The Structure and Legal Interpretation of Computer Programs

James Grimmelmann

Cornell Tech/Law Colloquium
October 24, 2017

Three vignettes

Video poker


GAME KING
MULTI-GAME
MULTI-GAME
Earn & Win!
WEDNESDAYS IN JUNE
INSERT TICKETS
GAME KING
MULTI-GAME
EARN
1000 POINTS
$20 FREE SLOT PLAY
WIN
ROYAL FLUSH NO DEUCES
4 DEUCES
1800
1200
2100
1400
2400
1600
2700
1800
3000
2000

Website scraping

```
User-agent: *
Disallow: /private/
```

```
User-agent: *
Dsallow: /private/
```

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."

The question stated

What are the legal effects of software?

- Distinguish the *factual* effects of software
  - E.g., self-driving cars, pacemakers, etc.
- Distinguish the *expressive* content of software
  - E.g., Facebook Like button, MS Paint, etc.
- These are cases in which the software itself changes legal relationships
  - In Hohfeldian terms, "exercises a power"

Obvious legal analogs

- Legislatures affect citizens via statutes
- Agencies affect citizens via regulations
- Private parties affect each other via contracts
- Testators affect heirs via wills
- Property owners affect users via licenses

A natural parallel

- A person exercises a legal power through a *text*
  - The person must actually hold the power
  - They must adopt the text in the correct way (e.g., a will must be signed and witnessed)
  - The *meaning* of the text determines its *effects*
- *What are the (legal) interpretive rules for software?*

Plan of attack

- Trace the techniques used by computer scientists to say what a program means
- Compare them to the techniques used by judges to say what a legal instrument means
- Determine how legal interpreters should rely on or depart from these techniques

Code is law?

Interpretation and construction

- Distinguish *interpretation* from *construction*
  - Interpretation: *determining linguistic meaning*
  - Construction: *determining legal effects*
- The conventional account is that construction is necessary because texts are sometimes vague
  - That is, the linguistic meaning "runs out" and it becomes necessary to fill in the gaps

Software is different

- Legal texts' *legal* effects are their *only* effects
- But software is primarily a *functional* artifact
  - It drives cars, turns lights on, modifies account balances, shows or hides data, etc.
  - In our vignettes, the principal thing software is expressing is how to use it
- What it (legally) *means* is derivative of what it *does*

Who is the interpreter?

- Legal texts are addressed to *people*: citizens, counterparties, guests, and especially judges
  - So we care about their meaning to people
- But software is addressed to *computers*: it consists of a series of commands to execute
  - I.e., the functional effects of a program derive from its meaning to a computer

Interpretation without construction?

- Natural language is inherently vague in a way that programming languages are not
  - Just run the code and see what it does!
  - A program's functional effects *are* its meaning
  - Programs require (functional) interpretation but not construction
  - How does this fact affect their *legal* treatment?

Proposition:
functional meaning ≠ legal meaning

- Naive interpretive strategy: *pure functional meaning*
  - Let the computer interpret the code for you, and do not engage in any further construction
    - What the code allows is what the law allows
    - Legal meaning = functional meaning
- This is obviously insufficient as a theory
  - Computers malfunction; software is buggy

Proposition:
functional meaning ~ legal meaning

- Not anything goes!
  - Video poker is not video backgammon
  - The DAO is incoherent unless there is some determinate content to "the smart contract code existing on the Ethereum blockchain at `0xbb9bc244d798123fde783fcc1c72d3bb8c18941`"
- Legal meaning is grounded in functional meaning

Functional meaning

From hardware to software

- Actual computers get hit by cosmic rays, suffer hard drive failures, drop packets, etc.
- Idea (Turing, von Neumann, etc..): idealize the computer as a mathematical abstraction
  - Abstract away from faulty hardware
  - Abstract away from specific hardware
  - Focus on the text of the software

Specification and semantics

- What does `2+2` mean?
- Three answers:
  - 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

Two problems

- Where do specifications and semantics come from?
  - Some people got together to define them
- What language are we running?
  - “CSS” is different in Safari and Chrome
  - “Python” is different in version 2.7 and version 3.6
- These questions can be answered only by reference to a community of programmers and users

Where we are

- 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
- New interpretive strategy: *literal functional meaning*, what a correctly functioning computer would do

Features and bugs

The price we pay

- Running a program produces *a* result—but not necessarily the *right* result
- This is characteristic of literal functional meaning as an interpretive strategy: 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

Fixing bugs

- People detect and correct misstatements by noticing incongruities and through discussion
- Computers are ill-positioned to do either
  - They can be coded to accept a wider range of inputs and make different assumptions
  - Or humans monitoring them can notice incongruities and engage in discussion

Ordinary meaning

- The ordinary legal meaning of a text is the meaning a reasonable audience would give it
  - The audience for a program consists of its users
    - Users expect that a program contains bugs
- So the ordinary functional meaning of a program is what reasonable people in the position of its users and knowing what they know would expect it to do, *if it were free of bugs*

Legal meaning

Back to legal meaning

- Computer programs can have legal effects because the legal system says they do
- There are contexts where it treats functional effects as the exercise of a legal power
- It does so because the *audience* for those functional effects—video poker players, web scrapers, DAO investors—can and should recognize them as bearing legal meaning

When should the law depart from actual meaning?

- With *legal* texts, this is the debate between textualism and more contextual theories
- Computers seem to offer a much stricter version of textualism: literal functional meaning, which treats the actual computer's interpretation as authoritative
- But sometimes computers fail to be competent interpreters, and sometimes users prefer the programmer's intended interpretation

Two reasonable interpretive theories

- *Literal functional meaning*: how a reasonable computer would understand the program
  - Abstracts away from malfunctions
- *Ordinary functional meaning*: what a reasonable user would understand the programmer to have meant
  - Abstracts away from malfunctions and bugs
- The choice between them depends on context

Video poker

- Reasonable video poker players understand:

  (1) They are expected and allowed to play as skillfully as they can to improve their payouts

  (2) Quitting and returning to a game after a hand has been played probably wasn't intended to change the payout multiplier

- The case looks hard because of the conflict between (1) and (2). But ordinary functional meaning controls, and the payout trick looks like a bug, not a feature.

Robots.txt

- Reasonable web scrapers understand:

  (1) `Disallow` was probably intended; `Dsallow` isn't a valid keyword in the robots exclusion standard

  (2) The standard is designed for bots to process automatically, not (primarily) for humans to read

- Literal functional meaning is appropriate because of (2). Without specific knowledge (maybe even with it), bot operators don't need to respect `Disallow`.

The DAO

- Reasonable blockchain investors understand:

  (1) The DAO contract was buggy

  (2) The DAO's legal instruments purported to make the contract judicially unreviewable

  (3) The DAO depends on Ethereum

- Whether (2) successfully selects literal functional meaning is a question of offline contract law. But (3) makes even literal functional meaning ambiguous!

Coda

Looking in the mirror

- Thinking about the power and limits of functional meaning sheds light on legal interpretation
- Legal texts, like programs, are hybrids:
  - They can be expressions of meaning to people
  - They can be "programs" to convey legal effects to judges and other officials
- How much do we want judges to act like *users*, and how much do we want them to act like *computers*?

On debugging

- Program analysis and testing are essential to modern software development
- They enable programmers to learn about what a program they wrote does *before* releasing it
  - (Cf., the Constitution forbids the federal courts from issuing pre-release "advisory" opinions)
- Can these software development techniques be pulled back into law?

Questions?