A formal model for ledger management systems based on contracts and temporal logic

Paolo Bottoni, Anna Labella

Department of Computer Science - Sapienza University of Rome - Italy

Remo Pareschi

Stake Lab - University of Molise - Italy

Abstract

A key component of blockchain technology is the ledger, viz., a database that, unlike standard databases, keeps in memory the complete history of past transactions as in a notarial archive for the benefit of any future test. In second-generation blockchains such as Ethereum the ledger is coupled with smart contracts, which enable the automation of transactions associated with agreements between the parties of a financial or commercial nature. The coupling of smart contracts and ledgers provides the technological background for very innovative application areas, such as Decentralized Autonomous Organizations (DAOs), Initial Coin Offerings (ICOs) and Decentralized Finance (DeFi), which propelled blockchains beyond cryptocurrencies that were the only focus of first generation blockchains such as the Bitcoin. However, the currently used implementation of smart contracts as arbitrary programming constructs has made them susceptible to dangerous bugs that can be exploited maliciously and has moved their semantics away from that of legal contracts. We propose here to recompose the split and recover the reliability of databases by formalizing a notion of contract modelled as a finite-state automaton with well-defined computational characteristics derived from an encoding in terms of allocations of resources to actors, as an alternative to the approach based on programming. To complete the work, we use temporal logic as the basis for an abstract query language that is effectively suited to the historical nature of the information kept in the ledger.

Preprint submitted to Blockchain: Research and Application *October 1, 2021*

Keywords: Ledger, Contract, Database, Transaction, Automata theory, Temporal logic.

1. Introduction

Blockchains and distributed ledgers are nowadays among the information technologies having the greatest impact. Initial success has come to public blockchains, starting with the mother of all of them, that of Bitcoin, which has cleared the large-scale practicability of cryptocurrencies, i.e., currencies freed from central issuing authorities and governed by the community of their users. Public blockchains of the next generation, such as Ethereum, have combined the community-based management of cryptocurrencies with the experimentation and practice of new forms of organization and finance, such as decentralized autonomous organizations (DAOs), initial coin offerings (ICOs) and decentralized finance (DeFi). Finally, private distributed ledgers, of which the best known and most practiced are the platforms developed within the Hyperledger open-source project, have released the potential for organizational innovation of these technologies from the support of cryptocurrencies, thus making it possible to involve brick-and-mortar companies in innovative business ecosystems.

At the base of all this there is a fundamental technological component, referred to here as the "digital ledger", which is nothing more than a database management system (DBMS) that preserves the history of all its records by appending new versions of records to previous ones and linking them together through identifiers such as hash pointers. Several DBMSs that predate the coming of age of blockchains work that way, like the popular Hadoop file system (HDFS) largely adopted for big data management. Distributed ledgers are a special case of digital ledger where the validation of transactions takes place through a consensus mechanism over a network of peer nodes. Public blockchains, in turn, specialise distributed ledgers by grouping transactions on records into blocks built by nodes rewarded for the construction with the native currency of the blockchain. Private distributed ledgers are yet another special-

ization, as are public distributed ledgers based on data arrangements alternative to blocks, such as the IOTA platform, aimed at supporting Internet-of-Things applications, that uses instead directed acyclic graphs.

However, there is more, in the latest ledgers that have entered the arena in the wake of the blockchain boom, than just decentralized validation of transactions through distributed consensus protocols. There are, in fact, smart contracts: conceptually introduced by the multifaceted scholar Nick Szabo (jurist, cryptographer, computer scientist) at the end of the 1990s for computer-based automated execution of legal contracts of a commercial nature [1], the idea stayed dormant until it was dusted off from oblivion by Vitalik Buterin, the founder and creator of Ethereum, in the mid-10s of this century, who transferred it to the eponymous blockchain by equipping it for implementation with Solidity, a contract-oriented programming language [2]. Ethereum's move was soon repeated on other platforms, both public and private, and there is now ample opportunity for smart contracts in blockchain and distributed ledger projects.

The reason why smart contracts lend themselves particularly well to implementation on a "distributed ledger" is actually more on the "ledger" than on the "distributed" part of the term. In fact, the permanence of data on the ledger makes the execution of smart contracts traceable, which of course is an advantage for auditing purposes whenever need may arise, as in a due diligence or a litigation. On the other hand, also distributed consensus can support traceability, albeit less essentially, by multi-checking the execution of the agreements made through many validators, which strengthens the claim to veracity of the transcribed data in the absence, by choice or necessity, of a reliable central authority. In any case, what really matters is operating in a context predisposed to auditing, such as a digital ledger by its nature is.

However, it is precisely the choices made for the implementation of smart contracts that have distanced the platforms that have adopted them both from the database characteristics of first generation distributed ledgers like the Bitcoin and from smart contracts as originally conceived. In fact, although referred to as a contract-oriented language, Ethereum's Solidity is actually a scripting

language that can write on the blockchain. Similarly, in other environments, such as Hyperledger, smart contracts are programs written in conventional languages that can access the ledger. These programming tools have certainly been useful in propelling the application of distributed ledgers beyond cryptocurrencies, but have also created reliability problems by making possible constructs of dubious semantics and arbitrary complexity; witness among all the notorious DAO exploit of 2016 in which a still unknown hacker exploited a bug in a Solidity smart contract to steal $ 50 million from the Ethereum common pot. Moreover, they have weakened the relationship between smart contracts and legal contracts as originally conceived by Szabo and taken up in intent, but not in practice, by Buterin; indeed, there is very little in common between the text of a commercial contract and a program in Solidity.

The purpose of this article is to define a general ledger model with contracts seen as declarative constructs on a database, rather than as arbitrary procedural programs, with well-defined operational characteristics in the tradition of advanced transaction approaches such as the renowned ACID model. To this end, we propose a bare-bones view of contracts based on the notion of *allocation* of resources to actors, so that a contract defines admissible evolutions of states of affairs defined by sets of such allocations.

At the same time, our model aims to make legal contracts in textual form effectively transferable and translatable into executable contracts, thus recovering the original idea of smart contract as introduced by Szabo. Hence, we propose two forms of contract-related automata: the first identifies the possible states of the contract with respect to the completion of some sets of obligations, where the discharge of an obligation is modeled as the transfering of some resource from some actor to another one; the second provides a refined view of the first, taking into consideration the possible orderings in which obligations can be discharged, thus turning a ledger into a faithful record of the sequence of transfers actually occurred during the execution of the contract.

On our way towards a model endowed with full-fledged DBMS characteristics, we will also deal with querying, by providing constructs to verify progress

in a contract execution, a functionality that is badly needed in practice, as well as to query the ledger by scrolling it back and forth in time, with advanced query modes ranging beyond the still rather limited solutions currently available. The formal cornerstones of our model are the theory of automata and temporal logic, formalisms that have been both rigorously systemized and theorized. As for distribution and centralization, we will be agnostic about the issue, since our model can be adapted to either one of the two options. For the organization of the data, we will maintain the minimal requirement that they are organized as in a ledger. This does not preclude more specialized organizations, like blocks, the model being easy to extend and adapt to ledgers structured as blockchains.

The rest of this article is structured as follows: Section 2 introduces some background notations and definitions. Section 3 provides the fundamental notions of *resource*, *actor* and *transfer*, so that a contract can be seen as defining a set of constraints on sequences of transfers of resources between actors. Section 4 shows how a legal contract understood as a set of interdependent obligations can be modeled in computational terms by defining admissible paths on some contract-related automaton, where transitions are associated with the discharge, through transfers of resources, of obligations. Section 5 then shows how contract automata can be integrated with a ledger so that their actions are transcribed as records into the ledger, while Section 6 builds, from the algebraic structure of ledgers and contracts, modal/temporal logics used in Section 7 to define an abstract query language that can be applied to extract information about both static (the records in the ledger) and dynamic (the states of contracts whose execution is in progress) aspects of a contract execution. Finally, Section 8 discusses related work and Section 9 concludes the article.

2. Background

We recall some basic notions and notations, useful in the rest of the paper.

For $n \in \mathbb{N}$, the set of all integers from 1 to n is denoted by $[n]$. An *alphabet* is a finite set $A = \{a_1, \ldots, a_k\}$, where each element a_i is called a *member* of A.

For $k \in \mathbb{N}$, a sequence $\omega = \langle x_1 \cdots x_k \rangle$ of elements from A is called a *word* on A of *length* k. We use the notation $\omega[i]$ to indicate the element x_i for $i \in [k]$. We denote the length of a sequence ω by $|\omega|$ and the unique word of length 0 by ϵ. Then, the set of all words on A (for each possible length) is denoted by A^*.

Given a partially ordered set $(X, \leq)$, if $(x, y) \in \leq$, then x is called a *prefix* of y and y is called a *prolongation* of x. Vice versa, given a prefix relation on a set X, its transitive closure defines a strict partial order.

It is easy to see that the above definition translates to the standard notion of prefix for the case of words. Indeed, given a word $\omega = \langle x_1 \cdots x_k \rangle$, for each $l \leq k$, the word $\omega_l = \langle x_1 \cdots x_l \rangle$ is the *prefix* of ω of length l, denoted by $pref(\omega, l)$. For any word ω, $pref(\omega, 0) = \lambda$. We denote by $PREF(\omega)$ the set $\{pref(\omega, l) \mid l \in \{0, \ldots, |\omega|\}\}$. A^* results thus partially ordered according to the prefix relation, i.e. $\omega \leq \omega'$ if and only if $\omega \in PREF(\omega')$.

A partially ordered set $(L, \leq)$ forms a *meet-semilattice*, $\mathbf{L} = (L, \leq, \wedge, \bullet)$, if it can be equipped with a *meet* operation, $\wedge$ (i.e., greatest lower bound), and a minimum element, $\bullet$. Then, a *tree* is a set of elements, called *paths*, in a meet-semilattice with the meet operation as a gluing function between them.

Definition 1 (L-tree). *Let* $\mathbf{L} = (L, \leq, \wedge, \bullet)$ *be a meet-semilattice. Then:*

- *A* deterministic **L**-tree *(*tree *for short) is a pair* $\mathcal{X} = (X, \wedge_X)$*, where* $X \subseteq L$ *and* $\wedge_X : X \times X \to L$ *is the restriction to* X *of the* meet $\wedge$*.*
- *An* **L***-tree is called* prefix-closed *if it contains all the prefixes of its paths.*
- *Given two* **L***-trees* $\mathcal{X} = (X, \wedge_X)$ *and* $\mathcal{Y} = (Y, \wedge_Y)$*, we say that* $\mathcal{X}$ *is a* subtree *of* $\mathcal{Y}$*, noted* $\mathcal{X} \subseteq \mathcal{Y}$*, if* $X \subseteq Y$ *(*$\subseteq$ *denotes set-theoretical inclusion).*

Given a meet-semilattice $\mathbf{L} = (L, \leq, \wedge, \bullet)$, the tree $\mathcal{L} = (L, \wedge)$ is canonically associated with $\mathbf{L}$ and formed by taking as set of paths the whole of L and defining the gluing of data between paths as given by the $\wedge$ operation.

Proposition 1. *Let* $\mathbf{L} = (L, \leq, \wedge, \bullet)$ *be a meet-semilattice and* $\mathcal{L} = (L, \wedge)$ *its canonically associated tree. Then:*

1. *a prefix-closed* **L***-tree is a meet-subsemilattice of* **L***;*

2. *the set of subtrees of $\mathcal{L}$, $Subtree(\mathcal{L})$, with set-theoretical inclusion as partial order, is a boolean algebra (see [3]); and*

3. *given $\mathcal{X} \in Subtree(\mathcal{L})$, the set of subtrees contained in $\mathcal{X}$, $Subtree(\mathcal{X})$, is a boolean algebra in turn (see [3]).*

We define an important property of monotonic functions between posets, by specialising to this case the general notion of adjunction between functors.

Definition 2 (Adjoints). *Given two posets, $P = (X, \leq)$ and $P' = (X', \leq')$, and two monotonic functions, $f : P \to P'$ and $g : P' \to P$, we say that f is* right adjoint to g *(g is* left adjoint to f*) whenever it happens that: for all $x \in X$ and $x' \in X'$, $x' \leq f(x)$ if and only if $g(x') \leq x$.*

In other words: $f(x)$ is the least upper bound of the set $\{x' \mid g(x') \leq x\}$. Dually, $g(x')$ is the greatest lower bound of the set $\{x \mid x' \leq fx\}$. In the case of posets, an adjunction is also called a *Galois connection*.

Fact 1. *The following hold:*

1. *An adjoint to a given function, if it does exist, is unique, hence it is characterised by this property.*

2. *Composition of two adjoints on the same side is still an adjoint on the same side. We say that the first one is preserved by the second one.*

3. *Boolean operators are adjoints on one side ($\wedge, \Rightarrow$ on the right, $\vee, \bot$ on the left), so that they are preserved by operators adjoint on the same side. In this sense a monotonic function between Boolean algebras which is a one-side adjunction, will provide what we call a* smooth translation *from one algebra to the other one, because it will preserve part of its structure;* a fortiori *if it is a two-side adjunction.*

3. Resources, actors, transfers

In this section, we introduce a bare-bones view of contracts, seen as constructs imposing constraints on admissible sequences of transfers of resources

(from some given set), among actors (from some given set), entitled to some form of ownership on these resources. The proposed model was first formulated in [4] and fully developed within the theory of reaction systems in [5].

The definition of these sets can occur either extensionally or intensionally. An example of a first case is provided by a loan contract, where both the lender and borrower parties are identified by names, the lent resource is identified by means of some title of property, the transfer of the right to its use by the possession of a copy of the text of the contract itself, and the number of instalments to be payed is defined and timestamped. An example of the second case is provided by bearer bonds, where the assets are identified, but the only identified actor is the emitter of the bond, or, vice versa, by a Memorandum of Understanding, whereby two actors commit to share future products, which are only partially defined, e.g., by mentioning their types, at the time of the contract.

The constraints set by the contract, on the other hand, range from general, overarching conditions, such as: "an actor k cannot transfer a resource r in a situation where k is not entitled to the use of r", to specific ones, as in the mentioned case of regular payments of a loan.

Another category of constraints may impose some kind of transactionality, such that a certain set of transfers have to occur "simultaneously" among some actors. In the loan example, with each payment received from the borrower, the lender must produce a receipt for it and give it to the borrower. Moreover, the execution of some transfer can be conditional on the occurrence of some event, for example the expiration of a deadline for payment of instalments, or a car accident for starting a damage compensation procedure.

Regardless of these differences, we assume that the general form of a transfer can be expressed as "actor $k1$ yields resource r to actor $k2$", assuming that r, $k1$, and $k2$ are all unambiguously identified at the time the transfer occurs.

Moreover, as we are interested here in the encoding of transfers on a digital ledger, we assume that each of r, $k1$, and $k2$ is suitably represented by some URI, corresponding, respectively, to a digital token representing the asset r, or to (possibly encrypted) accounts in a digital store associated with the ledger.

Example 1. *Let us consider the case of a contract binding* AL *to sell a house, say* house1 *(i.e., to transfer some property document* house1PropDoc*) to* PB*, and* PB *to pay a certain amount, say €500K, to* AL*. The actual payment (in the "real" world) is mediated through some identifiable resource, e.g., a set of banknotes, a cashier's cheque from* PB*'s account in the name of* AL*, a certain amount of bitcoins in* PB*'s wallet, etc., that we represent here abstractly as the unique token* PayKE500Doc1*, testifying that the payment has occurred (this is equivalent to, say, having a copy of the cheque taken in the real world and registered by the notar). Then, the two transfers, of* house1PropDoc *from* AL *to* PB *and of* PayKE500Doc1 *from* PB *to* AL*, constitute a* transaction*, as they must occur "at the same time", meaning that any observation of the execution of the contract which reports the first exchange must also be able to report on the second exchange, and vice versa. We say that these two transfers are* co-occurent*. On the other hand no transfer of* PayKE500Doc1 *from* PB *to some* k_1 *other than AL (since the token is produced for this specific transaction) or of* house1PropDoc *from* AL *to some* k_2 *other than* PB *(since the house cannot be sold two times) can be co-occurrent with the previous two.*

To sum up, we view any, virtual or tangible, asset or service mentioned in a contract, and whose creation, consumption, or *transfer*, is relevant to the contract, as a *resource*. More precisely, we consider that at each moment some *token* representing some form of (possibly shared) ownership of a resource is allocated to some actor also mentioned in the contract. Instances of such tokens are the payment instrument and the property document mentioned in Example 1.

In this paper, as we abstract away from the actual form and nature of the resources, being only interested in how the corresponding tokens are distributed among actors at any given moment (we call such a distribution a *state of affairs*), we use the terms (resource) "token" or "resource", indifferently. Hence, with each contract, we associate a nonempty set, $\mathbf{R}$, of resource tokens, constituting the overall universe of discourse.

The set $\mathbf{R}$ can be constructed so as to accommodate quotas of some bulk

resource, in analogy with shares of a company. So, for example, the right to use 100 liters of water delivered by *WaterInc.* can be represented as $[water, WaterInc, 100, R]$, where R is a unique identifier generated on the fly.

On the other hand, we see an *actor* as representing an individual entity bound by a contract and participating in transfers of resource tokens, whether at the yielding or at the receiving end. We assume that for any given contract the actors who may participate in it cannot be also regarded as resources, so that they are modeled as a nonempty set $\mathbf{K}$, with $\mathbf{K} \cap \mathbf{R} = \emptyset$.

We model the information that a token r is held by some actor k as the *allocation* of r to k, noted $[r, k]$. A *state of affairs on* $(\mathbf{R}, \mathbf{K})$ is then defined as a set of allocations such that each token in $\mathbf{R}$ is allocated to one and only one actor in $\mathbf{K}$. Hence, we model a state of affairs ς as the graph of a map $s_\varsigma : \mathbf{R} \to \mathbf{K}$, The set of all states of affairs on $(\mathbf{R}, \mathbf{K})$ is denoted by $\mathcal{S}(\mathbf{R}, \mathbf{K})$. Note that, according to the discussion above, although each token is unique, we can model joint ownership of a resource by generating a different token for each quota of that resource to which an actor is entitled.

A state of affairs evolves through exchanges of tokens among actors, possibly as part of complex transactions. Thus, the yielding of a token r by an actor k_1, originally holding it, to an actor $k_2 \neq k_1$, which becomes the holder of r, constitutes a *transfer*, noted $\theta = (r, k_1, k_2)$. Then, r is the *transferred resource*, denoted by $res(\theta)$. The set of all transfers on $(\mathbf{R}, \mathbf{K})$ is denoted by $TRA(\mathbf{R}, \mathbf{K})$.

Although the "real" transfer of resources may occur physically in some way not controlled by a program, a distributed ledger trusted to maintain information on the contract must be updated about such transfers as they occur, i.e., it has to maintain the record of the transfers undergone by the corresponding tokens. At any time instant, the current allocation of a resource can thus be reconstructed by following the sequence of transfers for its associated token(s).

To this end, we say that a transfer $\theta = (r, k_1, k_2) \in TRA(\mathbf{R}, \mathbf{K})$ is *applicable in* a state of affairs $\varsigma \in \mathcal{S}(\mathbf{R}, \mathbf{K})$ if and only if $[r, k_1] \in \varsigma$. Then, $APL(\varsigma)$ is the set of transfers applicable in ς. Given $\mathbf{R}$ and $\mathbf{K}$, for $\theta = (r, k_1, k_2) \in TRA(\mathbf{R}, \mathbf{K})$ and $\varsigma \in \mathcal{S}(\mathbf{R}, \mathbf{K})$, such that $\theta \in APL(\varsigma)$, the *application* of θ *to* ς, noted $apl_\theta(\varsigma)$,

produces $\varsigma' = (\varsigma \setminus \{[r, k_1]\}) \cup \{[r, k_2]\} \in \mathcal{S}(\mathbf{R}, \mathbf{K})$.

As stated before, contracts may require a given set of transfers to be co-occurrent. Their application must then occur in a *transactional* way, i.e., all transfers in the set are applied to a given state of affairs ς if they are all applicable in ς, and no transfer in the set is applied if any of them is not applicable in ς. Due to the overall constraint prohibiting multiple transfers of the same token from the same actor, and the constraint that each token can be held by only one actor at a time, the transfers in the set must refer to different resources.

Therefore, we define a *bundle* to be a nonempty set of transfers $\Theta = \{\theta_1, \ldots, \theta_n\}$ such that $res(\theta_i) \neq res(\theta_j)$, for $i \neq j$, $i, j \in \{1, \ldots, n\}$. The set of all bundles for $(\mathbf{R}, \mathbf{K})$ is denoted by $BUN(\mathbf{R}, \mathbf{K})$. A bundle $\Theta \in BUN(\mathbf{R}, \mathbf{K})$ is *jointly applicable in* $\varsigma \in \mathcal{S}(\mathbf{R}, \mathbf{K})$ if, for each $\theta \in \Theta$, $\theta \in APL(\varsigma)$. We then denote the set of bundles jointly applicable in ς by $JPL(\varsigma)$. The *joint application*, of $\Theta \in BUN(\mathbf{R}, \mathbf{K})$ to $\varsigma \in \mathcal{S}(\mathbf{R}, \mathbf{K})$, such that $\Theta \in JPL(\varsigma)$ (noted $jpl_\Theta(\varsigma)$), produces $\varsigma' = (\varsigma \setminus \{[r, k_1] \mid (r, k_1, k_2) \in \Theta\}) \cup \{[r, k_2] \mid (r, k_1, k_2) \in \Theta\} \in \mathcal{S}(\mathbf{R}, \mathbf{K})$.

During the execution of a contract, resources which were originally not in the availability of any participating actor can become available, as if produced in compliance with the contract. For example, a certain required document can be printed and kept by a notar as legal registration of some act; or a deliverable, required in a tender contract, hence typically not existing before the contract was put in place, can be produced as required during the execution of the tender. On the other hand, some resource can become no longer available for any exchange, as for example a car crashed in an accident covered by an insurance policy. In order to model such cases, we enrich the set $\mathbf{K}$ with the special symbols $\top$ and $\bot$, deemed *environment actors*, with the property that, for any resource $r \in \mathbf{R}$, both allocations $[r, \top]$ and $[r, \bot]$ are admissible, while no transfer of the form $(r, \bot, k)$ is admissible. We call the elements in $\mathbf{K} \setminus \{\top, \bot\}$ *proper actors* and allocations of the form $[r, k]$, for some $k \notin \{\top, \bot\}$, *proper allocations*.

The introduction of $\top$ and $\bot$ also provides a way to formalise the occurrence of events relevant to a contract (e.g., the deadline for an instalment is reached, an accident report is submitted, a cargo is delivered) in a way formally identical

to transfers of tokens. In particular, we define a set $\mathbf{Ev} \subseteq \mathbf{R}$ of *event tokens* (short, *events*), such that, for $e \in \mathbf{Ev}$, any $\varsigma \in \mathcal{S}(\mathbf{R}, \mathbf{K})$ can contain only one of $[e, \top]$, representing a situation where the event e has not occurred yet, or $[e, \bot]$, representing a situation where e has already occurred, Then, the only possible transfer for e is $(e, \top, \bot)$ modeling the occurrence of the event. As a consequence, an event occurs atomically and can never occur again.

A typical usage of transfers associated with event occurrences is to insert them into bundles, for transactions which must occur in correspondence with specific events. For example, in the loan contract, an instalment must be payed, and the corresponding receipt must be signed, with each 10th day of the month.

4. Modeling contracts

In this section, we show how a notion of contract, seen as a formal construct defining a structured collection of obligations, can be modeled in terms of resources, actors, and transfers, as introduced in Section 3.

In particular, we show how specific bundles of transfers, whose application transforms states of affairs into states of affairs related to a universe $\mathcal{S}(\mathbf{R}, \mathbf{K})$, correspond to transitions of a finite-state machine describing the possible executions of a contract on $(\mathbf{R}, \mathbf{K})$. This constitutes the basis for relating sequences of transfers encoded in a ledger, as discussed in Section 5, to the evaluation of queries on the excution of the contract, as discussed in Sections 6 and 7.

In essence, a contract must state, at least implicitly: the conditions of its validity; the acts through which the obligations of the contract are discharged and the conditions under which such acts can be carried out; and the situations corresponding to the completion of the obligations set by the contract (we then say that the contract is *honoured*) as well as those corresponding to a *breach* of these obligations. The latter may be associated with repair or compensation actions, which may thus be seen as alternate ways for honouring the contract, or as constituting the definition of a different contract altogether.

We can then encode the legal form of a contract $\mathcal{C}$ in a *legal contract au-*

tomaton, a finite state machine $\mathcal{M}^l_{\mathcal{C}} = (V^l, E^l, F, s^l, t^l, Act, TO, \lambda^l, \upsilon)$, where conditions correspond to states and the discharge of all of the obligations leading from a state to another corresponds to transitions. In particular: V^l is the set of *state nodes* (short, *states*); E^l is the set of *transition edges* (short, *transitions*), with $s^l : E^l \to V^l$ the *source map* and $t^l : E^l \to V^l$ the *target map*; $F \subseteq V^l$ is the set of *final* states; *Act* is the set of obligation-discharging *actions*, with $TO \subseteq Act$ the set of *timeout* actions; $\lambda^l : E^l \to \wp(Act)$ is a function labeling each transition with the set of actions needed for its firing; and $\upsilon : F \to \{HON, BRC\}$ is a function mapping each final state into the corresponding outcome (honoured or breach).

We assume that if a timeout action *to* is in $\lambda(\eta)$ for some $\eta \in E^l$, then *to* is the only action in $\lambda(\eta)$, i.e., the latter is a singleton. Moreover, we assume that $\mathcal{C}$ is such as to induce a strongly deterministic automaton $\mathcal{M}^l_{\mathcal{C}}$, in the sense that for $\eta_1, \eta_2 \in E^l$ such that $s^l(\eta_1) = s^l(\eta_2)$, neither $\lambda^l(\eta_1) \subseteq \lambda^l(\eta_2)$, nor $\lambda^l(\eta_2) \subseteq \lambda^l(\eta_1)$ hold, while the case $\lambda^l(\eta_1) \cap \lambda^l(\eta_2) \neq \emptyset$ is admitted.

The usual notion of trajectory between states, as given by the sequence of transitions leading from one to the other, can then be adapted to $\mathcal{M}^l_{\mathcal{C}}$. The set of all trajectories in $\mathcal{M}^l_{\mathcal{C}}$ is denoted by $\Pi^l_{\mathcal{C}}$ and the set of all initial trajectories (i.e., originating in the initial state of $\mathcal{M}^l_{\mathcal{C}}$) is denoted by $\Pi^{l,in}_{\mathcal{C}}$.

We introduce a sketch of a part of an insurance policy as a running example.

Example 2. Fred *has activated a comprehensive insurance policy for his car with the company* SURE!*, whereby a black box has been mounted on* Fred*'s car. As a part of this policy, the coverage of expenses to repair car damages, even if due to natural causes, is provided, according to a certain procedure, defining a contract* $\mathcal{C}_d$*. In particular, the procedure starts when a damage event occurs and the blackbox makes a report on it available to* Fred*. Then* Fred *has the possibility to file a claim for reimbursement on the* SURE! *system and to upload the report on its server. Then, the* SURE! *system will issue an offer for reimbursement to* Fred*, who might accept or reject it. If* Fred *communicates that he accepts the offer, the* SURE! *system issues both a refund order and a communication that*

the policy premium is increased. If Fred *communicates that he rejects the offer, no further action is needed. Each act must be performed within some deadline.*

We sketch here the content of states and transitions defining $\mathcal{M}^l_{\mathcal{C}_d}$.

1. The procedure enters the *Active* state when the damage event occurs AND the damage report is produced. A deadline is set to present the claim.
2. The *Claimed* state is reached when the claim is filed AND the report is acquired, before the deadline expires. At this stage, an obligation for *SURE!* to make an offer becomes active, which must be discharged before a set deadline, on pain of violating the contract.
3. The *Offered* state is reached when the reimbursement offer is made, so that the obligation is discharged. An obligation for *FRED* to respond to the offer is activated. This can be discharged by either accepting or rejecting the offer, but also by simply letting the deadline pass, after which the proposal becomes void (this would not constitute a violation).
4. The *Accepted* state is reached when *FRED* accepts the reimbursement offer. An obligation for the *SURE!* system to issue the refund AND to apply the raise to the premium is then activated. Also this obligation is under pain of violating the contract, if not discharged within the deadline.
5. The *Refunded* state is reached when the *SURE!* system issues the refund AND applies the raise in the premium, before the deadline.
6. The *Rejected* state is reached when *FRED* rejects the offer.
7. Each missed deadline brings into a corresponding state. Deadlines missed by *SURE!* represent a violation, while those missed by *FRED* do not.

Figure 1 depicts the resulting legal contract automaton $\mathcal{M}^l_{\mathcal{C}_d}$. Each state from which no transition starts, namely *Rejected*, *Refunded*, and each of the four *Out* states, is in F, as it represents a possible completion of the procedure. For $v \in F$, v is depicted in green if $\upsilon(v) = HON$, in red if $\upsilon(v) = BRC$.

It is to be noted that the procedure above is activated in the *SURE!* system each time a damage is recorded, and proceeds according to the same steps

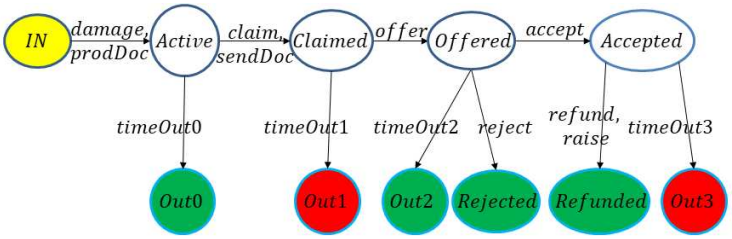

Figure 1: The finite state machine for the contract from Example 2.

independent of whether the customer is *FRED* or any other. Moreover, the contract might even be realised as a service and offered to insurance companies which might customise it, e.g., by setting deadlines, setting maximum refunds, or defining premiums and their raise. Without loss of generality, we can therefore consider that, for each insurer and customer, a fresh instance of the contract (and of the associated state machine) is always ready to control the next procedure, the state *IN* in Figure 1 representing the initial state for such an instance.

We now proceed to modeling contracts in terms of the approach presented in Section 3. Indeed, leveraging the notions of *resource*, *actor*, *transfer*, and *bundle*, we identify all types of conditions in the contract with predicates on states of affairs and the required actions with transfers. The latter is to say that an obligation is considered to be discharged when there is evidence that some actor has provided some resource (which might represent a physical asset, access to services, or execution of some task) to some other actor, i.e., a transfer of the corresponding token has been recorded. Where the contract requires a certain set of obligations to be completed before a different set comes into force, we model this in terms of a bundle of co-occurrent transfers.

Hence, the discharge of obligations in a legal contract automaton $\mathcal{M}^l_{\mathcal{C}}$ encoding a contract $\mathcal{C}$ realises some change in the states of affairs for an underlying system of tokens in a set $\mathbf{R}_{\mathcal{C}}$ (one for each piece of documentation mentioned in the contract) and actors in a set $\mathbf{K}_{\mathcal{C}}$ (one for each party involved in the contract). In particular, given a state of affairs in $\mathcal{S}(\mathbf{R}_{\mathcal{C}}, \mathbf{K}_{\mathcal{C}})$ holding in the initial state of $\mathcal{M}^l_{\mathcal{C}}$, each sequence of discharge actions (i.e., transfers) leads to the

production of some state of affairs in $\mathcal{S}(\mathbf{R}_\mathcal{C}, \mathbf{K}_\mathcal{C})$. It is now important to notice that, while a state transition is realised only when all the obligations needed are discharged (i.e., all the transfers in a bundle have been applied), it might be the case that other transfers, not composing a whole bundle, have been applied.

As an example, $Out0$ is reached whenever the event $(timeOut0, \top, \bot)$ occurs before *both* transfers corresponding to the discharge of the *claim* and *sendDoc* obligations have been applied. Hence, once $\mathcal{M}^l_{\mathcal{C}_d}$ reaches the state $Out0$, there are three possible corresponding states of affairs: one for not having applied either of *claim* or *sendDoc*, and one each for having applied just one of them.

Moreover, not all states of affairs in $\mathcal{S}(\mathbf{R}, \mathbf{K})$ can be mapped to some state of a legal contract automaton on $(\mathbf{R}, \mathbf{K})$. For example, a state of affairs where a claim has been filed but the report of the damage event has not been made available, or, vice versa, such a report is available, but no claim has been filed (yet) for that accident, does not correspond to any state of $\mathcal{M}^l_{\mathcal{C}_d}$, as a consistent state will be reached only when both of the associated transfers will have been applied (or the $timeOut0$ event will have fired.).

The model based on $\mathcal{M}^l_\mathcal{C}$ must therefore be integrated with a second finite state machine, univocally induced from $\mathcal{M}^l_\mathcal{C}$, the *contract execution automaton* $\mathcal{M}^e_\mathcal{C} = (V^e, E^e, F, s^e, t^e, Act, TO, \lambda^e, \upsilon)$, defined on the same sets of actions and final states as $\mathcal{M}^l_\mathcal{C}$, which considers all possible sequences of actions (transfers) in a concrete execution of the contract, compatible with its legal definition. In $\mathcal{M}^e_\mathcal{C}$, transitions are labeled with single actions, realised individually, and not with a set of actions to be completely realised for the transition to occur.

In particular (and specifying only the components not inherited from $\mathcal{M}^l_\mathcal{C}$): $V^e \supseteq V^l$ is the set of *states*; E^e is the set of *transitions*, with $s^e : E^e \to V^e$ the *source map* and $t^e : E^e \to V^e$ the *target map*; and $\lambda^e : E^e \to Act$ is a function labeling each transition with the specific action needed for its firing.

The derivation of $\mathcal{M}^e_\mathcal{C}$ from $\mathcal{M}^l_\mathcal{C}$ proceeds according to Construction 1.

Construction 1. *We isolate three components of the construction.*

1. *First, we include in V^e the whole of V^l. Then, for $\eta \in E^l$, we build*

all the "intermediate" states between $s^l(\eta)$ and $t^l(\eta)$, each corresponding to a nonempty proper subset of $\lambda^l(\eta)$. We call $Lin(\eta)$ the resulting set of states. For example, given the states Active *and* Claimed *of V_d^l, Lin*((Active,Claimed)) *contains the states* Active/Claim *and* Active/Sent*, corresponding to the partial realisations of the set of actions* $\lambda^l((\text{Active,Claimed})) = \{\text{claim}, \text{sendDoc}\}$.

2. *Then, for $\eta \in E^l$, we include in E^e all the transitions connecting $s^l(\eta)$ to $t^l(\eta)$ through the states in $Lin(\eta)$. That is to say that, for $\eta \in E^l$, and for $s_1, s_2 \in Lin(\eta) \cup \{s^l(\eta), t^l(\eta)\}$, respectively associated with sets $\alpha_1, \alpha_2 = \alpha_1 \cup \{a\}$ for $a \in \lambda^l(\eta) \setminus \alpha_1$, E^e will include a transition between s_1 and s_2 labeled with a. Note that this also includes transitions leaving $s(\eta)$ (where $\alpha_1 = \emptyset$) and transitions reaching $t(\eta)$ (where $\alpha_2 = \lambda^l(\eta)$). For example, in E_d^e a transition exists from* Active *to each of* Active/Claim *and* Active/Sent*, labeled with the corresponding action, and from each of* Active/Claim *and* Active/Sent *to* Claimed*, labeled with the action needed to complement the set in the original transition from* Active *to* Claimed *i.e.,* sendDoc *for* Active/Claim *and* claim *for* Active/Sent.

3. *The two items above are subsumed under a general construction, whereby V^e and E^e include all of the states and transitions needed to account for the possible interleaving of actions labeling different transitions in E^l originating from a given state, until the completion of one of these sets (no two transitions from the same state are labeled with the same set of actions), thus keeping track of the "progress" towards completion of each such set. To this end, for a state $v \in V^l$, we take all the transitions in E^l with source in v, let this be $T(v)$, and add to V^e a node for each set in $\wp(U(v)) \setminus Cmpl(v)$, where $U(v) = \bigcup\{\lambda^l(\eta) \mid \eta \in T(v)\}$ and $Cmpl(v) = \{X \in U(v) \mid (\nexists \eta \in E^l)[\lambda^l(\eta) \subseteq X]\}$. In other words, we consider all possible subsets in the union of all the sets of actions labeling these transitions, minus those subsets corresponding to the completion of one transition, as only one such set can be completed. Indeed, all and only*

the states in V^l are taken to correspond to the completion of $\lambda^l(\eta)$ for any $\eta \in T(v)$. (This also means that once a state $v' \in V^l$ is reached (i.e., $\lambda^l(\eta)$ has been completed for some $\eta \in T(v), v' = t^l(\eta)$), all the actions representing progress towards other states become irrelevant, as a different set of intermediate states is constructed for transitions in $T(v')$). Then, the set E^e includes all the needed transitions between these states, one for each action representing an increment towards completion of $\lambda^l(\eta)$, for some $\eta \in T(v)$. Transitions are only added towards reaching a state in V^l which is the target for the corresponding transition in E^l, as defeasing of actions is not admitted. Note that by definition of timeout state, each such state is reached with just one action, so that a timeout would interrupt any possible trajectory in $\mathcal{M}^e_{\mathcal{C}}$ not yet landed in a state from V^l. For example, irrespective of whether the timeOut0 *event occurs in* Active, Active/Claim, *or* Active/Sent, *it immediately completes the singleton set of actions needed to reach* Out0. *Hence, in addition to a transition from* Active *to* Out0, *generated in Item 2, E^e_d has a transition to* Out0 *from each of* Active/Claim *and* Active/Sent, *both of them labeled* timeOut0.

Similarly to the case for $\mathcal{M}^l_{\mathcal{C}}$, we denote the set of trajectories in $\mathcal{M}^e_{\mathcal{C}}$ by $\Pi^e_{\mathcal{C}}$ and the set of initial trajectories by $\Pi^{e,in}_{\mathcal{C}}$. The difference between considering all trajectories in $\Pi^{e,in}_{\mathcal{C}}$ or only those passing through only the intermediate states built in $Lin(\eta)$ for some $\eta \in E^l$, (which correspond to a trajectory in $\Pi^l_{\mathcal{C}}$) will play an essential role in the definition of the logics in Section 6.

Another caveat is in order. In principle, a state of affairs, even if containing allocations for all of the resources and actors involved in a contract, might include allocations for a set of resources concerning other aspects of their rapport.

For example, and extending Example 2, some tokens in the complete model for a car insurance policy between an *insurer* company and a *customer* might refer to topics such as *coverage of theft*, *extension of civil liability*, *deadlines for payments,* etc, not relevant to the procedures for damage management. Hence, an automaton $\mathcal{M}_{\mathcal{C}_x}$ (either legal or execution) might simply model some spe-

cific section $\mathcal{C}_x$ in a wider contract $\mathcal{C}$, relative to a universe of states of affairs $\mathcal{S}(\mathbf{R}_\mathcal{C}, \mathbf{K}_\mathcal{C})$. Then, we should regard states in $\mathcal{M}_{\mathcal{C}_x}$ as inducing some relation on (some subset of) $\mathcal{S}(\mathbf{R}_\mathcal{C}, \mathbf{K}_\mathcal{C})$. A compositional view of contracts can then ensue, resulting in progressive refinements of this relation.

In general, for a contract $\mathcal{C}$, we will identify the subset $\mathcal{S}_\mathcal{C} \subseteq \mathcal{S}(\mathbf{R}_\mathcal{C}, \mathbf{K}_\mathcal{C})$, of states of affairs *consistent with* $\mathcal{M}^l_\mathcal{C}$, in the sense that they can occur as the combined effect of a sequence of transfers which can occur as a possible execution of the obligations in the contract, according to $\mathcal{M}^e_\mathcal{C}$.

The translation of $\mathcal{M}^l_\mathcal{C}$ in terms of a $(\mathbf{R}_\mathcal{C}, \mathbf{K}_\mathcal{C})$ system proceeds as follows.

1. Each party involved in $\mathcal{C}$ is modelled as an actor $k \in \mathbf{K}_\mathcal{C}$.
2. Each resource to be produced or transfered in order to discharge an obligation in $\mathcal{C}$ is modelled as a token $r \in \mathbf{R}_\mathcal{C}$. In particular, the production of a document corresponds to a transfer from $\top$ to a proper actor.
3. Each accident to be documented in $\mathcal{C}$ is modelled as an event $e \in \mathbf{Ev}_\mathcal{C} \subseteq \mathbf{R}_\mathcal{C}$. The expiration of a deadline is also modelled as an event. Event occurrences are therefore modelled as transfers of the form $(e, \top, \bot)$.
4. Each action discharging an obligation (i.e., in Act) is modelled as a transfer, in a set $\mathrm{TRA}_\mathcal{C}$, of the resource associated with that obligation.
5. Each set of actions collectively ensuring a transition between states in $\mathcal{M}^l_\mathcal{C}$ is modelled as a bundle, in a set $\mathrm{BUN}_\mathcal{C}$, of the corresponding transfers.

We can now define the system of resources and actors modeling a contract.

Definition 3 (Resource-based contract model). *Let $\mathcal{C}$ be a contract and let $\mathcal{M}^l_\mathcal{C} = (V^l, E^l, F, s^l, t^l, Act, \lambda^l, \upsilon)$ be the associated contract automaton. Then a* resource-based contract model *is a tuple $\mathbf{RS}_\mathcal{C} = (\mathcal{S}_\mathcal{C}, BUN_\mathcal{C}, TRA_\mathcal{C})$, together with three mappings: $\gamma : \Pi_\mathcal{C} \to \wp(\mathcal{S}_\mathcal{C})$, $\beta : E^l \to BUN_\mathcal{C}$, and $\rho : Act \to TRA_\mathcal{C}$, collectively enjoying the following properties:*

1. *For $\tau \in \Pi_\mathcal{C}, \eta \in E^l, \Theta = \beta(\eta)$: $\langle \tau \cdot \eta \rangle \in \Pi_\mathcal{C} \Rightarrow (\forall \varsigma \in \gamma(\tau))[jpl_\Theta(\varsigma) \in \gamma(\langle \tau \cdot \eta \rangle)]$.*

2. For $\eta \in E^l$: $\beta(\eta) = \{\rho(act) \mid act \in \lambda(\eta)\}$[1].

We can now show the constrution of the legal contract automaton $\mathcal{M}^l_{\mathcal{C}_d}$ for the contract $\mathcal{C}_d$ of Example 2, managing damage events (the contract execution automaton $\mathcal{M}^e_{\mathcal{C}_d}$ is induced according to Construction 1). Since each arrow in E^l for $\mathcal{M}^l_{\mathcal{C}_d}$ is unique to an ordered pair of states, we identify here an edge in E^l with the corresponding pair. Moreover, since, for each state in V^l, there is a unique trajectory leading to it, we identify a state v and the corresponding trajectory, denoted by $\overline{v}$, in the definition of γ.

Example 3. *With reference to Example 2, we define:*

- $\mathbf{R}_{\mathcal{C}_d} = \{oldPrem, claim, damageEv, damageDoc, offer, reject, accept, raise, refund, out0, out1, out2, out3\}$.
- $\mathbf{K}_{\mathcal{C}_d} = \{customer, insurer, \bot, \top\}$.
- *The function γ is defined as follows*[2]*:*
  - $\gamma(\overline{In}) = \{[oldPrem, customer]\} \cup \{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d} \setminus \{oldPrem\}\}$.
  - $\gamma(\overline{Active}) = \{[oldPrem, customer], [damageDoc, customer], [damageEv, \bot]\} \cup \{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d} \setminus \{oldPrem, damageEv, damageDoc\}\}$.
  - $\gamma(\overline{Claimed}) = \{[claim, insurer], [damageDoc, insurer], [damageEv, \bot], [oldPrem, customer]\} \cup \{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d} \setminus \{claim, damageEv, oldPrem, damageDoc\}\}$;
  - $\gamma(\overline{Offered}) = \{[offer, customer], [damageEv, \bot], [damageDoc, insurer], [oldPrem, customer], [claim, insurer]\} \cup \{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d} \setminus \{offer, damageEv, damageDoc, oldPrem, claim\}\}$;
  - $\gamma(\overline{Accepted}) = \{[accept, insurer], [offer, customer], [oldPrem, customer], [damageEv, \bot], [claim, insurer], [damageDoc, insurer]\} \cup \{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d} \setminus \{claim, accept, damageEv, damageDoc, offer, oldPrem\}\}$.

[1] That is, β is completely consistent with λ^l.

[2] As is customary, we will write a singleton $\{\phi\}$ as simply ϕ, for a set ϕ.

- $\gamma(\overline{Refunded}) = \{[accept, insurer], [raise, customer], [damageEv, \bot], [damageDoc, insurer], [refund, customer], [claim, insurer], [oldPrem, \bot]\}, \cup\{[r, \top] \mid r \in \{reject, out0, out1, out2, out3\}$;
- $\gamma(\overline{Rejected}) = \{[damageEv, \bot], [damageDoc, insurer], [claim, insurer], [offer, customer], [reject, insurer]\}\cup\{[r, \top] \mid r \in \mathbf{R}_{\mathcal{C}_d}\backslash\{claim, reject, offer, damageDoc, damageEv\}\}$;
- $\gamma(\overline{Out0}) = \{\varsigma\in\mathcal{S}(\mathbf{R}_{\mathcal{C}_d}, \mathbf{K}_{\mathcal{C}_d}) \mid [out0, \bot]\in\varsigma, [outX, \top]\in\varsigma \text{ for } X\in\{1, 2, 3\}\}$.
- $\gamma(\overline{Out1}) = \{\gamma\in\mathcal{S}(\mathbf{R}_{\mathcal{C}_d}, \mathbf{K}_{\mathcal{C}_d}) \mid [out1, \bot]\in\gamma, [outX, \top]\in\varsigma \text{ for } X\in\{0, 2, 3\}\}$.
- $\gamma(\overline{Out2}) = \{\gamma\in\mathcal{S}(\mathbf{R}_{\mathcal{C}_d}, \mathbf{K}_{\mathcal{C}_d}) \mid [out2, \bot]\in\gamma, [outX, \top]\in\varsigma \text{ for } X\in\{0, 1, 3\}\}$;
- $\gamma(\overline{Out3}) = \{\gamma\in\mathcal{S}(\mathbf{R}_{\mathcal{C}_d}, \mathbf{K}_{\mathcal{C}_d}) \mid [out3, \bot]\in\gamma, [outX, \top]\in\varsigma \text{ for } X\in\{0, 1, 2\}\}$.

The set $BUN_{\mathcal{C}_d}$ and the function β are jointly defined as follows, implicitly defining also the set $TRA_{\mathcal{C}_d}$ and the ρ function:

- $\beta(In, Active) = \{(damageEv, \top, \bot), (damageDoc, \top, customer)\}$
- $\beta(Active, Claimed) = \{(claim, \top, insurer), (damageDoc, customer, insurer)\}$,
- $\beta(Claimed, Offered) = \{(offer, \top, customer)\}$,
- $\beta(Offered, Accepted) = \{(accept, \top, insurer)\}$,
- $\beta(Accepted, Refunded) = \{(refund, \top, customer), (oldPrem, customer, \bot), (raise, \top, customer)\}$,
- $\beta(Offered, Rejected) = \{(reject, \top, insurer)\}$,
- $\beta(Active, Out0) = \{(out0, \top, \bot)\}$,
- $\beta(Active, Out1) = \{(out1, \top, \bot)\}$,
- $\beta(Active, Out2) = \{(out2, \top, \bot)\}$,
- $\beta(Active, Out3) = \{(out3, \top, \bot)\}$,

By construction of $\mathbf{RS}_{\mathcal{C}}$, one can derive the sets of sequences of bundles and of transfers leading from any given state of affairs in $\mathbf{RS}_{\mathcal{C}}$ to any other state of affairs in $\mathbf{RS}_{\mathcal{C}}$ reachable from the first, following a trajectory in $\mathcal{M}^l_{\mathcal{C}}$.

The functions β and ρ then induce the two notions of *bundle* and *transfer trajectory-labeling*, for trajectories in $\mathcal{M}^l_{\mathcal{C}}$ and $\mathcal{M}^e_{\mathcal{C}}$, respectively.

Definition 4 (Trajectory labelings). *Let $\mathcal{C}$ be a contract, let $\mathcal{M}^l_{\mathcal{C}}$ and $\mathcal{M}^e_{\mathcal{C}}$ be the associated legal contract and contract execution automata, and let $\tau^l = \langle t^l_1 \cdots t^l_n \rangle \in \Pi^l_{\mathcal{C}}$ and $\tau^e = \langle t^e_1 \cdots t^e_m \rangle \in \Pi^e_{\mathcal{C}}$, be such that all and only the states in V^l reached with τ^e are also reached with τ^l, in the same order. In this case, we say that τ^e is an* unfolding *of τ^l, and we define:*

- $B^\beta_{\mathcal{C}}(\tau^l) = \langle \beta(t^l_1) \cdots \beta(t^l_n) \rangle$ *to be the* bundle trajectory-labeling of τ^l*;*
- $L^\rho_{\mathcal{C}}(\tau^e) = \langle \rho(t^e_1) \cdots \rho(t^e_n) \rangle$*, to be the* transfer trajectory-labeling *of τ^e.*

If $L^\rho_{\mathcal{C}}(\tau^e)$ is such that only states in $\bigcup_{\eta \in E^l} Lin(\eta)$ (see Construction 1) are visited, then we say that $L^\rho_{\mathcal{C}}(\tau^e)$ is a linearisation *of $B^\beta_{\mathcal{C}}(\tau^l)$.*

The set of all bundle trajectory-labelings for trajectories in $\Pi^l_{\mathcal{C}}$ is denoted by $\mathbf{B}^\beta_{\mathcal{C}}$ and the set of all bundle trajectory-labelings for initial trajectories in $\Pi^{l,in}_{\mathcal{C}}$ is denoted by $\mathbf{B}^{\beta,in}_{\mathcal{C}}$. Analogously, the set of all transfer trajectory-labelings for trajectories in $\Pi^e_{\mathcal{C}}$ is denoted by $\mathbf{L}^\rho_{\mathcal{C}}$ and the set of all trajectory transfer labelings for initial trajectories in $\Pi^{e,in}_{\mathcal{C}}$ is denoted by $\mathbf{L}^{\rho,in}_{\mathcal{C}}$. A prefix-induced partial order is therefore defined on both sets of labelings.

1. Given two trajectories $\tau^l_1, \tau^l_2 \in \Pi^l_{\mathcal{C}}$, $\tau^l_1 \leq^l_{\mathcal{C}} \tau^l_2$ *iff* $\tau^l_1 \in PREF(\tau^l_2)$
2. Given two bundle trajectory-labelings $\beta_1 = B^\beta_{\mathcal{C}}(\tau^l_1)$ and $\beta_2 = B^\beta_{\mathcal{C}}(\tau^l_2)$, $\beta_1 \leq_\beta \beta_2$ *iff* $\beta_1 \in PREF(\beta_2)$ *iff* $\tau^l_1 \leq^l_{\mathcal{C}} \tau^l_2$.
3. Given two trajectories $\tau^e_1, \tau^e_2 \in \Pi^e_{\mathcal{C}}$, $\tau^e_1 \leq^e_{\mathcal{C}} \tau^e_2$ *iff* $\tau^e_1 \in PREF(\tau^e_2)$.
4. Given two transfer trajectory-labelings $\rho_1 = L^\rho_{\mathcal{C}}(\tau^e_1)$ and $\rho_2 = L^\rho_{\mathcal{C}}(\tau^e_3)$, $\rho_1 \leq_\rho \rho_2$ *iff* $\rho_1 \in PREF(\rho_2)$ *iff* $\tau^e_1 \leq^e_{\mathcal{C}} \tau^e_2$.

Remark 1. *Strictly speaking, bundles and transfers are complex structures, so that in principle we should distinguish between their definition in terms of declarative specifications of behaviours modifying a state of affairs and their unique names to be used in labeling. This could be achieved by associating with each transfer θ its unique name $\overline{\theta}$ and with each bundle Θ its unique name $\overline{\Theta}$. For*

the sake of simplicity we do not introduce this distinction here, relying on the context to clarify whether we are referring to names or to specifications.

5. Encoding transfers on ledgers

In this section, we discuss how the model of contracts presented in Section 4, which views them as defining admissible evolutions of the state of affairs of a system of resources and actors, lends itself to the recording of sequences of contract-related actions on a ledger, making it amenable to forms of auditing on their compliance with the constraints set by the contract. As a consequence, we do not simply deal with the "current" set of allocations, as maintained in traditional databases, but we aim at reconstructing the whole "history" of transfers involving resources and actors pertaining to a contract.

Before moving on, it is important to clarify that we only consider the recording of actions on a ledger, while we are agnostic regarding the deployment of the contract which can itself reside on the ledger, as in some blockchains, rather than on a network node, as in other cases. In either case case the contract and its actions are at different levels and the focus here is on actions.

The *allocation history* of a resource $r \in \mathbf{R}$ can then be reconstructed by considering, in the sequence of encodings of applied transfers, those for which $res(\theta) = r$. We denote the set of sequences of (encodings of) transfers (irrespective of the contract for which it has been executed) in the ledger by LT. Each sequence $\sigma \in LT$ is called a *ledger state*. Note that LT is *prefix-complete* with respect to the standard prefix-order $\leq$ (i.e., for any $\sigma \in LT$, $PREF(\sigma) \subseteq LT$).

In a centralized ledger, the recording of transfers has a natural correspondence with contract execution. In a distributed environment, however, as they can originate in different nodes, we cannot assume that a log state records transfers in the exact order in which they occurred. We can assume, however, that a *resource-safeness* (see Definition 5) property holds in every encoding sequence registered in the ledger. That is, for $r \in \mathbf{R}$, a new transfer of r is not logged if the previous transfer of r has not yet been transferred to the ledger. This

property can be verified through a simple check during coding.

Besides resource-safeness, we consider other properties, providing the basis for some form of conformance-checking on the traces of transfers associated with a contract. Hence, compliance to a *wallet-safeness* property amounts to requiring that actors can only use resources they are entitled to (e.g., no form of double spending can be encoded in the ledger); compliance to a *bundle-safeness* property provides some form of transactionality (if we observe a transfer from a bundle Θ for some $r \in \mathbf{R}_{\mathcal{C}}$, then the next transfer for r can only appear after Θ is completed); and compliance to a *contract-safeness* property, with respect to a contract $\mathcal{C}$, means that the sequence of transfers encoded in the ledger is consistent with both partial orders, $\leq_\beta$ and $\leq_\rho$, induced by $\mathcal{M}^l_{\mathcal{C}}$ and $\mathcal{M}^e_{\mathcal{C}}$.

Definition 5 provides a formal account of these properties. From now on, we introduce a notion of *encoding* of a transfer $\theta = (r, k_1, k_2)$ on the ledger, noted $\varrho(\theta)$. An encoding is a construct maintaining information about r, k_1, and k_2, together with some metadata, such as a timestamp for its creation time, the identifier of some validation authority, or of the contract to which it refers.

Definition 5 (Properties of a ledger state). *Let $\mathbf{R}$ be a set of resources and let $\sigma = \langle \varrho(\theta_1) \cdots \varrho(\theta_k) \rangle$ be a ledger state, with $res(\theta_i) \in \mathbf{R}$ for $i \in \{1, \ldots, k\}$. Then, we define:*

- *For $r \in \mathbf{R}$, σ is* r-safe *if: for any $j > i \in [k]$, such that $\theta_i = (r, k_1, k_2), \theta_j = (r, k_3, k_4)$, for some $k_3 \neq k_2$, σ contains a sub-sequence $\langle \varrho(\theta_{l_1}) \cdots \varrho(\theta_{l_m}) \rangle$, $l_1 > i, l_m \leq j$, such that, for $s \in \{1, \ldots, m-1\}$, $\theta_{l_s} = (r, k_{l_s}, k_{l_{s+1}})$, with $k_{l_1} = k_2, k_{l_m} = k_3$.*
- *For a contract $\mathcal{C}$ (so that $\mathbf{R} = \mathbf{R}_{\mathcal{C}}$):*
  - *σ is* $\mathcal{C}$-wallet safe *if it is r-safe for any $r \in \mathbf{R}_{\mathcal{C}}$;*
  - *σ is* $\mathcal{M}^l_{\mathcal{C}}$-bundle safe *if it is $\mathcal{C}$-wallet safe and, for $\Theta \in \beta(E^l), \theta \in \Theta$, $\varrho(\theta)$ appearing in σ: no encoding $\varrho(\theta')$, with $res(\theta')$=$res(\theta)$, appears in σ before all transfers in Θ have been encoded in σ.*
  - *σ is* $\mathcal{M}^l_{\mathcal{C}}$-contract safe *if it is $\mathcal{M}^l_{\mathcal{C}}$-bundle safe and it is a prefix of*

some transfer trajectory-labeling of $\mathcal{M}^l_{\mathcal{C}}$.

Proposition 2. *Each of the properties in Definition 5 is decidable.*

Proof:[Sketch] It is easy to see that by inspecting the sequence $\sigma_{\mathcal{C}}$, of transfers relative to resources in some contract $\mathcal{C}$, extracted from a ledger state σ, and comparing it with the sequences of transfers in $\mathbf{L}^{\rho}_{\mathcal{C}}$ (using the induced contract execution automaton $\mathcal{M}^e_{\mathcal{C}}$), each of the considered properties can be assessed. In particular, a ledger state is $\mathcal{M}_{\mathcal{C}}$-contract safe *iff* its extracted sequence for resources in $\mathbf{R}_{\mathcal{C}}$ is in $\mathbf{L}^{\rho,in}_{\mathcal{C}}$. Since a ledger state is constituted of a finite number of transfers, the extracted sequence $\sigma_{\mathcal{C}}$ needs to be compared with a finite number of transfer trajectory-labelings[3] up to the length $|\sigma_{\mathcal{C}}|$. □

The proof is then immediate for Corollary 1.

Corollary 1. *All prefixes of a* $\mathcal{M}^l_{\mathcal{C}}$*-contract safe state are* $\mathcal{M}^l_{\mathcal{C}}$*-contract safe.*

In order to define the set of sequences of transfers encoded in a ledger, corresponding to possible contract executions, we have to go back to the considerations in Section 4 on the possibility that some bundles do not get completed, so that some of the transfers in them appear in a labeling in $\mathbf{L}^{\rho,in}_{\mathcal{C}}$, but not in any linearisation of a labeling in $\mathbf{B}^{\beta,in}_{\mathcal{C}}$.

In particular, resuming the arguments in Construction 1, for each state $v \in V^l$, the determinism of $\mathcal{M}^l_{\mathcal{C}}$ allows us to establish a bijection between the set $T(v)$, of transitions leaving v, and the set $Bun(v) = \{\beta(\eta) \mid \eta \in T(v)\}$ of bundles labeling these transitions. Then $Trf(v) = \bigcup Bun(v)$ is the set of transfers in all the bundles labeling transitions in $T(v)$. Let σ be a ledger state such that its last encoded transfer "completes" a bundle in $Bun(v)$, leading to a state $v' \in V^l$. No bundle which has not been completed in σ can then be completed in any prolongation of σ. As a consequence, σ comprises a sparse subsequence σ' of (encodings of) transfers which are parts of the transfer trajectory-labeling for a

[3]Remember that $\mathbf{L}^{\rho,in}_{\mathcal{C}}$ is prefix-complete.

trajectory $\tau \in \Pi^e_{\mathcal{C}}$ leading to v', and a sparse subsequence σ'' of (encodings of) transfers not on this trajectory, and consequently, not on any trajectory which is a prolongation of τ. We say that σ' is "useful" and that σ'' is "useless". The reasoning can be extended to sequences which proceed beyond the last completed bundle, i.e., after reaching a state $v \in V^l$. Let σ be one such sequence and let σ_v the suffix of σ following the completion of a bundle leading to v. Then, if a prolongation of σ_v, say σ'_v, leads to a new state v_2, the transfers in $\beta((v, v_2))$ will be incorporated in the useful part of $\langle \sigma \cdot \sigma'_v \rangle$, while the remaining transfers in σ'_v will be incorporated in its useless part.

6. An algebraic model of contracts and its logic

We can now go and resume our program of constructing a suitable logic for ledgers and contract automata based on their associated algebraic structures. The presented theory is inspired by, but not immediately reducible to, a general theory introduced in categorical terms in [6], used in [3] to provide a logic for (possibly nondeterministic) processes, and adapted to the study of contracts in [4]. In fact, the idea originated from the natural association of a Heyting first-order logic to a category of generalised labeled trees, proven to be a Heyting category and extended with several modal/temporal operators in [3]. There, the labeling of trees via a meet-semilattice was a crucial device, since we had to deal with a non-deterministic situation, which we modeled by allowing two different paths to be labeled via the same element in the meet-semilattice.

When, as here, one has a deterministic situation (i.e., one path, either in $\mathcal{M}^l_{\mathcal{C}}$ or in $\mathcal{M}^e_{\mathcal{C}}$ can only have a unique label), the labeling machinery can be dropped to consider the meet-semilattice itself as a tree, and the theory can be expressed in (po)set-theoretical terms. Indeed, the structure associated with the set of subtrees of a tree $\mathcal{X}$ is now that of a boolean algebra, so that, by using subtrees as interpretations of formulas,a boolean logic is obtained, (extendable with modal/temporal operators, see [3]), in analogy with the standard interpretation of formulas in terms of subsets of some universe of discourse.

In our approach, this kind of structure can be associated both with a ledger used to register (encodings of) transfers performed under the constraints set by (possibly more than) one pair of contract automata, the legal and the execution one, and with the behaviour of the contract automata themselves (or more of them). As a consequence, we are interested in two (or more) meet-semilattices at the same time, one derived from the sequence of transfers recorded in the ledger, the other ones derived from the admissible initial sequences of bundles (transfers) in a bundle (transfer) trajectory-labeling for some contract (or contracts) whose executions are recorded in the ledger.

Without loss of generality, we restrict ourselves to one ledger and one contract with its two automata. The involved meet-semilattices correspond to sorts in a category canonically associated with a classical many-sorted logic, namely, the category **TSL** of trees derived from meet-semilattices, with monotonic functions as morphisms. Indeed, for every object $\mathcal{X}$ of **TSL**, $Sub(\mathcal{X})$ is a Boolean algebra, and morphisms allow a canonical definition of quantifiers, as shown later. The logic associated with **TSL** will be expanded with modal/temporal operators, again defined canonically from the order relation on paths.

Given a finite alphabet of transactions T, the monoid T^*, of freely-generated sequences of transactions, is canonically defined. This is a meet-semilattice as well as a tree (rooted in the empty sequence ϵ) in our sense; all the prefixes of elements in T^* are still in T^*.

However, due to the condition of resource-safeness, not all possible sequences in T^* can actually constitute the record, in a ledger, of a sequence of transfers. Hence, the tree of possible ledger states forms a prefix-closed proper subtree of T^*, LT (as per the discussion in Section 5). When a new transfer is registered on the ledger, this causes the selection in LT of all and only those paths which present that transfer at the corresponding step, and which are like-wise consistent at all previous steps, thus eliminating all the paths which are not its prolongations from the possible evolutions of the ledger. This is precisely the novelty in this approach: that we consider a special subfamily of $Subtree(LT)$, namely what we call the *evolutions* of the ledger. If we take $\mathcal{E}_0 = LT$ to define

the set of ledger states which are still reachable at time 0, by repeating this operation any number of times n, we obtain a sequence of (*instantaneous*) *evolutions* $\langle \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_n \rangle$ (a subset of $Subtree(LT)$), with $\mathcal{E}_{i+1} \subseteq \mathcal{E}_i$, for $i \in \{0, \ldots, n{-}1\}$.

Remark 2. *We observe that, in general, one has* $\mathcal{E}_{i+1} \subsetneq \mathcal{E}_i$ *The case* $\mathcal{E}_{i+1} = \mathcal{E}_i$ *occurs only for* $i = n{-}1$*, in a situation where, after applying all transfers in* $\mathcal{E}_i$*, the only allocation of the form* (r, k)*, for* $k{\neq}\bot$*, in the resulting state of affairs is such that* $r \in \mathbf{Ev}$*, so that the only possible transfer left at step* $i{+}1$ *is* $(r, \top, \bot)$*.*

In this perspective, the overall ledger evolution is modeled as a sequence of trees on $(T^*, \leq, \wedge, \lambda)$, each containing the following one in the sequence: after having established a finite set of transfer records, only those paths which are prolongations of it are selected to generate the prefix-complete tree $\mathcal{E}_t$, representing the instantaneous evolution at time t. Figure 2 presents an intuitive representation of this sort of pruning, occurring from one step to the next.

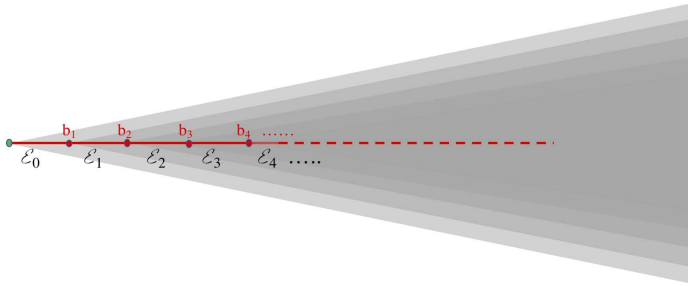

Figure 2: An evolving ledger.

The resulting chain of instantaneous evolutions will in turn represent the global evolution. Every instantaneous evolution $\mathcal{E}_k$ contains an *initial chain* of paths (the *established* part coloured in red in Figure 2), of length k.

On the other hand, we can easily see, on the basis of what discussed in Section 4, that both the set of all bundle initial trajectory-labelings in $\Pi_C^{l,in}$, $\mathbf{B}_C^{\beta,in}$, and the set of all transfer initial trajectory-labelings in $\Pi_C^{e,in}$, $\mathbf{L}_C^{\rho,in}$, with their partial orderings, are semilattices (and trees).

We will now define an *occurring function* for each of them: namely $\nu^l : LT \to \mathbf{B}_C^{\beta,in}$ (resp. $\nu^e : LT \to \mathbf{L}_C^{\rho,in}$). Intuitively, given a contract $\mathcal{C}$, an occurring

function extracts, from a given initial sequence σ of transfers recorded on the ledger, the subsequence σ' of those relative to $\mathcal{C}$, and associates with it the maximal initial trajectory of states in $\mathcal{M}_{\mathcal{C}}^{l}$ (resp. $\mathcal{M}_{\mathcal{C}}^{e}$) reached through σ'.

Definition 6 (Occurring maps). *Let $\mathcal{C}$ be a contract. Then, for any $\sigma \in LT$:*

1. *Let $\overline{\tau_\sigma^l}$ be the maximal trajectory in $\Pi_{\mathcal{C}}^{l,in}$ for which there exists a monotonic function from $B_{\mathcal{C}}^{\beta}(\overline{\tau_\sigma^l})$ into σ. Then the map $\nu_{\mathcal{C}}^{l} : LT \to \mathbf{B}_{C}^{\beta,in}$, defined by $\nu_{\mathcal{C}}^{l}(\sigma) = B_{\mathcal{C}}^{\beta}(\overline{\tau_\sigma^l})$ is called the* bundle occurring map *for $\mathcal{M}_{\mathcal{C}}^{l}$.*

2. *Let $\overline{\tau_\sigma^e}$ be the maximal trajectory in $\Pi_{\mathcal{C}}^{e,in}$ for which there exists a monotonic function from $L_{\mathcal{C}}^{\rho}(\overline{\tau_\sigma^l})$ into σ. Then the map $\nu_{\mathcal{C}}^{e} : LT \to \mathbf{L}_{C}^{\rho,in}$, defined by $\nu_{\mathcal{C}}^{e}(\sigma) = L_{\mathcal{C}}^{\rho}(\overline{\tau_\sigma^e})$ is called the* transfer occurring map *for $\mathcal{M}_{\mathcal{C}}^{e}$.*

"Maximal" here means with respect to the ordering defined on trajectories, while the existence of a monotonic function from $B_{\mathcal{C}}^{\beta}(\overline{\tau^l})$ (from $L_{\mathcal{C}}^{\rho}(\overline{\tau^e})$) into σ implies that the trajectory $\overline{\tau_\sigma^l}$ ($\overline{\tau_\sigma^e}$) exploits the useful part of σ.

Theorem 1. *Both $\nu_{\mathcal{C}}^{l}$:$LT \to \mathbf{B}_{C}^{\beta,in}$ and $\nu_{\mathcal{C}}^{e}$:$LT \to \mathbf{L}_{C}^{\rho,in}$ are monotonic functions.*

Proof: To prove that $\nu_{\mathcal{C}}^{l}$ is a function, we need to prove that $\nu_{\mathcal{C}}^{l}(\sigma)$ is uniquely determined. Indeed, consider the first transfer in σ completing a bundle, say the bundle $\bar{b}$; then $\bar{b}$ is the first label in the bundle trajectory labeling for the (only) trajectory τ' consistent with σ. In the same way, proceeding in σ to record the bundles progressively completed by the transfers in σ (remember that a bundle can only be completed after reaching a state in which a transition it labels is allowed) one obtains the bundle trajectory-labeling of τ'. But now, due to the automaton determinism, $\tau' = \overline{\tau_\sigma^l}$. Hence we get a monotonic function from $B_{\mathcal{C}}^{\beta}(\overline{\tau_\sigma^l})$ into σ, which selects that part of σ which is the linearisation of the bundles in the bundle trajectory-labeling for $\overline{\tau_\sigma^l}$ (the useful subsequence). Since $\mathcal{M}_{\mathcal{C}}$ is deterministic, $\overline{\tau_\sigma^l}$ is uniquely determined. The proof of the monotonicity of ν^l is immediate: if σ increases, its useful part cannot decrease and it is possible either to reach a further state in the automaton (due to determinism, divergence is impossible) or to remain on the same state. Transfers discarded

in the procedure belong to the useless part of σ. The part of the proof relative to $\nu^e_{\mathcal{C}}$ is trivial, since this map directly relates two sequences of transfers. $\square$

Being ν^l and ν^e monotonic, a smooth correspondence is established between subtrees of LT and of $\mathbf{B}^{\beta,in}_C$, as well as between subtrees of LT and of $\mathbf{L}^{\rho,in}_C$. A monotonic function $\nu^l_e : \mathbf{L}^{\rho,in}_C \to \mathbf{B}^{\beta,in}_C$ also exists, factorising ν^e through ν^l.

In this model the tree associated with the ledger will also play the role of "making time tick" with each registration of a new transfer; the present instant at time t is represented by the evolution $\mathcal{E}_t$.

Let us investigate more formally the underlying theory: for a given $\mathcal{X}$ which is an object of **TSL**, the system $Subtree(\mathcal{X})$ is seen as a boolean algebra, obtained by equipping the subtrees of $\mathcal{X}$ with natural inclusions between them. Hence, we can think of a subtree as of the interpretation of some formula. In this acception, a subtree which is the interpretation of a formula ϕ is denoted by $[\![\phi]\!]$. Starting from atomic formulas, more complex ones are interpreted via the boolean operators present in $Subtree(\mathcal{X})$.

We thus obtain a notion of satisfiability, by a path π, of a logical formula ϕ:

- $\pi \models_{\mathcal{X}} \phi$ *iff* $\pi \in [\![\phi]\!]$[4]
- $\pi \models_{\mathcal{X}} \phi \wedge \psi$ *iff* $\pi \in [\![\phi]\!] \cap [\![\psi]\!]$
- $\pi \models_{\mathcal{X}} \phi \vee \psi$ *iff* $\pi \in [\![\phi]\!] \cup [\![\psi]\!]$
- $\pi \models_{\mathcal{X}} \phi \wedge \neg\psi$ *iff* $\pi \notin [\![\phi]\!]$.

This logical structure is inherited by every subtree, so that we can relativise the interpretation of a formula to any given subtree, by simply taking the intersection between the extension of the formula and the subtree.

Thus, we have a rigorous tool to vary our satisfiability relation according to the evolution of a ledger, because inclusions between subtrees smoothly translate satisfiability at step t to satisfiability at step $t' < t$.

With every instantaneous evolution $\mathcal{E}_t$ an atomic formula Φ_t is associated.

[4]Read: π satisfies ϕ if and only if π belongs to the interpretation of ϕ.

Such a formula is taken to mean "the ledger state appears in (belongs to) $\mathcal{E}_t$". For $\mathcal{E}$ an instantaneous evolution (i.e., a tree) and ϕ an atomic formula, $[\![\phi]\!]_{\mathcal{E}}$ denotes the subtree of $\mathcal{E}$ providing the interpretation of ϕ in $\mathcal{E}$.

For the set of logical operators above (as well as for the temporal ones later on) one might define a separate notion of satisfiability for every instantaneous evolution $\mathcal{E}_t$. However, since each $\mathcal{E}_t$-interpretation of a formula ϕ can be embedded into $\mathcal{E}_0$ by taking the $\mathcal{E}_0$ interpretation of $\phi \wedge \Phi_t$ (see Definition 7), we prefer to adopt this convention and avoid a useless proliferation of operators.

Definition 7 (Satisfiability in evolutions). *Let ϕ be a formula, and let σ be a ledger state in $\mathcal{E}_0$. We say that σ* satisfies *ϕ in $\mathcal{E}_0$, noted $\sigma \models_{\mathcal{E}_0} \phi$, according to the following, by induction on the structure of ϕ:*

- $\sigma \models_{\mathcal{E}_0} \phi$ iff $\sigma \in [\![\phi]\!]_{\mathcal{E}_0}$,
- $\sigma \models_{\mathcal{E}_0} \phi \wedge \psi$ iff $\sigma \models_{\mathcal{E}_0} \phi$ *and* $\sigma \models_{\mathcal{E}_0} \psi$;
- $\sigma \models_{\mathcal{E}_0} \phi \vee \psi$ iff $\sigma \models_{\mathcal{E}_0} \phi$ *or* $\sigma \models_{\mathcal{E}_0} \psi$;
- $\sigma \models_{\mathcal{E}_0} \phi \Rightarrow \psi$ iff $\sigma \models_{\mathcal{E}_0} \phi$ *implies* $\sigma \models_{\mathcal{E}_0} \psi$
- $\sigma \models_{\mathcal{E}_0} \neg\phi$ iff *it is not the case that* $\sigma \models_{\mathcal{E}_0} \phi$.

We say that σ satisfies *ϕ in $\mathcal{E}_t$ (i.e., at step t), noted $\sigma \models_{\mathcal{E}_t} \phi$, as follows:*

- $\sigma \models_{\mathcal{E}_t} \phi$ iff $\sigma \models_{\mathcal{E}_0} \phi \wedge \Phi_t$.

A similar logical structure can be produced for the meet-semilattices $\mathbf{B}_C^{\beta,in}$ and $\mathbf{L}_C^{\rho,in}$, so that all the logical operators considered in this section for LT, are also definable in $\mathbf{B}_{\mathcal{C}}^{\beta,in}$ and $\mathbf{L}_C^{\rho,in}$.

We now investigate the relationships between these algebraic structures and their associated logics. We recall that, for a tree $\mathcal{X}$ which is an object in **TSL**, $Subtree(\mathcal{X})$ is a boolean algebra. Theorem 2 unveils a more stringent structure.

Theorem 2. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two trees and let $\mu : \mathcal{X} \to \mathcal{Y}$ be a monotonic function. Then, the following hold:*

- *There exists a monotonic function* $\mu^{-1} : Subtree(\mathcal{Y}) \to Subtree(\mathcal{X})$[5].
- *The left and the right adjoints to* μ^{-1} *exist, namely existential and universal quantifier (*$\exists_\mu$ *and* $\forall_\mu$*).*
- μ^{-1} *preserves all algebraic operators.*

Proof:(Sketch)

- We define μ^{-1} as the function such that, for $\mathcal{Y}'$ a subtree of $\mathcal{Y}$, $\mu^{-1}(\mathcal{Y}') = \{\pi \in X \mid \mu(\pi) \in Y'\}$, which is a subtree of $\mathcal{X}$. Monotonicity is immediate.
- The operator $\exists_\mu$ is defined, for $\mathcal{X}'$ an object of **TSL**, by $\exists_\mu(\mathcal{X}') = \{\eta \in Y \mid (\exists \pi \in X')[\mu(\pi) = \eta]\}$. This is the left adjoint to μ^{-1}, while the operator $\forall_\mu$ is defined, for $\mathcal{X}'$ an object of **TSL**, by $\forall_\mu(\mathcal{X}') = \{\eta \in Y \mid (\forall \pi \in X)[\mu(\pi) = \eta \text{ implies } \pi \in X']\}$, which is the right adjoint to μ^{-1}.
- Preservation of operators is a consequence of having left and right adjoints.

As all items have been proven, this concludes the proof. □

Hence, we can smoothly translate formulas in one logic to formulas in the other one, the application of μ^{-1} maintaining their syntactical form.

As we define our logic on a tree-shaped model with a discrete structure, Definition 8 introduces the modal/temporal operators of interest, which complement the standard logical connectives.

Definition 8 (Modal/temporal operators). *Let* $\mathcal{X}$ *be a prefix-closed tree, let* $\leq$ *be the prefix relation between its paths and let* π *a path in* $\mathcal{X}$*. Then:*

- $\pi \models_\mathcal{X} \Diamond^u \phi$ iff $(\exists \pi' \in \mathcal{X})[\pi \leq \pi' \wedge \pi' \models_\mathcal{X} \phi]$. *In other words: "there is a future of* π *when* ϕ *becomes true".*
- $\pi \models_\mathcal{X} \Box^d \phi$ iff $(\forall \pi' \in \mathcal{X})[\pi \leq \pi' \Rightarrow \pi' \models_\mathcal{X} \phi]$. *In other words: "in all the possible futures of* π*,* ϕ *is true".*
- $\pi \models_\mathcal{X} \Diamond^d \phi$ iff $(\exists \pi' \in \mathcal{X})[\pi' \leq \pi \wedge \pi' \models_\mathcal{X} \phi]$. *In other words: "there is a past of* π *when* ϕ *was true".*

[5]The function μ is called a *substitution*.

- $\pi \models_{\mathcal{X}} \Box^u \phi$ iff $(\forall \pi' \in \mathcal{X})[\pi \leq \pi' \Rightarrow \pi' \models_{\mathcal{X}} \phi]$. *In other words: "for all pasts of* π ϕ *was true"*[6].

Using the strong partial order $<$ *canonically associated with the weak partial order* $\leq$*, we define the conditions for the relation* $Succ(\pi', \pi)$ *to hold as:* $\pi < \pi'$ *and there does not exist* π'' *such that* $\pi < \pi'' < \pi'$*. The following ensues:*

- $\pi \models_{\mathcal{X}} NXT_{\Diamond} \phi$ iff $(\exists \pi')[Succ(\pi', \pi) \wedge \pi' \models_{\mathcal{X}} \phi]$*, i.e., "there is an immediate future of* π *when* ϕ *becomes true".*
- $\pi \models_{\mathcal{X}} NXT_{\Box} \phi$ iff $(\forall \pi')[Succ(\pi', \pi) \Rightarrow \pi' \models_{\mathcal{X}} \phi]$*, i.e., "for all immediate futures of* π*,* ϕ *becomes true".*

Analogously, if we define that the relation $Pred(\pi', \pi)$ *holds if* $Succ(\pi, \pi')$ *holds, we have the following:*

- $\pi \models_{\mathcal{X}} PRV_{\Diamond} \phi$ iff $(\exists \pi')[Pred(\pi', \pi) \wedge \pi' \models_{\mathcal{X}} \phi]$*, i.e., "there is an immediate past of* π *when* ϕ *was true".*
- $\pi \models_{\mathcal{X}} PRV_{\Box} \phi$ iff $(\forall \pi')[Pred(\pi', \pi) \Rightarrow \pi' \models_{\mathcal{X}} \phi]$*, i.e., "for all immediate pasts of* π ϕ *was true"*

Theorem 3. *The temporal operators in Definition 8 enjoy the following algebraic properties:*

1. *They are all monotonic functions.*
2. *Squares and diamonds form temporal doctrines (see [3]). In other words, they are left (right) adjoint (and also left (right) inverse) to inclusions of up or down completion: namely*
   - $\Diamond^u \phi$ *corresponds to the minimal subobject containing* $[\![\phi]\!]$ *complete w.r.t. prefixes.*
   - $\Box^u \phi$ *corresponds to the maximal subobject contained in* $[\![\phi]\!]$ *complete w.r.t. prefixes.*

[6]Note that "future" and "past" correspond to a "d"-index (for down) and to a "u"-index (for up), respectively, for the $\Box$ operators, while it is the other way around for the $\Diamond$ operators.

- $\Diamond^d\phi$ *corresponds to the minimal subobject containing* $[\![\phi]\!]$ *complete w.r.t. prolongations.*
- $\Box^d\phi$ *corresponds to the maximal subobject contained in* $[\![\phi]\!]$ *complete w.r.t. prolongations.*

3. *Next and previous operators also form temporal doctrines: they correspond to left (right) adjoint to inclusions of next step or previous step completion.*

Proof: The results can be obtained routinely in analogy with those in [3]. □

Corollary 2. *All of the properties of the operators in Definition 8 (in particular their interrelations) are completely defined by their being associated with adjoint functions, hence uniquely determined.*

Now we are ready to show how to define a property in one tree also using the other one. To this end, we look at the interpretation in LT of the atomic formula corresponding to a property w.r.t. a given $\mathcal{M}^l_{\mathcal{C}}$.

- $\sigma \in LT$ is *bundle-complete* (noted $\sigma \models bc$) *iff* $\sigma' < \sigma \Rightarrow \neg(\nu^l(\sigma') = \nu^l(\sigma))$.
- σ is *contract-safe* if it satisfies $\Diamond^u bc$.

The first condition means that σ reaches a stable state of the automaton, while the second one means that σ does not contain any spurious transfer.

7. The associated language at work

The definition of the logical (modal/temporal) operators in Section 6 immediately lends itself to the definition of a query language, where a query is formulated in terms of paths in a suitable tree, satisfying a given formula. To start with, the language contains two atomic sentences, presented in Definition 9, to be interpreted in the structure LT.

Definition 9 (Atomic sentences). *Let LT be the tree corresponding to the evolution $\mathcal{E}_0$ for a given ledger built on an alphabet of transfers T. Then the following atomic sentences are defined by producing the respective interpretations.*

- *For any ledger state σ, we consider the property $\leq_t$, which is satisfied if $|\sigma| \leq t$. In LT this corresponds to identifying the subtree $\chi_t = [\![\leq_t]\!]_{LT}$ composed of all the sequences in LT which are not longer than t.*
- *For a transfer $\theta \in T$, the operator app_θ is interpreted in LT as the set of sequences in which θ appears*[7]*, i.e., it is the subtree $[\![app_\theta]\!]_{LT}$.*
- *For a transfer $\theta \in T$, the operator $app_{\theta,n}$ is interpreted in LT as the set of sequences in which θ appears in position n, i.e., as the subtree $[\![app_{\theta,n}]\!]_{LT}$.*

The following facts test the expressivity of the language built with the atomic sentences of Definition 9 and the operators of Definition 8.

Fact 2. *Given a set of ledger states LT for a given ledger and a contract $\mathcal{C}$, we can define the following subtrees as interpretations of the respective formulas (the first one refers to the ledger, the other ones to a contract):*

1. *The interpretation in LT of $appLst_{\theta,t} = \bigvee_{1 \leq n \leq t}(app_{\theta,n} \wedge \chi_n)$ is given by the tree of ledger states σ, of length at most t, with θ as last transfer in σ.*
2. *Given a bundle initial trajectory-labeling z_B in $\mathbf{B}_{\mathcal{C}}^{\beta,in}$, let ζ_B be the formula interpreted into the corresponding singleton. Then the set of bundle initial trajectory-labelings in $\mathbf{B}_{\mathcal{C}}^{\beta,in}$ such that they start with z_B, i.e. the set of prolongations of z_B, is the interpretation in $\mathbf{B}_{\mathcal{C}}^{\beta,in}$ of $\Diamond^d \zeta_B$.*
3. *Given a transfer initial trajectory-labeling z_L in $\mathbf{L}_{\mathcal{C}}^{\rho,in}$, let ζ_L be the formula interpreted into the corresponding singleton. Then the set of transfer initial trajectory-labelings in $\mathbf{L}_{\mathcal{C}}^{\rho,in}$ such that they start with z_L, i.e. the set of prolongations of z_L, is the interpretation in $\mathbf{L}_{\mathcal{C}}^{\rho,in}$ of $\Diamond^d \zeta_L$.*
4. *Given the translation function ${\nu_{\mathcal{C}}^l}^{-1}$ (resp. ${\nu_{\mathcal{C}}^e}^{-1}$) and z_B a labeling in $\mathbf{B}_{\mathcal{C}}^{\beta,in}$ (resp. z_L a labeling in $\mathbf{L}_{\mathcal{C}}^{\rho,in}$), the set of ledger states where a prefix of z_B (resp. z_L) has been performed reaching a "legal" state in $\mathcal{M}_{\mathcal{C}}^l$ (resp. a possibly "intermediate" state in $\mathcal{M}_{\mathcal{C}}^e$), is the interpretation in $\mathcal{E}_0$*

[7] Actually, the paths in LT contain encodings of transfers, but for the sake of simplicity in the rest of the section we identify the two notions.

of ${\nu^l_{\mathcal{C}}}^{-1}(\Diamond^d(\zeta_B))$ *(resp. of* ${\nu^e_{\mathcal{C}}}^{-1}(\Diamond^d(\zeta_L))$*).*

5. *The set of ledger states in LT, noted* $hol_{(r,k,t)}$*, such that, as a result of the transfers in this sequence, an agent* k *holds a resource* r *at time* t *(w.r.t. a contract* $\mathcal{C}$*), is the interpretation of the formula* $\Diamond^d(\bigvee_{k'} \bigvee_{t' \leq t} appLst_{(r,k',k),t'} \wedge \Box^d(\neg \bigvee_{k''} \bigvee_{t' \leq t'' \leq t} appLst_{(r,k,k''),t''}))$*. (That is to say: at time* t*, the token* r *is with the agent* k*, since at some previous time* t' *it was transfered to* k *from* k'*, and in no prolongation of the state* t'*, namely at* t'' *which is a prefix of the state at* t*, a transfer towards some actor* k'' *has been recorded.)*

We can now express in the language (and, possibly, use as axioms in deductions) the properties required in Section 3, and assume that the following formulas are always verified when interpreted in $\mathbf{B}^{\beta,in}_{\mathcal{C}}$:

- $\bigwedge_r \bigwedge_k \neg app_{(r,k,k)}$, meaning: "no transfer of the form (r, k, k) occurs";
- $\bigwedge_r \bigwedge_k \neg app_{(r,\perp,k)}$ meaning: "no transfer of the form $(r, \perp, k)$ occurs";
- $\bigwedge_r \bigwedge_k (app_{(r,k,\perp)} \Rightarrow \Box^d \bigwedge_{k'} \bigwedge_{k''} \neg app_{(r,k',k'')})$, meaning: "once $(r, k, \perp)$ has occurred, then no transfer for r can appear in any future".

Similarly, we can assume that the formula $\Phi_{t+1} \Rightarrow \Phi_t$ is always verified when interpreted in $\mathcal{E}_0$, thus formalising as an axiom of the logic the fundamental property of *immutability* of ledgers. Indeed, the formula expresses that each evolution at time $t + 1$ is contained in the evolution at time t, thus formalising the fact that "once a ledger state is reached, it is reached forever".

7.1. Formalising queries

We now give some examples of queries we can ask a ledger w.r.t. a contract. Let us assume that, when we write σ_i for a ledger state, we intend $|\sigma_i| = i$, i.e. σ_i satisfies $\chi_i \wedge \neg \bigvee_{0 \leq k \leq i-1} \chi_k$. Hence σ_i satisfies ϕ will mean that this fact happens at the instant i. Recall that an evolution $\mathcal{E}_j$ has only one state σ_i for $i \leq j$ (a past, established state) and only prolongations of σ_j as future states.

1. Suppose we start recording on a ledger the transfers relative to contract $\mathcal{C}_d$ of Example 1 performed according its associated contract execution automaton $\mathcal{M}^e_{\mathcal{C}}$ (with the useful part reaching a state in $\mathcal{M}^l_{\mathcal{C}}$). The encodings of transfers can be registered on the ledger at different (not necessarily consecutive) instants of time. We can then ask "which is the state of the legal contract (resp. contract execution) automaton for a state of the ledger σ_8, in an evolution $\mathcal{E}_n$, with $8 \leq n$?" The answer is given by evaluating the formulas $\exists_{\nu^e}(\Phi_n \wedge \chi_8)$, with respect to the contract execution automaton, and $\exists_{\nu^l}(\Phi_n \wedge \chi_8)$, with respect to the contract legal automaton.

2. A simple historical query made possible by our approach is the following: given an interval $[i \dots j]$ of time instants, and fixed an evolution $\mathcal{E}_s$, $j \leq s$, we can ask whether a formula ϕ is true for each ledger state σ_k with $k \in [i \dots j]$ in the evolution $\mathcal{E}_s$. A typical example for ϕ is: $hol_{(\mathit{Eiffel},a,all(X[i,j]))}$, meaning that "In the interval between i and j, Alice (represented as a) owns the non-fungible token certifying the property of Eiffel tower". Note that this notation seems to use a new universal quantification, but it is actually syntactic sugar: the query can be rolled out into a finite conjunction of queries the form $hol_{(\mathit{Eiffel},a,i)} \wedge \dots \wedge hol_{(\mathit{Eiffel},a,j)}$, as we have made in other cases. Hence, the query is answered by the satisfiability of $\phi \wedge \Phi_s$.

3. Many variations on this theme are possible. For example: given σ_i, we could verify whether, in the future of it there will be a state σ_j verifying ϕ, provided that, in its past the series of transfers $\theta_1, ..., \theta_k$, at times $j_1 \leq ... \leq j_k$, has been recorded. This is equivalent to verifying the formula: $\Diamond^u(\Diamond^d(\bigwedge_{1 \leq n \leq k} app_{(\theta_n, j_n)}) \Rightarrow \phi)$.

4. A further variation is a form of hypothetical reasoning: given an evolution $\mathcal{E}_i$ and a formula ϕ which is not true in $\sigma_i \in \mathcal{E}_i$, is there a prefix σ_k of σ_i such that ϕ could be true in some future of σ_k? In this case, we are dealing with a ledger state σ_i in the interpretation of $\neg\phi \wedge \Diamond^d\Diamond^u\phi \wedge \Phi_i$.

5. A simple, but interesting, type of query is about the possibility of verifying, at given instant i, the presence of sequences of transfers registered on the ledger. Without loss of generality, we consider sequences composed of two transfers: $\langle\theta_1\theta_2\rangle$. Then the occurrence of such a sequence on a state is expressible by the formula: $\Diamond^d(app_{(\theta_2,n)} \wedge \Diamond^d(app_{(\theta_1,n')}))$. Queries of this kind can be structured in more complex ones like $\Diamond^d(app_{(\theta_2,n)} \wedge \Diamond^d(app_{(\theta_1,n')})) \wedge \Diamond^u(\Diamond^d(app_{(\theta_2,m)}) \wedge \Diamond^d(app_{(\theta_1,m')}))$, looking for repetitions of the same sequence later in the time. One could weaken the condition that the transfers occurring in the two sequences are ordinately equal, by imposing only a certain kind of similarity: e.g., the same actors, or the same resource, etc.

Some remarks are in order.

Technically, the construction of the interpretation of the formula in Item 3 is equivalent to a "temporal projection via regression": transfers are not executed, but consequences of their hypothetical execution are verified. In practice, temporal regression queries must be handled adequately to ensure that meaningful answers return and unnecessary computational burdens are avoided. In fact, in themselves they are subject to infinite answers: think of the case in which, starting from a state of affairs in which Bob is in possession of a token that certifies his ownership of the Colosseum and Alice is in possession of a token that certifies her ownership of the Eiffel tower, we want to verify the possibility of switching ownerships, with Bob and Alice taking possession, respectively, of the French Belle Epoque building and of the Roman era stadium. One way for this to happen is by exchanging tokens, if contracts have been defined that enable such exchanges; but this may happen an arbitrary number n of times, by sending them back and forth for cases greater than 1, while one is presumably interested solely in the case $n = 1$. In a concrete ledger management system, managing queries of this type could be delegated to the user, by allowing her to place limits on the time j_k when the hypothesized situation occurs in the future, or to the system, by constraining it to return only the shortest answer.

Queries of the type discussed in Item 5 are particularly useful and interesting because they can act as tools for auditing and process mining on the activities of the ledger. For example, they could be used to verify the repetition over time of the sending of quantities of money from Bob to Alice and then from Alice to George, which could be indicative of a laundering activity. Or they could identify the repetition, for instance, of supplies of kitchen plinths from company A to company B, which in turn uses them to assemble kitchen furniture which then routinely supplies to company C that produces turnkey kitchens, fully furnished and equipped with appliances; these repetitions may suggest connecting the three companies in a single supply chain through optimized revenue sharing contracts as illustrated in [7, 8].

8. Discussion and related work

Not long after its inception, database technology produced several temporal data models [9, 10, 11]. However, these models apply to data, like bank accounts and document versions, produced in application contexts where time is treated as an add-on to basic data models bereft of a time dimension. By contrast, in ledgers time is not an option, but stands out as a fundamental, albeit implicit, aspect in the overall functioning of the system. Hence, our approach differs substantially from models such as those mentioned above precisely because we treat time as a universal factor independent of specific data types, starting from the assumption that all data in a ledger is set in a time dimension.

Much more recently,the works in [12] and [13] describe systems that use relational database technologies to extract information from the Ethereum and Bitcoin blockchains by leveraging the available meta-information, such as hash numbers of blocks and transactions. Thus, while having significant practical usability, they are very coarse-grained and very specific to the environments they are meant for, hence they do not provide for generalizable data models with a built-in management of time.

Quite closer to our approach is the treatment of time in the Situation Cal-

culus [14, 15], a formal framework that provides a general theory of action by viewing the world as a succession of situations in consequence of the execution of admissible actions. It has in this sense influenced our elaboration of similar formal and computational tools to support querying in ledgers. On the other hand, the primary applications of Situation Calculus are in the areas of knowledge representation for planning by artificial agents, thus requiring a rich vocabulary made up of fluent predicates describing various aspects of the world relevant for the agents' actions. This results into "frame problems" and "ramifications problems" in the update of such representations that need to be tackled whenever even seemingly minimal changes in the state of affairs take place. By contrast, our lean representation structure links actors and resources through a single graph, thus staying at large from laborious update processes. At a logical level, the Situation Calculus can be reconstructed through modal/temporal logics [16, 17] bearing a number of resemblances with, and sharing characteristics of, the logics used here to provide the basis to query past, present and future state of affairs of the ledger. In particular, the modal/temporal logic versions of the Situation Calculus share with the approach we have presented here the management of state of affairs as "possible worlds" rather than as reified entities referred to through second-order variables as in the original version of the Situation Calculus, with considerable simplifications in the technical treatment of the underlying semantics. Temporal and dynamic logics are used in [18, 19, 20] to address dynamic aspects of the blockchain, orthogonal to those discussed here, such as, respectively, the guarantees given by the validation protocols to agents involved in contractual transactions and the management of provisional blocks.

The relationships between legal contracts, smart contracts, transactions and blockchains are wide and varied and unfold from a multiplicity of applications and of disciplinary contexts. Long before blockchains became commonly used technologies, and quite independently of them, the concept of smart contract was introduced by Nick Szabo in the 1990s [1]. The focus and fundamental motivation for this conceptualization stems from the observation made by Szabo that the contracts of the legal tradition, especially those pertinent to the

commercial sphere, establish that transactions, in the commercial sense of the term, must be carried out in conjunction with given events, such as the payment deadline for a rent, mortgage or leasing contract. In the treatment given by Szabo, a contract can therefore be seen as the automation of the transactional part of a legal contract, with the guarantee that the transactions will be effectively carried out, as well as the ability to automatically manage infringements of the agreed conditions without having to resort straight away to legal proceedings - for example by automatically sending a warning letter in the case of an overdue payment, or by electronically locking access to rented premises in case of accumulation of non-payments. With this conceptualization of smart contracts, Szabo aimed to increase the efficiency in the execution of the underlying legal agreements, as well as to decrease legal disputes arising from any infringements. However, time was not yet ripe for the concept to pass into practice and implementation, and it will take more than two decades for Szabo's pioneering contribution to gain light and consideration.

In the meantime, the idea was born of transferring the technology of advanced DBMS transactional models beyond the fences of the single company, so as to leverage them to support business eco-systems made possible by the concurrent advent of technological trends such as Web services, service-oriented architectures and, generally, of distributed systems hinging on cross-entity computer interactions. Early attempts to adapt the most advanced of such models, based on the properties of Atomicity, Consistency, Isolation and Durability (ACID), to implement complex e-commerce transactions spanning multiple businesses are described in [21, 22, 23]. However, substantial difficulties faced by such models were rooted in the fact that the short-lived nature of ACID transactions as conceived for single corporate domains did not fit well with the longer-life requirements of multi-business transactions. Alternatives to the timed-out resource-locking of traditional ACID models were thus introduced, in particular compensating transactions [24], although these too proved insufficient to make the envisaged business model fly, in the absence of a trustable infrastructure for cross-enterprise transaction execution. Meanwhile, a transaction model such as

BASE (Basic Availability, Soft-state, Eventual consistency)[8], characterized by weaker constraints with respect to ACID, was successful in the practice of distributed applications such as social networks and e-commerce stores, that can be handled without taking care of demanding contractual requirements.

But the coming of age of blockchains showed that the last word had yet to be said about the feasibility of bringing together the world of commercial contracts and of digital transactions. Indeed, second-generation blockchains like Ethereum revived and brought all the previous attempts together, by leveraging the digital trust afforded by blockchain technology to resurrect and implement Szabo's proposal. Yet, here too there are a number of weaknesses to address, as smart contracts implemented in Ethereum and other blockchains [25] bring together basic transactions but are not inherently transactional, in the sense of being bound by a well-defined and verifiable set of properties that define the effects of their execution on the database, while they are instead treated as programming constructs in dedicated or conventional programming languages. Specialised languages and frameworks for smart contracts include Solidity on Ethereum[9] and private distributed ledgers maintaining records of all the performed (atomic) transactions like Fabric from the Hyperledger software ecosystem [10]. As these languages and frameworks usually lack a proper formal foundation, but may, on the other hand, be Turing-complete [26], problems arise with respect to verification, both of their behavioural properties and of the conformity between a contract's textual definition and its programmatical implementation. This has led to a number of problems like the already mentioned notorious concurrency bug that affected the DAO smart contract through which Ethereum itself was funded [27]. By contrast, our formalization of a contract as an automaton forces a rigorous division of its various execution phases, so as to provide a declarative model transparent to aspects of access to resources. Fur-

[8] www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing

[9] https://docs.soliditylang.org/en/latest/#,https://ethereum.org/en/

[10] https://www.hyperledger.org/use/fabric, https://www.hyperledger.org

thermore, the ability, intrinsic to our formalization, to impose interdependencies between the various transactions that make up a smart contract substantially reproduces the atomic characteristics of the ACID paradigm, without however binding them to the release and roll-back of resources after time-out, which would disagree with contexts of use populated by actors that can be flexible on their times to respond to requests.

At the same time, our model is sufficiently general to be independent from any implementation infrastructure (also the original formulation by Szabo abstracted from specific execution models), and yet compatible with various deployments on ledgers, distributed ledgers and blockchains. Finally, it could also be synthesized from optimization procedures as in the case of the "intelligent smart contracts" for innovative supply chain management illustrated in [7, 8] with implementation, respectively, in a public blockchain (Ethereum) and a private distributed ledger (Hyperledger Fabric).

Another aspect for which the implementation of smart contracts as programs is unsatisfactory is that the relationship with legal contracts is substantially weakened, if not lost altogether, a relationship whose reconstruction is a specific objective of this work. In this sense, there are parallels and convergence with other efforts in a similar direction. The work in [28] first recognized that to make a legal contract computer-executable requires the availability of suitable formal models for specifying its intended (legal) semantics. As concerns this latter point, two proposals, both named Contract Automata and based on similar principles, are contained in [29] and [30].

In [29], states of the automaton correspond to sets of *permissions* and *obligations* to execute some actions. A state transition occurs when a permitted action in a state is performed, giving rise to a new configuration of permissions/obligations. The main goal is to verify the soundness of the contract, for example to check that any action for which there is an obligation in a given state is also permitted in that state. The treatment is restricted to two-party contracts, so that a state is annotated with a four-tuple of sets of deontic clauses, describing, for each party, what its permitted actions and its obligations are.

No explicit consideration is given to the resources needed to perform the action, as the assumption is that if a part p has permission to perform an action a, p will succeed in performing a, if it tries. The authors suggest that the possibility of checking the actual feasibility of a could be modeled by introducing a different action $attempt\_a$ which then becomes the object of the permission. This extension could open the way for an explicit discussion of the conditions, i.e., of the resources needed, for an attempt to be successful.

The model in [30] considers multi-party contracts, where each participant is modeled by an automaton, and a contract describes the conditions under which transitions in the different automata can be synchronised. Here, each party *offers* or *requests* the execution of some action, and synchronisation is achieved when requests by one party are matched by offers from another one. In this case, offers and requests may concern resources produced by a participant and consumed by another one. Synchronisation can also be weakened, provided that requests and offers are matched in the end, thus introducing a form of credit, to be honoured before the end of the contract. Various logics for reasoning on contracts are then derived and compared.

Finite-state machines are also at the basis of VeriSolid [31, 32], a tool to specify contracts to be deployed on Ethereum. The generated contracts are secure by design, as they do not allow some sequences of actions (invocations of a Solidity function), thus reducing the expressiveness of Solidity. Plugins can impose further constraints, for instance requiring that functions are invoked according to the linear order provided by some shared program counter.

These models do not consider resources in the definition of state, and, lacking a notion of transaction as set of actions, transitions occur on single actions.

A connection between the notion of contract (actually, the almost equivalent notion of *policy*) and the notion of resource has been drawn in [33]. There, policies are defined in terms of admissible paths in the space state of some actor subject to the policy. State transitions have to be synchronised with corresponding transitions in the state of some resource necessary to realise the transition (where the resource state typically describes the availability of the resource for

some actor). This is achieved by annotating states in a policy with resources, and expressing synchronisation through constraints on valid annotations. The model of allocations of resources to actors adopted in this paper can be seen as equivalent to the form of annotation adopted there, especially since allocation systems fully support the notion of state for both resources and actors.

9. Conclusions

The approach to reasoning on evolutions of ledgers introduced in this paper starts from a formalisation of actions required by a contract in terms of transfers of tokens representing specific resources between specific actors; the recording of these transfers on ledgers marks the advancements in the contract's execution. This provides the basis for the construction of computational solutions that appear advantageous and appropriate for the evolution of the technological paradigm to which these constructs are associated. These solutions have been expressed at a level of maximum generality and abstraction so as not to place restrictions on their transferability to a wide variety of concrete implementation contexts. Specifically, we have shown that the resulting notion of ledger state is compatible with query mechanisms that can be translated into a modal/temporal logic where it is possible to query the present, the future and the past of the ledger, as well as its "present perfect" and "nearest future", and also to reason counterfactually with respect to what would have happened if the course of things, that is, of transactions, had been different. Maximum flexibility is thus guaranteed in the auditing of activities on the ledger. Moreover, the formal characterization of contracts as automata allow us to treat them as advanced transactional models, rather than as programs, as is the case in the current practice of smart contract implementations. This model reproduces some of the properties of the well-established ACID model in the context of ledgers and contracts, such as consistency, atomicity in the sense of interdependence between transactions, declarativeness in the access to resources used for the completion of contractual obligations. It therefore generalizes, at an ab-

stract and formal level, indications and proposals for an effectively transactional treatment of smart contracts, which is desirable in order to avoid programming errors that have plagued their practice in the past, sometimes with disastrous financial consequence, and to boost effectiveness in contract execution. The two contributions come together, as the query logic can be used to audit the progress and actual execution of contracts built on the basis of these formal criteria.

Acknowledgements

Work partially supported by Sapienza, project "Consistency problems in distributed and concurrent systems". We thank the anonymous referees for indication on how to improve this paper.

References

[1] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9).

[2] V. Buterin, Ethereum whitepaper (2013).
URL `https://ethereum.org/en/whitepaper/`

[3] P. Bottoni, D. Gorla, S. Kasangian, A. Labella, A doctrinal approach to modal/temporal heyting logic and non-determinism in processes, Mathematical Structures in Computer Science 28 (4) (2018) 508–532.

[4] P. Bottoni, D. Gorla, S. Kasangian, A. Labella, Modal epistemic logic on contracts: A doctrinal approach, in: Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday, Vol. 11665 of LNCS, Springer, 2019, pp. 298–314.

[5] P. Bottoni, A. Labella, Transactions and contracts based on reaction systems, Theoretical Computer Science (2021) 1–50`doi:10.1016/j.tcs.2021.07.012`.

[6] S. Kasangian, A. Labella, Observational trees as models for concurrency, Mathematical Structures in Computer Science 9 (6) (1999) 687–718.

[7] P. Bottoni, N. Gessa, G. Massa, R. Pareschi, H. Selim, E. Arcuri, Intelligent smart contracts for innovative supply chain management, Frontiers in Blockchain 3 (2020) 52.

[8] P. Bottoni, N. Gessa, G. Massa, R. Pareschi, D. Tortola, Distributed ledgers to support revenue-sharing business consortia: a hyperledger-based implementation, in: Proc. BRAIN 2021, 2021, to appear.

[9] R. T. Snodgrass, Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, 1999.

[10] R. T. Snodgrass, The TSQL2 Temporal Query Language, Kluwer, 1995.

[11] A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, R. T. Snodgrass, Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993.

[12] S. Bragagnolo, H. Rocha, M. Denker, S. Ducasse, Ethereum Query Language, in: Proc. WETSEB@ICSE 2018, ACM, 2018, pp. 1–8.

[13] K.-B. Yue, K. Chandrasekar, H. Gullapalli, Storing and querying bitcoin blockchain using SQL databases 17 (2019) 24–41.

[14] J. McCarthy, P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. L. Webber, N. J. Nilsson (Eds.), Readings in Artificial Intelligence, Morgan Kaufmann, 1981, pp. 431–450.

[15] R. Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, The MIT Press, 2001.

[16] J. van Benthem, McCarthy variations in a modal key, Artificial Intelligence 175 (1) (2011) 428–439.

[17] G. Lakemeyer, The situation calculus: A case for modal logic, Journal of Logic, Language, and Information 19 (4) (2010) 431–450.

[18] J. Y. Halpern, R. Pass, A knowledge-based analysis of the blockchain protocol, Electronic Proceedings in Theoretical Computer Science 251 (2017) 324–335.

[19] B. Marinkovic, P. Glavan, Z. Ognjanovic, T. Studer, A temporal epistemic logic with a non-rigid set of agents for analyzing the blockchain protocol, J. Log. Comput. 29 (5) (2019) 803–830.

[20] K. Brünnler, D. Flumini, T. Studer, A logic of blockchain updates, Journal of Logic and Computation 30 (8) (2020) 1469–1485.

[21] J. Andreoli, S. Freeman, R. Pareschi, The coordination language facility: Coordination of distributed objects, Theory Pract. Object Syst. 2 (2) (1996) 77–94.

[22] J. Andreoli, F. Pacull, R. Pareschi, XPECT: A framework for electronic commerce, IEEE Internet Comput. 1 (4) (1997) 40–48.

[23] J. Andreoli, F. Pacull, D. Pagani, R. Pareschi, Multiparty negotiation of dynamic distributed object services, Sci. Comput. Program. 31 (2-3) (1998) 179–203.

[24] R. Karlsen, T. Strandenæs, Trigger-based compensation in web service environments, in: Proc. ICEIS, 2003, pp. 487–490.

[25] M. Garriga, S. Dalla Palma, M. Arias, A. De Renzis, R. Pareschi, D. A. Tamburri, Blockchain and cryptocurrencies: A classification and comparison of architecture drivers, Concurrency and Computation: Practice and Experience e5992 (2020) 1–21.

[26] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, M. Imran, An overview on smart contracts: Challenges, advances and platforms, Future Generation Computer Systems 105 (2020) 475–491.

[27] Q. DuPont, Experiments in algorithmic governance: A history and ethnography of "The DAO", a failed Decentralized Autonomous Organization, in: M. Campbell-Verduyn (Ed.), Bitcoin and Beyond, Routledge, 2017, pp. 157–177.

[28] D. Magazzeni, P. McBurney, W. Nash, Validation and verification of smart contracts: A research agenda, IEEE Computer 50 (9) (2017) 50–57.

[29] S. Azzopardi, G. J. Pace, F. Schapachnik, G. Schneider, Contract automata - an operational view of contracts between interactive parties, Artif. Intell. Law 24 (3) (2016) 203–243.

[30] D. Basile, P. Degano, G. L. Ferrari, Automata for specifying and orchestrating service contracts, Log. Methods Comput. Sci. 12 (4) (2016) 1–51.

[31] A. Mavridou, A. Laszka, Designing secure Ethereum smart contracts: A finite state machine based approach, in: S. Meiklejohn, K. Sako (Eds.), FC 2018, Revised Selected Papers, Vol. 10957 of LNCS, Springer, 2018, pp. 523–540.

[32] A. Mavridou, A. Laszka, E. Stachtiari, A. Dubey, Verisolid: Correct-by-Design smart contracts for Ethereum, in: I. Goldberg, T. Moore (Eds.), FC 2019, Revised Selected Papers, Vol. 11598 of LNCS, Springer, 2019, pp. 446–465.

[33] P. Bottoni, A. Fish, A. Heußner, F. Parisi-Presicce, Resource-aware policies, J. Vis. Lang. Comput. 38 (2017) 84–96.