Building Trust in Smart Legal Contracts

Alessandro Parenti and Marco Billi

A. Introduction

Since its birth, the interest around blockchain technologies has experienced a continuous growth. Especially in the last 2–3 years, this sector received high fundings by many venture capital firms and other investors (23 billion in 2021 for the whole sector).[1] We can mention two domains above all that gained the most success worldwide: the NFTs market, where the new interest for digitalized ownership drew investments in the millions for many projects,[2] and Decentralized Finance (DeFi).

Both technologies, like many others within the blockchain sector, are built upon software running a distributed ledger, a distributed, immutable database, on which information can be registered, such as *smart contracts*.

The term "smart contract" was originally coined by computer scientist and cryptographer *Nick Szabo* in 1994 as "a computerized transaction protocol that executes the terms of a contract".[3] He explains his idea by bringing forward the example of a vending machine: the vending machine is programmed to automatically perform (dispense the product) when certain conditions are met (a coin is inserted). This mechanism has the advantages of removing the need for intermediaries thus reducing transaction costs and making breaches of contract expensive or non-convenient (the cost of breaching the machine would likely be higher than the amount in the till). As we can notice also from the wording, *Szabo*'s idea of smart contract was closely related to the legal domain.

After the 90s, the term smart contract remained unused for more than 15 years, also because of the lack of technologies capable of fully realizing this theoretical idea in all its features. In 2014, however, *Vitalik Buterin*

---

1 *Team Blockdata*, The 10 biggest funding rounds in blockchain / crypto ever, 2021, available at https://www.blockdata.tech/blog/general/top-10-funding-rounds-in-blockchain-crypto

2 Although lately is has seen a decline in interest (last access: 05.09.2022).

3 *N. Szabo,* Smart Contracts, 1994, available at https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html (last access: 05.09.2022).

used *Szabo*'s idea when he published Ethereum's whitepaper. One of the main innovative features of *Buterin*'s blockchain enabled the implementation of smart contracts on it.

The concept that originated from Ethereum smart contracts, however, diverges from *Szabo*'s original idea in that it loses the link with the legal field. *Buterin* refers to smart contracts as "cryptographic 'boxes' that contain value and only unlock it if certain conditions are met".[4] Therefore, for today's understanding, we need to bear in mind *Stark*'s[5] distinction between a *smart contract* (code), which simply refers to a piece of software running on a blockchain that self-executes its code once certain conditions (pre-defined inside it) get satisfied; and a *smart legal contract*, i.e., a smart contract used to represent and automatically execute an agreement enforceable by law.

As we will explain throughout the present paper, a smart legal contract, given its intrinsic features, raises an explainability issue, especially because of the potential impact that such a tool could directly have on people's personal legal sphere.

Traditionally, as far as the field of xAI (explainable AI (Artificial Intelligence)) is concerned, the focus is on providing the user with a rationale behind the outcome of AI system, often seen in decision support systems and other user-oriented applications. There are two main ideas in this research field. The first focuses on building additional systems that mimic the original, showing to which extent the model and/or its predictions are human understandable.[6] The two most common methods to achieve this are (1) by either creating a second model that provides a global explanation to the opaque system, achieving transparency and interpretability, (2) explaining only the reason for the prediction on a specific instance, basically providing a justification for the outcome of the black box considering a pair of input and decision.[7] The second research current, instead of

---

4 *V. Buterin*, Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform, 2014, available at https://ethereum.org/en/whitepaper/ (last access: 05.09.2022).

5 *J. Stark*, Making Sense of Blockchain Smart Contracts, Coindesk.Com, 2016, available at https://www.coindesk.com/making-sense-smart-contracts (last access: 05.09.2022).

6 *R. Guidotti/A. Monreale/ S. Ruggieri/ F. Turini/ F. Giannotti/ D. Pedreschi*, A Survey of Methods for Explaining Black Box Models, in ACM Computing Surveys 2019, 1 (1 et seqq.).

7 *H. Prakken/R. Ratsma*, Case-based argumentation for explanation: formalisation and experiments, Argument and Computation 2021.

an ex-post explanation, tries to embed in the original system a symbolic representation of the knowledge-base.

With regard to smart contracts, we depart from this distinction, as explainability may be replaced with intelligibility. The parties of a contract may not require, strictly speaking, an "explanation" behind what the contract does, instead the focus shifts to describing, in computable and human terms, what the contract states. The step of contract creation can be quite unclear for non-software developers as it occurs in a programming language, aiming at guaranteeing the automated execution of the contract.[8] This paper aims at providing a solution to help these contracting parties communicate in a common language, understandable to both users and programmers alike.

In Smart Legal Contract research, there are two main currents that try to solve these issues from different points of view. The most classical *hybrid* approach is Ricardian Contract,[9] which places all information from the legal document in a format that can be executed by software. The code and the legal prose are then connected through the use of parameters, keeping the two separate and connected at the same time.

Our contribution aims to move beyond this approach, through the so-called *standalone* approach, using a sole source to represent both the contract and the code. Particularly, we shall focus on (1) what legal concepts can be successfully represented using a domain-specific language and (2) whether it is possible to isomorphically translate a legal contract into code using a general-purpose language.

B. Smart Legal Contracts

As we mentioned above, when we use smart contract technology for the purposes of representing legal agreements between two or more parties enforceable by law, we refer to Smart Legal Contracts. For purposes of completeness, we shall also say that, just like any other contract, they must satisfy a certain number of conditions that are usually laid down by the relevant legal provisions, in order to be considered valid by the legal system and thus to be enforceable before a court of law.

---

8 *P. Qin/W. Tan/J. Guo/B. Shen*, Intelligible Description Language Contract (IDLC) – A Novel Smart Contract Model, Information Systems Frontiers 2021, available at https://link.springer.com/article/10.1007/s10796-021-10138-4 (last access: 13.10.2022).

9 *I. Grigg*, 2004; *I. Grigg,* 2017.

In literature we can find different approaches to the use of this techno-logy. First, we shall highlight the distinction defined by the ISDA-Link-laters Whitepaper[10] between external and internal model of smart contract. In the former, code is not part of the legal contract and is employed simply as a means for the execution of some parts of it. The latter refers to a *"legal contract rewritten in a more formal representation than the current natural human language form. A computer would then take that more formal representation and execute the conditional logic automatically."* In this model the code is a necessary part of the contract, as agreed and signed by the parties.

Furthermore, we can distinguish three ways to implement internal mo-dels of smart contracts, taking into account the approaches identified by *Clack*.[11] First, it is possible to link the written contract with its associa-ted code through the use of markup languages, which have the task of annotating certain parts of the code and the text, providing a direct link between the two. This approach is represented by the concept of Ricardian Contracts[12] introduced by *Grigg* in 1996 and by all the evolutions that stemmed from it. These include Smart Contract Templates,[13] Intelligible Contracts[14] and the Accord Project.[15] We will not go into further detail, as this approach falls outside the scope of this contribution.

The other two approaches are (1) domain specific programming langua-ges and (2) controlled natural language. The former helps towards the creation of a single artefact expressing both the contractual obligations and the computer code (in the light of the concept of *Computable* Contracts).[16] Moreover, thanks to its specificity, it is better suited for expressing and identifying aspects (e.g., payments, assets, and logic) that are proper to the legal domain of smart legal contracts, thus facilitating the activity of contract drafting. This approach was particularly explored for the financial

---

10 *ISDA/Linklaters*, Whitepaper: Smart Contracts and Distributed Ledger-A Legal Perspective, 2017, available at https://www.isda.org/2017/08/03/smart-contracts-and-distributed-ledger-a-legal-perspective/ (last access: 05.09.2022).

11 *C. D. Clack*, Languages for Smart and Computable Contracts, 2021.

12 *I. Grigg*, The Ricardian Contract, 2021, available at https://iang.org/papers/ricardian_contract.html (last access: 05.09.2022).

13 *H. Haapio/J. Hazard*, Wise contracts: smart contracts that work for people and machines, in: Trends and communities of legal informatics. Proceedings of the 20th international legal informatics symposium IRIS, 2017, 425 (425 et seqq.).

14 *L. Cervone/M. Palmirani/F. Vitali*, Intelligible Contracts, 53rd Hawaii International Conference on System Sciences, 2020, 1780 (1780 et seqq.).

15 Available at https://www.accordproject.org/ (last access: 05.09.2022).

16 *H. Surden*, Computable Contracts, UC Davis Law Review 2012, 629 (626 et seqq.).

sector: we can mention the *Marlowe*[17] language for the Cardano block-chain and *Goodenough et. al.*[18] with a state-transition system to represent a financial contract.

The controlled natural language approach aims at making lawyers, and non-technical people in general, comfortable in the use of the programming language, thus contributing to the creation of the smart contract code directly, or at least in their ability to understand the written contract. Furthermore, this would help remove the error-prone step of manual conversion from natural language to a specification or programming language. An example of this approach is the Lexon language.[19]

C. Intelligibility

The concept of Smart Legal Contracts necessarily raises questions on whether it could fit in the current contract law framework. The main issue is represented by the language in which these are written, i.e., code, unintelligible for non-experts, aka those without a background in computing or logic.

On the one hand, we must say that current contract law (from a European point of view) is firmly based on the principle of freedom of form. This means that a contract, as a general rule, is not subject to any formal requirement.[20] National laws can then require a specific form for particular types of contracts, both *ad substantiam* and *ad probationem*. Moreover, the EU Blockchain Observatory & Forum has affirmed that blockchains fall under the scope of the e-IDAS regulation as an electronic document and that, consequently, "the data, including smart contracts, contained

---

17 *P. Lamela Seija/S. Thompson.* Marlowe: Financial contracts on blockchain. In International Symposium on Leveraging Applications of Formal Methods, Cham 2018, p. 356 (356 et seqq.).

18 *M. D. Flood/O. R. Goodenough*. Contract as Automaton: The Computational Representation of Financial Agreements (Office of Financial Research Working Paper), 2021. Although recently Goodenough has tried the logic programming approach.

19 *F. Idelberger*, Merging Traditional Contracts (or Law) and (Smart) e-Contracts – a Novel Approach, in: Proceedings the 1st Workshop on Models of Legal Reasoning, 2020, available at https://lawgorithm.com.br/wp-content/uploads/2020/09/MLR2020-Florian-Idelberger.pdf (last access: 05.09.2022).

20 PECL 2:101 (2); DCFR II – 1:106; UNIDROIT 1.2.

therein cannot be denied legal force solely because of their electronic nature".[21]

On the other hand, the fact that a contract is expressed in a language that is inaccessible to the average person may affect the correct formation of contractual intent that, together with a sufficient agreement, represents the only necessary requirements.[22] In fact, the difficulty in ascertaining the mutual expression of the intention to be legally bound is highlighted when the parties cannot read or understand their obligations.[23] A solution was found by assimilating smart contracts to adhesion contracts, for which the expression of the intention to be bound was undoubtably recognized, and by applying the relative discipline to it. This means requiring the drafting party to take reasonable steps to bring terms not individually negotiated or imposed by one party to the other party's attention, before or when the contract is concluded.[24] The EU Directive of Unfair Contract Terms states that the consumer should have a 'real opportunity of becoming acquainted' with the terms in order for those not to be considered unfair.[25] This provision has been considered satisfied for wrap contracts (Click-wrap or browse-wrap),[26] where the expression of consent has been recognized just by using a website where the terms and conditions are accessible through hyperlinks. The EU directive on consumer rights takes one more step by also providing that, in order for the consumer to be bound by a contract falling under the scope of the directive (B2C relationships), the trader shall provide certain information to the consumer "in a clear and comprehensible manner".[27]

---

21 *EU blockchain Observatory & Forum*, REPORT – Blockchain and digital Identity, 2018.
22 *G. Christandl*, Art 2:101 (1): Conditions for the Conclusion of a Contract, in: N. Jansen/R.Zimmermann (eds.), Commentaries on European Contract Laws, Oxford 2018, p. 236 (236 et seqq.).
23 *B. Carron/ V. Botteron*. "How smart can a contract be?" Blockchains, Smart Contracts, Decentralized Autonomous Organizations and the Law, Cheltenham 2019, p. 128.
24 *N. Jansen*, Art 2:104: Terms not Individually Negotiated, in: N. Jansen/R. Zimmermann (eds.), Commentaries (n. 23), p. 272 (272 et seqq.).
25 Directive 1993/13/EC, annex 1(i).
26 "Wrap contracts" are adhesion contracts concluded online. The most common are "click-wrap" and "browse-wrap" agreements. In the former the user accepts the terms by clicking on an "I agree button", while in the latter, he does so by simply continuing using the website. C. *Bomprezzi/G. Finocchiaro*, A legal Analysis of the use of blockchain technology for the formation of smart legal contracts, mediaLAWS 2020, 122.
27 Directive 2011/83/EU, Art. 5–6.

Under such legal framework, the most commonly proposed solution, at least in B2C relationships, entails accompanying the smart contract code with a natural language translation of the contract.[28] In B2B relationships such need does not arise since the parties are deemed to have equal contracting power. The legislator assumes that they had enough resources to understand the contents of the agreement, without having to directly intervene in order to protect the weak party.

The present contribution follows instead the abovementioned *standalone* approach, where the whole agreement is expressed directly in the smart contract code. By using programming languages whose understandability level is far higher than usual smart contract languages the goal is to build trust in the source code for both the end user and for the legal professional.

It is argued that with a hybrid smart contract there is no way to determine whether the code behaves according to what is written in the natural language section of the document (i.e.t, the original contract).[29] Such a situation can occur both because of intentional deceit by the drafting party, who taking advantage of the other, or just because of a translation mistake made by the programmer. In fact, where two parties decided to conclude an agreement in the form of a smart contract, they would need, other than a lawyer to lay out the contract terms, also a programmer to write the agreement into computer code. This new level of intermediation increases both the transaction costs and the risk of errors. Not having a legal background, a programmer could be unaware of the different meanings a term can have and thus implement code that is slightly different in its legal effect.

For all the above-mentioned reasons, we believe that approaching trust in the system by using human readable code could empower both the user to be more aware of the behavior of the program, and the legal professionals to build or understand the content of contracts on their own. In case of B2C relationships intelligibility of the source code is necessary for a standalone smart contract to be compliant with EU provisions on consumer rights, according to which "Before the consumer is bound by […], or any corresponding offer, the trader shall provide the consumer with the following information in a **clear and comprehensible manner…**". This principle can and should be applied also outside the B2C field, rather in

---

28 *C. Bomprezzi*, Implications of Blockchain-Based Smart Contracts on Contract Law, Baden-Baden 2021, p. 140.

29 *F. Idelberger*, Merging Traditional Contracts (n. 22).

all cases in which intelligibility of the code could foster trust between the parties, as well as in the contract itself.

In order to properly reach our goal, it would be important to transpose in the smart contract code the whole agreement, at least to the extent permitted by the programming language. This means having a smart contract representation in an intelligible form both legal and non-legal elements, i.e., the parts exclusively necessary for the execution of the code and that would not be mentioned in a traditional contract; as well as elements that do not affect the execution of the transactions, but that are nonetheless part of the contract, such as the header or the competent forum. By doing so, one can acknowledge all the distinct aspects of the agreement in relation to the execution of the obligations.

In the following sections we employ two newly developed programming languages for the purpose of testing our approach to smart legal contract drafting. We will start with a simple contract example and evaluate its transpositions in both languages explaining their functioning.

D. Example

For the purpose of displaying the effect of the two methods we will analyse throughout this paper, we must first present the running example, taken from *Governatori et al.*[30] and readapted to better show what we are proposing. The contract concerns a license agreement between two parties. The licensor is willing to grant the licensee a temporary license to test the product, only under the conditions that the licensee pays the full fee in advance. From the moment he receives the temporary license, the licensee has a limited amount of time (called evaluation period) to test the product and either send confirmation of his intention to buy the full version or let the evaluation period expire and get reimbursed.

Below, the contract in full:

1. Licensor is willing to grant to the Licensee a License to use the Product for the term and specific purpose set forth in this Agreement, subject to the terms and conditions set out in this Agreement.

---

30 G. *Governatori/F. Idelberger/Z. Milosevic/R. Riveret/G. Sartor/X. Xu.*, On legal contracts, imperative and declarative smart contracts and blockchain system, Artificial Intelligence and Law 2018, 377 (377 et seqq.).

2. Licensor grants Licensee a temporary license to evaluate the Product and fixes (i) the evaluation period – in 10 days – and (ii) the cost of the Product – in 1000 Euros -, in case the Licensee will buy the Product.

3. In consideration of the License Product described in Clause 1 of this License Agreement, Licensee shall pay in advance the License fee as stated in Clause 2 of the Agreement.

4. The licensee can decide to purchase the permanent version of the license by sending an explicit communication to this end to the licensor, within the end date of the evaluation period.

5. This Agreement and the License granted herein commences upon the moment the payment has been received by the Licensor.

6. This Agreement shall terminate upon (a) the moment of purchase or, in any case, (b) once the evaluation period has expired.

7. Once the evaluation period has expired, the Licensee will be reimbursed if the Product has not been bought.

I. TECHNOLOGIES – *Stipula*

The first approach we present is represented by a domain-specific programming language, *Stipula*. It was recently developed by professors *Cosimo Laneve*[31] and *Silvia Crafa*[32] in collaboration with *Giovanni Sartor*[33], designed for the creation of Smart Legal Contracts. Stipula's distinctive characteristic is that it was built starting from a small set of abstractions aimed at capturing the main concepts of contract law. These include elements such as *permission, prohibition, obligation, agreement* or *alea*: basic patterns that can be found in any legal contract. Each of these are represented in code with a specific primitive. The idea is that such a structure would make the code more understandable and, especially, easier to handle for legal professionals when drafting smart legal contracts. Moreover, Stipula is based on a relatively straightforward syntax, consisting of terms that recall their commonly understood meaning.

In the next sections we put forward the implementation of our example using Stipula. We will explain the meaning of the different elements constituting the language while describing the various steps of the contract execution.

---

31 cosimo.laneve@unibo.it
32 silvia.crafa@unipd.it
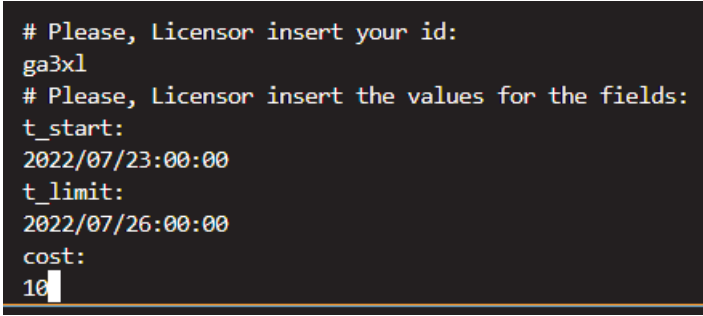33 giovanni.sartor@unibo.it

1. Agreement

Every Stipula contract begins with the execution of the *Agreement* constructor. This represents the moment in which the parties have reached a consensus on the contents of the arrangement they want to create, such as the end of negotiation.

At this stage, each party is asked to either set and/or accept certain values of the contract terms. In Stipula, these values are called *fields,* and in our example, these are the cost, the deadline for the activation of the contract and the deadline of the evaluation period.

```
stipula License {

asset balance, token

field t_start, t_limit, cost, code

init Inactive


agreement (Licensor, Licensee)(t_start, t_limit, cost){

Licensor , Licensee : t_start, t_limit, cost

} ==> @Inactive
```

Fig. 1. Contract header and agreement constructor in Stipula's code. Asset values and agreement constructor are highlighted.

```
# Please, Licensor insert your id:
ga3xl
# Please, Licensor insert the values for the fields:
t_start:
2022/07/23:00:00
t_limit:
2022/07/26:00:00
cost:
10
```

Fig. 2: Stipula's interface during the execution of the agreement constructor.

As you can see from the contract header, field values must be distinguished from *asset* values, which refer to the actual goods managed by the smart contract like a currency or a token representing a good or a right. Such resources have to preserve a total supply in the context of the program: "the sender of the asset must always relinquish the control of the transferred asset".[34] This feature implements, by design, a safety against the risks of double-spending, accidental loss, or lock-in assets, since there is a finite predetermined amount that can be exchanged.

In the license agreement, the assets are the *Balance* (money sent to the contract) and a *token* representing the permanent license. Note how only fields, and not assets, are part of the agreement constructor.

2. *Offer*

An offer is an expression to another party or to the community at large, to be bound by the stated terms. The deployment of smart contract code on a distributed ledger is generally deemed to correspond to an offer, at least for those individuals who are allowed to interact with the smart contract.[35] When any participant in the network can interact with it and conclude the agreement, then the uploading represents an offer to the public. In order to be valid, an offer must contain all the terms of the agreement (*essentialia negotii*).

After the agreement, the contract's state is set to "Inactive". The first action can be taken by the licensor by calling the "offerLicense" function, through which he or she is asked to send the token representing the license to the smart contract. By doing so, where the licensee accepted, the contract could automate the licensor's performance of granting the license.

---

34 *S. Crafa/C. Laneve/G. Sartor,* Pacta sunt servanda: smart legal contracts in Stipula, 2021 available at https://arxiv.org/abs/2110.11069 (last access: 13.10.2022).

35 *M. Durovic/A. Janssen*, The Formation of Blockchain-based Smart Contracts in the Light of Contract Law. European Review of Private Law 2019, 753 (753 et seqq.).

```
@Inactive Licensor : offerLicense (x)[n]{

n --o token

x --> code;

now + t_start >> @Proposal{

token --o Licensor

} ==> @End

} ==> @Proposal
```

Fig. 3: offer function in the Stipula's syntax. The 'state' is highlighted

```
# Please, choose which function should run:        ############
        Licensor.offerLicense(Type4 x)[Asset n]  Next state Proposal
ga3xl.offerLicense(useCode)[100]                   ############
```

Fig. 4. Stipula's interface during the execution: Offer.

After the offer the contract's state switches to "Proposal". *States* are another basic feature of Stipula. They can be seen as the various stages of a contract's lifecycle and the transition between one another is triggered by the occurrence of an event (external or produced by one party). Events are indicated in the source code with an "@".

From the legal standpoint, *States* are used to implement the concepts of *prohibition* or *permission*. In every state, certain actions (i.e., functions call) are allowed, while others are precluded. As an example, in the license agreement, the Licensee is allowed to buy the license only in the "@Trial" state and not before.

3. Acceptance

An acceptance is a form of statement or conduct that indicates assent to the offer. In a smart contract, parties express acceptance by signing a transaction with their cryptographic keys and by sending it to the contract address. As it was noted, the majority of today's smart contracts represent unilateral contracts, stating, for example, that if X happens, I will give

you Y.[36] With these types of contracts, acceptance always comes through the act of performance. The transaction expressing consent will likely represent the transfer of control over a digital asset to the smart contracts as, for example, money, cryptocurrency, or digital token representing a material good.

In our example, after the licensor's proposal, the licensee can accept it by calling the function "activateLicense". He or she receives a usecode which grants a temporary access to the software. At the same time, the licensee is required to send to the contract the price necessary to buy the permanent license. This money will be credited automatically to the licensor if the licensee buys the permanent license by calling the relative function (in another version of the same contract, the upfront escrow can be used to issue a penalty where the licensee infringed some contract terms). Should the licensee not buy the permanent license within the evaluation period, the money will automatically be sent back to him and the token back to the licensor.

```
@Proposal Licensee: activateLicense()[b] (b == cost) {

b --o balance

code --> Licensee;

now + t_limit >> @Trial {

balance --o Licensee

token --o Licensor

} ==> @End

} ==> @Trial
```

Fig. 5: the acceptance function in Stipula's syntax. The 'event' is highlighted.

Such a procedure is implemented through an *event*, another Stipula's primitive. Events are employed to schedule the execution of future operations if specific preconditions are met. Usually, preconditions are represented by the different states defined in the contract. From the legal perspective, events are used to enforce contractual *obligations*. They consist of a timeout (timer), which is initiated when the function is called, a precon-

---

36 *M. Durovic, A. Janssen,* 2019 (n. 33).

dition (state) and the consequential operation that may be triggered (a statement). According to an event, if the timer expires and the contract state is still "@X", then the contract executes a certain operation such as terminating the contract, issue an automatic penalty etc. In our example, the event is used to invalidate the temporary usecode for the software at the expiration of the evaluation period and, as already mentioned, to send the resources managed by the contract to the respective owner.

4. Purchase

The licensee can decide to purchase the permanent license by calling the "Buy" function. The money previously escrowed by the contract is sent to the licensor address and the licensee receives the token representing the permanent license. By receiving the token, the licensee acquires a full right on the license which cannot be limited by other actors.

```
@Trial Licensee : buy ()[]{

balance --o Licensor

token --o Licensee;
```

Fig. 6. The buy function in Stipula's Syntax. The lollypop operator is highlighted.

This particular kind of transfer is indicated by a special operator, the lollypop (⊸). This operator is used for the movement of *assets*. This means that the location previously holding the asset is emptied and it loses control over it. On the legal side, this represents the translative effect of a previously existing right.

II. TECHNOLOGIES – *Logical English*

Logical English is a controlled natural-language interface that provides syntactic sugar for logic programming languages (namely Prolog and sCASP). It enables the user to write logic rules in quasi-natural language form, which is then translated internally in Prolog, evaluated, and the solution presented to the user in natural language form. This language,

108

developed by *Robert Kowalski* in 2020,[37] is aimed at providing the ability to understand the code and what it represents even to non-programmers, thus virtually giving everyone access to the underlying logic of a program. This language differs from Stipula, as while the latter excels at inferring legal meaning and concepts from the syntax, this approach is focused on representing the source code as faithfully as possible to the contract as written in natural language.

We will now present a brief overview of the technical aspects of this language.

A LE (Logical English) document consists of a knowledge base of facts and rules, scenarios, queries, and templates. The templates are declarations of the predicates contained in the knowledge base and scenarios, such as

> *"according to clause1 *a licensee* and *a licensor* conclude *a contract* for *a product*"*

A template identifies a sentence and the variables contained in it. The arguments are identified by a being surrounded by asterisks and starting with an indefinite article "a" or "an". The predicate itself is represented by the rest of the template. The templates are used to identify instances of the predicates in the knowledge base and elsewhere. For example, the template above can be used to identify a sentence such as

> *"according to clause1 Ale and Marco conclude Contract1 for licenseNFT"*

as an instance of a predicate, which is translated into Prolog or s(CASP) resulting in the symbolic representation:

> *"according_to_clause1_and_conclude_for('Ale', 'Marco', 'Contract1', 'licenseNFT')".*

The translation can be then processed by any standard Prolog interpreter.

This next section will focus on the conversion process, e.g., how to translate the natural language text into Logical English. In legal logic programming one of the key conditions for the representation of legal rules in code is isomorphism, defined as one-to-one correspondence between norms in the formal model and natural language. Each concept or condition must be accurately translated, not only the automated process that is the foundation of a smart legal contract, but also the surrounding

---

37 *R. Kowalski*, Logical English A position paper prepared for Logic and Practice of Programming 2020.

information that the parties decide is useful to better interpret the contract itself.

For example, let us take a look at clause 1 of this hypothetical contract (Figure 6).

```
according to clause1 a licensee and a licensor conclude a contract for a product
    if the licensee is of type licensee
    and the licensor is of type licensor
    and the parties agree to the contract
    and the product is the object of the contract.
```

Fig. 7: the logical transposition of clause 1.

The representation of this clause exemplifies the process behind it. First, we must recall Article 1321 of the Italian Civil Code, which states that a contract is an agreement between two or more people, to establish, regulate or terminate a patrimonial legal relationship.

After having established the minimum requirements behind the definition of a contract, it becomes easier to visualize the purpose of the clause. The LE representation, as seen in Figure 6, clearly identifies the conditions behind the applicability of the clause and therefore of the contract. The parties have been identified as 1) the licensor and 2) the licensee. Furthermore, the parties have agreed to the contract. Finally, the object of the contract is the license to use the product, here referred to as simply "product", which has monetary value and establishes a legal relationship between the two parties.

The same process applies to all other contract clauses. We shall take a slightly deeper look at clause 6, which states that the Agreement shall terminate upon (a) the moment of purchase or, in any case, (b) once the evaluation period has expired.

```
63 according to clause6 the agreement shall terminate by purchase on a date D
64     if according to clause4 a licensee purchases a permanent license on the date D.
65
66 according to clause6 the agreement shall terminate by expiration on a date D
67     if according to clause2 a licensor grants a product to a licensee for a number days for an amount on a date D0
68     and according to clause5 an evaluation period starts on a date D1
69     and D is the number days after D1
70     and it is not the case that
71         according to clause6 the agreement shall terminate by purchase on a date D2.
```

Fig. 8: the logical transposition of clause 6.

Since clause 6 introduces two moments in time related to the termination of the contract, we have decided to divide this rule into two. The first is related to the moment of purchase, while the second relates to the

expiration date. It is interesting to introduce at this point one of the logical features of LE, which is negation as failure, and is written as "it is not the case that", followed by the negated literal. Through negation as failure, just like traditional Prolog programs, it is possible to write exceptions to norms as defeasible rules.

For this clause, we have stated that in all cases where the contract has not terminated with the purchase of the product by the licensee, it shall terminate by expiration of the trial period.

1. Execution

Now that the contract has been represented into computational language, the next step concerns telling the systems which steps must be taken to execute the contract itself. Basically, how to go from "what the contract states" to "what the contract does"?

```
86  a contract sends NFTuseCode to a licensee on a date D
87      if according to clause1 the licensee and a licensor conclude the contract for a product
88      and according to clause2 the licensor grants the product to the licensee for a number days for an amount on D.
89
90  a contract sends a product to a licensee on a date D
91      if according to clause1 the licensee and a licensor conclude the contract for the product
92      and according to clause4 the licensee purchases a permanent license on D.
93
94  a contract sends an amount to a licensee on a date D
95      if according to clause1 the licensee and a licensor conclude the contract for a product
96      and according to clause7 the licensee gets refunded the amount on the date D.
97
98  a contract sends a product to a licensor on a date D
99      if according to clause1 a licensee and the licensor conclude the contract for the product
100     and according to clause7 the licensee gets refunded an amount on the date D.
```

Fig. 9: the transposition of the execution rules.

Each goal originates from the same template – "*a contract* sends *a thing* to *an agent* on *a date*" – which is being recalled, providing a trace of all token movements. Furthermore, it can be observed that each action is linked to the computable representation of the legal prose. Particularly, each transaction can be directly connected to an article being applied in the current case.

For example, the first rule in Figure 8, lines 86–88, states that a contract sends a temporary use code to the licensee if clauses 1 and 2 of the contract are positively instantiated. Each executable step is triggered by the satisfactory outcome of a clause in the contract.

2. Explaining the contract


```
E =
  It is the case that: Contract1 sends 1000 to Ale on 2022-7-20T0:0:0.0 as proved by KB Text
because
  • It is the case that: according to clause1 Ale and Marco conclude Contract1 for licenseNFT as proved by KB Text
    because
      ◦ It is the case that: Ale is of type licensee as proved by hypothesis in scenario
      ◦ It is the case that: Marco is of type licensor as proved by hypothesis in scenario
      ◦ It is the case that: the agreement is Contract1 as proved by hypothesis in scenario
      ◦ It is the case that: licenseNFT is the object of Contract1 as proved by KB Text
        because
          ▪ It is the case that: Marco transfers licenseNFT to Contract1 on 2022-7-2T0:0:0.0 as proved by hypothesis in scenario
  • It is the case that: according to clause7 Ale gets refunded 1000 on 2022-7-20T0:0:0.0 as proved by KB Text
    because
      ◦ It is the case that: clause3 applies to Ale to Marco about 1000 on 2022-7-10T0:0:0.0 as proved by KB Text
        because
          ▪ It is the case that: Ale sends 1000 to Marco on 2022-7-10T0:0:0.0 as proved by KB Text
            because
              ▪ It is the case that: Ale sends 1000 to Contract1 on 2022-7-10T0:0:0.0 as proved by hypothesis in scenario
      ◦ It is the case that: according to clause6 the agreement shall terminate by expiration on 2022-7-20T0:0:0.0 as proved by KB Text
        because
          ▪ It is the case that: according to clause2 Marco grants licenseNFT to Ale for 10 days for 1000 on 2022-7-10T0:0:0.0 as proved by KB Text
            because
              ▪ It is the case that: Marco has set 10 days as an evaluation period as proved by hypothesis in scenario
              ▪ It is the case that: Marco has set 1000 as price as proved by hypothesis in scenario
          ▪ It is the case that: according to clause5 an evaluation period starts on 2022-7-10T0:0:0.0 as proved by KB Text
          ▪ It is NOT the case that: according to clause6 the agreement shall terminate by purchase on "a date" ~ KB Text
```
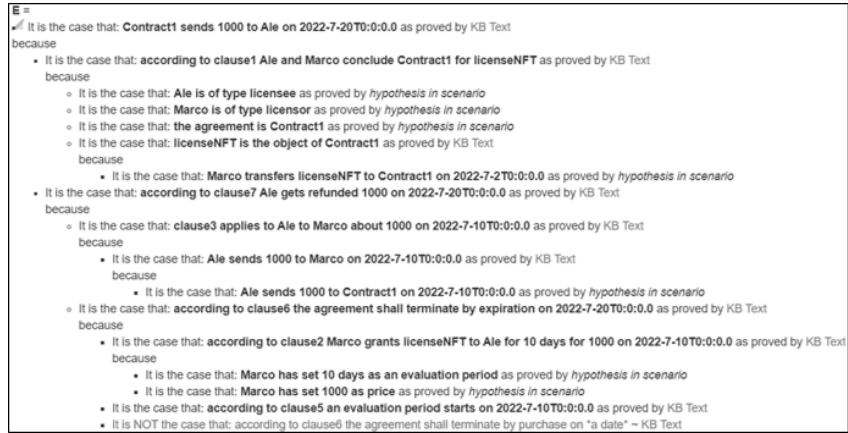
Fig. 10: the results of the execution.

In figure 9 we can visualize one of the advantages provided by mixing declarative programming with quasi-natural language translation. It is possible to query the system regarding the actions taken by the contract, as well as whether any clause applies in the present case. The system looks through all known facts, inserted by the user, as well as any inferences that can be made based on the applicable rules.

Based on the information which has been given as input, the system runs through the available options, checking which articles can be applied in the current situation, and by tracing the executable steps, gives a complete overview of the relevant clauses and the actions that can be taken. For example, by looking at Figure 9, we can visualize that the contract has sent the price of the contract back to the licensee (*Ale*) on a date since 1) a contract was successfully concluded (clause 1 was correctly applied) and 2) the licensee is supposed to be refunded the price on this date (clause 7 was correctly applied). Each of these clauses has sub-conditions for their applicability, and it is possible to verify each step of the reasoning process.

Each indentation reflects a goal that shall be satisfied by the conditions in the next indentation level. The system also returns negative conditions, highlighted in red, which tell the user which requirements were not met.

Therefore, by approaching smart legal contracts from a declarative-isomorphic point of view, it is possible to both provide the user with a trace of the movement of tokens from and to the contract, as well as enable the parties to interact with the contract directly, asking questions regarding

112

the applicability of certain clauses, their effect on the overall validity of the contract, and the conditions that must be satisfied or not satisfied.

E. Conclusion & discussion

In this paper we have presented Stipula, a domain specific language aimed at supporting lawyers in the drafting of smart legal contracts, by capturing the essential concepts of contract law and transposing them into an easy-to-use programming language. We highlighted certain concepts, such as what constitutes an agreement, an offer or the acceptance of the contract, and the way Stipula conveys such information to the user.

Furthermore, we experimented with Logical English, a logic programming language. Transposing a contract by maintaining the same structure and syntax sacrifices certain semantic and contextual definitions in order to provide non-IT experts with the ability to read the code and understand the logical steps required in order to fulfill the obligations of the contract.

We believe that these solutions could greatly contribute to the transparency of this innovative technology, thus fostering people's trust in it. Moreover, these can become useful for legal practitioners, them to either draft smart legal contracts directly, or to validate the program written by a programmer.

An observation that stemmed out from our work, particularly from the implementation of Logical English, is that the translation of a contract in logic form is facilitated when the original contract has been written computable logic in mind. With regard to execution steps, certain logical structures must be present in the text in order to facilitate the transposition. Generally, having a clear distinction between goals (the actions that must be automatically executed) and requirements (which conditions must apply) is essential to write a precise isomorphic representation of the contract clauses.

These two solutions can help contracting parties communicate in a common language, available to both users and programmers alike, and the use case shown supports, at least preliminarily, this conclusion.