Evaluation of Logic-Based Smart Contracts for Blockchain Systems

Florian Idelberger[1(✉)], Guido Governatori[2(✉)], Régis Riveret[2(✉)], and Giovanni Sartor[1(✉)]

[1] European University Institute, Fiesole, Italy
{florian.idelberger,Giovanni.Sartor}@eui.eu
[2] Data61 - CSIRO - NICTA, Brisbane, Australia
{guido.governatori,regis.riveret}@data61.csiro.au

Abstract. While procedural languages are commonly used to program smart contracts in blockchain systems, logic-based languages may be interesting alternatives. In this paper, we inspect what are the possible legal and technical (dis)advantages of logic-based smart contracts in light of common activities featuring ordinary contracts, then we provide insights on how to use such logic-based smart contracts in combination with blockchain systems. These insights lead us to emphasize a fundamental challenge - algorithms for logic approaches have to be efficient, but they also need to be literally cheap as measured within the environment where they are deployed and according to its economic rules. We illustrate this with different algorithms from defeasible logic-based frameworks.

Keywords: Smart contract · Blockchain · Programming paradigm · Logic

1 Introduction

A smart contract is a computer program that both expresses the contents of a contractual agreement and operates the implementation of that content, on the basis of triggers provided by the users or extracted from the environment. Smart contracts are currently promoted as means to leverage efficiency, security and impartiality in the execution of an agreement, thereby reducing the costs in implementing contracts and increasing trust between parties.

While imperative languages, especially procedural languages, are mostly used to code smart contracts in current blockchain platforms, declarative languages for such contracts, and in particular logic-based rule languages, should also be considered to better represent and reason upon them, towards a concept that we may call *declarative smart contracts*, in particular the concept of *logic-based smart contracts*.

Combinations of logic frameworks and blockchain systems may lead to smart contracts that are easier to work with for jurists and developers and have technical advantages over procedural coding of the contracts. These combinations may also lead to new opportunities for applications for these logic frameworks.

J.J. Alferes et al. (Eds.): RuleML 2016, LNCS 9718, pp. 167–183, 2016.
DOI: 10.1007/978-3-319-42019-6_11

In this paper, we investigate the utility of logic-based smart contracts and possible ways to use them in combination with blockchain systems:

- to understand what legal and technical (dis)advantages logic-based smart contracts can provide w.r.t. their procedural counterparts, we structure this investigation in light of a common contract lifecycle;
- to show how logic-based smart contracts can be used in combination of blockchain systems, we inspect different combinations for leveraging logic-based languages to operate smart contracts in combination with such systems.

These insights will lead us to emphasize a foundational challenge to fully take advantage of logic-based smart contracts with blockchain systems: algorithms for logic approaches have to be efficient, but they also need to be literally cheap in execution. Since logic models of defeasible reasoning are often advocated to capture legal knowledge and reasoning (see e.g. [16]), we will illustrate our discourse with defeasible rules and associated logic frameworks.

This paper is organised as follows. In Sect. 2, we outline some basic elements and mechanisms of blockchain systems. In Sect. 3, we define and illustrate logic-based smart contracts and in Sect. 4 we examine the possible legal and technical utility of such logic-based smart contracts compared to procedural smart contracts, and we do so in light of common legal activities. In Sect. 5, we investigate different options for the operation of smart contracts in combination with blockchain systems, before concluding.

2 Blockchain Systems

A blockchain system consists of a network of computing nodes, sharing a common data structure (the blockchain) with consensus about the state of this structure.

The most prominent example of such a system is Bitcoin [13], which established a distributed network of accounts and transactions (a ledger), where revisions or tampering is made prohibitively difficult due to the algorithm used in conjunction with economic consensus. Since Bitcoin is the most prominent example, most explanations regarding blockchain systems below will be based on this system; the functioning of different blockchains may differ in detail but such differences fall outside of the scope of this paper.

The data structure backing a blockchain system is distributed because it is replicated amongst the nodes (i.e. computers) of the system. As new blocks of recent transactions are added to the distributed data structure, they include a reference back to the previous blocks, so that any node can consequentially verify the integrity of the data structure. This chain of blocks of transactions is called a blockchain.

Transactions on the blockchain are not cost-free. Miners have to spend computing power (tied to hardware) and energy to integrate blocks of transactions into the blockchain. As incentives, if a miner ‘discovers’ the solution of the problem to include a block, such miner receives economic incentives in the form of

new Bitcoins (block reward) and transaction fees. It is presently unclear how this system will function once the algorithmically predetermined number of Bitcoins has been reached.

The transaction fee is an incentive for a miner to include this transaction in their block. For advanced blockchain systems, the fee *may* also cover the cost of the computational steps required to operate the transaction, in particular when the transactions are associated with extra instructions. The computation of the amount of the fee is outside of the scope of this paper, but as rule of thumb, the simpler a transaction in terms of computational complexity, the less it costs.

Since transactions can be costly, it is often advanced that heavy computation should occur 'off-chain' instead of 'on-chain'. In off-chain scenarios, computation is performed outside the blockchain-based system, e.g. on the server of an intermediation service, while, in on-chain scenarios, computation is performed and validated in the blockchain-based system by the miners. Of course, off-chain computation results can be recorded in a blockchain, however parties may prefer to avoid off-chain intermediation services that can be performed on-chain, for example to increase trust.

While blockchain technology was initially used as a distributed ledger of crypto-currency transactions (namely Bitcoin transactions), such a technology can also be used to manage smart contracts and associated transactions.

3 Logic-Based Smart Contracts

The term *smart contract* was originally introduced in the 90s by Szabo [17], stemming from the idea that a more technological legal framework would help commerce and cut down on disputes. Lately the idea came to popularity again with the rise and expanding capabilities of blockchain based systems. Parts of a *smart contract* can correspond to a legal contract or a clause in a legal contract, but they do not have to.

When there is a condition to which certain legal consequences are attached, the *smart contract* executes the corresponding statements and any potential contractual consequences. Examples for applying *smart contracts* are programmatic banking functions (see e.g. Automated Escrow, Savings), decentralized markets (e.g. OpenBazaar, EtherMarket), prediction markets (Augur, Gnosis), distribution of music royalties (Ujo) and encoding of virtual property (Ascribe).

Smart contracts in blockchains are typically programmed in a *procedural* language. On the platform Ethereum [5,18], developers can encode smart contracts in a procedural language called Solidity[1]. When programming in a procedural language, the programmer writes an explicit sequences of steps that are executed to produce what has to be done. The programmer has to write *what* has to be done and *how* to achieve it.

Example 1. This example is based on the structure of the example provided in [7] to illustrate some intricacies of the logical formalisation of legal reasoning

[1] Solidity. Available at https://ethereum.github.io/solidity.

Let us consider the following licensing contractual clauses for the evaluation of a product.

Article 1. The Licensor grants the Licensee a licence to evaluate the Product.
Article 2. The Licensee must not publish the results of the evaluation of the Product without the approval of the Licensor; the approval must be obtained before the publication. If the Licensee publishes results of the evaluation of the Product without approval from the Licensor, the Licensee has 24 h to remove the material.
Article 3. The Licensee must not publish comments on the evaluation of the Product, unless the Licensee is permitted to publish the results of the evaluation.
Article 4. If the Licensee is commissioned to perform an independent evaluation of the Product, then the Licensee has the obligation to publish the evaluation results.
Article 5. This license will terminate automatically if Licensee breaches this Agreement.

Suppose that the licensee evaluates the product and publishes on their website the results of the evaluation without having received an authorisation from the licensor. The licensee realises that they were not allowed to publish the results of the evaluation, and they remove the published results from their website within 24 h from the publication. Is the licensee still able to legally use the product? Since the contract contains a compensatory clause, it is possible to argue that the license to use the product still holds. Suppose now that the licensee, right after publishing the results, posted a tweet about the evaluation of the product and that the tweet counts as commenting on the evaluation. In this case, we have a violation of Article 3, since, even if the results were published, according to Article 2 the publication was not permitted. Thus, they are no longer able to legally use the product under the term of the license. The final situation we want to analyse is when the publication and the tweet actions take place after the licensee was commissioned to perform an independent evaluation from the licensor. In this case, the licensee has the obligation to publish the result, which then means that they were also permitted to publish the result, and thus they were free to post the tweet. Accordingly, they can continue to use the product under the terms of the licence. □

Algorithm 1 gives a pseudo-code example of how a procedural smart contract can implement the contractual clause of Example 1. The smart contract includes a sequence of instructions updating the normative states (obligations, prohibitions and permissions in force) depending on what actions have been done and then the current state. The program has to set the initial state for the contract, then the procedure EvaluationLicenseContract has to be invoked every time there is a trigger for the program. Notice that the order of the instructions in the procedure does not reflect the natural order of the contract clauses expressed in natural language. The programmer has to come up with such an order, and also the programmer has to manually determine how a trigger changes

Algorithm 1. Pseudo-code of the licensing contractual clauses.

```
Initialise getLicence, getApproval, getCommission, use, publish, comment, remove
[Forb_licensee] use ← true
[Forb_licensee] publish ← true
[Forb_licensee] comment ← true
violation ← false

procedure EvaluationLicenseContract
    if getLicence = true then
        [Forb_licensee] use ← false
        [Perm_licensee] use ← true                                   ▷ Article 1

    if getLicence = true and (getApproval = true or getCommission = true) then
        [Forb_licensee] publish ← false
        [Perm_licensee] publish ← true                               ▷ Article 2, 4

    if getLicence = true and
       getApproval = false and
       getCommission = false and
       publish = true then
        [Obl_licensee] remove ← true                                 ▷ Article 2

    if [Perm_licensee] publish = true then
        [Forb_licensee] comment ← false
        [Perm_licensee] comment ← true                               ▷ Article 3

    if getLicence = true and getCommission = true then
        [Forb_licensee] publish ← false
        [Obl_licensee] publish ← true
        [Perm_licensee] publish ← true                               ▷ Article 4

    if ([Forb_licensee] use = true and use = true) or
       ([Forb_licensee] publish = true and publish = true) or
       ([Obl_licensee] publish = true and publish = false) or
       ([Forb_licensee] comment = true and comment = true) or
       ([Obl_licensee] remove = true and remove = false) then
        violation ← true
    if violation = true then
        [Forb_licensee] use ← true
        [Forb_licensee] publish ← true
        [Forb_licensee] comment ← true
        [Perm_licensee] use ← false
        [Perm_licensee] publish ← false
        [Perm_licensee] comment ← false
        [Obl_licensee] publish ← false                               ▷ Article 5
```

the state of the normative provisions (i.e., obligations, permissions and prohibitions), and to propagate the changes according to the meaning. This means that the programmer is responsible to perform the legal reasoning implied by the contract clauses. For example, when a permission becomes true, the corresponding prohibition should be set to false; similarly, when we set an obligation as true, the corresponding permission should be set to true as well. For large and complex smart contracts, an alternative is to set an auxiliary procedure to be invoked, when the state of a normative provision has to be changed, and propagate the changes to all related normative provisions.

The process of writing a procedural program corresponding to a contract can be cumbersome and error prone since the order of instruction affects the correctness of the resulting smart contract. A possible solution to alleviate this problem is to create a state machine for the contract (Fig. 1 shows a state machine for the contract in Example 1). Then, the programmer can use the state machine as a guide to derive the procedural code. Alternatively, the state machine could be represented directly in the program and a state machine engine could then be used to execute the resulting smart contract. This approach can grow exponentially large in the number of states and transitions for non-trivial contracts, and the programmer still remains in charge of the legal reasoning implied by the contract.

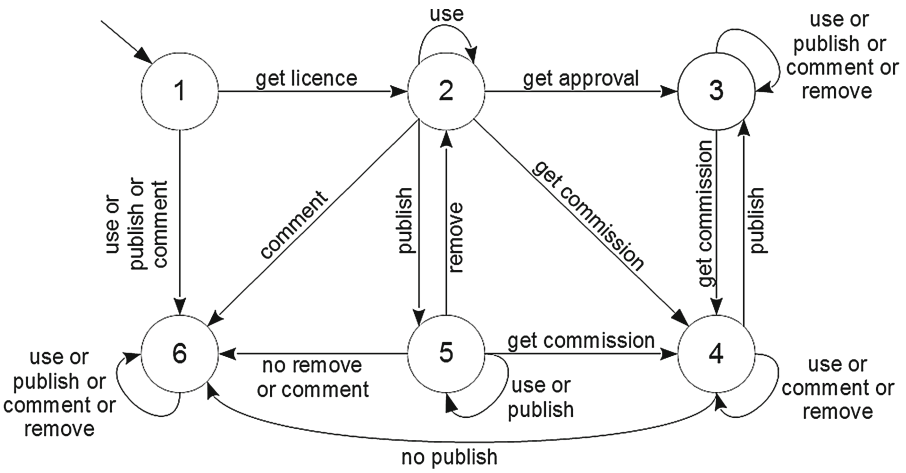

Fig. 1. State machine of the licensing contractual clauses.

Besides imperative languages for smart contracts, one may consider *declarative* languages (in particular logic-based languages). When programming in a declarative language, the programmer does not have to explicitly write the sequence of steps to produce what has to be done. Instead the programmer describes what has to be done, but not how to do it. In particular, languages

for logic programming can be used to represent and reason upon the rules represented by smart contracts. With the logic approach, contractual clauses are rephrased into explicit formal statements which are separated from the embedding program, and the program has inferential functionalities to reason upon these statements. In practice, the contractual clauses would be encoded into logic rules, and a rule-based engine would reason upon the rules.

Example 2. Since logic-based models of defeasible reasoning are often advocated to capture legal knowledge and reasoning (see e.g. [16]), let us consider the representation of the contract given in Example 1 provided by the (deontic) defeasible logic (Formal Contract Logic, FCL) of [6] and implemented by the defeasible logic engine SPINdle [10].

```
Article1.0: => [Forb_licensee] use
Article1.1: getLicense => [Perm_licensee] use
Article2.1: => [Forb_licensee] publish [Compensated] [Obl_licensee]remove
Article2.2: getApproval => [Perm_licensee] publish
Article3.1: => [Forb_licensee] comment
Article3.2: [Perm_licensee] publish => [Perm_licensee] comment
Article4.1: getCommission => [Obl_licensee] publish
Article4.2: getCommission => getLicense
Article5: violation => [Forb_licensee] use
% Superiority relation
Article1.1 > Article1.0,  Article5 > Article1.1,
Article2.2 > Article2.1,  Article3.2 > Article3.1
```

The order of the rules is irrelevant, and as should be visible, the declarative rules are shorter than procedural code and easier to use, they would later then be evaluated by a rule engine on the blockchain or deployed in conjunction with a rule engine.

If there are no triggers, then `Article1.0`. `Article2.1` and `Article3.1` fire and we conclude the prohibitions of `use`, `publish` and `comment`. When, `getLicense` and `publish` hold, then `Article1.1` overrides `Article1.0` thus we have the permission of `use`, but we continue to have the prohibition to `publish`, thus the publication contravenes Article 2, and we can use rule `Article2.1` to derive the mandated compensation, that is the obligation of removing the material is now in force, i.e., we conclude `[Obl_licensee] remove`. See [6] for the details of FCL. □

4 Utility of Logic-Based Smart Contracts

The successes of blockchain-based systems for smart contracts, or at least the amounts of investments in such systems suggest the viability of 'procedural smart contracts', while the utility of logic-based smart contracts has been hardly investigated.

In this section, we consider the utility of the logic approach w.r.t. its procedural counterpart. Too bridge the gap between smart contracts and legal contracts, this is done in the light of the lifecycle of a contract.

Formation and Negotiation. Based on the 'freedom to contract', any legal entities are free to form contracts, within the limits of the law. Necessary conditions for the formation are the 'meeting of the minds' (i.e. the parties have the intentions to form the contract) and the 'Offer and Acceptance' (i.e. the expression of an offer to contract on certain terms by one person to another person, and its acceptance of those terms). In practice, parties often negotiate the terms until they reach an agreement.

As any ordinary contracts, a smart contract can be negotiated i.e. the smart contract program is coded, and this creation can occur through a negotiation. In a blockchain system, agreement about what a contract should perform is defined before deploying the contract in the blockchain system. After creation and giving assent by calling the contract in the required way, the contract establishes legal relations between the parties. Often, a contract is first created in a natural language (as in the case of the creation using a template), and then this contract is translated into a smart contract. However, a smart contract program can be created without a natural language counterpart, the same as normal computer programs.

Using procedural languages, fairly sophisticated smart contracts can be formed already. However, the procedural coding of a smart contract may appear difficult to apprehend, slowing down its negotiation and formation. As the procedural code may appear difficult to understand, one can wonder whether the contractual clauses are properly coded. In this regard, the procedural code can be 'validated' (unit testing etc.) to determine whether this smart contract is fit for use, but testing procedural code is well-known to be time consuming and error prone. In logic-based smart contracts, as logic statements can be understood as high-level specifications, they constitute executable specifications of smart contracts, i.e. specifications that can be directly executed by the smart contracts, thereby decreasing the risks of errors in the implementation. Moreover, a logic representation can ease validation by taking advantage of logic-based techniques, such as formal verification, to detect if certain properties hold. This can be automated, but since such techniques are often heavy in terms of computation, they will most likely occur off-chain. Furthermore, a logic representation may ease the formation of a smart contract resulting from a negotiation between parties. When the formation and negotiation are delegated by humans to artificial agents, the logic approach may particularly facilitate these activities (w.r.t. a procedural counterpart) since in this case such activities require, presumably, some artificial intelligence to represent and reason upon contractual terms.

Contract Storage/Notarizing. A contract can be binding in many forms, such as by oral agreement, hand shake or intangible agreement. Thus, in principle there are little formalities required (though exceptions apply). The real problem arises when there is contention on whether there was a contract or not, and what its contents were. In those cases, it helps to have a written record of what was agreed stored and certified. To be extra certain, contracts can be certified by a trusted third party, a notary. For non-digital contracts, the content has to be described in natural language and a date manually inserted.

Contract storage can be straightforwardly related to the storage of smart contracts using file systems or database systems. Instead of storing the smart contract into the machine(s) of particular entities (such as the parties and intermediaries), one can use a blockchain system to store it (its bytecode) with a relatively accurate timestamp.

There are no particular restrictions on the types of data that can be stored in blockchains, and therefore smart contracts with logic statements can be stored in them. As logic statements (e.g. the set of rules stored within a procedural smart contract and meant to be passed to a rule engine) are generally more compact than its procedural counterpart, the logic approach may decrease the cost of storage, in particular when there is an explosion of possible states on which rules can be applied.

Enforcement and Monitoring. Once a contract is formed, it has to be performed; the parties have to take appropriate actions to fulfil the contractual clauses. If parties are encouraged or forced to perform their required actions, this is called enforcement for the purposes of this paper.[2] Monitoring is the activity of checking whether the appropriate actions are taken. Enforcement and monitoring can be described as the deployment and the execution of a program, which can to some degree be automated by the blockchain consenus code.

The efficient execution and monitoring of a smart contract is a necessary condition for the use of such a contract, in particular in regard to the worst-case scenarios. While the computational complexity of the execution of a procedural smart contract can be quite easily controlled, the complexity of a logic-based smart contract relies on the complexity of the underlying inference mechanisms (we will further instigate this point in the next section). Concerning monitoring, 'controls' can be typically integrated in the procedural code of a smart contract, while in logic contracts, monitoring can take advantage of more formal run-time compliance techniques. Furthermore, the execution and monitoring of a contract is not necessarily meant to occur in isolation. On the contrary, when executing smart contracts, contractual clauses may have to be considered w.r.t. exogenous (legal) information, such as rules from other contracts or the embedding normative environments (the law in particular). While procedural smart contracts can interact with each other rudimentarily, a logic approach would take advantage of efforts in rule interchange languages (such as LegalRuleML [2]) to express rules and ease interoperability amongst the contracts and other rule systems.

Modification. If all parties perform their contractual duties, then a contract may in principle not be unilaterally modified. If a party fails to perform or if a predetermined condition in the contract is activated, then a change in the contractual relationship can be invoked. If all parties to a contract agree to a change, the contract can be amended accordingly.

These considerations for non-smart contracts also hold for smart contracts. In current blockchain systems, a contract cannot be modified but the data stored in it can be updated. As such, one model to enable flexible solutions is the 'hub

[2] While encouragement is not enforcement in all meanings of the word, it is either a precursor or a part of it.

and spoke' model where one main smart contract holds addresses/pointers to all other necessary contracts that contain the specific clauses and functionality.

The hub and spoke model allows the modification of smart contracts, but it may appear quite coarse. In logic-based smart contracts, the statements of the knowledge base can be coded as 'public' variables, thus allowing more fine-grained updates. A modified knowledge base can also be passed to an existing contract, which then acts accordingly, similar to how in the hub and spoke model addresses of subcontracts are exchanged. Moreover, the order of instructions and procedures is fundamental in the procedural approach (as illustrated in Algorithm 1), and thus the hub and spoke model may cause some issues in that regard. As the order of the statement in a knowledge base does not matter w.r.t. the conclusions that can be derived from it, a logic-based language can greatly help to tackle modifications.

Dispute Resolution. A dispute regarding a contract may occur, and thus such a dispute has to be resolved. Two major types of dispute resolution exist: (i) adjudicative resolution, such as litigation or arbitration, where a judge, jury or arbitrator determines the outcome, and (ii) consensual resolution, such as collaborative law, mediation, conciliation, or negotiation, where the parties attempt to reach agreement.

Smart contracts can be disputed too, and adjudicative resolution as well as consensual resolution can be attempted. The final arbiter of legal technological innovation is always acceptance by the courts. At the moment there is no useful case law on this for smart contracts, but this would also depend strongly on the nature of the smart contract, i.e. whether it is linked to a contract in natural language as well as other factors. In principle, based on Bitcoin case law and the freedom to contract, it can be said that smart contracts are binding [19, pp. 11–24].

With regard to a consensual resolution, a smart contract could specify a committee of human or computational arbitrators that should be consulted first. It is unclear at present how a court would interpret such a choice of law or arbitration clause in a smart contract.

In principle smart contracts can be considered to be legally valid (exceptions notwithstanding); to this end, it likely does not matter if the smart contract is programmed using an imperative or a declarative language. Nevertheless, one may argue that, as some imperative code (and, to a lesser extent, some procedural code), may be difficult to comprehend, it may be the case that the control structures of these smart contracts rebut jurists and hamper their interpretation of the contract (this would lead to the emergence of case law setting precedent on how to interpret smart contracts; however so far this does not exist). On the contrary, as logic rules are meant to reflect contractual clauses, their logic representation will ease the work of jurists, in particular to structure, evaluate, and compare legal arguments constructed from formal statements. However, if there are legal rules that a human has to be told to what he agrees to, there has to be a natural language equivalent anyway. Then the logic rules might make the implementation or the interpretation of the contract easier, but they may not

be close enough to natural language to be a substitute, particularly to people who might not be technical experts.

In summary, the logic approach has the potential to advantageously complement its procedural counterpart for each activity thereof. Whilst advantages are clearly backed by technical considerations, it is less evident whether a logic approach provides a stronger legal foundation to smart contracts. As previously alluded to, one may argue that a full representation of a smart contract has to explicitly establish and link the normative effects (rights, obligation, transfers of entitlement) resulting from the contract, and the procedure for implementing these rights and obligations though the computational actions performed by the contract, in the given infrastructure. Thus, a hybrid approach combining logic and procedural components may help to bridge the gap between smart contracts and their legal counterparts.

5 Use of Logic-Based Smart Contracts with Blockchain Systems

In this section, we investigate different technical options to use logic-based smart contracts in combination with blockchain systems, and we will discuss these options w.r.t. the legal activities we previously identified.

Given a set of statements, inferences can be performed in different manners. Every inferential mechanism has its own characteristics, and the adoption of a particular mechanism to execute logic contractual clauses should be based on these characteristics.

Example 3. Considering a defeasible logic framework for the representation of the contractual clauses; conclusions can be derived by using dialectic proofs (DPs) [14] or an algorithm based on the fixed-point of the characteristic function of the grounded semantics [4] (FP), see e.g. [12], more efficient algorithms stemming from Defeasible Logic (DL) [1,11] or even equation-based approaches (EB), see e.g. [15] and neuro-symbolic systems (NS), see e.g. [3]. In most cases, it is preferable to use the algorithm with the lowest computational complexity, but for some reasons, one may prefer other algorithms to provide, for example, more intelligible inferences. How to use these mechanisms to deal with smart contracts in blockchain-based systems? □

Beside the characteristics of the inferential mechanisms, it is important to notice that inferences can occur on-chain or off-chain.

1. On-chain: inferences are made within the blockchain platform;
2. Off-chain: inferences are made outside the blockchain system, e.g. on a third party server.

The distinction of on-chain and off-chain inferences leads us to distinguish off-chain options for logic-based smart contracts and on-chain options.

5.1 Off-Chain Options

When miners are processing transactions into blocks to append to the blockchain, the security model of the virtual machine in which smart contracts on existing blockchain platforms operate and the co-processing by nodes does not allow to call outside resources. Thus, we must discard the option where an off-chain inferential mechanism is called by the smart contract.

Though an off-chain inferential mechanism cannot be called from a smart contract, another off-chain option simply consists in recording the smart contract (i.e. knowledge base and the reference to the semantics) and the inferential conclusions in the blockchain. On the basis of the inferential conclusions, procedural code of the contract can then execute particular transactions. Activities that we identified in the previous section are accommodated as follows.

Formation and negotiation. The contract can be formed and negotiated off-chain or on-chain.

Contract storage/notarizing. A contract is stored off-chain (so that it can be executed off-chain) and in the blockchain.

Enforcement and monitoring. Enforcement and monitoring are achieved off-chain, the conclusions can be stored in the blockchain.

Modification. If a contract is modified, then the off-chain smart contract will be updated, and stored in the blockchain. If the knowledge base can be updated, then the smart contract can be updated without interrupting associated processes.

Dispute resolution. One can check whether an off-chain contract matches a blockchain code (bytecode). Thus in case of a dispute, the parties can check whether the recorded conclusions are proper conclusions of the smart contract (w.r.t. the given semantics).

The main advantage of this off-chain option is the lower cost of associated transactions, since the inferences are performed off-chain. The disadvantage is that such an off-chain inferential mechanism may be simply seen as an intermediary service, while the parties may prefer to avoid such intermediation and associated costs or trust issues.

5.2 On-Chain Options

Instead of an off-chain inferential mechanism, one may prefer an on-chain mechanism. The availability of a logic-based language to program smart contracts shall facilitate such options, but a procedural language can also be used to write meta-programs (i.e. programs with the ability to treat programs as their data). For example, a rule-engine can be integrated in a smart contract to derive some conclusions given a particular knowledge base. Based on the results, some procedural code can execute the transactions. The rule-engine can also be a smart contract script of its own, so that smart contracts can always refer to this smart contract. Having the inference engine as an immutable contract on the blockchain

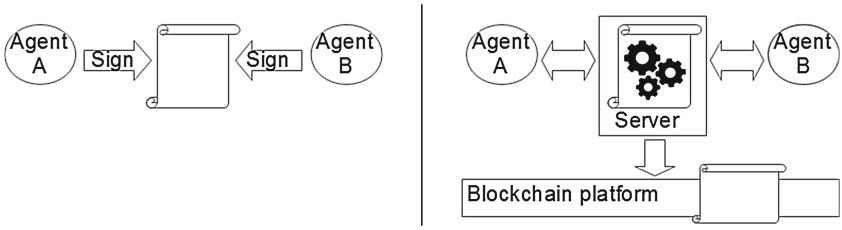

Fig. 2. Off-chain option. Agents A and B form a (smart) contract which is stored on a blockchain. The contract is executed in a server external to the blockchain system, and transactions can be recorded in the blockchain.

allows participants' confidence into the smart contract engine to increase over time (test once, utilize n-fold).

Formation and negotiation. The contract can be formed and negotiated off-chain or on-chain.

Contract storage/notarizing. A contract can be stored off-line, but it has to be stored in the blokchain (so that it can be executed on-chain).

Enforcement and monitoring. Enforcement and monitoring is achieved on-chain, the conclusions can be stored in the blockchain.

Modification. If the knowledge base can be updated, then the contract can be updated without interrupting associated processes.

Dispute resolution. One can check whether an off-chain contract matches a blockchain code. Thus in case of a dispute, the parties can check whether the recorded conclusions are proper conclusions of the smart contract (w.r.t. the given semantics).

The major advantages of on-chain solutions is that some off-chain intermediation services are eliminated, and the inferential mechanisms (e.g. the rule engine) are themselves recorded in the blockchain, resulting into more scrutinizable and trustful inferences.

The main disadvantage of on-chain solutions may regard the costs. To address the costs of on-chain inferences, algorithms with low computational complexity shall be favoured. If the selected algorithm provides inferences which appears sufficiently efficient but insufficiently intelligible for human operators, then more intelligible inferences can be used to explain the results off-chain.

Example 4. Considering DPs, FP or DL algorithms for the on-chain option, DPs have higher complexity than FP algorithms, which have higher complexity than algorithms from DL [8]. Consequently, one shall prefer DL algorithms to derive conclusions on-chain. □

Interestingly, it is also possible to propose an on-chain option, that we may call the 'on-off' option where, given a knowledge base, this knowledge is converted (let's say 'compiled') into a lower-level representation to increase the speed of

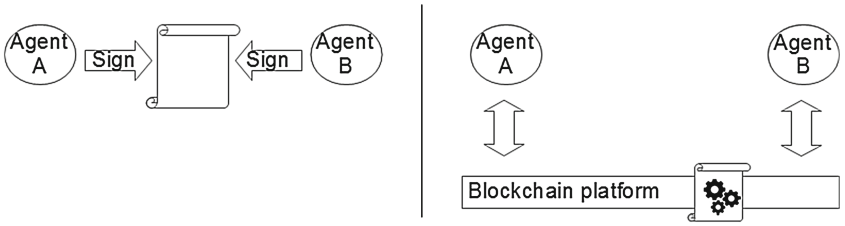

Fig. 3. On-chain option. Here, agents A and B form a (smart) contract which is stored and executed in a blockchain platform.

inferential computation, and this compiled code is part of the smart contract (this smart contract is eventually recompiled to run on the virtual machines of the blockchain network).

Formation and negotiation. The contract can be formed and negotiated off-chain or on-chain. The compiled code can be generated off-chain or on-chain. If compilation occurs off-chain then third party services may again appear, along with the associated disadvantages. If compilation is done on-chain then the compiler may be scrutinised and gain trust from the parties, at the expense of extra costs for compilation.

Contract storage/notarizing. A contract and its compiled code can be stored off-chain, but the compiled code has to be stored in the blockchain (so that it can be executed on-chain).

Enforcement and monitoring. Enforcement and monitoring is achieved on-chain, the conclusions can be stored in the blockchain.

Modification. If a contract is modified, then the logic statements have to be recompiled. If the compiled knowledge base can be updated, then the contract can be updated without interrupting associated processes.

Dispute resolution. One can check whether the compiled off-chain contract matches a blockchain code. Thus in case of a dispute, the parties can check whether the recorded conclusions are proper conclusions of the smart contract (w.r.t. the given semantics).

Compared to the off-chain option, the need for intermediation services is mitigated since inferences are achieved on-chain. Compared to the on-chain option, the costs of transactions may be decreased because the compiled knowledge base is meant to lower the computational complexity. The costs will be presumably higher than the off-chain option, therefore, such on-off approaches shall have a cost intermediate between off-chain and on-chain solutions.

Example 5. EB and NS approaches can be considered for 'on-off' solutions. In the EB approach, the considered knowledge base is 'compiled' into a set of equations, and these equations are stored into the smart contract to compute conclusions given a set of facts. In the NS approach, the knowledge base is 'compiled' into a neural network instead. While such approaches are interesting,

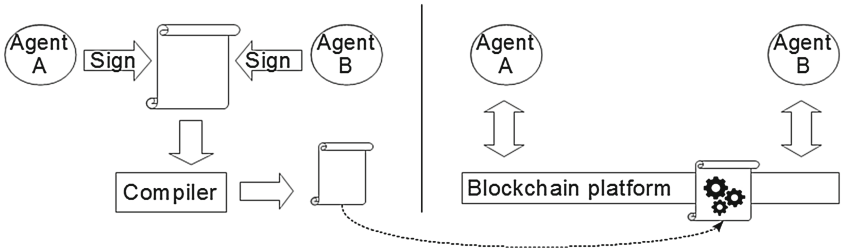

Fig. 4. On-off option. Agents A and B form a (smart) contract which is compiled. The compiled contract is stored and executed in a blockchain platform.

they may be quite limited in terms of expressiveness; for example we know neither EB nor NS approaches able to deal with temporal aspects for any defeasible rule-based framework, whereas there are works introducing temporal extensions to DL, see e.g. [9]. □

Whatever the option, and as previously mentioned, verification of the conclusions should be possible, and understandable by humans. In this regard, given some semantics, the conclusions of efficient but unintelligible approaches can be verified off-chain by more comprehensible proof systems.

Example 6. DPs clearly provide more intelligible proof systems for human operators w.r.t. other solutions. Hence, one may use efficient algorithms such as DL algorithms for routine operations, and human operators can rely on DPs to verify results if necessary. □

So, if comprehensible proof systems are available for the considered logic-based system, then the choice of the option to execute a logic-based smart contract in combination with a blokchain system largely depends on the costs of such execution. As revealing experiment, we compared the cost of the procedural code (PC) for a modus ponens inference (from the premises 'a' and 'if a then b', then we derive b) with a rule reduction as used in a reasonably efficient algorithm for DL, and with an EB approach. The estimated cost for PC was 1480, 11859 for DL and 1418 for an EB approach.[3] For a simple modus ponens inference, the reduction rule was thus approximately 8 times more costly than the two other approaches. This result suggests that blockchain systems bring a new important technical challenge which is hardly addressed by the community: algorithms for a logic approach will not only be required to be efficient, but they also are required to be cheap as measured within the environment where they are deployed and according to its economic rules.

[3] This comparison was conducted by writing the basic solidity code for the requisite modus ponens inference and then comparing the 'gas' cost as estimated by the official solidity compiler.

6 Conclusion

While procedural languages are commonly used to program smart contracts in blockchain systems, logic-baed languages have been hardly explored. For this reason, we investigated the utility and possible ways to use logic-based smart contracts with such systems. We structured this investigation in light of a common contract lifecycle. We have shown that a logic approach can advantageously complement its procedural counterpart w.r.t. the negotiation, formation, storage/notarizing, enforcement, monitoring and activities related to dispute resolution.

To show how logic-based smart contracts could be used, we inspected different combinations for leveraging logic programming languages to operate smart contracts with such blockchain systems, and we illustrated our discourse with different algorithms from defeasible logic frameworks. This led us to emphasize a fundamental challenge to fully take advantage of a combination of logic-based smart contracts and blockchain systems: algorithms for logic-based approaches have to be efficient and cheap as measured within the environment where they are deployed and according to its economic rules, to ensure feasability in an environment where economic governance and consensus is used to ensure a working system and abuse prevention.

Finally, we have to emphasize that the logic and procedural approaches are not incompatible, on the contrary, they have the potential to advantageously complement each other. By providing a declarative specification of the content of the contract, to be complemented with a procedural definition of the steps needed to perform the obligations in the contract — either automatically or through specification introduced by the parties — more clarity is established, and a criterion is provided for matching automatic execution and shared intention of the parties, as expressed in the declarative specification.

Acknowledgements. NICTA is funded by the Australian Government through the Dept of Communications and the Australian Research Council through the ICT Centre for Excellence Program.

References

1. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Trans. Comput. Log. **2**(2), 255–287 (2001)
2. Athan, T., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: LegalRuleML: design principles and foundations. In: Faber, W., Paschke, A. (eds.) Reasoning Web 2015. LNCS, vol. 9203, pp. 151–188. Springer, Heidelberg (2015)
3. d'Avila Garcez, A.S., Gabbay, D.M., Lamb, L.C.: A neural cognitive model of argumentation with application to legal inference and decision making. J. Appl. Log. **12**(2), 109–127 (2014)
4. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995)

5. Ethereum Foundation. Ethereum's white paper
6. Governatori, G.: Representing business contracts in RuleML. Int. J. Coop. Inf. Syst. **14**(2–3), 181–216 (2005)
7. Governatori, G.: Thou shalt is not you will. In: Atkinson, K., (ed.) Proceedings of the Fifteenth International Conference on Artificial Intelligence and Law, pp. 63–68. ACM, New York (2015)
8. Governatori, G., Pham, D.H.: DR-CONTRACT: an architecture for e-contracts in defeasible logic. Inter. J. Bus. Process Integr. Manag. **5**(3), 187–199 (2009)
9. Governatori, G., Rotolo, A., Riveret, R., Palmirani, M., Sartor, G.: Variants of temporal defeasible logics for modelling norm modifications. In: Proceedings of the 11th International Conference on Artificial Intelligence and Law, Stanford, California, USA, pp. 155–159. ACM (2007)
10. Lam, H.-P., Governatori, G.: The making of SPINdle. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 315–322. Springer, Heidelberg (2009)
11. Maher, M.J.: Propositional defeasible logic has linear complexity. Theor. Pract. Log. Program. **1**(6), 691–711 (2001)
12. Modgil, S., Caminada, M.: Proof theories and algorithms for abstract argumentation frameworks. In: Simari, G., Rahwan, I. (eds.) Argumentation in Artificial Intelligence, pp. 105–129. Springer, Heidelberg (2009)
13. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008). (The Nakamoto paper)
14. Prakken, H., Sartor, G.: A dialectical model of assessing conflicting arguments in legal reasoning. Artif. Intell. Law **4**(3–4), 331–368 (1996)
15. Riveret, R., Rotolo, A., Sartor, G.: Probabilistic rule-based argumentation for norm-governed learning agents. Artif. Intell. Law **20**(4), 383–420 (2012)
16. Sartor, G.: Legal Reasoning: A Cognitive Approach to the Law. Springer, Heidelberg (2005)
17. Szabo, N.: The idea of smart contracts (1997)
18. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2014)
19. Wright, A., De Filippi, P.: Decentralized Blockchain Technology and the Rise of Lex Cryptographia. SSRN Scholarly Paper ID 2580664, Social Science Research Network, Rochester, NY, March 2015