Programming Languages and Law: A Research Agenda for a New Field

James Grimmelmann

ProLaLa22

January 15, 2022

In this talk

- Law using/of/and x …
  - … for x = programming languages
- 10 big ideas for the future of PL+law

Ten big ideas for the future of PL+law

1. Everything law always wanted to know about PL
2. Hybrid contracts
3. And you get a legal DSL
4. Orthogonal legal primitives
5. Legal drafting languages
6. Legal design patterns
7. Legal design principles
8. An IDE for lawyers
9. Legal Jupyter notebooks
10. What is interpretation?

Law of x

Law using x

Law and x

Law <u>of</u> *x*

- Law directly regulates the field of *x*
- E.g., let *x* = "medicine" in …
  - … licensure, malpractice, billing, etc.
- Hopefully the legislatures know something about the field they're regulating

Law <u>using</u> *x*

- Law asks questions that *x* can answer
  - Administrative law, evidence law, etc. deal with how to incorporate technical expertise
- E.g., let *x* = "marine ecology" in …
  - … what is the thermal impact of power-plant discharge on "a balanced indigenous population of shellfish, fish, and wildlife"?

Law and *x*

- The methods of *x* provide novel ways to formulate and answer legal questions
- E.g., let *x* = "microeconomics" in …
  - … what is the most efficient measure of damages in case of contractual breach?
- E.g., let *x* = "corpus linguistics" in …
  - … what was the original public meaning of "keep and bear Arms"?

Successful examples of x

- Anthropology
- Economics
- History
- Linguistics
- Literature
- Philosophy
- Political Science
- Psychology
- Sociology

Law and PL

Zooming in on law and PL

- Law and …
  - … technology
  - … computers and the Internet
  - … computer science
  - … programming languages

CS subfields

- Some CS fields treat law as a fruitful problem domain
  - Symbolic AI models legal reasoning
  - ML etc. treat law as data
- Some legal areas draw on ideas from CS fields
  - Privacy law draws on security
  - Telecom law draws on networking
  - Antidiscrimination law now draws on ML

“The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.”

–Frederick P. Brooks, Jr., *The Mythical Man-Month*

"The programmer, like the ~~poet~~ lawyer, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures."

–Frederick P. Brooks, Jr., *The Mythical Man-Month*?

Program text ~ legal text

- CS and law are both *linguistic* professions:
  - They use language to create, manipulate, and interpret complex abstractions
  - A programmer who uses the right words in the right way makes a computer do something
  - A lawyer who uses the right words in the right way changes people's rights and obligations

The PL advantage

- Some fields (e.g., AI) deal with legal structures
- Other fields (e.g., NLP) deal with legal language
- PL provides a principled, systematic framework to analyze legal structures in terms of the linguistic expressions lawyers use to create them
- PL abstractions have an *expressive power* in capturing the linguistic abstractions of law

Past, present, and future

Law of PL

- Programming is not a regulated profession
- But *programs* are subject to law in many ways
- And some of those laws care about how programs work, what they mean, how programmers write them, etc.
- PL experts have relevant expertise here!

(1) Everything law always wanted to know about PL

- *Patent*: Does software consist entirely of math? In what sense is an algorithm an abstract idea?
- *Copyright*: Which parts of a program are standard? Required for compatibility?
- *First Amendment*: What constitutes the "speech" in writing, sharing, or running code?
- etc. …

Contract DSLs

- *Composing Contracts* (2000): combinators for financial option contracts with evaluation semantics
- Extensive work on logics to model deontic, multi-party, and event-driven structure of contracts
- "Smart" "contract" languages substitute automation for legal enforcement
- Hybrid projects combine machine-readable logic with human-readable terms

(2) Hybrid contracts

- Write contractual terms once and compile them to “legal code” or “computer code”
- Prevent or at least detect bugs in legal logic
- Automate execution of *parts* of a contract
- Have a clear story about the legal effects of executing the automated part, and vice versa

Legal DSLs

- Orlando: future interests in property
  - Translation from natural-language-like PL to trees of interests, with operational semantics
- Catala: legislative defaults
  - Translation from declarative PL with exceptions to lambda calculus with default
  - Being used to rewrite and verify the implementation of the French tax code

(3) And *you* get a legal DSL

- Which legal areas are already PL-like?
  - They use *rules* rather than *standards*
  - They have *recurring patterned structures*
  - Their participants *highly value clarity*
- Low-hanging fruit:
  - Anything transactional (e.g. IP licensing, wills)
  - Lots of property law (e.g. title assurance)

Legal logic programming

- Long history of symbolic AI for legal reasoning
- But distinctive characteristics of legal reasoning are not well supported out of the box
    - E.g., counterfactuals, non-monotonic inference, deontic modalities, hierarchies of authority
- Response: adopt PLs with appropriate language features for the problem domain
    - E.g. PROLEG, L4, DCPL, LLD …

Legal calculi

- *Composing Contracts* uses combinators
- Catala uses lambda calculus with exceptions
- Orlando uses a non-Turing-complete calculus with sequencing and termination
- There seems to be something fruitful about compiling legal expressions to this form

(4) Orthogonal legal primitives

- Examples:
  - Sequencing, conjunction, disjunction
  - Defaults and exceptions
  - Rights and duties, powers and liabilities
- Do they have clean minimal formalizations?
- Can they be combined?
- Do we need legal-specific back-end support?

Language features

- Lawsky (2017) observes that the *scope* of statutory definitions is often ambiguous
  - Some drafting choices (e.g. "reasonable") are vague, but these should be unambiguous
  - Proposes logical forms with explicit scoping
- In other words, *use a programming language whose features promote clear and correct code!*

(5) Legal drafting languages

- What other language features would help clean up legal drafting?
  - Variables and binding
  - Cross-referencing
  - Counterfactuals and reflection
  - Substitution
  - …

Design patterns

- General, repeatable template for solutions to a commonly occurring class of design problems
  - Roots in (physical) architectural theory
    - Adapted to object-oriented software design and to UI/UX, with a strong PL angle
- Nascent legal-design-patterns literature
  - With its own debts to architectural theory

(6) Legal design patterns

- Identify proxies, decorators, singletons, etc. in regulatory regimes and complex transactions
- Craft pattern languages for specific legal areas

Design principles

- Example: a *modular* system decomposes into subsystems that are weakly coupled to each other
  - Roots in cybernetics
  - A central principle of computer system design
  - Often achieved with explicit PL features
- Smith: extensive theory of modularity in property
  - Other applications: legislative drafting, contracts

(7) Legal design principles

What other PL design principles are relevant?

- Recursion
- Compositionality
- Type safety
- Extensibility
- …

Where else in law do they show up?

- Legal drafting
- Commercial law
- Corporate law
- Evidence
- …

Programming tools

- Modern IDEs and toolchains are outstanding
- But legal drafting often means Microsoft Word
  - If you're lucky, XML
  - If you're not, WordPerfect
- This is a moment of rapid legaltech ferment
  - Some of which is based on serious PL work

(8) An IDE for lawyers

- What kinds of language-aware IDE support would be useful to legal drafters?
  - Syntax highlighting, auto-formatting, …
  - Type-checking, linting, static analysis, …
  - Version control with branches, diffs, …
  - Unit tests, fuzz testing, …
  - Interactive debugging, breakpoints, …

Tools for teaching and modeling

- PL approaches improve understanding:
  - Visualization — Littleton uses a railroad-diagram library to visualize interests
  - Interactivity — Lawsky Practice Problems
  - Literate programming — Catala, Cicero

(9) Jupyter notebooks for law

- How to visualize legal structures using PL concepts like ASTs, control-flow graphs, etc.?
- How to tighten the feedback loop between writing down legal text and seeing what it does?
- How to mix the human-facing and computer-facing parts of literate legal programs?
- How to do *all three at once?*

Philosophical questions

- Legal and PL scholars study interpretation: how to understand what a text means
  - Maybe they can learn from each other?
- Of intellectual interest to both fields
- Of practical interest for PL+law systems

(10) What is interpretation?

- What is the difference between how a person interprets a text and how a computer does?
- How (if at all) should computers change how legal texts are written and interpreted?
- Does legal interpretation have any lessons for PL interpretation?
- My book on this was due a year ago …

Closing thoughts

Interdisciplinary work is *hard*!

- Different fields have different methods, concepts, standards of rigor, authorities, writing styles, etc.
- Legal scholars can be overconfident amateurs outside of their own field ("law-office history")
  - And *vice versa* (*cough* economics)
- Collaborations and dual training help a lot

Principled > ad hoc

- PL has a long history of developing languages, tools, and concepts to tame the chaos of coding
  - Law and legal tech could use some of that
    - E.g., Catala and the French tax code
- Formalizing a body of law forces you to *understand* it in a much deeper way
  - Knuth: "Science is what we understand well enough to explain to a computer."

Discussion