From Legal Agreements to Blockchain Smart Contracts

Ravi Rahman*, Kevin Liu†, and Lalana Kagal‡
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA USA
Email: *r_rahman@csail.mit.edu, †kevinliu@csail.mit.edu, ‡lkagal@csail.mit.edu

***Abstract*—Complex legal agreements enable many real-world applications, from data sharing systems to financial transactions. However, legal expenses scale with complexity because of the manual processes to draft, revise, and enforce agreements. To reduce such costs, we propose a new framework for lawyers to develop machine readable legal agreements, which are automatically verified and deployed on the Ethereum blockchain. Specifically, our framework introduces domain specific repositories to store human and machine readable legal language, a web interface and Python API to draft legal agreements, correctness checking via formal verification, and a voting system for blockchain based adjudication. Experimental evaluation found that our proposed framework offers an efficient verification system, incurs linear scaling of Ethereum blockchain gas consumption in terms of agreement size, and correctly models 81% of conditions in real-world agreements through the domain specific repositories. These results suggest a practical approach for developing verifiable and blockchain compatible legal agreements.**

***Index Terms*—Ethereum blockchain smart contracts, machine readable legal agreements, formal verification**

I. Introduction

Formal legal contracts govern many real-world applications, ranging from data sharing systems to complex financial transactions. For all types of agreements, lawyers employ similar, manual drafting and revision processes. When disputes arise, lawyers must re-read agreements to present arguments for settlement or trial. Each of these manual steps during creation, validation, and enforcement of an agreement incurs significant legal fees for lawyers' time. Moreover, no manual check will catch all errors, and inadequate agreements resulted in multi-million dollar lawsuits [1], [2].

While some have attempted to use natural language processing to automatically interpret legal agreements, such systems are limited to certain legal domains [3] and are not focused on detecting errors or easing with litigation. Computer systems cannot extract the logic from text based legal agreements. This limitation prevents automation of legal processes, and by extension, significant reduction of legal expenses.

We introduce a new framework that aims to reduce the time and cost of drafting, checking, and resolving disputes. Our framework, through its web interface and Python API, enables lawyers to write agreements by importing customizable pieces of legal language from repositories. Studies have shown that similar language is in more than 60% of agreements in the same domain [4]. Our repositories store language in human and machine readable forms as text templates and combinational logic, respectively.

Our framework uses formal verification to check legal agreements. Formal verification can both identify unintended loopholes and prove that intended uses cases are permissible. For example, in a data sharing agreement, the data provider could ensure that, under all circumstances, there is no loophole that allows data to be used for marketing purposes without end user consent. Likewise, the data recipient can prove that there exists a procedure to use end user data for marketing purposes.

Our framework converts the machine readable representation of agreements into SMT-LIBv2 for formal verification via Z3 [5] and Vyper [6] for Ethereum blockchain smart contract deployments [7]. Unlike traditional text based agreements, our smart contract legal agreements include a dispute resolution mechanism. This system simplifies the adjudication process, thereby lowering enforcement costs.

We evaluated the framework on randomly generated agreements and sample data sharing agreements in terms of formal verification performance, Ethereum blockchain gas consumption, and empirical applicability. These metrics measure the scalability of our framework with respect to agreement size and the ability to represent real-world legal agreements.

Section II describes works similar to our framework. In Section III, we detail the implementation. In section IV, we evaluate formal verification performance, Ethereum blockchain gas consumption, and empirical applicability on randomly generated agreements and sample data sharing agreements.

II. Related Work

Our framework spans document assembly, formal verification, structured representations for legal agreements, and blockchain based adjudication. We could not identify any other single system that integrates these four domains into one unified framework.

A. Document Assembly

Document assembly systems use interview responses to populate templates to generate unique documents. While these systems are popular among legal professionals [8], [9], there are significant issues with agreements generated through document assembly that have not undergone manual legal counsel

978-1-7281-6680-3/20/$31.00 ©2020 IEEE

review [10]. As such, these systems only reduce drafting costs, not verification nor enforcement expenses.

Our framework incorporates the benefits of document assembly by providing repositories to store customizable legal language in both human and machine readable forms. Moreover, our framework automatically verifies the machine readable representation to eliminate drafting errors.

B. Formal Verification

Formal verification verifies the functionality of deterministic systems through mathematical proofs. Precision critical systems, such as those in aerospace or defense, employ this technique to detect and eliminate design errors [11].

As errors in legal agreements can also incur significant repercussions, our framework employs formal verification to validate agreements. We use the Z3 prover developed by Microsoft Research [5]. Given a logical statement, Z3 returns either a combination of inputs to satisfy the logical statement or a proof that no such combination of inputs exists.

C. Structured Representation of Legal Agreements

Existing projects, such as Ergo, model legal agreements as software programs by embedding machine readable functions inside legal clauses [12]. Ergo provides a repository of clauses that users can import into their agreements, and it supports Coq [13] for formal verification. Unlike Ergo, which focuses on representing the commonly used parts of a legal agreement via a functional programming language, our framework models the entire agreement via combinational logic and thus can formally verify the entire agreement.

Others have attempted to use natural language processing (NLP) on legal agreements. Such systems are generally limited to specific domains of legal documents, such as leases [3], and to information retrieval [14], [15]. As formal verification requires a highly structured representation, current NLP techniques would not be able to satisfy the requirements for our framework. As such, our framework does not currently support generating the machine readable representation via NLP.

D. Blockchain Adjudication

Studies have noted the importance of allowing human judgment to serve as inputs to programs that represent legal agreements [16]. For example, while it may be trivial to calculate late fees, it is difficult to calculate a monetary award for emotional damages. Our framework addresses these concerns by supporting user specified inputs. Should the users disagree on the appropriate value for such inputs, they can invoke our framework's blockchain based adjudication.

We incorporate our earlier work [17], which proposed a method for adjudicating disputes via an Ethereum blockchain smart contract. During creation of the smart contract, both parties agree on a set of arbiters. Should a dispute arise, the pre-selected arbiters can vote, within a specified time period, either yes or no on whether a breach occurred. Should a minimum number of arbiters vote, and the majority of those who voted agree that a breach occurred, the smart contract can automatically enforce monetary penalties.

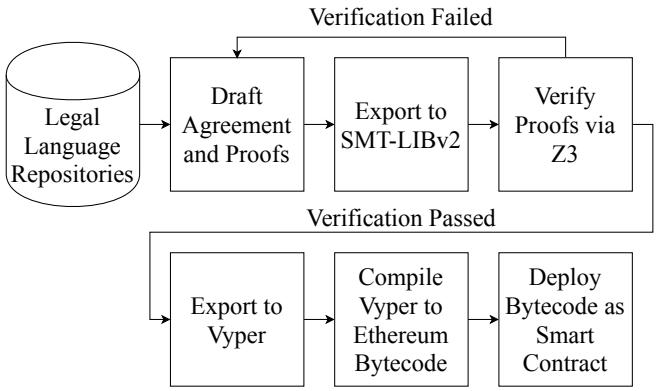

Figure 1: **System architecture.** Lawyers import legal language from repositories to draft agreements. Our framework converts agreements to SMT-LIBv2 for formal verification via Z3. Verified agreements are exported to Vyper for Ethereum blockchain smart contract deployments.

III. Implementation

A. Legal Language Repositories

Our framework stores legal language as both text templates and machine readable combinational logic in repositories. Combinational logic reduces Boolean inputs, through logical connectives, into a single Boolean output. We refer to the Boolean inputs as actions, and the logical connectives as clauses. Both actions and clauses are stored in the repositories.

Actions are inputs to an agreement and must resolve to a Boolean value (`true` or `false`). They represent whether a specific event occurred; for example, in data sharing agreements, an action could be whether the data was shared. In the repositories, actions contain a text template describing the event (e.g. "{party_a} shared {data_type} with {party_b}") and a default state (e.g. `false`). When instantiated, the lawyers must provide parameters to populate the text template, a list of parties who can update the state of the action, and a list of independent arbiters who can adjudicate disputes. When deployed as an Ethereum blockchain smart contract, the parties can propose updates to the state of any action. The other parties, and arbiters in the case of a disagreement, vote to decide whether to update the action state. The design of an action was based on concerns raised in [16] that some inputs into programmatic agreements require human judgment.

Clauses use deterministic combinational logic to resolve combinations of actions or other clauses (recursively) to a single Boolean output. Unlike actions, clauses are not disputable. However, disputing and changing the state of an action contained within a clause could change the outcome of the clause; a new outcome would propagate through the entire contract. Each clause in a repository contains a combinational logic implementation, a set of proofs, and a text template description. Proofs assert that specific input values result in the expected outcome. The natural language description is not verified by the proofs and instead must be proofread manually. When using a clause in an agreement, lawyers must specify

the input actions and sub-clauses, as well as parameters for the text template description. When verifying an agreement, a clause's proofs ensure that the clause does not conflict with another clause in an agreement.

B. Agreement Drafting

Our framework presents two interfaces, a Python API and point-and-click web frontend, in which one can write a legal agreement. The Python interface exposes all features of the framework. The web-based interface supports drafting an agreement by importing clauses from the repositories, writing custom clauses, and converting agreements to Vyper for deployment on the Ethereum blockchain. It does not support writing custom proofs.

Behind the scenes, the Python backend powers the web frontend. The web frontend and Python backend communicate via JSON as suggested by [18], [19].

C. Formal Verification

Our framework verifies an agreement by recursively converting the logical representation of the actions and clauses into SMT-LIBv2, the language for the Z3 prover [5].

Each proof, which is contained within a clause in a repository or is written by the user, is checked individually by Z3. A proof fixes the result of one or more actions or clauses and asserts whether the overall agreement should be satisfiable. Given these fixed values, Z3 verifies that the agreement is either satisfiable under at least one scenario or unsatisfiable under all scenarios [5]. Finally, the satisfiability is compared to the expected outcome from the proof.

Discrepancies indicate that the logical implementation of a clause is incorrect, that there might be a contradiction in the agreement, or that, in the case of a proof written by a user, that the logic of the agreement differs from what the user intended to write. Note that formal verification does not verify that the populated text templates within the clauses or actions are correct. The prover cannot determine whether the textual and logical representation of a clause align. One must verify these components manually.

D. Ethereum Smart Contracts

Our framework automatically generates Ethereum blockchain smart contracts [7] which model the underlying legal agreements. First, agreements are converted into Vyper, an Ethereum smart contract programming language [6]. The Vyper compiler then generates Ethereum bytecode. This two step process enables us to leverage the advantages of Vyper, including its integer overflow detection and Ethereum gas requirement calculations. These features protect against integer and gas limit attacks [20].

To minimize Ethereum blockchain costs and preserve privacy, only the contract logic – not the textual descriptions – are stored in the Ethereum contract. Our framework generates a map between the original descriptions and the corresponding state variables for off chain storage.

After deploying a smart contract, parties can propose state updates for actions. If all other parties agree, then the state of the action is updated. Otherwise, disagreement among the parties starts the blockchain based adjudication process, which is based on [17]. During the pre-voting period, other parties can include additional actions in a dispute. This period allows for counterclaims. After the pre-voting period, all arbiters vote on what they believe the correct state to be for each disputed action. Finally, all votes are tallied after the voting period. Only if a minimum number of arbiters have voted, each action state is updated to what a simple majority have decided the "correct" state to be. Note that the pre-trial period duration, voting period duration, and arbiter quorum threshold must be specified during compilation of the Ethereum smart contract.

The smart contract exposes the overall breach status of the agreement via a public method named `computeContract`. This function allows other smart contracts on the Ethereum blockchain to determine whether the agreement was breached.

IV. Evaluation

A. Metrics

We evaluated our framework in terms of formal verification performance, Ethereum blockchain gas costs, and empirical applicability. These metrics reflect the scalability of our framework and ability to model real-world legal agreements.

B. Formal Verification Performance

As SMT formal verification is an NP-complete problem, it is unlikely that formal verification performance can scale in polynomial time with the number of actions in an agreement. However, the Z3 prover incorporates optimizations for efficient SMT solving, including Boolean constant propagation, clause deletion, and relevancy propagation [21], to reduce the search space. To determine the scalability of the proof mechanism, we measured how the number of actions and clauses on randomly generated agreements affects the execution time of Z3.

We randomly generated 150 agreements containing a variable number of two party, three arbiter actions and `AND`,

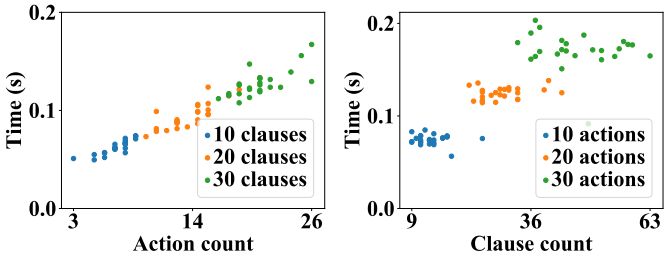

Figure 2: **Formal verification performance**. The above charts illustrate how the number of actions (left) and clauses (right) affect the time required by Z3 to verify 150 randomly generated agreements. The number of actions and clauses respectively resulted in a correlation of $r = 0.97$ and $r = 0.86$ with the time performance of Z3. These results illustrate the more significant impact on prover performance from each action, or free variable, which increases the search space.

Method	Gas Consumption
File a Dispute	91,412
Add a Counterclaim	108,939
Vote	41,611

Table I: **Gas consumption for fixed cost methods**. Our implementation uses constant time methods, independent of agreement size, to file a dispute, add counterclaims to a dispute, and vote on a dispute. These figures were reported by the Vyper compiler.

Method	Marginal Gas Consumption for Each			
	Party	Arbiter	Action	Clause
Deploy Contract	24,336	1,720	192,872	12,769
Compute Breach Status	0	0	0	432
Close Dispute	0	0	93,980	0

Table II: **Gas consumption for variable cost methods.** The above table shows the marginal gas costs incurred for each party, arbiter, action, and clause in a contract. We obtained these results by randomly generating 250 agreements and performing linear regressions. Web3 [24] provided gas estimates for contract deployment, and the Vyper compiler [6] provided upper-bound gas limits for contract methods. All regressions had $R^2 \geq 0.99$.

`OR`, and `NOT` clauses. In addition, we generated another 100 one-action, one-clause agreements with between 10 and 100 parties and arbiters. For each randomly generated agreement, we measured the execution time of Z3 via Python's `timeit` library [22]. Data was collected on Amazon Web Services dual core Intel Xeon E5-2666 v3 Haswell machines (instance type `c4.large`) [23] running Ubuntu Linux 18.04. Figure 2 illustrates how verification time scales with the number of actions and clauses.

C. Blockchain Costs

Executing instructions, reading from, and writing to the Ethereum blockchain require gas, which is used to reward the Ethereum blockchain miners. Lower gas consumption incurs lower real-world costs. The amount of gas consumption is independent of the market price of Ethereum.

We measured how the number of parties, arbiters, actions, and clauses on randomly generated agreements affected gas costs for deployment onto the Ethereum blockchain and for dispute adjudication. The Web3 framework [24] provides estimates for the gas needed to deploy a smart contract, and the Vyper compiler provides upper bound gas limits for contract methods [6]. Results are shown in I and II.

D. Empirical Applicability

We measured the empirical applicability of our framework by evaluating sample agreements freely available online [25], [26]. We designed actions and clauses for our repositories such that the logical representation matches our interpretation from the original agreements.

From these agreements, we identified 93 actions and clauses, with 31 from [25] and the remaining 62 from [26]. Out of these 93 actions and clauses, only 21 were distinct (i.e. excluding repetitions). Of these 21, we successfully modeled 17 (81%) in our framework. For these 17 we could model and add to our repositories, each was used on average 4.5 times, with the "Share Data Action" being most commonly used with 16 repetitions. Considering we evaluated our framework on data sharing agreements, it is unsurprising that the "Share Data Action" was the most commonly used. The level of repetition for this action and others, both within an individual agreement and between agreements, shows the effectiveness of the repositories.

However, of the 21 clauses, four (19%) were incompatible and represent the limitations of our framework. Specifically, our framework cannot cleanly model repeating events, for-each statements, and changes to the parties and arbiters. These types of legal constructs were found in the evaluated agreements. For-each statements would enable the modeling of infinitely repeating events. For example, in the context of data sharing agreements, the imposition of penalties *for each* data breach cannot be accurately modeled through our current framework. Finally, agreements generated through our framework require the explicit names and Ethereum addresses of all parties and arbiters to be specified at compile time. Should the parties change after the agreement is created, a new contract would have to be deployed. However, external smart contracts can proxy the parties and arbiters. These contracts could then delegate voting authority to real parties and arbiters.

V. Conclusion

Our framework enables lawyers to use a web interface to write legal agreements that can be automatically verified via formal verification and deployed on the Ethereum blockchain for lower adjudication costs when disputes arise. Storing legal language in repositories enables reuse of human and machine readable legal language across agreements. By having both a web interface and a Python API, our framework is accessible to both legal professionals and software developers alike. We aim to reduce the manual work involved with drafting, checking, and enforcing complex agreements.

The evaluation indicates that the framework, through combinational logic, can accurately model 81% of legal constructs found in real-world agreements. The formal verification runtime and blockchain gas costs for generated agreements scale linearly with agreement complexity. These results suggest a practical approach for developing verifiable and blockchain compatible legal agreements.

Future work includes extending our framework to support for-each type statements and integer inputs. These improvements would enable our framework to model more types of agreements. In addition, we are interested in performing an end user evaluation among legal processionals. Our current evaluation has focused on the performance and expressivity of our framework. User testing would enable us to validate our underlying hypothesis that machine readable legal agreements would reduce time and costs needed to draft, check, and enforce contracts.

REFERENCES

[1] Bourke v. Dun & Bradstreet Corp, 59 F.3d 1032 (7th Cir. 1998). [Online]. Available: https://casetext.com/case/bourke-v-the-dun-bradstreet-corp

[2] Baybank v. Vermont National Bank, 118 F.3d 30 (1st Cir. 1997). [Online]. Available: https://caselaw.findlaw.com/us-1st-circuit/1057693.html

[3] Lease abstraction. Kira Systems. [Online]. Available: https://kirasystems.com/solutions/lease-abstraction/

[4] E. Garoufallou, S. Virkus, R. Siatri, and D. Koutsomiha, Eds., *Toward a Metadata Framework for Sharing Sensitive and Closed Data: An Analysis of Data Sharing Agreement Attributes*. Cham: Springer International Publishing, 2017. [Online]. Available: http://cci.drexel.edu/media/19189/metadata-for-sharing-closed-data-grabus-greenberg-56-mtsr2017.pdf

[5] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-78800-3_24

[6] (2020) Vyper. Vyper Team. [Online]. Available: https://github.com/vyperlang/vyper

[7] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[8] Hotdocs benefits. HotDocs. [Online]. Available: https://www.hotdocs.com/benefits

[9] Elite. Thomson Reuters. [Online]. Available: http://www.elite.com/3e/matter-management/documents/

[10] I. Figueras, "The legalzoom identity crisis: Legal form provider or lawyer in sheep's clothing comment," *Case Western Reserve Law Review*, vol. 63, p. 1419, 2012-2013. [Online]. Available: https://scholarlycommons.law.case.edu/cgi/viewcontent.cgi?article=1234&context=caselrev;The

[11] V. Wiels, R. Delmas, D. Doose, P. Garoche, J. Cazin, and G. Durrieu, "Formal Verification of Critical Aerospace Software," *AerospaceLab*, no. 4, pp. p. 1–8, May 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01184099

[12] (2019, 05) Introduction to ergo. Accord Project. [Online]. Available: https://docs.accordproject.org/docs/ergo.html

[13] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner, "The Coq Proof Assistant Reference Manual : Version 6.1," INRIA, Research Report RT-0203, May 1997, projet COQ. [Online]. Available: https://hal.inria.fr/inria-00069968

[14] I. Chalkidis, I. Androutsopoulos, and A. Michos, "Extracting contract elements," in *Proceedings of the 16th Edition of the International Conference on Articial Intelligence and Law*, ser. ICAIL '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 19–28. [Online]. Available: https://doi.org/10.1145/3086512.3086515

[15] J. Nay, "Natural language processing and machine learning for law and policy texts," *Available at SSRN 3438276*, 2019. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3438276

[16] E. T. T. Tai, "Formalizing contract law for smart contracts," *Tilburg Private Law Working Paper Series*, no. 6/2017, September 2017. [Online]. Available: https://ssrn.com/abstract=3038800

[17] H. Desai, K. Liu, M. Kantarcioglu, and L. Kagal, "Adjudicating violations in data sharing agreements using smart contracts," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, July 2018, pp. 1553–1560. [Online]. Available: https://ieeexplore.ieee.org/document/8726492

[18] *Comparison of JSON and XML data interchange formats: A case study*, 01 2009. [Online]. Available: https://www.cs.montana.edu/izurieta/pubs/IzurietaCAINE2009.pdf

[19] J. N. Dewey. (2017, October). [Online]. Available: https://github.com/jndewey/cctools

[20] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks and defenses," *arXiv preprint arXiv:1908.04507*, 2019. [Online]. Available: https://arxiv.org/abs/1908.04507

[21] L. De Moura and N. Bjørner, "Relevancy propagation," Technical Report MSR-TR-2007-140, Microsoft Research, Tech. Rep., 2007.

[22] timeit - measure execution time of small code snippets. Python Software Foundation. [Online]. Available: https://docs.python.org/3.6/library/timeit.html

[23] Amazon ec2 instance types. Amazon Web Services. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[24] Web3. Ethereum Foundation. [Online]. Available: https://github.com/ethereum/web3.js

[25] C. for Disease Control and Prevention, "Sample data-sharing usage agreement." [Online]. Available: https://www.cdc.gov/cancer/ncccp/doc/sampledatasharingusageagreement.doc

[26] C. C. of Healthcare Providers, "Sample data sharing agreement," May 2017. [Online]. Available: https://www.nationalcomplex.care/wp-content/uploads/2018/06/Appendix-B-Data-Sharing-Agreement-Template.pdf