All Smart Contracts Are Ambiguous

James Grimmelmann

Princeton CITP
November 6, 2018

In this talk

- Legal contracts vs. smart contracts
- Ambiguity and interpretation
- Legal governance vs. blockchain governance

Smart contract basics

Key blockchain ideas

- A transactional ledger that …
  - … is cryptographically secure
  - … does not require a centralized recordkeeper
  - … uses incentives to ensure consensus
- Reduce need to trust:
  - Your counterparties
  - Centralized recordkeepers

Key smart contract ideas

- Embed terms in hardware/software:
  - Automatic interpretation
  - Automatic monitoring
  - Automatic enforcement
- Reduce need to trust:
  - Your counterparties
  - The legal system

Blockchain + smart contract

- “X on the blockchain” is often silly
    - But not for X = smart contract
- Ethereum-style: specify a virtual machine
    - VM primitives can affect shared blockchain resources
    - Blockchain transactions update the VM
    - Blockchain protocol forces VM consistency
    - VM specification provides a programming model with desired security properties

Legal contracts
vs.
Smart contracts

Three sources of uncertainty

(1) Natural language is *ambiguous*

- A contract's meaning might be unclear

(2) Courts make *mistakes*

- A court might fail to enforce the contract

(3) Parties can escape *enforcement*

- A party might ignore a court's judgment

(1) Ambiguity

- *Frigalament v. B.N.S.*: what does "chicken" mean in a contract for the sale of 75,000 pounds of "US Fresh Frozen Chicken, Grade A, Government Inspected"?
  - Buyer: "a young chicken suitable for broiling and frying"
  - Seller: "any bird of that genus"
- Saying "chicken suitable for broiling" doesn't help
  - What's "suitable"? What's "broiling"?

(2) Mistakes

- Litigation is uncertain
  - The *parties* might agree on a meaning when they write a contract … but one of them might later persuade a court otherwise
  - Or there may be external pressure on a court
- Litigation is slow
- Litigation is expensive

(3) Enforcement

- Some defendants are "judgment-proof"
  - They have no assets to pay a damage award
  - They have fled the jurisdiction
  - They have vanished into thin air
- You can't always get your property back
  - It might have been destroyed or hidden
  - It might have been sold onwards

The fix:
smart contracts on a blockchain

(1) Write the contract as a computer program

- The program is *unambiguous*

(2) Put the program on a blockchain

- Miners will collectively *prevent mistakes*

(3) Give it control of the relevant assets

- Results are enforced *automatically*

(1) Ambiguity

- The meaning of “chicken” is a social fact
  - There are dictionaries, patterns or speech, usage in multiple trades, etc.
  - Its meaning can vary and be misunderstood
- The meaning of `2+2` in Python is a technical fact
  - This expression will always evaluate to `4`
  - Its meaning never changes, and if you think it evaluates to `5` that is your mistake

(2) Mistakes

- Any one judge might be corrupt or confused
  - So might any single smart-contract enforcer
- But on blockchain, miners must agree
  - The consensus protocols rewards miners who agree with their peers
  - The vast majority of honest and competent miners converge on the correct result of a program
  - Those who disagree go broke

(3) Enforcement

- Physical assets can be hidden or destroyed
    - This is an important reason why money contract damages are even necessary
- But digital assets on a blockchain can't
    - As long as a smart contract keeps assets in escrow, they can automatically be returned
    - They're released only to a parties who have fully complied with the contract's terms

Summary

- Legal contracts are obviously run by humans:
  - They are fallible in all the ways people are
  - People are vague and confused
  - People can be bribed or pressured
- Smart contracts are run by computers:
  - Computers are precise and deterministic
  - And (we hope) they can be secured well enough

Everything I have just said is wrong

Where does program meaning come from?

- Why *doesn't* 2+2 in Python evaluate to 5?
- Not because that's what "2+2" inherently means
  - Any more than "chicken" inherently means any *gallus gallus domesticus*, even one that is wholly unsuitable for cooking
- In 1991, GvR picked + as the addition operator
  - He could have picked ++ instead

Usual sources of program meaning

- Use a program: a *reference implementation* whose behavior is by stipulation treated as correct
- Use natural language: a *specification* that defines the behavior of a correct implementation
- Use mathematics: a *formal semantics* that identifies programs with abstract entities

Three questions

- Where do these come from?
  - Some people got together to write them
- What makes one of them definitively correct?
  - Because people agree that it is
- What language are we running?
  - “Python” 2.7 is different from “Python” 3.6
- These questions can be answered only by reference to a community of programmers and users

Program meaning is a social fact, too

- Yes, 2+2 in Python is unambiguously 4
- But that's only because Python users have already agreed on what "Python" is
- If they agreed differently, "Python" would be different, and so might 2+2
- This happens *every time* there's a new version
- Technical facts depend on social facts!

Fixing program meaning

- A technical community agrees on a process for deriving a functional meaning from texts
- Developers implement that process on different computers, with different tools, etc.
- Most of the time, running a program on most implementations yields the same result
- We perceive as fixed technical facts the *successful* result of coming to a social consensus

Implicit agreements

- The statement “Program *P* does *X* when run” requires some shared assumptions:

(1) *P* is written in a specific language *L*

(2) *L*’s semantics specify that P does X

(3) The hardware environment in which *P* runs is free from shorts, bit flips, meteors, etc.

(4) The software+hardware environment in which *P* runs is a correct implementation of *L*

Back on the blockchain

- A blockchain where consensus holds resolves (1)–(3)
  - The social fact of agreeing on blocks *de facto* standardizes every user on the same virtual machine
  - Its semantics are whatever they agree on — typically the output of the reference VM implementation
  - Idiosyncratic hardware faults do not threaten consensus
- But (4) is harder
  - The resolution of (1)–(3) might be incorrect!
  - What is “correct” can be in the eye of the beholder

Features and bugs

The price we pay

- Running a program produces *a* result—but not necessarily the *right* result
- Specifying in advance the resolution of all possible ambiguities is a recipe for predictably getting many of them wrong
- The concept of a “bug” assumes a distinction between *actual* and *intended* program behavior

What is a bug?

- A programmer could:
  - Type the wrong expression
  - Misunderstand how the language works
  - Misunderstand the algorithm they chose
  - Misunderstand the problem they're solving
  - Fail to anticipate a possible input
  - Make an incorrect assumption about the world
  - Misunderstand a tool (library, API, etc.) they relied on
  - Miscommunicate with a colleague
  - Forget what they were doing and do something inconsistent
  - Run the program on hardware that violates expectations
  - Regret doing something they fully intended at the time
  - …

This sounds familiar

- This list bears more than a passing resemblance to the list of ways to misspeak
- The distinction between actual and intended meaning carries over from natural to programming languages
    - A speaker might produce an utterance her human audience understands differently than she intended
    - A programmer might produce a program that computers interpret differently than she intended

Which bugs count?

- A bug in a *smart contract* might be regarded as its author's or user's fault
  - They had a communicative goal, but failed to express themselves as they wanted
- A bug in a *blockchain client* goes much deeper
  - The client — or the language — fails to do what its community of users expects
- It's not always clear which is which

Blockchain governance

Does this matter?

- We might be able to ignore all of this if smart-contract blockchains never had trouble
- But in fact, there are fights over the meanings of blockchain programs *all the time*

The DAO

"The terms of The DAO Creation are set forth in the smart contract code existing on the Ethereum blockchain at `0xbb9bc244d798123fde783fcc1c72d3bb8c189413`. Nothing in this explanation of terms or in any other document or communication may modify or add any additional obligations or guarantees beyond those set forth in The DAO's code."

Ethereum Classic

- Following the DAO hack, Ethereum upgraded to a new version that *specifically unwound the DAO transactions*
- Not everyone was happy with this, and some users were unhappy enough to fork Ethereum Classic, which didn't have this "upgrade"
- The two blockchains *have different semantics*

Bitcoin Cash

- A long-running dispute over Bitcoin block size caused some users to fork Bitcoin Cash
  - Bitcoin has ~1MB blocks
  - Bitcoin Cash had 8MB blocks (now 32MB)
- The two blockchains have *different semantics*

The once and future fork

- If blockchains *can* change, then every blockchain *could* change
- If anyone objects enough to walk away, the dispute becomes visible as a fork
    - Each blockchain is "unambiguous" but the *choice of blockchains* expresses a patent ambiguity
    - Literally anything on a blockchain is subject to the latent ambiguity that the blockchain itself could change out from underneath it

The DAO (legal) contract

- The English phrase "the smart contract code existing on the Ethereum blockchain at `0xbb9bc244d798123fde783fcc1c72d3bb8c18941`" is ambiguous
- "the Ethereum blockchain" does not uniquely refer: do you mean ETH or ETC?
- It uniquely referred when the contract was drafted, but no longer

Where to go from here?

All is not lost

- Smart contracts are based on social facts
  - Social facts are empirically contingent: they are always open to contestation and change
- Legal contracts are based on social facts, too
  - And a lot of the time, they work just fine!
- Smart contracts *cannot* be perfectly unambiguous
  - But they can be unambiguous enough

Focus on the consensus

- Blockchains make consensus explicit
  - The mechanism that holds them together is the protocol for agreeing on the next block
- Put another way, every smart contract is vulnerable to a "51%" attack
  - Where the "attack" could happen through persuasion as well as raw computational power

The contractual is political

- A blockchain whose governance fails will collapse, fork, be hijackable, etc. — all of which threaten the smart contracts that run on it
- Contract law depends on social institutions that establish and limit government
- Smart contract code depends on social institutions that establish and limit blockchain governance
- There is no escape from politics

Good blockchain citizenship

- (Practically, not perfectly) unambiguous smart contracts require correct, stable blockchains
- Blockchain correctness and stability require a good blockchain community
  - Correctness from making good changes
  - Stability from not making bad ones
- Not just consensus protocols — it's also the mailing lists, the depth of developer knowledge, user commitment to long-term health, etc.

Conclusion

Blockchains are made out of people

Questions