Legal Framework For Crypto-Ledger Transactions

I. Integration of Law and Computer Technology

The marriage of law and technology in the late 20th and early 21st century can be divided into three phases.

1. Phase one involved digitizing information itself: taking paper and ink and storing as computer readable information. That phase is well under way: copies of cases, statutes, and regulations have been available on-line for decades in large databases, accessible at first for a fee, and now mostly for free. Yet, there is a big difference between digitized information (that can be stored and displayed on a computer) and semantically meaningful information (that can be also processed and understood by a computer).
2. Phase two has involved automating decision-making. Much legal informatics research to date has focused on transforming legal provisions into computer code. This is a hard project for many reasons, including the ambiguity of the human language and the need for legal norms to be flexible and fact dependent. Despite these challenges (and limitations) government institutions and businesses worldwide increasingly rely on rule-based representations of specific knowledge domains - such as health care, tax and financial regulations - for automated decision-making (see e.g. specific tools for taxation, accounting and credit-score assessment).
3. Phase three - which is just beginning - involves self-enforcing smart contracts. We are moving quickly towards a system where technology will not only aid in decision-making but will also be employed to enforce rules. A notable example can be found in Digital Rights Management systems which incorporate the provisions of copyright law into technological measures of protection. But technology can also implement its own technical rules, which do not reflect any underlying legal rule. In order to be legally enforceable, these rules need to be wrapped-up into a legal framework capable of making smart contracts actionable under the law. A lot of work has yet to be done in that area.

II. Smart Contracts: self-enforceability vs. legal enforceability

Smart contracts were first described by Nick Szabo in the late 1990s. He envisioned placing contracts into code that could be both "trustless" and self-enforcing, enhancing efficiency and removing ambiguity from contractual relationship. Beyond increased speed and efficiency, an important benefit of smart contracts over traditional contracts is the lack of textual ambiguity. In the long run, this might reduce the needs for canons of construction and other textual interpretation techniques -- although factual ambiguity (i.e. did a real world event happen or not) will obviously remain.

Smart contracts also eliminate the need for trust amongst the parties, who can be sure that the contract will be performed exactly as agreed. Indeed, as opposed to traditional legal contracts, smart contracts are always and necessarily deterministic - i.e. all possible outcomes of the contract (including penalties

for breach of contract) must be explicitly stipulated in advance -- something akin to what liquidated damages clauses and letters of credit do in the paper world.

Discussion is currently very focused on the technical aspects of smart contract deployment and their implementation within a particular technological framework. Smart contract proponents claim that many contractual clauses could be made partially or fully self-executing, self-enforcing, or both. Smart contracts aim to provide security superior to traditional contract law and to reduce other transaction costs associated with contracting. Some even propose a future where self-enforcing algorithmic law would supplant traditional law.

Yet, many people forget that these applications are meant to operate in a world that is regulated by traditional rules of law. While smart contracts are increasingly able to handle complex deal logics, many kinds of transactions do - eventually - have to interface with the "real world". It is at those "choke points" that the legal system will ultimately have a say in the context of a breach.

In this regard, one important question is to determine whether smart contracts are in fact actionable in the real world. While they can be regarded, at their core, as a written contract drafted in a computer language, it is not clear - at this date - whether their code is "legally binding" upon the parties interacting with these contracts.

This is a critical issue because the provisions of a smart-contract are self-enforceable only to the extent that they account for all possible deviations from the agreed-upon terms - e.g. by implementing a complex system of collaterals, which might considerably increase the complexity of these contracts. Were they enforceable under the law, smart-contracts could be drafted in a much simpler manner - e.g. by only incorporating a basic set of conditions into the code and subsequently relying on the legal system in order to enforce these conditions, in the event of a breach. Legal enforceability would essentially allows for only the main-case set to be handled by smart-contracts, leaving the edge-case set to be handled by the courts.[1]

Most importantly, in many real-world situations, contracts are performed without the need to be enforced by law, as the threat of one party resorting to the legal system is sufficient for the other party to comply with the contracted terms. In order to be as effective as their traditional counterparts, smart contracts must therefore also be actionable in the real world. This might, of course, requires them to comply with all the standard formalities required for a court to enforce a contract under the law.

We present here a way to extend the reach of cryptoledger-based transactions by having smart contracts render into conventional legal documents. This allows for the smart contracts (as the administration layer) to remain simple, while relying on the legal contracts (as the enforcement layer) to handle the edge cases.

[1] Think of a stock market. There are deterministic 'dry' elements, but there are also legal 'wet' rules. While virtually all transactions are handled without express reference to these rules, occasionally, something unexpected happens and the managers of the exchange or the lawyers work have to work it out by relying on the legal provisions.

III. Common Accord: turning smart contracts into legally enforceable contracts

CommonAccord (http://commonaccord.org) plans to "open source" and automate the drafting of legal documents by creating a global template system of codified legal texts (*i.e.* something akin to the Incoterms rules for commercial transactions). The goal is to make the documents so modular that much of the text disappears, leaving parties with only specific deal points and clear relationships. These relationships can be 'rendered' at any time into full legal documents, for verification and enforcement. Technically, this is a data-model for text, an extremely simple and expandable data-model that consists of a series of nested lists that render into texts. The texts can be improved, extended and forked by the community. As such, *CommonAccord* is expected to play the same role in facilitating and accelerating collaboration on legal texts as git has played for code.
For more details on the *CommonAccord* data model, see:
http://commonaccord.org and http://beta.commonaccord.org/wiki/Main_Page.

Examples of use cases (switch between "Document" and "Source" to see the model in action)

- ❏ Frameworks of trust: NDAs; Master Agreements
- ❏ Licenses; Contribution Agreements
- ❏ Crowd-funding (Eris Assurance contract)

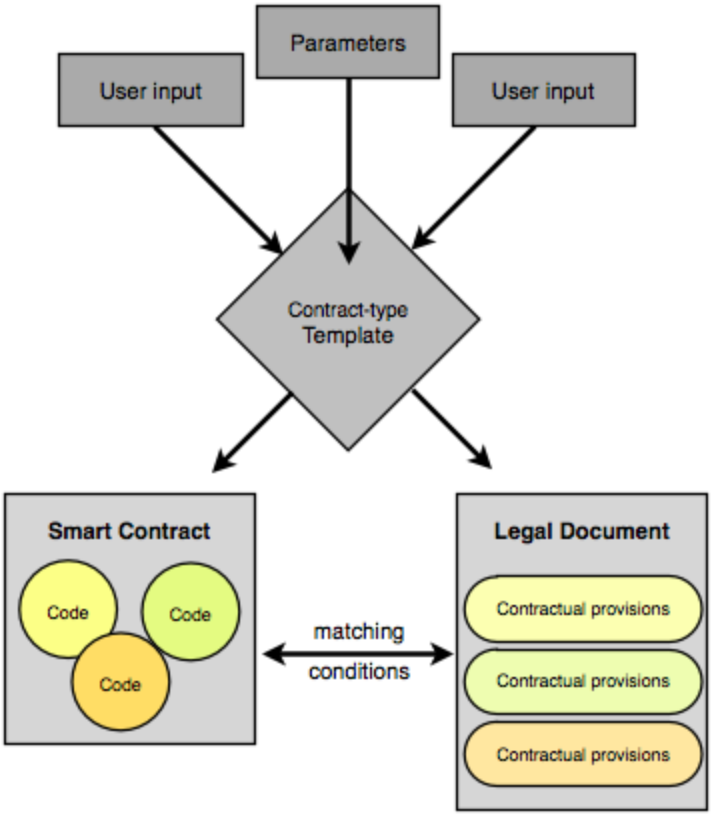
CommonAccord's list-based template system provides a flexible, robust format for codifying and rendering legal texts from code. This can be achieved in at least two ways:

- Manual Matching
  CommonAccord's framework can be used to produce a legal document (whose torrent/hash can be incorporated into the blockchain as a proof of existence) to tie every smart contract to a real world one. Through inheritance, *CommonAccord*'s data model can express any smart contract as simple nested lists that can be rendered into legal documents, according to the selected jurisdiction and circumstances. This is a manual process that requires matching the prototype of the smart contract with the specific *CommonAccord* template that goes along with it.

- Automated Matching
  CommonAccord's framework can be used as an API that returns matching snippets of code (both legal and technical code) whenever someone submits a request for a particular template, together with the necessary parameters requested by the template. This requires creating a database of independent 'contract modules' linking standardized code-snippets to their corresponding contractual provisions, which can be combined together into a collection of 'contract-type templates' (i.e. documents instructing how these modules can or should be incorporated into either a smart contract or a legal document).

  Traditional contracts often share common "boilerplate" elements, and smart contracts are no different. *CommonAccord*'s inheritance system enables code reuse; this provides more convenience, as well as increased legal certainty. Many contracts will have relatively simple and easy to understand logic built on top of well-known and widely used modules. Modules

could encompass basic functionality, or more advanced features such as a standard auction, escrow, or bond implementation.



The goal is to implement a modular template system which, after having been fed with specific parameters, can be rendered into both technical 'dry' code (i.e. self-contained code snippets) and legal 'wet' code (i.e. specific contractual provisions translating the code snippets into *legalese*) whose overall structure can be established directly through the *CommonAccord* template system. Ideally, this would allow for the dynamic creation of sophisticated and highly customized smart-contracts, which are automatically rendered into a legal document enforceable between the parties.

Of course, both versions of the contract - the smart-contract, written into code, and the legal contract, written in human language - should cross-reference each other. The legal contract should include a reference to the smart-contract code (i.e. its blockchain address), whereas the smart-contract should incorporate a reference to the hash of the digitally-signed legal document, which should remain available on a decentralized storage system (which can be either public or private, according to the confidentiality rules). Those two versions may differ in certain regards. While there should be a 1:1 correspondence between the code and the specific template texts, the relationships established between these templates are uniquely defined in the code. As a result, the code version might be extremely unambiguous, but the generated legal text less so.

The question therefore arises as to which version should be relied upon in court in order to settle a dispute between the parties. Considering the current courts' inability to understand computer code, it will most likely be the legal document. Yet, although it might be ignored at first, incorporating a reference to the smart-contract into the real-world legal contract might - in the long run - expand the opportunities for smart-contract enforcement, insofar as it might lead courts to consider the smart-contract code as evidence of how the legal contract should be effectively construed.