“Programming using logic”

Nik Sultana

With thanks to Andy Rice and Alastair Beresford
Aims

• Learn Prolog

• Reflect about deep questions
  • What’s a declarative language?
  • How does Prolog compare/relate to other languages I know?
  • What kind of problems would I use this language for?

• As a side-effect: learn how to model problems using logic (a.k.a write programs in Prolog).
Why?

- **Technical depth**, and exposure to different ideas.
- Prolog **brings theory to life**: discrete maths, database theory, program semantics, logic and proof.
- Prolog-based languages are **useful**. (Next slide)
Prolog is a living language

Prolog and its derivatives are used in Software-defined networking, various products (Datomic, LogicBlox, SEMMLE), resource planning, automated trading, law enforcement, ...

You don’t have to be a logician to use Prolog.
(You don’t have to be French to speak French…)
How did we get here?

• Prolog emerged from research on logic, programming and linguistics.

• Many years of research by many people in different disciplines.

• For more background see:
  • “The early years of logic programming” by Kowalski
  • “The birth of Prolog” by Colmerauer
This course is taught differently to all the others
Course content will be through videos rather than lectures

• Videos of all the content are available online.

• You can watch them whenever you want.

  • Useful for revision.
Course content will be through **videos** rather than lectures

- Videos of all the content are **available online**.
- You can watch them whenever you want.
- Useful for revision.

"I can watch them naked at 4am while eating cereal. Also, I can re-watch the parts that I haven't understood ... I can pause the lecture and google something ... or even write some relevant code while the lecture is ongoing, all without missing anything!"
We are collecting statistics about how you use the site

• Your supervisor can see how you are progressing.

• We will anonymise the data once the course is over.

• The data will not be used to assess you.
Why?

- **Technical depth**, and exposure to different ideas.

- Prolog *brings theory to life*: discrete maths, database theory, program semantics, logic and proof.

- Prolog-based languages are *useful*. e.g., Software-defined networking, NLP, recent products (Datomic, LogicBlox, SEMMLE), resource planning, automated trading, law enforcement.

(Recall slide. Let’s explore the bigger **Why** next.)
Programming is about processing information
Programming is about processing information based on some model of the problem.
Programming is about processing information based on some model of the problem, and involves changing the world to achieve a result.
Programming **languages** support us in lifting this **cognitive** load.

Computers then carry out automated processing.
• Programming ultimately results in **changing the world** (side-effects are necessary)...

• ...if only by **redistributing voltages** on tiny scraps of silicon!

• But we don’t **think** at that low-level scale! “We don’t design bridges by using quantum physics”.

• We need **abstractions** — programming languages provide us with these.

• **Prolog’s abstractions**: terms (including variables, which have a special role) and clauses. **Simple but powerful.**
Make Programming Great Again
Beware oversimplification

e.g., “Best language ever”
Now with more lambdas!
Cultivate judgement.

i.e., The best language for the task at hand, given its expressiveness, compiler targets, toolchain support, abundance of trained programmers, who else uses it, ..., budget and deadlines.
Spoiler: 3 things about Prolog

• Program with relations, not functions. We need to think slightly differently from “calling a function” or “passing parameters”.

• Giving a procedural interpretation to logic. As a result, lose some properties (e.g., commutativity) but gain others (better control over search space).

• Computation by deduction. Compute with knowledge, as formalised in FOL. In Prolog, truth = provability, falsehood = unprovability, both in a “small world”.
Example

List membership.
Relation vs Function

\[ \text{member}_R \subseteq (\text{Thing} \times \text{ListThing}) \]

\[ \text{member}_f \in (\text{Thing} \times \text{ListThing} \Rightarrow \text{bool}) \]

(Excepting side-conditions, since they’d apply similarly to functions and relations)

- With **functions**, you have inputs and outputs.
- With **relations**, any of the **parameters** could be the inputs or outputs.
Everything
Non-List Things

List Things

Everything
Everything (a.k.a. Herbrand Universe)
Everything (a.k.a. Herbrand Universe)

Non-List Things

1
2
3
...

Lists Things

[1]
[1,1]
...
[1,2,3]
[1,2,3,1]
...
[[1],[2,3],1]
...

26
Every possible assertion (a.k.a. Herbrand Base)
Every possible assertion (a.k.a. Herbrand Base)

member(1, [1])
member(1, [1, 1])
member(1, [1, 2, 3])
member(1, [1, 2, 3, 1])
member(1, [[1], [2, 3], 1])
...
member(2, [1])
member(2, [1, 1])
member(2, [1, 2, 3])
member(2, [1, 2, 3, 1])
member(2, [[1], [2, 3], 1])
...
But what makes some assertions true but others false?
But what makes some assertions true but others false?

\[
\text{member}(1,[1])
\]
\[
\text{member}(1,[1,1])
\]
\[
\text{member}(1,[1,2,3])
\]
\[
\text{member}(1,[1,2,3,1])
\]
\[
\text{member}(1,[[1],[2,3],1])
\]
\[
\vdots
\]
\[
\text{member}(2,[1])
\]
\[
\text{member}(2,[1,1])
\]
\[
\text{member}(2,[1,2,3])
\]
\[
\text{member}(2,[1,2,3,1])
\]
\[
\text{member}(2,[[1],[2,3],1])
\]
\[
\vdots
\]
Which is the ‘membership’ relation I seek?

Relations between Things and Lists

R1(Thing,List)
R2(Thing,List)
R3(Thing,List)
...

31
Which is the ‘membership’ relation I seek?

Relations between Things and Lists

R1(Thing,List)
R2(Thing,List)
R3(Thing,List)
...

Example behaviours

• Empty relation
• Thing is an element of List
• Thing is NOT an element of List
• Thing is the GREATEST element of List
• Thing is the 1st element of List
• Thing is the 2nd element of List
...

...
Which is the ‘membership’ relation I seek?

Answer: this is what your Prolog implementation specifies.
All other “lectures” will take place in Intel Lab

• Bring your own **laptop+headphones**.

• We'll be there to answer your questions and talk to you about the course.

• You are free to work through the course in your own time if you prefer.
Question book

• Help you review the course material.

• Questions categorised
  {Bookwork, Shallow, Deep, Open}.

• Can be used for supervisions, self-study, and revision.
Feedback and problems

• If you are **stuck**:  
  • come to a lab session (Intel Lab)  
  • talk to your peers  
  • talk to your supervisor  
• If the software isn’t working then email me.

(Not actual student experiencing Prolog)
Assessment is through practical exercise (Tick) and an exam question

• Tick’s due date announced shortly by email. Exercise details on the course website.

• You need to complete either the Prolog or the C/C++ exercise.

• There is one Prolog question in the exam.
In Tick we’ll also look for good programming style.

- **Simple is often best.** “Clever” often leads to overengineering, technical debt, ….

- **Readability is essential.**
  - Reason: it makes it easier to evaluate correctness, and maintain the program.
    cf. “Obfuscated C contest”
  - For coding style advice, see “Coding Guidelines for Prolog” by Covington et al.
    - Or “The Elements of Programming Style"
  - Practice empathy, think of the reader (of your code).
Send me your videos

• Explain something in the course.
  • Worked solution to supervision question or a tripos question
  • If they are clear and correct I'll include them in the course and credit you as a contributor at the top of the course webpages. (CV points!)
• Avoid filming naked at 4am..
  • (Explaining something is a good way to learn it)
Wrap-up: Objectives

• be able to write programs in Prolog using techniques such as **accumulators** and **difference structures**;

• know how to model the **backtracking** behaviour of program execution;

• appreciate the **unique perspective** Prolog gives to problem solving and algorithm design;

• understand how **larger programs** can be created using the basic programming techniques used in this course.