Synthesizing Abstract Specifications

William Schultz

September 28, 2021

When writing a formal specification, one always makes a choice about what level of abstraction the specification is written at. Generally, it may be considered better to have a specification that is sufficiently abstract, in the sense that it permits a variety of possible implementations without specifying too many low level implementation details. Of course, specifications that are too abstract are also not useful, since they may not provide enough details to implement a concrete algorithm. For example, if we were to specify a C compiler, a specification that states only that the "generated assembly code permits only machine states that are permitted by the program semantics" is useless to an implementer. However, there is an art to writing specifications that are not overly restrictive e.g. that do not rely on architecture specific subtleties or that assume some particular implementation approach. Specifications also serve as artifacts for understanding how an algorithm or protocol works. If a specification is written without a sufficient level of abstraction, it can leave little room for modifications or extensions, making an implementer wary to extend or optimize a complex, detailed protocol.

Choosing the right level of abstraction, however, is a difficult and subjective task, and it is a skill that may only be developed after years of writing specifications or with expertise in a particular domain. So, we consider the following problem. Given a formal specification S, automatically generate more abstract specification S_H, where $S \Rightarrow S_H$.