Logic and Modelling

Todd Davies

December 18, 2014

Introduction

This is a unique course developed at the University of Manchester. It explains how implementations of logic can be used to solve a number a number of problems, such as solving hardest Sudoku puzzles in no time, analysing two-player games, or finding serious errors in computer systems.

Aims

This course intends to build an understanding of fundamentals of (mathematical) logic as well as some of the applications of logic in modern computer science, including hardware verification, finite domain constraint satisfaction and verification of concurrent systems.

Additional reading

Contents

1 Setting the stage

Before it is possible to understand some of the content in these notes, it is required to have a basic knowledge of other content that was not covered in my previous notes. This section attempts to give an overview of the relevant topics.

1.1 Decision problems

A decision problem is essentially a question, described by some formal language (such as that of propositional logic) with a yes/no answer. In order to obtain whether the decision problem yields yes or no, input parameters must be specified and the problem evaluated according to the language it is written in.

Another requisite of a decision problem is that the possible set of inputs must be infinite. That is to say that $5 = 5$, or "Is the sky blue?" would not be a decision problem, since the set of inputs in both cases is the $\emptyset$. Likewise, if a problem doesn't yield a yes/no answer (for example, a quadratic equation), then it is not a decision problem.

1.1.1 Decidable problems

A problem is classed as decidable if it is both a decision problem and there exists and algorithm that will be able to compute the correct answer for all (infinite number of) inputs to the problem. Such an algorithm can be described by any Turing complete programming language, in fact, it works both ways; if there is a computer program which can find the correct yes/no answer for a decision problem over all inputs, then the problem must be decidable.

Some decision problems are undecidable. It is impossible to create an algorithm that can always solve the problem for all of its inputs. One such problem is the Halting problem, which asks:

> Given the description of an arbitrary program and a finite input, decide whether the program finishes running or will run forever.

It has been proven that the Halting problem is undecidable when running on a Turing machine. The essence of the proof is that the algorithm evaluating whether the input program will halt or not could be made to contradict itself.

1.2 Mappings

A mapping is a function that takes an input from one set and returns an output from another (or the same) set. Conversely, you could describe a function as mapping one set to another.

The notation $f'(x) = f(x) + \{a \to b\}$ means:

$$f'(x) \overset{\text{def}}{=} \begin{cases} b & \text{if x = a} \\ f(x) & \textit{otherwise} \end{cases}$$

1.3 Binary relations

Binary relations are often denoted by $\Rightarrow$, and it's reverse is denoted by $\Leftarrow$ such that $y \Leftarrow x \overset{\text{def}}{=} x \Rightarrow y$. Along a similar train of thought, the symmetric closure of $\Rightarrow$ is $\Leftrightarrow$, where $\Leftrightarrow \overset{\text{def}}{=} \Rightarrow \cup \Leftarrow$

1.4 Orders

An order is a relation that is irreflexive and transitive. This means that each pair in the relation cannot be constructed of only one element (e.g. $x > x$ wouldn't be allowed) and if $((x > y) \wedge (y > z)) \implies x > z$.

1.5 Directed graphs

A directed graph consists of a set N and a binary relation on the set R. The elements in N are nodes, and the relation R defines the edges between the nodes. A directed graph is finite if it's set is finite.

A *path* is a subset of R where each element of the path will end at the start of another element with the exception of the start and end elements. A cycle is a path where there is no start and end pairs.

1.5.1 Directed Acyclic Graphs (DAG's)

A Directed Acylcic Graph is a directed graph that has no cycles. If a dag has a node n such that every other node in the dag is reachable by n, then the dag is *rooted at* n.

1.6 Directed acylcic graphs

2 Propositional logic

3 Evaluating formulae

4 Satisfiability

5 CNF

6 Definitional Clausal Form Translation

7 DPLL

8 Encoding problems in SAT

9 Randomly generated clause sets

10 Randomised algorithms for satisfiability checking

11 Signed Formulae

I, then it is denoted by $I \models A = b$, and consequently, I is a model of the signed formula $A = b$.

Note:

This is also when $I(A) = b$.

If a signed formula has a model, then it is specifiable.

11.1 Finding a model of a specifiable formula

If we had a signed formula such as $A \Leftrightarrow B = 1$, the three interpretations that model it are:

A	B
0	0
0	1
1	1

11.2 Tabelau

A tableau is a tree with each node being a signed formula. The tableau for the signed formula $A = b$ would have the root node as $A = b$.

The notation for a set of branches is $B_1|...|B_n$, where each B_i is a branch.

11.2.1 Branch expansions

There are a number of rules that can be used to expand the branches of a tableau.