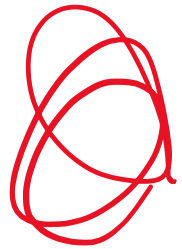
CS 295A/395D: Artificial Intelligence



Application: Law and Logic Programming

Prof. Emma Tosch

3 February 2022

The University of Vermont

Questions about homeworks, exams, schedules...


YOU GET AN A! AND YOU
GET AN A!

Agenda

Topic: how logic fits in with the bigger picture in AI

- SAT/SMT solvers today
- History of logic in AI
- Focus: Symbolic AI for Law
    - Prolog
    - Active research

SAT/SMT: how it's used in real life

- Symbolic execution/Test case generation
- Circuit equivalence
- Games (sudoku, n-queens)
- "Running programs backwards"
- AWS IAM Access Analyzer
- Anaconda package management
- Finding bugs in legal code
- Google OR-tools
- Neural Network Verification

Few are traditional "AI" domains!

This list crowdsourced via Alastair Donaldson's tweet: https://twitter.com/afd_icl/status/1475641756209364996

How are SAT/SMT solvers useful?

Known: SAT/SMT is NP-Hard

→ No known sub-exponential algorithm for solving

… So how can we actually use these tools in practice?

1990s – "phase transition" phenomena in SAT problems

- Generate random k-SAT problems in CNF
- Ratio of clause size to number of variables correlates with runtime
- Theme: AI focus on "features of problem instances," rather than ML focus on "features of data"

History of Logic in AI

Agent-based reasoning

- First logical agent-based: McCarthy (1950s-1960s)
- Agents behavior in terms of *declarations* rather than *procedures* (1980s-1990s)

Natural Language

- SHRDLU: Winograd, 1971
- As an alternative to SQL: 1970s-1980s

Law – Topic of this lecture!

robotics
~~robots~~
RL
Multi agent
reasoning

THE BRITISH NATIONALITY ACT AS A LOGIC PROGRAM

The formalization of legislation and the development of computer systems to assist with legal problem solving provide a rich domain for developing and testing artificial-intelligence technology.

M. J. SERGOT, F. SADRI, R. A. KOWALSKI, F. KRIWACZEK, P. HAMMOND, and H. T. CORY

There are essentially two kinds of law, *case law*, determined by earlier court decisions, and *statutes*, determined by legislation. Substantial amounts of statutory law are basically definitional in nature and attempt to define more or less precisely some legal relationship or concept. The British Nationality Act 1981 [20] defines British citizenship and is a good example of statutory law. The act embodies all the characteristics of statutes in general: syntactic complexity, vagueness, and reference to previously enacted legislation.

In the course of this article, we will describe how the text of a large part of the British Nationality Act 1981 was translated into a simple form of logic, and we will examine some possible applications of this translation.

The form of logic used is that on which the programming language Prolog is based. Later in the article, we will describe how our translation of the act can be executed as a program by an augmented Prolog system, so that consequences of the act can be determined mechanically.

Although Prolog logic is severely restricted, it proved to be sufficiently high level so that our implementation could resemble the style and structure of the actual text of the act. Such a resemblance is important because it helps increase confidence in the accuracy of the implementation and makes the implementation easier to maintain as the legislation changes and as case law evolves to augment the legislation.

Our implementation of the British Nationality Act 1981 was undertaken as an experiment to test the suitability of Prolog logic for expressing and applying legislation. The British Nationality Act 1981 was chosen for this experiment for a number of reasons. At the time it was first proposed, the act was a controversial piece of legislation that introduced several new classes of British citizenship. We hoped that formalization of the various definitions might illuminate some of the issues causing the controversy. More importantly, the British Nationality Act is relatively self-contained, and free, for the most part, of many complicating factors that make the problem of simulating legal reasoning so much more difficult. Furthermore, at the time of our original implementation (summer 1983) the act was free of the complicating influence of case law.

A complication that we anticipated was the presence of vagueness. The act contains such vague phrases as "being a good character," "having reasonable excuse," and "having sufficient knowledge of English." These concepts are not defined in the act and occur only at the lowest level of detail. At higher levels, the question of whether a person is a British citizen depends primarily on concrete, easily

This work was supported by the Science and Engineering Research Council.

Verification → French tax code

Catala

Famous Application

We believe that the formalization of legislation and legal reasoning offers potential contributions to computing technology itself. It should help to discriminate, better than other applications, between different knowledge representation formalisms and problem-solving schemes in artificial intelligence. Moreover, the rules and regulations that govern the management of institutions and organizations have exactly the same character as legal provisions. This suggests, therefore, an unconventional approach to the construction of software for data-processing applications. A payroll system, for instance, could be based directly on tax and sick-pay legislation, could include a representation of the company pension scheme, the rules that govern holiday allocation, and promotion regulations. We have already constructed a number of experimental systems dealing with some of these topics.

Tax Software

Example 1 : Below food table shows the facts, rules, goals and their english meanings.

Facts	English meanings
food(burger).	// burger is a food
food(sandwich).	// sandwich is a food
food(pizza).	// pizza is a food
lunch(sandwich).	// sandwich is a lunch
dinner(pizza).	// pizza is a dinner

Rules	
meal(X) :- food(X).	// Every food is a meal OR Anything is a meal if it is a food

Queries / Goals	
?- food(pizza).	// Is pizza a food?
?- meal(X), lunch(X).	// Which food is meal and lunch?
?- dinner(sandwich).	// Is sandwich a dinner?

EXAMPLE – 1 EXPLANATION & MORE

Is a hotdog a sandwich?

Core logic language:

- Rules and facts form a knowledge base (stored in a database)
- Every rule is an implication, written "backwards"
- Queries evaluate truth values or find satisfying assignments
- All facts, rules, and queries are implicitly universally quantified

https://athena.ecs.csus.edu/~mei/logicp/prolog/programming-examples.html

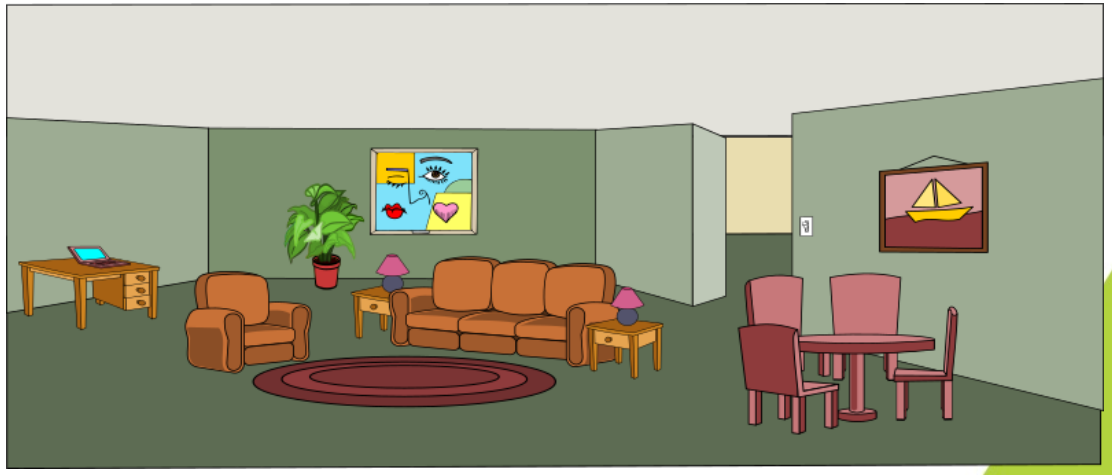
Prolog activity

Go to: http://tau-prolog.org/sandbox/

I will assign you to random pairs

Select a domain, e.g., CS major requirements or

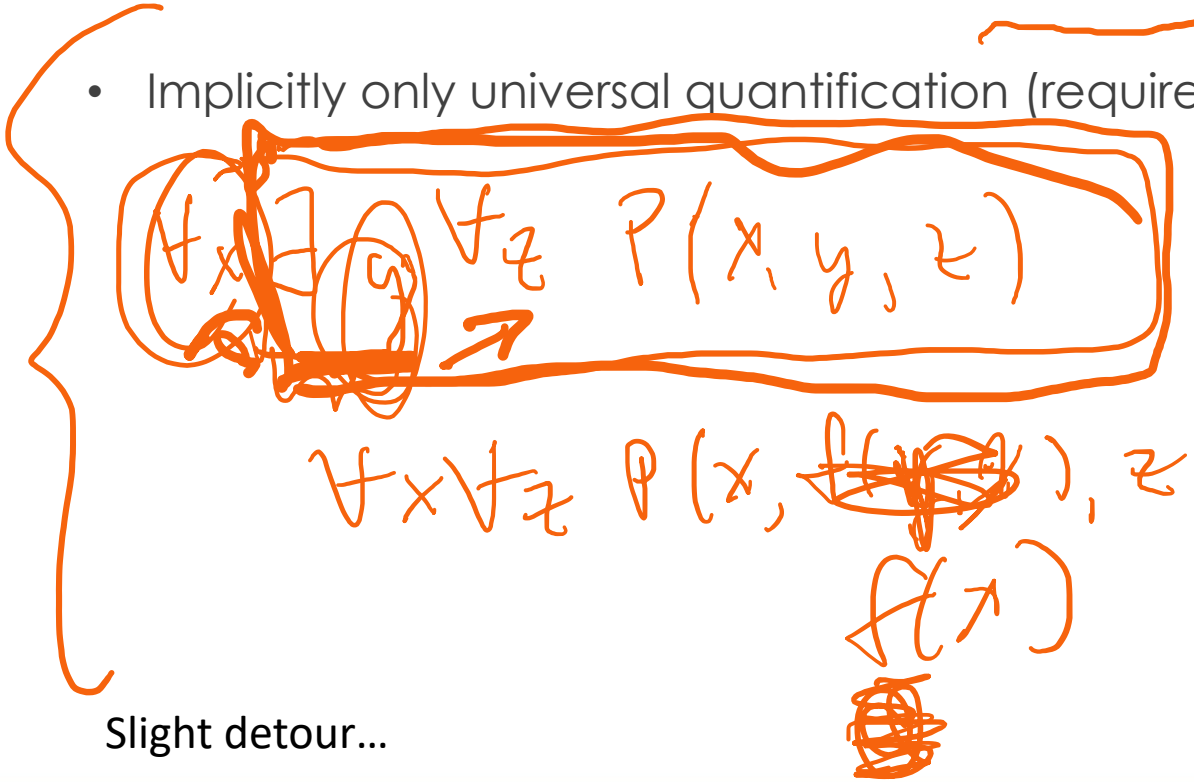
Encode facts and rules:

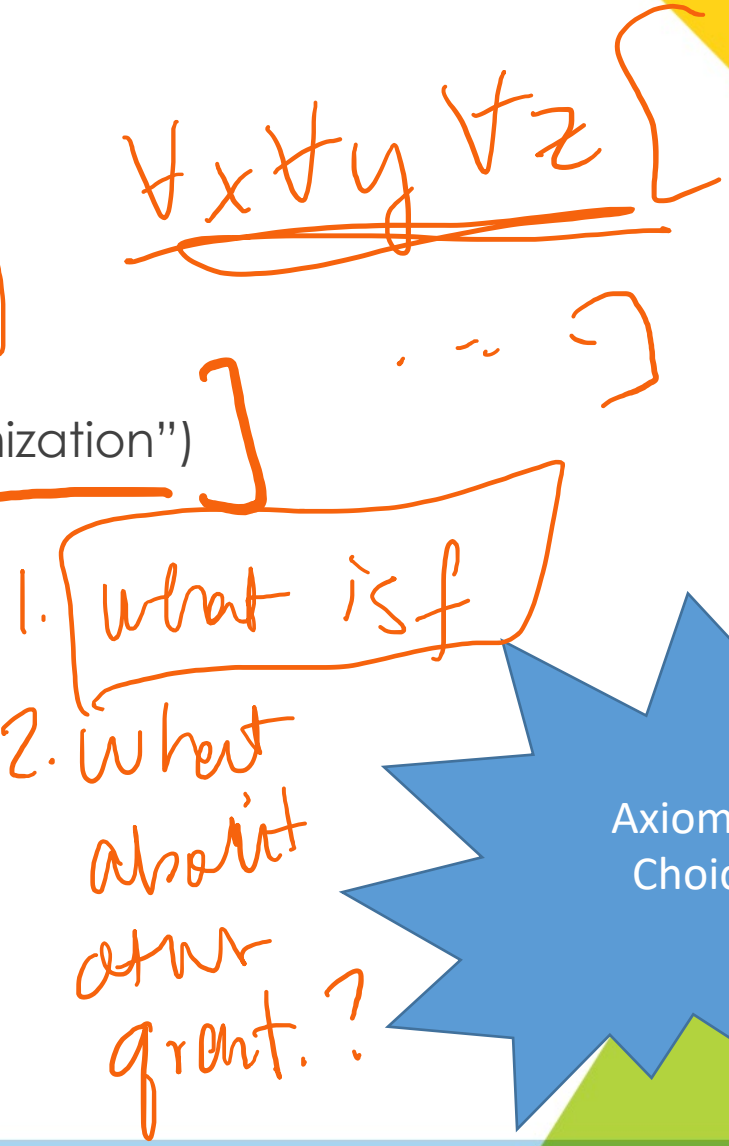


https://www.cis.upenn.edu/~matuszek/Concise%20Guides/Concise%20Prolog.html

Prolog limitations I

- Implicitly all quantifiers be "in front" ("prenex form")
- Implicitly only universal quantification (requires "Skolemization")





Axiom of Choice

Slight detour…

THE EARLY YEARS OF LOGIC PROGRAMMING

This firsthand recollection of those early days of logic programming traces the shared influences and inspirations that connected Edinburgh, Scotland, and Marseilles, France.

ROBERT A. KOWALSKI

The name *Prolog* is ambiguous. It was originally intended as the name for the programming language developed by Alain Colmerauer and Phillipe Roussel in the summer of 1972. The name was suggested by Roussel's wife, Jacqueline, as an abbreviation for *programmation en logique*. In time, however, this abbreviation has been used to refer to the concept of logic programming in general. It is a confusing notion, as claims made for the general concept of logic programming do not always hold for the programming language, Prolog, and vice versa. In an attempt to minimize such confusion, I shall reserve the term Prolog to refer to the programming language alone.

This is not the place for an extensive discussion of what should or should not be regarded as *logic programming*, a term that is equally ambiguous. However, without wanting to stir further controversy, let me hazard the following rough characterization: Logic programming shares with mechanical theorem proving the use of logic to represent knowledge and the use of deduction to solve problems by deriving logical consequences. However, it differs from mechanical theorem proving in two distinct but complementary ways: (1) It exploits the fact that logic can be used to express definitions of computable functions and procedures; and (2) it exploits the use of proof procedures that perform deductions in a goal-directed manner, to run such definitions as programs.

A consequence of using logic to represent knowledge is that such knowledge can be understood *declaratively*. A consequence of using deduction to derive consequences in a computational manner is that the same knowledge can also be understood *procedurally*. Thus, logic programming allows us to view the same knowledge both declaratively *and* procedurally.

The most straightforward case of logic programming is when information is expressed by means of Horn clauses and deduction is performed by backwards reasoning embedded in resolution [29]. But logic programming can also be understood more generally, for example, to include negation by failure [3], set construction [4, 32], or goal-directed reasoning with equations. The advantage of the more liberal notion of logic programming is that it points the way for further developments to encompass richer fragments of logic and give a computational interpretation to a greater variety of proof procedures.

The liberal notion of logic programming does not include a number of related uses of logic in programming. It excludes, for example, systems of constructive logic in which *proofs* are interpreted as programs, and it excludes uses of logic in which computation is construed model-theoretically as evaluating a formula in an interpretation.

This article is a personal account of some of the early history of logic programming, ending with my move from Edinburgh to London in December 1974. The chronicle is unavoidably biased toward my own recollection of events at the University of Edinburgh. I am especially conscious that it does not do justice to related activities that took place during that time at the Université d'Aix Marseilles.

THE EDINBURGH–MARSEILLES CONNECTION

My first contact with the Marseilles group was a three or four day visit in the summer of 1971 at the invitation of Colmerauer, who was then head of the artificial intelligence (AI) team at the university. The group, which consisted of Bob Pasero, Roussel, and Colmerauer, was developing a natural language question-answering system. Roussel and Jean Trudel, a colleague visiting from the University of Montreal, had read [21], which describes the SL-resolution theorem prover, and Roussel was interested in using it for the deductive component of the question-answering system.

Most of my visit consisted of intensive discussions

The liberal notion of logic programming does not include a number of related uses of logic in programming. It excludes, for example, systems of constructive logic in which *proofs* are interpreted as programs, and it excludes uses of logic in which computation is construed model-theoretically as evaluating a formula in an interpretation.

Prolog limitations I

- Requires all quantifiers be “in front” (“prenex form”)
- Only universal quantification (requires “Skolemization”)
- All logical formulas can be converted to CNF
- Only some CNF formulas are Horn clauses

Law + Logic: Active Research

Programming Languages and the Law 2022

ProLaLa 2022

About | Program | Accepted presentations | Call for submissions

Important Update:

> The workshop will be held as a hybrid event with full support for remote participation, following the latest update from the main POPL conference. To attend in person, choose "in-person POPL" option from the main registration page, which later will prompt you to select the specific meetings you want to attend. To participate remotely, choose "Virtual POPL" option which is common to all POPL-week events. Note that in both cases, you will automatically get all the benefits of "Virtual POPL" option, namely, remote access to all POPL-week events as well as the POPL Virtual Workshop. If you are already registered, and wish to either update your information or switch between the in-person and virtual options, use "Update Information" option.

Law at large underpins modern society, codifying and governing many aspects of citizens' daily lives. Oftentimes, law is subject to interpretation, debate and challenges throughout various courts and jurisdictions. But in some other areas, law leaves little room for interpretation, and essentially aims to rigorously describe a computation, a decision procedure or, simply said, an algorithm.

The programming languages community has so far brought very few answers to the problem of having a transparent, accountable implementation of computational law. The current state of affairs is concerning: in many cases, human-critical systems are implemented using technology that is several decades old, resulting in e.g. the IRS relying on assembly code from the 60s or its French counterpart relying on a home-made language from the 90s with tens of thousands of global variables. For institutions stuck with this unfortunate status quo, consequences are many: legacy systems cannot be evolved, in spite of hundreds of millions of dollars spent on "modernization" budgets; mistakes are made and rarely noticed; automatic analyses remain elusive, meaning policymakers are "flying dark"; and in the worst case, as happened with the French military pay computation, families are on the verge of bankruptcy because of incorrect code. However, there is hope. Recent papers published at PL venues (A Modern Compiler for the French Tax Code, CC'21; Catala: a Programming Language for the Law, ICFP'21; Property conveyances as a programming language, Onward!'19), along with a recent NSF proposal for

Important Dates AoE (UTC-12h)

Sun 16 Jan 2022
Workshop

Thu 18 Nov 2021
Notification of acceptance

Thu 28 Oct 2021
Submission deadline

Submission Link

https://prolala22.hotcrp.com

Program Committee

Sarah Lawsky Program Co-Chair
Northwestern University
United States

Jonathan Protzenko Program Co-Chair
Microsoft Research, Redmond
United States

Timos Antonopoulos
Yale University

Denis Merigoux
@DMerigoux

[3/15] The logic programming community identified some key needs for translating law into code:
- defeasibility (or negation-as-failure)
- unification (for dealing with legal qualifications)
- modalities (or some form of deontic logic)

6:03 AM · Jan 31, 2022 · Twitter Web App

2 Likes

https://twitter.com/DMerigoux/status/1488105678799577091

Defeasibility

Open vs. closed worlds

- prolog, homework → if not explicitly true, then false
    - Introduces bias in our knowledge base

“Non-montonicty”

Idea: define some inference

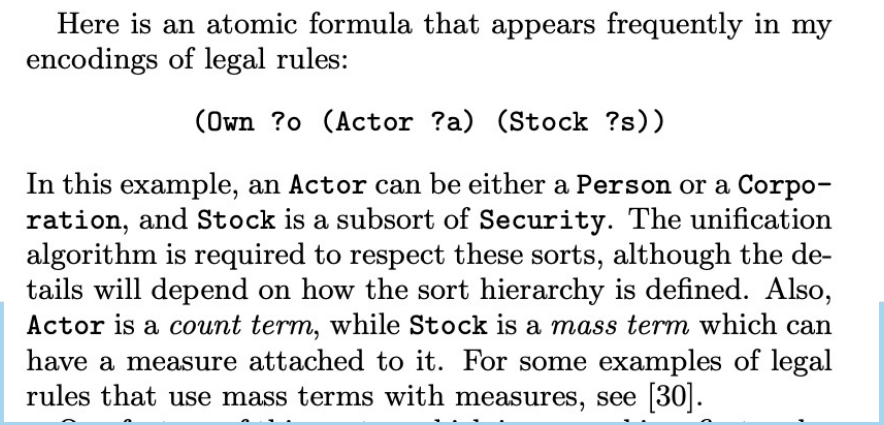
- has4legs(f) → table(f) \/ chair(f)
- …but there is a table here that has one leg!

Unification

Step in resolution for predicate logic

- Algorithm for *binding* variables in order to "melt" two clauses together in resolution
- Fairly simple without functions; very algorithmic

Hard part in law: generalization vs. specialization, things vs. stuff*



Here is an atomic formula that appears frequently in my encodings of legal rules:

```
(Own ?o (Actor ?a) (Stock ?s))
```

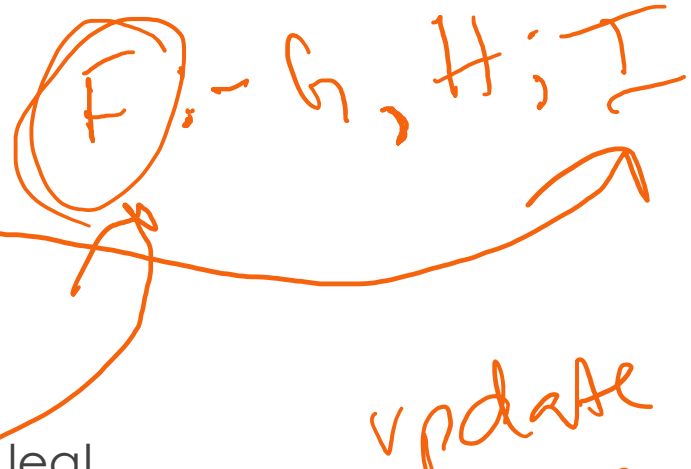
In this example, an **Actor** can be either a **Person** or a **Corporation**, and **Stock** is a subsort of **Security**. The unification algorithm is required to respect these sorts, although the details will depend on how the sort hierarchy is defined. Also, **Actor** is a *count term*, while **Stock** is a *mass term* which can have a measure attached to it. For some examples of legal rules that use mass terms with measures, see [30].

Modal logic

We will cover some modal logics later in the semester:

- Necessity and sufficiency
- Time
- Knowledge (epistemic)

Prolog limitations II



Default logic

Idea: define some inference

- has4legs(f) → table(f) \/ chair(f)
- …but there is a table here that has one leg!

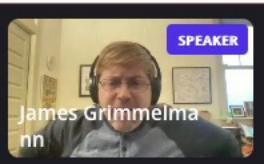
update inferences in light of new info

"Non-montonicty" → a reasonable way to build a knowledge base with "background assumptions"

Law + Logic: Active Research

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 …



Necessary features of logic in AI?

- Non-montonicity
- Law of excluded middle
- Axiom of choice
- Modal expressions

will come up later in time, necessity/sufficiency, semester knowledge