CO141 - Reasoning About Programs

Prelude

The content discussed here is part of CO141 - Reasoning About Programs (Computing MEng); taught by Sophia Drossopoulou, and Mark Wheelhouse, in Imperial College London during the academic year 2018/19. The notes are written for my personal use, and have no guarantee of being correct (although I hope it is, for my own sake). This should be used in conjunction with the (extremely detailed) notes.

Material Order

These notes are primarily based off the notes on CATe, as they cover the lecture slides in great detail. This is the order in which they are uploaded (and I'd assume the order in which they are taught).

1. *Introduction and Motivation (full notes).pdf*
2. *Stylised Proofs (full notes).pdf*
3. *Induction over natural numbers (full notes).pdf*
4. *Induction over Haskell data structures (full notes).pdf*
5. *Induction over recursive relations and functions (full notes).pdf*
6. *Java - Program Specifications (full notes).pdf*
7. *Java - Conditional Branches (full notes).pdf*
8. *Java - Method Calls (full notes).pdf*
9. *Java - Recursion (full notes).pdf*
10. *Java - Iteration Informal (full notes).pdf*
11. *Java Reasoning - summary.pdf*
12. *Loop case study.pdf*
13. *Java - Iteration Formal (full notes).pdf*
14. *Case Studies - overview (full notes).pdf*
15. *Case Studies - Dutch Flag Problem (full notes).pdf*
16. *Quicksort (full notes).pdf*

Introduction

This module will cover Proof by Induction from first principles, and shows how a recursive definition can implicitly introduce an inductive principle, how the inductive principle introduces a proof schema, and how the schema can be used to prove a property of a inductively defined set, relation or function. This will go into more detail regarding valid uses of quantifiers, when we're able to use the induction hypothesis, how auxiliary lemmas can help, as well as what cases we will need to strengthen properties to prove weaker ones.

Binding Conventions

The binding conventions in this module are the same as the ones used in **CO140 - Logic**; with the addition of $\forall x$, and $\exists x$ before $\neg$.

Formalising a Proof

For this section, we'll work on one example proof, with the given facts;

(1) a person is happy if all their children are rich

(2) someone is a supervillain if at least one of their parents is a supervillain

(3) all supervillains are rich

We want to show that "all supervillains are happy".

Proof in Natural Language

The given argument is that "All of a supervillain's children must therefore also be supervillains; and as all supervillains are rich, all the children of a supervillain are rich. Therefore, any supervillain is happy". However; we've made a few assumptions in this proof - we assume that a supervillain is always a person, and that a supervillain has children (as well as the fact that parent, and child aren't formally defined to be related concepts).

Therefore, we need to generalise statement (1) OR add an additional assumption (4);

(1) **someone** is happy if all their children are happy

(4) a supervillain is also a person

Formal Argument

Given:

(1) $\forall x[\text{person}(x) \land \forall y[\text{childof}(y, x) \to \text{rich}(y)] \to \text{happy}(x)]$

(2) $\forall x[\exists y[\text{childof}(x, y) \land \text{supervillain}(y)] \to \text{supervillain}(x)]$

(3) $\forall x[\text{supervillain}(x) \to \text{rich}(x)]$

(4) $\forall x[\text{supervillain}(x) \to \text{person}(x)]$

To show:

(α) $\forall x[\text{supervillain}(x) \to \text{happy}(x)]$

(Stylised) Proof:

	take arbitrary G	
(a1)	$\text{supervillain}(G)$	
(5)	$\text{person}(G) \land \forall y[\text{childof}(y, G) \to \text{rich}(y)] \to \text{happy}(G)$	from (1)
(6)	$\text{person}(G)$	from (a1), and (4)
	take arbitrary E	
(a2)	$\text{childof}(E, G)$	
(7)	$\text{supervillain}(E)$	from (a1), (a2), and (2)
(8)	$\text{rich}(E)$	from (3), and (7)
(9)	$\forall y[\text{childof}(y, G) \to \text{rich}(y)]$	from (a2), (8), and arbitrary E
(10)	$\text{happy}(G)$	from (5), (6), and (9)
(α)		from (a1), (10), and arbitrary G

While this can be proven fairly easily, and with great confidence, via first-order natural deduction, the proof is often tedious, and the intuition might be lost. On the other hand, stylised proofs have an explicit structure, few errors (compared to free-form) - although errors are still possible.

Our goal for our proofs are that they should only prove valid statements, are easy to read / check, and are able to highlight intuition behind arguments. The rules for a stylised proof are as follows;

1. write out, and name each given formula
2. write out, and name each goal formula
3. plan out proof, and name intermediate results
4. justify each step
5. size of each step can vary as appropriate

Planning, and justifying the the steps follow extrmeely similar rules to natural deduction - the rules for proving P are as follows;

• $P = Q \wedge R$	prove both Q, and R ($\wedge$I)
• $P = Q \vee R$	prove either Q, or R ($\vee$I)
• $P = Q \to R$	prove R from assuming Q ($\to$I)
• $P = \neg Q$	prove $\bot$ from asuming Q ($\neg$I)
• $P = \forall x[Q(x)]$	show $Q(c)$ from arbitrary c ($\forall$I)
• $P = \exists x[Q(x)]$	find some c, and show $Q(c)$ ($\exists$I)
• P	prove $\bot$ from assuming $\neg P$ (PC)

On the other hand, if we have proven P, we can do the following;

• $P = Q \wedge R$	both Q, and R hold ($\wedge$E)
• $P = Q \vee R$	case analysis ($\vee$I)
• $P = Q \wedge (Q \to R)$	R holds ($\to$E)
• $P = \forall x[Q(x)]$	$Q(c)$ holds for any c ($\forall$E)
• $P = \exists x[Q(x)]$	$Q(c)$ holds for some c ($\exists$E)
• $P = \bot$	anything holds ($\bot$E)
• $P = \neg Q$	$Q \to \bot$ holds ($\neg$E)
• P	use a lemma, or any logical equivalence