Programming Languages and Law: A Research Agenda

James Grimmelmann

ACM CS+Law

November 2, 2022

In this talk

- Law and PLT
- Some examples of successful combinations
- Ten big ideas for the future of PL+law

Law and PLT

Zooming in on law and PLT

- Law and …
  - … technology
  - … computers and the Internet
  - … computer science
  - … programming language theory

“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

Why PLT?

- Some fields (e.g., AI) deal with legal structures
- Other fields (e.g., NLP) deal with legal language
- PLT 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

Examples

TAXMAN (McCarty 1977)

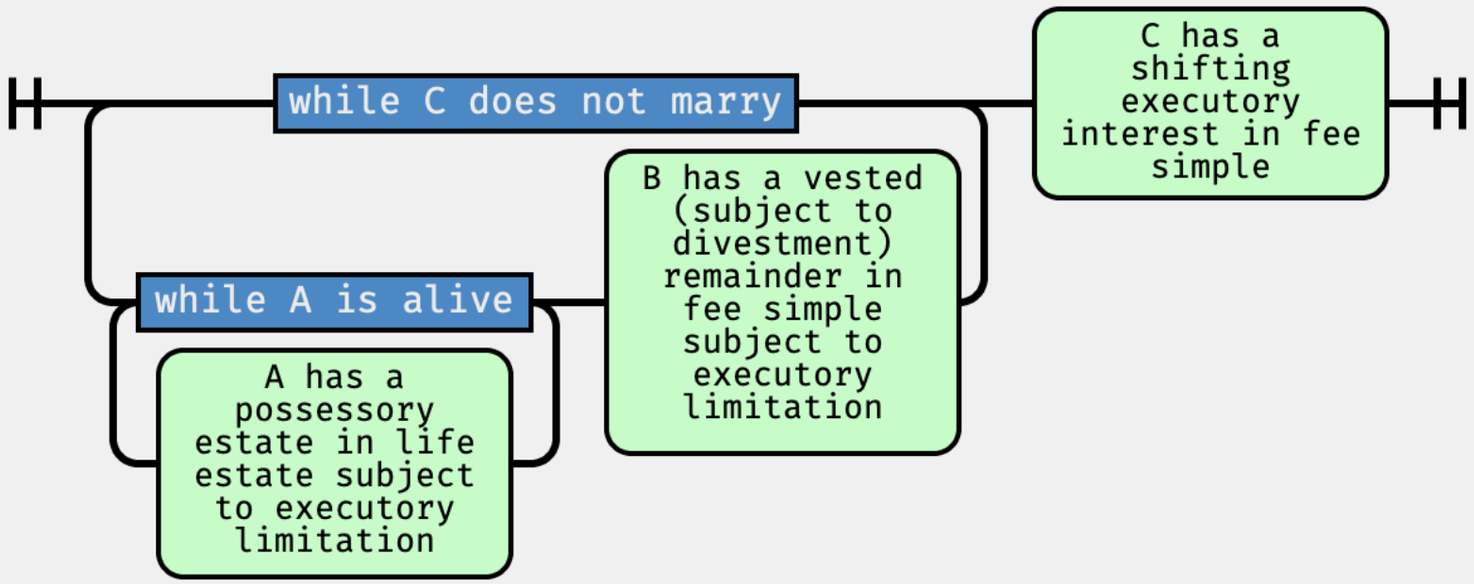
```
(PROG (S P)
   (GOAL (ISSUE NEW-JERSEY ?S))
   (GOAL (STOCK ?S))
   (GOAL (PIECE-OF ?P ?S))
   (GOAL (OWN PHELLIS ?P)))
```

M++
(MMP 2021)

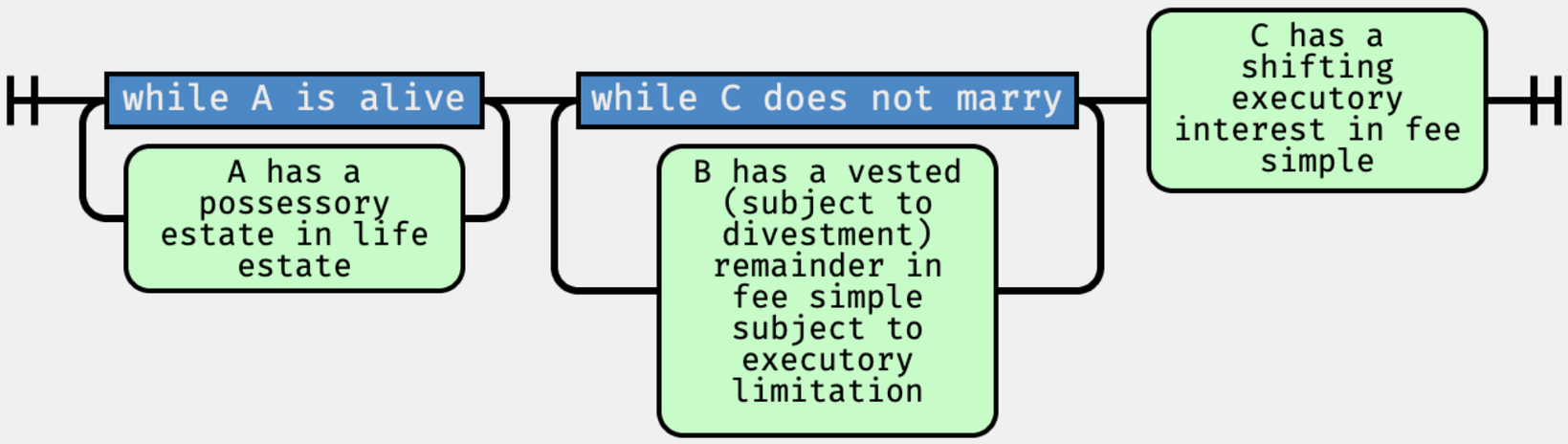
```
compute_benefits():
  exists(taxbenefit) or exists(deposit):
    V_INDTEO = 1
    V_CALCUL_NAPS = 1
    partition with taxbenefit:
  NAPSANSPENA, IAD11, INE, IRE, PREM8_11
        <- call_m() iad11 = cast(IAD11)
ire = cast(IRE)
ine = cast(INE)
prem = cast(PREM8_11)
V_CALCUL_NAPS = 0
V_IAD11TEO = iad11
V_IRETEO = ire
V_INETEO = ine
PREM8_11 = prem
```

Orlando (BFGPR 2022)

(To A for life, then to B), but if C marries to C.



To A for life, then (to B, but if C marries to C).



Research directions

(1) Legal DSLs

- 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)

(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

(3) Orthogonal legal primitives

- Catala uses a lambda calculus with exceptions
- Orlando uses a calculus with early termination
- Can we identify other orthogonal primitives, formalize them cleanly, and combine them?
- E.g., sequencing, conjunction, disjunction, defaults, exceptions, privileges and powers …

(4) Legal drafting languages

- Lawsky (2017): the scope of statutory definitions is often ambiguous
  - Proposes logical forms with explicit scoping, i.e., *use a PL whose features promote correct code!*
- What other language features would help clean up legal drafting?
  - Variables and binding, cross-referencing, counterfactuals and reflection, substitution …

(5) Legal design patterns

- General, repeatable template for solutions to a commonly occurring class of design problems
  - Applied to object-oriented software design and to UI/UX, with a strong PL angle
- Are there pattern languages for legal fields?
  - E.g., UK Office of the Parliamentary Counsel's *Common Legislative Solutions*

(6) Legal design principles

- Smith: property law is *modular*
- What other design principles show up in law?
  - Privilege law in evidence law is *extensible*
  - Good contract drafting is *type-safe*
  - Recursion, compositionality, …

(7) An IDE for lawyers

- Programmers have outstanding toolchains
  - Lawyers have Microsoft Word and "Save As"
- 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 and diffs, interactive debugging, breakpoints, unit tests …

(8) Legal Jupyter notebooks

- Can we visualize legal structures using PL concepts like ASTs, control-flow graphs, etc.?
- Can we tighten the feedback loop between writing legal text and seeing what it does?
- Can we mix the human-facing and computer-facing parts of literate legal programs?

(9) The law of software

- *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. …

(10) Philosophical questions

- 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?

(Summary)

1. Legal DSLs
2. Hybrid contracts
3. Orthogonal legal primitives
4. Legal drafting languages
5. Legal design patterns
6. Legal design principles
7. An IDE for lawyers
8. Legal Jupyter notebooks
9. The law of software
10. Philosophical questions

Closing thoughts

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., M++ 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