Computer Programs as Dialogue Games

Martin Churchill
Supervisors: Guy McCusker and Jim Laird
Department of Computer Science

4th June, 2009
Meeting of Minds
University of Bath

Computer Crash

- On June 4th, 1996 the Ariane 5 expendable launch system was launched into space.
- 37 seconds later the rocket veered off its flight path and was destroyed by its automated self-destruct system.
- This was caused by the control software trying to fit a 64-bit number into a 16-bit piece of memory.
- Sometimes a bug is more than just a nuisance.

Quality Control

- Computers are everywhere, including many safety-critical situations!
- So it's important to avoid "bugs"
- Quality control exists — but prone to complacency / human error
- Better if we can formally and mechanically check programs.

Formal Methods

- We study computer programs as objects in their own right.
- We use
  - *Mathematics* to describe our programs, how they behave, and what they mean.
  - *Computer science* to motivate our “design” choices, and implement/use the resulting results

Types and Terms

- We describe what computer programs look like by using *formal grammars*
  - $\vdash 3+3 : \texttt{Num}$ says that $3+3$ is a valid computer program of type "number"
  - $\vdash x := x+1 : \texttt{Com}$ says that the program that adds 1 to the value of x has type "command"
  - $\vdash \_ + \_ : \texttt{Num} \times \texttt{Num} \rightarrow \texttt{Num}$ says that $+$ is a valid program takes two numbers as input and outputs a number.

Semantics

- We give *operational semantics* to these well-typed terms to describe how programs *behave*
  - $3 + 3 \Rightarrow 6$ tells us that the program "3 + 3" reduces to final answer "6"
  - $(x := x + 1, \{x \mapsto 6\}) \Rightarrow \{x \mapsto 7\}$

Formal Reasoning

- We now wish to reason about these programs:
  - Does this program meet its specification?
  - Is the behaviour of (obviously correct) Program A equivalent to the behaviour of the (more efficient but less clear) Program B?
  - Is it possible that Program A will go into an infinite loop and get stuck?

Program Equivalence

- We need to formalise this notion of *equivalence* of two programs
- Not enough to simply talk about "returning the same answer" — makes sense for Num, but what about Num $\to$ Com?

We say that M_1 for M_2 are *equivalent* iff

Replacing M_1 by M_2 in any program of type Num yields the same answer (and vice-versa).

A Large Quantification

- Reasoning about program equivalence directly is complex
  - We have a large quantification — "in **any** program"
- So we seek *models* of the language that identify equivalent programs

Game Semantics

- We can model programs as *dialogue games* between the program and its environment.
- **Program** and **Environment** alternately play *moves*, which might represent an input request or the final answer.
- Programs are represented as *strategies* for **Program** — recipes that express which move the program should play based on what's happened so far.

Example — Addition

For example, we might consider a program for addition which takes two numbers as input and returns a number ($+ : \mathtt{Num} \times \mathtt{Num} \to \mathtt{Num}$).

$\mathtt{Num}$	$\times$	$\mathtt{Num}$	$\to$	$\mathtt{Num}$	
				q	E
q					P
n					E
		q			P
		m			E
				$m + n$	P

Game Semantics 1

- We represent types as games, and terms as strategies.
- These models are *fully abstract* — equivalent programs are represented by the same strategy.
- Programming language features correspond to constraints on strategies

$\Rightarrow$ A flexible and accurate way to model programs.

Game Semantics 2

The main reasons this works:

- **Compositional reasoning** — the meaning of a program is given by the meaning of its components.
- **Operational, sequential content** — the models are sequential in nature, giving fine grained control of *when* things happen.
- **Flexibility of constraints** — the models are rich in structure, leading to flexibility for modelling different programming languages

Verification

- We can check if two programs are correct by seeing if the corresponding strategies are the same.
- Elsewhere[1], work analyses how this can be done mechanically (and for which languages)
- We can further use these ideas to check if programs satisfy specifications (tranforming program properties to properties on strategies)

[1] Ong, Ghica...

Current Work in Bath

Includes:

- Giving a higher-level, algebraic account of many of the characteristics of game semantics.
- Investigating the use of game semantics for analysing *access control* and *interference*

Questions?