IN5100: Formal Semantics of C in 𝕂

Peter Ölveczky

October 18, 2023

Curriculum

The curriculum is the paper

> Chucky Ellison, Grigore Rosu: *An executable formal semantics of C with applications.* In Proc. POPL 2012. ACM. Available, certainly from UiO, via `https://dl.acm.org/doi/10.1145/2103621.2103719`.

As always, I do not expect students to be experts in C or 𝕂, although some basic knowledge of C might be useful. Try to get the big picture, especially being able to answer the following "questions."

What to know?

- (Section 1) *Why* is a formal semantics of C needed? What benefits will/should such a semantics yield?
- (Section 2) What are the main differences between the presented C semantics and previous efforts? For example in terms of executability.
- (Section 3.1) Why does the C standard allow behaviors (programs, I guess) that are undefined or only partially defined?
- Why shouldn't we require one fixed semantics for all C programs? That is, why shouldn't we require all C code be portable?
- (Section 3.2) Try to understand the three code fragments in Section 3.2, and explain why the result is as it is.
- (Section 3.3) No need to understand this in detail. And no need to read the last paragraph in 3.3.
- (Section 4.2 and Figure 2) Try to have a brief overview/understanding of what "kinds" of information (about a C program execution) that is maintained in the global state of the semantics.
- (Section 4.3) Try to have some understanding of how a memory (location) is represented in the semantics.

- (Section 4.4.1) What do you think is represented (or "what happens") in the $\mathbb{K}$ "code" in the middle of the right column on page 537?
- Otherwise, no need to focus on the details in Section 4.4.
- (Section 4.6) Why are some of the C expressions towards the end of page 539 undefined?
- Why is the "entire semantics" needed to check whether the code sketch on top of page 540 is undefined?
- No need to read the last four paragraphs in Section 4.6.
- (Section 5) What do they test their semantics on?
  - Why do they exclude some programs from this "torture set"?
  - Why do the authors first only "get inspired by" 30% of the torture test programs when developing their semantics?
- What are the results of their evaluations, compared to existing compilers.
- (Section 6) What are some applications of their semantics? No details needed; just get the basic ideas/applications.
- (Section 8) What was the effort needed to develop the semantics of the standard?