Interpretation of Package Specifications *

Topics:
- what interpretations are;
- why interpretations are useful;
- adequate hidden annotations of a package;
- constructing interpretations of visible specifications;
- interpretation of subprogram annotations;
- full specification of subprogram bodies in a package;
- interpretation of axioms;
- interpreting dependent specifications.

Each visible annotation in a package specification imposes a constraint on the package hidden part. This constraint is expressed by a hidden annotation, which is called an interpretation of the visible annotation. The interpretation of an annotation of a private type is an annotation of the corresponding full type definition. The interpretation of an annotation of a subprogram is an annotation of the subprogram body. The interpretation of an axiom is, in the most general case, a modified type constraint on the state type. However, interpretations of axioms are often equivalent to fully quantified boolean assertions.

Interpretations have two important applications. First, they can be used to guide the construction of a package body. Secondly, they can be used to divide the complicated problem of proving (as opposed to checking at runtime) consistency between the visible and hidden parts of a package into simpler problems.

Interpreting visible annotations is a nontrivial process for humans. In all but the simplest examples, the number of details involved makes the chance of errors likely. Luckily, constructing interpretations is automatable. In this chapter, we outline how to do it in simple cases. Examples of interpreting subprogram annotations and axioms are described. The goal of this chapter is to give the reader an idea of what interpretations look like and how they can be applied in the process of constructing a package hidden part. This chapter provides an introduction to the topic and motivation for the next chapter, which integrates interpretation into the process of constructing the hidden part of a package. It will be quite clear that the use of interpretations in general can be — and must be — supported by automated tools.
definitive treatment of general methods of constructing interpretations is beyond the scope of our discussion.

9.1 Why Interpretations Are Useful

Let us digress for a moment to discuss why it is useful to have interpretations available when constructing a package body. Suppose we are given the package specification SETS with the private part in Section 8.1.1, and that we are about to implement subprogram bodies for ADD and REMOVE. These procedures will be implemented as operations on set records. But their visible specifications do not mention set records; they are specified entirely in terms of IS_MEMBER. How do we decide, for example, which procedure should increase the value of a LAST_ELEM component of a set record, and which should reduce it? Obviously we would like to guard against a misunderstanding whereby the meanings of the two operations are reversed!

Such decisions must be based on an earlier decision about how to represent an abstract set by a set record — i.e., how the basic IS_MEMBER function is interpreted on set records.

Now consider what happens if we disregard the interpretations of the visible specifications of ADD or REMOVE. This means that we disregard how the specification

\[
\text{procedure ADD (E : in ELEM; S : in out SET);}
\]

\[
\text{where out (for all U : ELEM => IS_MEMBER(U, S) \rightarrow (U = E or IS_MEMBER(U, in S));}
\]

can be expressed as an equivalent annotation involving internal set record structures. Similarly, we disregard the specification for REMOVE. If we do this, we must guess how to implement the bodies, because we have no other information about ADD and REMOVE. When we are finished, we will test them against the visible specifications. That is, we would implement bodies for ADD and REMOVE and then execute both their bodies and their visible specifications independently on test cases and compare the results.\(^1\)

This approach is rather haphazard. It has the drawback that misunderstandings will not be detected until we have finished an implementation.\(^2\)

Suppose, on the other hand, that we do consider the interpretations of the visible specifications. The one above, for example, is interpreted (in the SETS package body where the set record representation and the hidden specification of IS_MEMBER are visible — see Sections 8.1.1 and 8.4) as

\(^1\)This assumes we write a body for IS_MEMBER.

\(^2\)The most expensive software errors are those that go undetected for the longest time.