Some logic puzzles in Prolog

Mark Armstrong

September 18, 2020

Contents

1 Introduction

These notes were created for, and in some parts **during**, the lecture on September 18th and the following tutorials.

2 Motivation

Today, we begin investigating another non-imperative paradigm other than functional programming: *logical programming*.

As we've seen in the previous hands-on lecture, functional programming takes advantage of *immutability* in order to make reasoning about programs easier. It also focuses on *compositionality*, including in its use of *higher-order functions*, all of which makes programming very much like writing functions in mathematics (where there is no mutable state).

Logical programming also assumes immutability, but instead of compositionality as a method of computation, it uses a (Turing-complete) subset of first-order predicate logic. Programs are databases of *inference rules* describing the problem domain, and programs are initiated by *queries* about the problem domain which the system tries to prove are true (a logical consequence of the rules) or false (refutable by the rules).

To put it simply, in logical programming, you describe the problem, rather than the solution.

3 A note about fonts

In these notes, I use plaintext blocks in order to write the inference rules, and use em-dashes (—) to create the horizontal rules.

In some cases, the em-dashes may not show quite correctly. For instance, in the PDF version of these notes, there is a small space between each dash. In some browsers, they may not show at all (they work in my install of Chrome, at least.)

Apologies for any issues reading these notes caused by this.

4 (Re)introduction to inference rules

Recall: an inference rule has the form

```
 Premise    Premise    …   Premise
------------------------------------- Rule Name
            Conclusion
```

where `Premise` , `Premise` , …, and `Premise` are some statement in our domain, and `Conclusion` is a statement that can be concluded from the premise statements.

In the domains of logics, the statements range over formulae (i.e., boolean expressions built up from boolean constants, predicates and propositional connectors), and we may have rules such as

```
  P     Q
------------  -Introduction
   P   Q
```

which says “given `P` and `Q`, we may conclude `P   Q`”,

```
   P   Q
-----------  -Elimination
     P
```

which says "given `P   Q`, we may conclude `P`", and

```
  P      P   Q
--------------- Modus Ponens
     Q
```

which says, by translating the   to English, "given `P` and if `Q` follows from `P`, then we can conclude `Q`".

(Technically, these are *rule schemas*; the *meta-variables* `P` and `Q` can be instantiated to obtain specific rules.)

Note that in these rules, we have the following *meta-syntax*:

1. Whitespace between premises is understood as a form of conjunction.
2. The horizontal rule is understood as a form of implication.

Any rule which does not have a premise is called an *axiom*; axioms are the known results of our domain, which do not need to be proven. For instance,

```
---------  -Introduction
  true
```

A collection of inference rules (or rule schemas) and axioms gives us a *proof system*.

See, for instance, the natural deduction proof calculus for classical logic.

You have likely seen the *equational* approach to proofs which is favoured by Gries and Schneider's "A Logical Approach to Discrete Math", and used in CS/SE 2dm3 at McMaster using the CalcCheck tool. Proof systems are an alternate approach to proof; see Musa's notes on the relationship from this year's 2dm3.

Inside of a proof system, we may construct proofs of statements via *proof trees*, which are trees where every node is a statement, and the connections between nodes correspond to the use of rules.

For instance, we have the following silly proof of `true` which uses the rules given above.

```
------  -introduction        ------   introduction
 true                         true
```

```
----------------------------------- -Elimination
          true   true
       ----------------- -Elimination
              true
```

Notice, by convention, we write proof trees from the **bottom up**. The root, at the bottom, is what we intend to prove. The leaves, at the top, must either

1. be axioms, or
2. be local assumptions.

Proof trees may be read from the top down, to see how the conclusion follows from the axioms and assumptions. It is generally better to read from the bottom up, though; otherwise the proof often seems to make unwarrented steps, or informally, it "pulls a rabbit from its hat".

5 Inference rules in other domains

The use of inference rules is not limited to the domain of logics.

It is perhaps better not to think of inference rules as defining a *proof system* (which makes us think of truth values and logics), but as defining a *game*: starting from the rules and axioms, what can we obtain?

For instance, consider the following problem.

5.0.1 The two bucket problem

1. The problem Suppose you are given two buckets,

    - one of which holds 5 units of water, and
    - one of which holds 3 units of water.

    You are tasked with collecting exactly 4 units of water; no more, and no less. You begin with 0 units in both buckets.

    You may at any point

    - fill one bucket entirely from a tap,
    - pour the water out of a bucket, emptying it entirely.
    - pour one bucket into another until either the first is empty or the second is full, or

You are tasked with collecting exactly 4 units of water using only those three kinds of actions.

2. The rules Let us represent the state of the bucket by a pair of numbers. In these rules, we will use

   - `x` as a meta-variable for the amount in the bucket which can hold 5 units, and
   - `y` as a meta-variable for the amount in the bucket which can hold 3 units.

   We can begin only if we have 0 units in both buckets.

   ```
             --------- Start
               0 , 0
   ```

   The action of filling a bucket replaces its current amount with the maximum amount.

   ```
     X , Y                  X , Y
   --------- Fill         --------- Fill
     5 , Y                  X , 3
   ```

   :TODO:

   ```
     X , Y                  X , Y
   --------- Empty        --------- Empty
     ? , ?                  ? , ?
   ```

   :TODO:

   ```
     X , Y
   --------- Pour   (provided ???)
     ? , ?

     X , Y
   --------- Pour   (provided ???)
     ? , ?
   ```

6 Prolog

Prolog programs are simply databases of inference rules. An inference rule

```
 A    A    …    A
--------------------
      B
```

is written in Prolog

```
b :- a1, a2, …, an.
```

(notice the period ending the rule). As with our inference rule, this rule states that `b` is true if all of the `a` are true. So we can think of `:-` as , and , as .

An axiom

```
--------------
      C
```

can be written in Prolog as

```
c :- true.
```

or, more simply,

```
c.
```

:TODO:

6.1 The two bucket problem in Prolog

We can begin only if we have 0 units in both buckets.

```
bucket(0,0).
```

The action of filling a bucket replaces its current amount with the maximum amount.