Proof rules for VDM statements

Robert Milne
STC Technology Ltd, London Road, Harlow, Essex, CM17 9NA, United Kingdom

1 Introduction

1.1 Scope

Certain dialects of VDM include constructive declarations of operations using statements, as well as implicit declarations using pre-conditions and post-conditions. This note concerns how such statements are to be understood for a language in which the primary stress is not on transformations of states but on assertions about states.

In the absence of a standard for VDM, the dialect considered is the STC VDM Reference Language [1,2]. This is based on the VDM of Jones [6] and intended for use in the STC group of companies. Most of the note should be equally applicable to other dialects.

1.2 Why statements are needed

Much teaching material for VDM deals with implicit declarations of functions and operations in terms of pre-conditions and post-conditions. Some deals with constructive declarations of types using maps (which are general arrays), lists and sets, and with the formation of subtypes using invariants. Only very little deals with the constructive declaration of functions and operations using expressions and statements, presumably because expressions and statements are thought to be familiar to users already.

This emphasis in the teaching material does not mean that statements can be omitted from VDM. Without statements, VDM would be unable to provide a clear development path from implicit specifications to constructive ones. Though it would let users make assertions about operations it would not let them compose operations from other operations. (Combinations of pre-condition and post-condition 'quotations' can serve as compositions of operations but they are often long-winded and sometimes unimplementable.)

In fact without statements VDM would be curiously lop-sided. For control flow it would permit only implicit declarations, whilst for data structuring it would permit at least:

- implicit declarations (in which a type would be marked as not yet defined and would be constrained by the pre-conditions and post-conditions of functions and operations acting on its elements);
- constructive declarations (in which a type would be described using maps, lists and sets and would be subject to an invariant);

- proof rules for justifying the refinement of implicit declarations of types into constructive ones.

The first of these usually has a counterpart for operations in VDM. Statements provide a counterpart to the second. Proof rules like those in this note provide a counterpart to the third: they allow implicit declarations of operations to be refined into constructive ones. They can also be viewed as defining conditions in which combinations of pre-condition and post-condition 'quotations' are implementable (which here means essentially that they are monotonic with respect to theory inclusion [10]).

1.3 How the proof rules are expressed

The proof rules need to be expressed in a compositional manner: for a composite statement the pre-condition and post-condition must be composed from the pre-conditions and post-conditions of its components. When this is done, the proof rules can be used to justify the refinement of an implicit declaration of an operation into a constructive one by decomposing the pre-condition and post-condition of the operation into pre-conditions and post-conditions of statements.

Here pre-conditions and post-conditions are formulated using equations which are like those for synthesised attributes in attributed grammars: there are functions pre and post which calculate a pre-condition and post-condition for a statement in terms of pre-conditions and post-conditions of the components of the statement. These components are not necessarily themselves statements, so pre and post are overloaded.

An equivalent formulation of the pre-conditions and post-conditions could be based on inference rules. These would let a pre-condition and a post-condition for a composite statement be inferred from pre-conditions and post-conditions for the components.

Either of these formulations produces rather untidy results. This is due partly to the emphasis on variables in the proof rules and partly to the VDM distinction between pre-conditions and post-conditions. Two somewhat different approaches to proof rules for statements are described in 2.1 and 2.2; these can produce quite tidy results when they are applied to languages that have been designed suitably.

The proof rules given here differ from those suggested before for VDM [6,7], because, for example:

- they rely on 'syntactic' entities such as variables instead of 'semantic' entities such as states (as exemplified in the treatment of assignment statements in 6.1.3, conditional statements in 6.1.4, and sequencing statements in 6.5.3);

- they handle loops by introducing general induction rules (given in 6.1.6) that avoid explicit uses of well-founded relations;

- they deal with a crude modularity mechanism (discussed in 6.2.2).