Bringing it all together

- Defining the semantics brings many concepts together
  - Concepts we have seen before: identifier, variable, environment, instruction, procedure value, kernel language
  - New concepts: semantic instruction, semantic stack, memory, execution state, execution, abstract machine
- We gave semantic rules for the kernel language instructions, to show how they execute in the abstract machine
- We used the semantics to prove program correctness, by using it as bridge between specification and program

$$\begin{tikzcd}\boxed{\text{Specification}} \arrow[r, leftrightarrow, "\text{Semantics}"] & \boxed{\text{Program}}\end{tikzcd}$$

Discrete mathematics

- The abstract machine is built with discrete mathematics
- For our students at UCL, it is the first time they see a complicated system built with discrete mathematics!
  - Even engineering students, who are quite used to integrals, differential equations, and complex analysis, which are all continuous mathematics
- Discrete mathematics is important because that's how computing systems work (both software and hardware)
  - Surprising behavior and bugs become less surprising if you understand the discrete mathematics of computing systems
  - Too often, continuous models are used for computing systems
  - All this applies to the real world as well (beyond computing systems)

Why semantics is important

- Semantics is part of programming
  - As a programmer, you are extending the system's semantics: you are writing specifications, designing and implementing abstractions (which we will see soon), and reasoning about your work
- The design of any complicated system with parts that interact in interesting ways (like programming languages and programs) should be done hand in hand with designing a semantics
  - Designing a *simple* semantics is the only way to avoid unpleasant surprises and to guarantee a simple mental model
  - You don't need to understand the semantics to take advantage of it: its mere existence is enough
    - So users of your system will also reap the benefits of a simple semantics
- « Semantics is the ultimate programming language »
  - Invariants as the ultimate loop construct
  - Data abstractions as new kernel language instructions

Using semantics

- Semantics has many uses:
  - For design (ensuring the design is simple and predictable)
  - For understanding (the nooks and crannies of programs)
  - For verification (correctness)
  - For debugging (a bug is only a bug with respect to a correct execution)
  - For visualization (a visual representation must be correct)
  - For education (pedagogical uses of semantics)
  - For program analysis and compiler design
- We don’t need to bring in details of the processor architecture or compiler in order to understand many things about programs
  - For example, our semantics can be used to understand garbage collection (explained in the textbook)
  - We will use the semantics when needed in the rest of the course