Assessing Z Specifications

As with all specification methodologies we examine, creating a Z specification is not synonymous with creating a “correct” Z specification. Completeness and consistency remain as two unavoidable general issues. Completeness is about whether we have written enough system properties so that all the desired system properties are implied to be true (if incomplete, an implementation meeting the specification still is not right). Consistency is about whether we have written too many system properties so that it is impossible for them all to be simultaneously true (i.e., if inconsistent, an implementation meeting the specification is unattainable).

Specification Animation

Specification animation, or prototyping, consists of the direct execution of a specification to observe its effect. An animation system is a computer tool that takes a specification “as is” (or as close to that ideal as possible) and uses it to apply operations and observe their results.

Consistency looms as a major concern since its ultimate resolution is the construction of an implementation (a model) that conforms to the specification. However, finding it impossible to complete a conforming implementation is a catastrophic way in which to detect an inconsistent specification —the work on both the specification and the program has been in vain. Specification animation is one approach that aids in early detection of inconsistency, and we will see that languages such as Miranda that support a high level of abstraction can facilitate the development of prototypes.

However, even after we are convinced of the completeness and consistency of a specification, the primary question remains — does the formal specification accurately capture our original intentions? This question is more elusive to answer. Normally our “intentions” have no other formal embodiment, and we have no firm basis for assessing the formal specification. Again, an animation that permits us to witness system behavior is a major aid in this regard. However, this approach suffers the same deficiency as program testing — it may detect errors, but will never guarantee their absence.

Automated Deduction

One natural way to use a formal specification is to deduce from it that critical properties of the desired system must be true. If this can be successfully accomplished, we gain confidence that we have expressed our intentions. For these reasons, computer tools for Z specification sometimes are packaged with a general automated theorem prover. This is to facilitate the mechanical verification that properties of a specified system are indeed logical consequences of the system’s specification. However, the confluence of the current state of the art in

automated theorem proving and the diversity and complexity of Z components and assertions has prevented progressing beyond experimental versions of such tools for Z. Other specification languages provide a more tractable environment, but further research on automated deduction is still needed. Tools specifically tailored to deduction in a particular specification language, general theorem proving systems, and generic tools such as Prolog have all been utilized.

Model Checking

An alternative approach to specification assessment is referred to *as model checking*. Rather than the formula manipulation strategy used in proving, model checking considers interpretations and truth-oriented analysis. This is the logic counterpart to program testing. We earlier noted the inadequacy of this approach — in general, there are an unlimited range of interpretations, so it's impossible to check for truth in them all. Therefore, as with program testing, with model checking we set the more modest goal of attempting to locate errors. As with program testing, the failure to locate errors does not assure us of correctness. So we need to avoid the risk of being misled, but to the extent it can help identify errors, it can be useful.

Inconsistency is revealed by finding that no interpretation is a model and so must involve the eventual examination of all interpretations. Verifying a property is a logical consequence involves insuring that *every* model of the specification is also a model of the property. Hence in either situation, the model checking approach requires unlimited checking. Even severely limiting the interpretations under consideration in software systems fails to significantly ease the problem of too many cases. Suppose we are considering a property of a binary relation. If we consider only binary relations on a 5 item set, there are 2^{25} > 33 million such relations. Even with this tiny beginning, checking that all are models (or none is) is already a sizable task.

Despite these caveates, the model checking approach has been found useful. The identification of any error in a specification we thought finished leads to wide ranging reconsideration. The Nitpick system is an example of a model checking system for a dialect of Z. There is a link to the homepage of this system on our class Web page.