL`
Output 1	Value		`300000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_DUP OP_HASH160 <Botnet agent public key hash> OP_EQUALVERIFY OP_CHECKSIG`
Locktime			`0`

The sequence of actions is similar as in the previous case. First, the Botnet system creates the DFA structure, stores the transition table externally (e.g. in a DHT), determines which agents will process the transactions and retrieve/generate their public keys, which are then included in the redeem scripts for each possible state of the DFA, note that these scripts can be stored externally and need not be transmitted securely. Next, a Botnet agent creates the origination transaction as specified in Table 2. At this stage the contract is incarnated as a structure on the blockchain and is in its first state s_0.

The only change in the transaction with respect to that in Table 2 is *output* 0, which now is of type P2SH and includes the public keys of the acknowledged agents as discussed above. Although the changes are completely analogous, for completeness we present in Tables 6 and 7 examples of transition transactions and completion transactions based on the P2SH transaction type. The flow of funds is the same as before.

Table 6 Example analogous to that in Table 3 but using the P2SH transaction type for the DFA states

Transaction identifier			`transition` t_f
Version number			`<version number>`
Number of inputs			`1`
Input	Previous transaction	Hash	`<6 mBitcoin from origination` o`>`
		Output index	`00`
	Length of signature script		`<unlocking script length>`
	Signature script		`OP_0 <Botnet agent signature> <state` s_0 `redeem script>`
	Sequence number		`<sequence number>`
Number of outputs			`2`
Output 0	Value		`400000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_HASH160 <state` f_f `redeem script hash> OP_EQUAL`
Output 1	Value		`100000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_DUP OP_HASH160 <Botnet agent public key hash> OP_EQUALVERIFY OP_CHECKSIG`
Locktime			`0`

3.3 Example

Imagine Alice has some savings and wants to invest them in a discount bond from a certain bond issuer. She can contact a service provider of blockchain-based DFA contracts, provides the appropriate (discounted) price funds (possibly through another Bitcoin transaction which can be integrated in the DFA as well), and have it create and automatically execute the contract. If everything goes well (happy path) she will automatically receive the face value of the bond back at maturity. If, for example, the bond issuer happens to default while Alice holds the bond, the Botnet system will automatically take the appropriate action as defined in the contract.

4 Summary and Conclusion

The invention relates to a technique for implementing, controlling and automating a task or process on a blockchain such as, but not limited to, the Bitcoin blockchain. The invention is particularly suited for, but not limited to, automated execution of

Table 7 Example analogous to that in Table 4 but using the P2SH transaction type for the DFA states

Transaction identifier			`completion` c_f
Version number			`<version number>`
Number of inputs			`1`
Input	Previous transaction	Hash	`<4 mBitcoin from transition` t_f`>`
		Output index	`00`
	Length of signature script		`<unlocking script length>`
	Signature script		`OP_0 <Botnet agent signature> <state` f_f `redeem script>`
	Sequence number		`<sequence number>`
Number of outputs			`2`
Output 0	Value		`100000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_DUP OP_HASH160 <originator public key hash> OP_EQUALVERIFY OP_CHECKSIG`
Output 1	Value		`200000`
	Length of public key script		`<locking script length>`
	Public key script		`OP_DUP OP_HASH160 <Botnet agent public key hash> OP_EQUALVERIFY OP_CHECKSIG`
Locktime			`0`

contracts such as smart contracts for financial agreements. However, other types of tasks and non-financial contracts can be implemented. The invention can be viewed as the implementation or incarnation of a state machine or DFA on a blockchain by using the unspent outputs of blockchain transactions to represents the states of the machine, and spending of those outputs as the transition of the machine from one state to another. The invention provides a technical realisation and implementation of a mathematical model of computation conceived as an abstract machine that can be in one of a finite set of states and can change from one state to another (transition) when a triggering event of a finite set (called input) occurs. The invention comprises compilation and codification techniques for the DFA implementation.

As detailed, this method is a means to implementing a DFA on a blockchain, comprising the listed steps. It is a method of implementing a DFA on a blockchain, comprising the steps of associating a portion of data in the locking script of an unspent output ($UTXO_1$) of a blockchain transaction (Tx_1) with a given state of the DFA.

A method further comprising the step of using a further transaction (Tx_2) to make a transition from the state of the DFA to a further state by spending the output ($UTXO_1$) of the transaction (Tx_1); wherein the further state is associated with a portion of data provided within a locking script of an unspent output ($UTXO_2$) of the further transaction. The process is completed using a portion of code to implement or represent at least one state transition trigger which, when executed, causes a further transaction (Tx_2) to spend the output ($UTXO_1$) of the transaction (Tx_1) and thus move the DFA to another state.

The process is a method wherein the portion of code comprises a machine-testable condition which provides a Boolean result based upon an input. In this, the input is determined at run time and is used by the portion of code to determine whether or not the unspent output ($UTXO_1$) should be spent so as to move the DFA to the further state such that the portion of data in the unspent output ($UTXO_1$) is provided in a locking script and data is a tag, label or a portion of metadata. In the model, the DFA is a model of a machine-executable smart contract.

This is conducted successfully as the unspent output ($UTXO_1$) comprises a locking script which includes a hash of a puzzle, the solution of which must be provided by an input of a further transaction in order to spend the output ($UTXO_1$) and transition the DFA to another state.

In the system, the unspent output ($UTXO_1$) comprises a locking script which includes a hash of a redeem script which must be provided by an input of a further transaction in order to spend the output ($UTXO_1$) and transition the DFA to another state. Further, the redeem script comprises a cryptographic key. From this, we have a step of further comprising the step of using one or more computing agents to perform the step of any preceding state.

The DFA is constructed on a blockchain, the method comprising the steps of executing a program which is arranged to monitor for and/or receive an input signal and, responsive to the input signal, generate a blockchain transaction Tx_2 which comprises an unspent output (UTXO) and spends an output of a previous transaction Tx_1; wherein the output of previous transaction Tx_1 comprises a locking script which includes an identifier associated with a first state of the DFA, and the unspent output (UTXO) of transaction Tx_2 comprises a locking script which includes a further identifier associated with a further state of the DFA.

A method allows for the implementing a DFA on a blockchain, comprising the steps using at least one input signal to execute at least one condition and, based on the outcome of the execution of the condition, perform an action in accordance with a state transition table for the DFA, wherein performance of the action is identifiable from the state of a blockchain ledger.

This system comprises of a blockchain platform and at least one computing agent arranged to implement the DFA via the blockchain.

References

1. T. Hvitved, *Contract Formalisation and Modular Implementation of Domain-Specific Languages*. Faculty of Science, University of Copenhagen, Ph.D. Thesis (2012)
2. C. Molina-Jimenez et al., Run-time monitoring and enforcement of electronic contracts. Electron. Commer. Res. Appl. **3**, 108 (2004)

3. M.D. Flood, O.R. Goodenough, *Contract as Automaton: The Computational Representation of Financial Agreements*. OFR Working Paper 1504 (2015)
4. Botman, *nCrypt: Umbrella Document*, WP0238 (2016)

The following resources provide background material relating to the technological background of the present process

5. XBRL Homepage. https://www.xbrl.org/. Last accessed 26 Jan 2019
6. On Bitcoin script. Bitcoin Wiki Homepage. https://en.bitcoin.it/wiki/Script. Last accessed 26 Jan 2019
7. On BIPs. Github Homepage. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki. Last accessed 26 Jan 2019
8. On "salt." Aspheute Homepage. http://www.aspheute.com/english/20040105.asp. Last accessed 26 Jan 2019
9. On "salt." Jasypt Homepage. http://www.jasypt.org/howtoencryptuserpasswords.html. Last accessed 26 Jan 2019