The Structure and Legal Interpretation of Computer Programs

James Grimmelmann

ProLaLa23
January 15, 2023

Motivation

- Judges interpret legal texts:
  - Does "no vehicles in the park" include electric scooters?
- Judges also interpret software:
  - Does the functionality of `sendmail` and `fingerd` allow the Morris Internet worm?
- *What is the legal meaning of a program?*

Who is the interpreter?

- Legal texts are addressed to *people*: citizens, counterparties, guests, and especially judges
  - They mean what they mean to people
- Programs are addressed to *computers*: they consists of a series of commands to execute
  - Do they mean (only) what they cause computers to do?

Interpretive strategy 1: naive functional meaning

- A program's meaning is the effects it has on the computer running it
  - Conceptually simple: meaning = effects
  - Operationally simple: run it and find out
- Natural language is vague and ambiguous
  - But it's easy to observe a computation, and people will agree on what its outputs are

Objection

- Real-world computations often fail
  - But naive functional meaning says that the failure mode *is* the program's meaning
- For legal purposes, this is often clearly wrong
  - E.g., if the ATM crashes before dispensing your cash, *you're still entitled to the money*

Response

- Programming-language definitions distinguish *correct* from *incorrect* executions
  - Natural-language specifications
  - Formal mathematical semantics
  - Reference implementations
  - Test cases
- So: *derive program meanings from language definitions*

Interpretive strategy 2: literal functional meaning

- A program's meaning is the effects it would have on a computer that correctly implements the language in which the program is written
- Based on abstract language definitions
- Rather than on concrete executions

Challenge

- Language semantics are defined the people who agree on what the language semantics are
  - This agreement can change or break down!
    - E.g., `1_000_000` is invalid in Python 3.5.2 but valid in Python 3.6.1
    - E.g., Firefox and Chrome implement CSS differently

The problem of bugs

- I write a program to draw an octagon, but it draws an eight-pointed star instead
  - I wrote `135` instead of `45`
    - So I write a new program and fix the bug
- Naive and literal functional meaning treat the two programs as equally valid
  - But to me, one is buggy and one is correct

Interpretive strategy 3: ordinary functional meaning

- Legal theorists distinguish the *literal* meaning of a text from its *ordinary* meaning to a reasonable reader who ignores mistakes, etc.
- Ordinary functional meaning = what a reasonable reader would expect the program to do if it were free of bugs

Which interpretative strategy is right?

- *They all are*
  - Programmers switch between them frequently
  - Judges and lawyers need to use all three
- E.g., Morris involved a distinction between literal functional meaning (he installed the worm via sendmail) and ordinary functional meaning (that's wasn't the "intended function")

Discussions