Setting Standards for Altering and Undoing Smart Contracts

Bill Marino[1(✉)] and Ari Juels[2]

[1] Cornell Tech, New York, USA
wlm67@cornell.edu
[2] Cornell Tech (Jacobs Institute), New York, USA
juels@cornell.edu

Abstract. Often, we wish to let parties alter or undo a contract that has been made. Toward this end, contract law has developed a set of traditional tools for altering and undoing contracts. Unfortunately, these tools often fail when applied to smart contracts. It is therefore necessary to define a new set of standards for the altering and undoing of smart contracts. These standards might ensure that the tools we use to alter and undo smart contracts achieve their original (contract law) goals when applied to this new technology. This paper develops such a set of standards and, then, to prove their worth as a framework, applies to them to an existing smart contract platform (Ethereum).

Keywords: Smart contracts · Contract law · Blockchain · Ethereum

1 Introduction

> If a covenant be made wherein neither of the parties perform presently, but trust one another, in the condition of mere nature … upon any reasonable suspicion, it is void: but if there be a common power set over them both, with right and force sufficient to compel performance, it is not void.
>
> — Thomas Hobbes, *Leviathan* (1651)

> The purpose of contracts is to solve a game-theoretic problem: it is to our mutual benefit to cooperate in some way. But if we cooperate, then one of us can do even better by defecting.
>
> — sirclueless [psued.], comment on *What is Ethereum?*, Hacker News (2015)

Tyrell Corporation, manufacturer of replicant humans in Philip K. Dick's *Do Androids Dream of Electric Sheep?*, famously touted their wares as "more human than human". Riffing on that motto, we might say that smart contracts are able to beat analog contracts at their own game and are therefore "more contract than contract".

This is because the "fundamental function of contract law (and recognized as such at least since Hobbes's day) is to deter people from behaving opportunistically toward their contracting parties" [1]. And that is something a smart contract — at least, in its paradigmatic form — does better than any analog contract ever could. In fact, a

J.J. Alferes et al. (Eds.): RuleML 2016, LNCS 9718, pp. 151–166, 2016.
DOI: 10.1007/978-3-319-42019-6_10

well-designed smart contract drives the probability of opportunistic breach toward zero as such behavior becomes impossible or, at least, "expensive (if desired, sometimes prohibitively so) for the breacher" [2].

Mind you, this feat is not possible for a contract that is merely "a set of promises, specified in digital form" [2] — i.e., a digital contract. Breaching a contract recorded in binary is no harder than breaching one recorded in ink. What lets smart contracts rise above their brethren is that they additionally include "protocols within which the parties perform on … promises" [2]. These protocols beget smart contracts' hallmark ability to "automatically enforce" [2] themselves, a quality that, in turn, eliminates the need for "trusted intermediaries" [3] and, of course, court enforcement [4].

Smart contracts' performance protocols take many forms, as there are countless ways to embed promises in technology so as to make breach infeasible or unduly expensive. They include the controller and motors of the "humble vending machine" [5], embedding, as they do, the promise of the vendor to deliver a Mr. Pibb to anyone inserting a dollar. They include the blockchain-dwelling bytecode of an evergreen loan contract on Ethereum, embedding, as it does, a creditor's promise to issue a new cryptocurrency loan to the debtor who repays a prior one [6].

Observe what these examples share: security. When promises are embedded in technology, one (perhaps the only) way to breach them is to disrupt that technology. Most smart contracts include security measures aimed at deterring this type of breach. To breach the vending machine's smart contract, you must break into its lockbox. To breach the blockchain loan contract, you must compromise the blockchain's consensus protocol. In this manner, security mechanisms form the archstone in the promise of smart contracts to transcend analog contracts. The problem, however, is that securing contracts against disruption for the purpose of breach often means securing them against disruption of any sort. And that is not always a desirable result.

The fact is, as "performance unfolds, circumstances change, often unforeseeably" [7]. External events like price shifts may degrade a contract's value in the eyes of the parties. It may come to light that there is a typo in the contract, or that one party was defrauded during its creation. When such events arise, the parties — and sometimes courts and or even the public — may find themselves wanting the contract to be performed differently (or not at all). This is why contract law has a well-honed set of tools for undoing and altering contracts, including termination and rescission (for undoing contracts) as well as modification and reformation (for altering contracts).

Unfortunately, these traditional tools often fail when applied to smart contracts. True, they successfully undo the legal agreement that a smart contract manifests. If these tools are exercised, no court will enforce the agreement. The problem, of course, is that technology still might. What is needed, then, is to define new standards for these tools as applied to smart contracts, making sure they remain robust for this new medium. That is the goal of this paper.

2 Termination and Rescission of Smart Contracts

2.1 Termination and Rescission Generally

"Rescission", the 1912 edition of *Black's Law Dictionary* tells us, "is where a contract is canceled, annulled, or abrogated by the parties, or one of them" [8]. Importantly, this definition turns out to be somewhat half-baked, with another corner of Henry Campbell Black's own oeuvre — 1916's *A Treatise On the Rescission of Contracts and Cancellation of Written Instruments* — cautioning that "[t]o rescind a contract is not merely to terminate it" and "release the parties from further obligation" but to "restore the parties to the relative positions which they would have occupied if no such contract had ever been made" [9]. After highlighting this restorative aspect of rescission, the latter text outlines three situations in which rescission may be implemented. These, like the definition that precedes them, have endured:

First, rescission may be implemented when "a right to take this action [is] reserved to either or both of the parties in the contract itself [9]. If reserved, such a right "may then be exercised without other grounds for it than the mere will of the party rescinding" [9]. Today, this is called "termination by right" ("Termination by Right").

Second, there may be a "rescission by the mutual agreement of the parties to the contract" [9]. This is, "in effect the discharge of both parties from the legal obligations admittedly existing thereunder, by a subsequent agreement made before the complete performance of the original contract" [9]. In modern times, this is called "mutual rescission" or "rescission by agreement" ("Rescission by Agreement").

Third, "one of the parties may declare a rescission of the contract, without the constant of the other … if a legally sufficient ground therefor exists, such, for instance, as fraud, false representations, [unilateral] mistake, duress, or infancy" [9]. The rescinding party may then ask a court to "set aside" the contract "by the equity decree" [10]. The modern label for this is simply "rescission" ("Rescission by Court").

Let us examine each of these three versions of rescission as applied to smart contracts, sketching new standards for each as we proceed:

2.2 Termination by Right

At law, Termination by Right is implemented passively: it bars future breach of contract actions [11]. For smart contracts, unfortunately, this approach often fails. If a smart contract is terminated at law, and nothing more is done, the smart contract will still automatically perform ("auto-perform") the parties' promises (as it is designed to do), negating the termination. Accordingly, the first standard we will set for smart contract Termination by Right is that, when the right is exercised, auto-performance indeed ceases. To permit otherwise means the termination is an empty gesture.

A second standard is this: the smart contract must ensure that Termination by Right is implemented if and only if the party holding the right exercises it. No other party may initiate termination. To permit otherwise, again, undermines the goals of contract law by rewarding opportunistic breach by non-right holding parties.

A third standard is this: echoing Black's emphasis on restoration, before implementing a Termination by Right, the smart contract's machinery must ensure that all partial performance that has occurred is compensated. For example, partial payments sent by either party must be returned. If this is not done, then parties will resort to courts to enforce restitution of that partial performance, undoing one of the primary efficiency benefits of smart contracts.

A fourth standard is this: the smart contract's machinery must ensure that all conditions placed on the termination right are met before termination is implemented. For example, if payment of a termination fee is a condition of the right, the contract must pay such a fee to the appropriate party (or otherwise ensure that it is paid) before initiating termination. To permit otherwise undermines the aim of contract law by rewarding opportunistic breach by the right holder. To summarize:

- Smart contract Termination by Right halts all auto-performance;
- Smart contract Termination by Right is enabled if and only if the party holding that right exercises it;
- Smart contract Termination by Right automatically compensates partial performance;
- Smart contract Termination by Right is enabled if and only if any termination conditions are satisfied.

2.3 Rescission by Agreement

Rescission, like termination, is implemented passively at law: when there has been a valid rescission, there is "no longer a cause of action for breach" [12]. Again, for smart contracts, this passive approach fails. Accordingly, our first standard for smart contract Rescission by Agreement is the same as our first standard for smart contract Termination by Right: automated performance must be halted.

Our second stand is unique to Rescission by Agreement: unlike with Termination by Right, the power to rescind a smart contract by mutual agreement may not lie with one party. An agreement to rescind, like the initial contract, takes the "form of an offer by one and an acceptance by the other" [13]. So this brand of smart contract rescission must be conditioned, by the smart contract, on mutual agreement: an offer to rescind by one party and acceptance of that offer by all other parties. To allow otherwise contravenes the goals of contract law by encouraging opportunistic breach.

Our final standard for smart contract rescission is this: smart contract Rescission by Agreement, like smart contract Termination by Right, should include restoration of any partial performance. To summarize:

- Smart contract Rescission by Agreement halts all auto-performance;
- Smart contract Rescission by Agreement is enabled and if all parties mutually agree to it;
- Smart contract Rescission by Agreement automatically compensates partial performance.

2.4 Rescission by Court

Of the grounds for rescission, unilateral mistake (when one party *thinks* the smart contract does one thing, while the other party *knows* it does another) is of particular interest to smart contracts. Due to the introduction of code to the agreement-making process, unilateral mistake may be a greater danger than ever before. Few feel confident reading "legalese"; even fewer feel confident reading code.

In light of this, our first standard is a familiar one: when there is a unilateral mistake — or when any of the other bases for Rescission by Court exist — and a court orders a smart contract rescinded, auto-performance must cease.

Our second standard is more unique: the power to order Rescission by Court may only lie with and be exercised by the appropriate court. Neither party may have the power to jeopardize that right. Naturally, that would undermine the goals of contract law (by encouraging opportunistic breach).

Our third demand is this: upon rescission by a court, restoration must occur, just as it would in the case of Termination by Right or Rescission by Agreement. If partial performance is not automatically compensated, parties may petition the court to enforce restitution of that performance, erasing one of the primary efficiency benefits of smart contracts. To summarize:

- Smart contract Rescission by Court halts all auto-performance;
- Smart contract Rescission by Court is enabled if and only if triggered by an appropriate court;
- Smart contract Rescission by Court automatically compensates partial performance.

3 Modification and Reformation of Smart Contracts

3.1 Modification and Reformation of Smart Contracts

Sometimes, we do not wish to wholly discard an agreement, but merely wish to alter *some* of its terms. Such alteration provides an "efficient mechanism for changing agreements in response to altered circumstances … saving a deal that would otherwise have ended in an inefficient breach" [14]. Like the undoing of a contract, the alteration of a contracts comes in three flavors:

The first is where "[u]nilateral-modification clauses give one party the unfettered right to amend … the underlying contract, often with neither notice to, nor consent from, the other party [15]. This is called "modification by right" ("Modification by Right"). (Note that some courts will uphold this right, while others will not [16]).

Second, contracting parties have a well-established right "to modify their original contract by mutual agreement" [17]. Such a modification is itself a contract [18] and must be based on mutual assent and supported by its own consideration [19]. This is referred to as "modification by agreement" ("Modification by Agreement").

Third, a court may, in some cases, order a modification of a contract even over the objections of one or more parties. It may do so based on three grounds: mistakes mutual to all parties [20], fraud [21], and "unconscionable" terms — i.e., terms born out of "an absence of meaningful choice" for one party and "unreasonably favorable to the other" [22]. This is type of modification is called "reformation" ("Reformation").

3.2 Modification by Right

If undoing a smart contract calls for an axe — a blanket action turning the entire contract off all at once — modifying it calls for a scalpel. Specifically, modification must halt auto-performance of only the terms that are intended to be modified while simultaneously initiating auto-performance of the new versions intended to replace them.

With that said, our first standard is this: upon Modification by Right of a smart contract term, auto-performance of that term's original iteration must cease, while auto-performance of its new iteration must, concurrently, initialize.

Our second standard is a familiar one: modification of a term can be initiated if and only if a party holding the right to modify that term wills it.

Our third standard is also a familiar one: if the modification is conditioned on the occurrence of events, such as the payment of a modification fee, those events must occur before modification can take place.

Our final standard is a twist on a standard previously forth for the ways of undoing smart contracts: a smart contract must automatically compensate for any partial performance that has occurred and which is tied to obligations embedded in the terms being removed during modification. (It need not compensate for partial performance of any terms that, though modified, remain active; those terms will be compensated through the performance of the contract.) To summarize:

- Smart contract Modification by Right simultaneously halts auto-performance of original, modified terms and instantiate auto-performance of new ones;
- Smart contract Modification by Right is enabled if and only if the party holding the right exercises it;

- Smart contract Modification by Right automatically compensates partial performance *of deleted terms*;
- Smart contract Modification by Right is enabled if and only if any modification conditions are satisfied.

3.3 Modification by Agreement

The issues faced when implementing Modification by Agreement resemble those faced during Modification by Right. So our standards are similar. The key difference is that a Modification by Agreement must be approved by all parties. To summarize:

- Smart contract Modification by Agreement simultaneously halts auto-performance of original, modified terms and instantiate auto-performance of new ones;
- Smart contract Modification by Agreement is enabled if and only if all parties mutually agree to it;
- Smart contract Modification by Agreement automatically compensates partial performance *of deleted terms*.

3.4 Reformation

Some grounds for reformation are of special interest to smart contracts. That includes mutual mistake, which covers the so-called "scrivener's error", an "accidental deviation from the parties' agreement" made while recording the agreement in writing [19]. In smart contracts, the risk of this error is high because of, again, the introduction of code to contracting. Fraud and unconscionability are high risks for the same reason: code-savvy parties are in a position to defraud or force unconscionable terms on code-naive parties. For these reasons, Reformation of smart contracts is likely to occur.

Our first standard for Reformation is familiar: it must halt auto-performance of the original versions of modified terms and instantiates auto-performance of the new version of modified terms.

Second, the power to reform the contract, like the power for Rescission by Court, must lie strictly with the court. And our third standard is a familiar one as well: once triggered, the Reformation must compensate for the partial performance of any terms that are being deleted. To summarize:

- Smart contract Reformation simultaneously halts auto-performance of original, reformed terms and instantiate auto-performance of new ones;
- Smart contract Reformation is enabled if and only if triggered by an appropriate court;
- Smart contract Reformation automatically compensates partial performance *of deleted terms*.

4 Testing Our New Standards on Ethereum

4.1 Ethereum Generally

Let's put our new standards for altering and undoing smart contracts to the test on an existing smart contract platform: Ethereum. Can smart contract alteration and undoing on this platform meet our standards? How?

Ethereum, "arguably the most ambitious crypto-ledger project," [25] is built on a blockchain. Ethereum blockchain stores both transaction data (concerning its native cryptocurrency, Ether) and the code of computer programs called, for better or for worse [26], "contracts." The code for these contracts is injected onto the blockchain when a personal account sends contract code in the data field of an unaddressed transaction. After this, the contract is added to a block and assigned an address, at which point its code becomes immutable [27]. Importantly, what is not immutable is the contract's state. Specifically, the nodes in the Ethereum network, besides being able to add transactions to the ledger, also run contract code and maintain and adjust contract states in a virtual machine they all host, the Ethereum Virtual Machine.

Contracts on Ethereum can hold balances of Ether. Like objects in object oriented programming, they can also have variables and functions that, if called, adjust those variables or do other nifty things, like send Ether to other contracts or accounts on Ethereum. Note that these functions cannot "wake" on their own and, in order to execute, must be called (by parties to the contract, third parties, or other contracts).

One of Ethereum's high level languages, Solidity, is a cross "between JavaScript and C++ but with a number of syntactic additions to make it suitable for writing contracts within Ethereum" [28] and is what we will use to prototype below.

4.2 Undoing Contracts on Ethereum

There are at least two ways to *undo* contracts (i.e., implement Termination by Right, Rescission by Agreement, or Rescission by Court) on Ethereum. The first, the global selfdestruct function, is easy to implement and effective. That said, it is also a blunt instrument, lacking the nuance of the second way, which is to turn the entire contract "off" at the function level using a combination of Solidity's modifiers and enums.

Undoing Contracts on Ethereum Using Selfdestruct. As stated, Ethereum contract code, once on the blockchain, cannot be altered. But it can be deleted. The global selfdestruct function, if called from inside a contract, sends the contract's Ether balance to the address this function takes as its sole argument, then deletes the contract's code from the blockchain going forward. This means the contract's functions cannot be called. Since Ethereum contract functions cannot self-wake, this halts auto-performance and thus satisfies the first (shared) standard we set for smart contract Termination by Right, Rescission by Agreement, and Rescission by Court.

It is also easy, on Ethereum, to satisfy the second (shared) standard for each of these tools by granting the power to selfdestruct a contract only to those entities that should have it. If that is a single party (which is the case for Termination by Right and Rescission by Court), this can be done by wrapping selfdestruct function inside a

conditional statement that checks if the address calling it belongs to the rightful exerciser:

```
function terminate() {
if (partyWithTerminationRightOrCourt == msg.sender) {
selfdestruct( partyReceivingContractBalance ); }}
```

If multiple parties must approve the undoing, as in Rescission by Agreement, there are a few ways to achieve this. One is to use Solidity's modifiers and enums (user defined types) to create states that log the consent of parties and then to throw exceptions when selfdestruct is called and those states do not reflect the necessary values:

```
contract Undoable {
address partyWithTerminationRight;
address partyWithoutTerminationRight;

enum State {RescissionByAgreementSuggested}
State public state;

modifier inState(State _state) {
if (state != _state) throw;
       _ }

function suggestRescissionByAgreement() {
  if ( partyWithoutTerminationRight == msg.sender) {
state = State.RescissionByAgreementSuggested; }}

function approveRescissionByAgreement()
inState(State.RescissionByAgreementSuggested) {
  if (partyWithTerminationRight == msg.sender) {
selfdestruct(partyWithoutTerminationRight); }}}
```

Next is the third standard, shared by each of these tools, that demands that any partial performance that has occurred be compensated automatically before the contract is undone. With selfdestruct, this is easy to engineer. All that is needed is a variable that tracks the level of performance, a function that lets one party suggest a new value for that variable, and a second function that lets the counterparty approve the new value. When the contract is undone, the latest value for the variable will be paid out.

Termination by Right is the only version of contract undoing with a fourth standard. It comes in many shapes, but we can address the simplest here. This is where the right is conditioned on the payment of a termination fee. To satisfy this standard, we can use a much more streamlined version of this approach used to satisfy the third standard.

Here is contract code that ties together all of the above, satisfying the conditions for our three methods of undoing contracts by creating functions for Termination by Right, Rescission by Agreement, and Rescission by Court, giving the power over those

functions to the right parties, and paying out termination and partial performance fees when required. To simplify things, let us assume partial performance is only possible for one party (e.g., it is a labor contract that the hirer has endowed with the full payment, such that partial performance is only an issue for the laborer):

```
contract Undoable {
address partyWithTerminationRight;
address partyWithoutTerminationRight;
address partialPerformanceApprover;
address court;
uint terminationFee;
uint suggestedPartialPerformanceCompensation;

enum State {RescissionByAgreementSuggested,
PartialPerformanceCompensationSuggested,
PartialPerformanceCompensationApproved}
State public state;

modifier inState(State _state) {
if (state != _state) throw;
       _ }

function terminateOrRescind(uint _terminationFee, address
_partyWithTerminationRight, address
_partyWithoutTerminationRight, address _court, address
_partialPerformanceApprover) {
terminationFee = _terminationFee;
partyWithTerminationRight = _partyWithTerminationRight;
partyWithoutTerminationRight =
_partyWithoutTerminationRight;
partialPerformanceApprover = _partialPerformanceApprover;
court = _court;}

function suggestPartialPerformanceCompensation(uint
_suggestedPartialPerformanceCompensation) {
if ( partyWithoutTerminationRight == msg.sender) {
suggestedPartialPerformanceCompensation =
_suggestedPartialPerformanceCompensation;
state = State.PartialPerformanceCompensationSuggested;}}

function approvePartialPerformanceCompensation()
inState( State.PartialPerformanceCompensationSuggested){
if (partyWithTerminationRight == msg.sender) {
state = State.PartialPerformanceCompensationApproved ;}}
```

```
function terminate()
inState(State.PartialPerformanceCompensationApproved){
if (partyWithTerminationRight == msg.sender) {
    partyWithoutTerminationRight.send(terminationFee);
    partyWithoutTerminationRight.send(
suggestedPartialPerformanceCompensation);
selfdestruct(partyWithoutTerminationRight);}}

function rescindByCourt()
inState(State.PartialPerformanceCompensationApproved){
if (court == msg.sender) {
    partyWithoutTerminationRight.send(
suggestedPartialPerformanceCompensation);
selfdestruct(partyWithoutTerminationRight); }}

function suggestRescissionByAgreement()
inState(State.PartialPerformanceCompensationApproved){
if (partyWithoutTerminationRight == msg.sender) {
state = State.RescissionByAgreementSuggested; }}

function approveRescissionByAgreement()
inState(State.RescissionByAgreementSuggested){
if (partyWithTerminationRight == msg.sender) {
    partyWithoutTerminationRight.send(
    suggestedPartialPerformanceCompensation);
selfdestruct(partyWithoutTerminationRight); }}}
```

While this code does not cover edge cases (such as the situation where conditions placed upon Termination by Right represent the occurrence of real world events), we have shown that our standards can reasonably be applied — and to some extent, satisifed — using one of the methods for undoing smart contracts (selfdestruct) on Ethereum. Now let us repeat the process for a second (and arguably superior) method for undoing smart contracts on the same platform:

Undoing Contracts on Ethereum Using Modifiers and Enums. The selfdestruct function is a convenient "one-stop" solution for undoing contracts. But Solidity's modifiers and enums are a more nuanced tool for this — one that, as we will see later, meshes well with existing tools for altering contracts.

We used modifiers and enums above, in conjunction with selfdestruct. We can extend the same strategy to implement Termination by Right, Rescission by Agreement, and Rescission by Court without selfdestruct. Specifically, we can create two states: one for

a contract that has been undone — let's call it ContractUndone — and one for a contract that is not undone — let's call it ContractNotUndone. Upon instantiation, we can set the state as ContractNotUndone. Then, we can create a function that enables the state to be toggled to ContractUndone (but not toggled in the other direction). Lastly, we can can cause all other functions to throw if the ContractUndone state exists. This will halt performance of the contract, satisfying the first standard for our three methods of undoing contracts. Then we can also satisfy the other standards for undoing smart contracts in the same ways we set forth above for selfdestruct.

4.3 Modifying Contracts on Ethereum

Modifying contracts (i.e., implementing Modification by Right, Modification by Agreement, and Reformation) on Ethereum is more nuanced than undoing contracts on Ethereum. Roughly speaking, there are three ways to achieve modification on Ethereum: modification of *variable*-captured terms, deletion of *function*-captured terms, and addition or alteration of *function*-captured terms.

Modifying Variable-Captured Terms. Contract terms like price (or labor hours, etc.) will often be captured as variables in smart contract code. When this is the case, modifying these terms is simple as assigning a new value to the variable using a set-type function. If such a function exists, this method of modification satisfies the first (shared) standard of our three flavors of modifying contracts: it halts performance of the old term and instantiates performance of the new one. If the set function is narrowly tailored to this variable, then this method of modification also satisfies the second standard of Termination by Right: that the scope must be hard-coded into the smart contract during formation. Satisfying the remaining standards for Modification by Right, Modification by Agreement, and Reformation can all be accomplished in much the same way there were accomplished above, for Termination by Right, Rescission by Agreement, and Rescission by Court.

Modifying Function-Captured Terms. Sometimes, contract terms are captured by functions and not variables. In that case, modification means deleting, adding, or swapping the relevant function(s). This must be handled differently than variable-level modification because, while variables can be changed freely, the functions in an Ethereum contract code are immutable once it is issued to the blockchain.

Deleting Function-Captured Terms. Of the types of function-level changes, the easiest to implement is deletion: i.e., subtraction of terms. For that, we can recycle the approach taken for Termination by Right, Rescission by Agreement, and Rescission by

Court: using modifiers and enums to create and toggle states, then causing functions to throw exceptions if the states do not exist. Using this method, we can build functions that can be turned off, on demand, if the parties agree to a deletion-style modification. This will halt performance much as it did above, satisfying the first standard for our three ways of undoing a contract. Beyond that, the remaining standards can be satisfied much as they were above for variable-captured functions.

Adding or Modifying Function-Captured Terms. Adding wholly new functions and replacing existing functions is accomplished in a similar fashion. The difference between the two that, if a function is being replaced, the initial version of it must also be turned off. (This can be accomplished using the methods described above for deletion of functions.) On Ethereum, there are at least two ways to add or swap functions in a contract. Both demand a bit of prognostication.

The first is to use modifiers and enums can be used to turn functions "off", they can be used to turn functions "on". Of course, in order to be turned on, those functions must be in the contract to begin with. Since contract code is immutable after initialization, this means functions that the parties suspect they may later wish to turn "on" during a modification must be included in the initial contract *in an "off" state*. That said, if this can be accomplished, then the standards for all three flavors of contract modification can be satisfied much as they would be for variable-captured functions.

A second way to add or modify function-captured terms — and, seemingly, the one endorsed by Ethereum's architects [6] — is to create, at the outset, satellite contracts that capture certain function-terms. The addresses of these satellite contracts can be stored in address variables or an arrays of address variables in the central contract. Using these pointers, the central contract can to call out to the satellite contracts when it needs to reference certain terms. If this is architected properly, modifying function-terms is as simply as changing the pointers.

As an example, suppose the parties wish to build flexibility into their price term. They can initialize a central contract with pointers to a satellite price calculation contract. Changing the price calculation method (e.g., swapping price datafeeds or formulas) is as simple as changing the pointers in the central contract. The code for such a contract might look like the code below, which contains a pair of functions that let one party suggest a new satellite contract and let the other approve the suggestion before making the change (note that, in order to call an outside contract's functions on Ethereum, the code for the outside contract must appear in the code for your present contract):

```
contract Satellite {
function returnPrice() returns (uint _price){
//calculate price
}}
contract Modifiable {
uint price;
address party1;
address party2;
address satelliteAddress;
address suggestedSatelliteAddress;

function setPrice(){
Satellite m = Satellite(satelliteAddress);
price = m.returnPrice();}

enum State {ModificationSuggested,ModificationApproved}
State public state;

modifier inState(State _state) {
if (state != _state) throw;
       _ }

function suggestModification(address
_suggestedSatelliteAddress){
if (party1 ==  msg.sender){
suggestedSatelliteAddress = _suggestedSatelliteAddress;
state = State.ModificationSuggested;}}

function approveModification()
inState(State.ModificationSuggested){
if (party2 == msg.sender){
satelliteAddress = suggestedSatelliteAddress;}}}
```

As is, this contract satisfies the first standard for all three flavors of modifying contracts; by de-linking the original satellite contract and linking the new one, it simultaneously halts auto-performance of the original versions of modified terms and instantiates auto-performance of the new versions. If it contains code to ensures that the party initiating the pointer swap is the correct one and additionally contains code that tracks and compensates partial performance in the event of a modification, then it can satisfy the second and third conditions of all three flavors of modification as well. Finally, it can satisfy the fourth condition of Modification by Right by additionally including code that prohibits modification unless certain conditions have been met.

5 Conclusion and Future Work

Contract law has developed a well-honed and necessary set of tools for altering and undoing contracts. Unfortunately, these tools often fail when applied to smart contracts. It is therefore crucial to define a new set of standards against which we can create a similar set of tools for altering and undoing smart contracts. These standards should be drawn from the principals of contract law but work for the new technology. We have sketched such standards. Further, by applying these standards to the present methods for altering and undoing smart contracts on Ethereum, we have proven that there is value to such a framework. Let the smart contract community take note. It is essential that the architects of this new technology, like the architects of contracts, create viable ways to alter and undo them.

References

1. Posner, R.: Economic Analysis of Law. Little Brown and Co., Boston (1986)
2. Szabo, N.: Smart Contracts: Building Blocks for Digital Markets (1996)
3. Juels, A., Kosba, A., Shi, E.: The Ring of Gyges: Investigating the Future of Criminal Smart Contracts (2015)
4. Szabo, N.: Smart Contracts (1994)
5. Szabo, N.: The Idea of Smart Contracts (1997)
6. Buterin, V.: Ethereum White Paper (2014)
7. Posner, R.: Let us never blame a contract breaker. Mich. Law Rev. **107**, 1360 (2009)
8. Black, H.C.: Black's Law Dictionary, p. 1025 (1910)
9. Black, H.C.: A Treatise on the Rescission of Contracts and Cancellation of Written Instruments, vol. 1 (1916)
10. Koford, H.S.: Recessions at law and in equity. Calif. Law Rev. **36**, 608 (1948)
11. Atlas Trucking v. City of Lompoc, S224878, 2015 Cal. LEXIS 2165 (Sup. Ct. Cal., 15 April 2015)
12. Great American Ins. v. General Builders, 934 p. 2d 257, 262 n. 6 (Nev. 1997)
13. Corbin, A.L.: Corbin on Contracts, vol. 5A (1964)
14. Russell, I.S.: Reinventing the deal: a sequential approach to analyzing claims for enforcement of modified sales contracts. Fla. Law Rev. **53**, 51 (2001)
15. DeMichele, M.L., Bales, R.A.: Unilateral-modification provisions in employment arbitration agreements. Hofstra Employ. Law J. **24**, 64 (2006)
16. Carey v. 24 Hour Fitness, USA, Inc., 669 F.3d 202 (5th Cir. 2012)
17. Christine, C.: Contracts as bilateral commitments: a new perspective on contract modification. J. Legal Stud. **26**, 204 (1997)
18. Hillman, R.A.: A study of uniform commercial code methodology: contract modification under article two. N. C. Law Rev. **59**, 339 (1981)
19. Williston, S., Lord, R.: Williston on Contracts (1992)
20. Moffett, Hodgkins & Clarke Co. v. Rochester, 178 U.S. 373, 385 (1900)
21. Link v. Kroenke, 909 S.W.2d 740, 745 (Mo. App. W.D. 1995)
22. Williams v. Walker-Thomas Furniture Co., 350 F.2d 445, 449 (D.C. Cir. 1965)
23. The Great Chain of Being Sure About Things. The Economist (2015)

24. Marino, B.: https://medium.com/@ConsenSys/unpacking-the-term-smart-contract-e63238f7db65
25. Delmolino, K., Arnett, M., Kosba, A., Miller, A., Shi, E.: Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab (2015)
26. Wood, G.: https://github.com/ethereum/wiki/wiki/Solidity,-Docs-and-ABI