UNIVERSITY OF
Southampton

School of Electronics
and Computer Science

COMP2212
PROGRAMMING LANGUAGE CONCEPTS

Julian Rathke and Pawel Sobocinski

INTRODUCTION TO SEMANTICS

BUT WHAT DOES IT ALL MEAN?

- In this lecture we look at the topic of Semantics of Programs.
- "Semantics" refers to the **meaning** of programs.
  - a semantics of a program is a specification of a program's runtime behaviour. That is, what values it computes, what side-effects it has etc.
  - the semantics of a **programming language** is a specification of how each language construct affects the behaviour of programs written in that language.

- Perhaps the most definitive semantics of any given programming language is simply its compiler or interpreter.
  - If you want to know how a program behaves then just run it !
- However, there are reasons we shouldn't be satisfied with this as a semantics.

WHY WE NEED FORMAL SEMANTICS

- Compilers and interpreters are not so easy to use for reasoning about behaviour. Why ?
  - not all compilers agree!
  - compilers are large programs, it is possible (and common) that they contain bugs themselves. So the meaning of programs is susceptible to compiler writer error!
  - the produced low-level code is often inscrutable. It is hard to use compiler source code to trace the source of subtle bugs in your code due to strange interpretations of language operators.
  - compilers optimise programs (allegedly in semantically safe ways) for maximum efficiency. This can disturb the structure of your code and make reasoning about it much harder.

ADVANTAGES OF FORMAL SEMANTICS

- In contrast, a formal semantics should be precise (like a compiler) but written in a formalism more amenable to analysis.
  - this could be some form of logic or some other mathematical language.
  - don't need to worry about efficiency of execution and can focus on unambiguous specification of the meaning of the language constructs.
  - can act as reference 'implementations' for a language: any valid compiler must produce results that match the semantics.
  - they can be built in compositional ways that reflect high-level program structure.

APPROACHES TO SEMANTICS

- There are three common approaches to giving semantics for programs:

- **Denotational Semantics** advocates mapping every program to some point in a mathematical structure that represents the values that the program calculates.
  - e.g. [[`if (0<1) then 0 else 1`]] = *0*

- **Operational Semantics** uses relational approaches to specify the behaviour of programs directly. Typically inductively defined relations between programs and values they produce, or states the programs can transition between are used.
  - e.g. `if (0<1) then 0 else 1 → if (true) then 0 else 1 → 0`

- **Axiomatic Semantics** take the approach that the meaning of a program is just what properties you can prove of it using a formal logic.
  - e.g. Hoare Logic.

- We'll look at the first two of these in a little more detail in the next two lectures.