An Extended VDM Refinement Relation

Morten Elvang-Góransson¹ and Robert E. Fields²

¹ CRI Space Division, Computer Resources International A/S, Bregnerødvej 144, DK-3460 Birkered, Denmark. meg@csd.cri.dk.
² BAe Dependable Computing Systems Centre and Human-Computer Interaction Group, Department of Computer Science, University of York, YO1 5DD, UK. bob@minster.york.ac.uk.

Abstract. The original definition of refinement proof obligations in VDM is reviewed and examples are discussed which, while being intuitively sensible, pose problems for this definition of refinement. An extended VDM refinement relation is introduced, to cope with the problems. Some non-standard applications of the extended refinement proof obligations are discussed.

1 Introduction

In [3] extended normal refinement is suggested as a natural characterization of refinement in VDM. The extended normal refinement proof obligations are generalizations of the normal VDM refinement proof obligations. In addition to having the normal VDM refinement proof obligations as special cases, the extended normal refinement proof obligations also account for refinements which are biased (towards a particular implementation) and partial (in that not all abstract states are represented). These two special types of refinement cannot be handled by the normal refinement proof obligations.

Other more general definitions for refinement relations (e.g., Schoett [8], Nipkow [7], Dahl [2]) are able to handle a wider class of refinements than the normal refinement proof obligations can, and for some of the definitions a wider class, in principle, than the extended normal refinement proof obligations.

In [4] we made the claim that the retrieve function of the normal refinement proof obligations is central to software developers’ way of thinking, and that an essential feature of the original VDM refinement relation was lost in the transition from retrieve functions to abstraction relations. It has never been clear to the present authors what the real gain in introducing a relation has been (apart from, of course, mathematical beauty!), and this paper suggests that there is, in fact, no such benefit. In contrast to other generalizations of normal VDM refinements, the extended normal refinements are expressed using a retrieve function and not a relation.

We claim that extended normal refinement is a sufficient and very usable extension of the normal refinement and that it should be preferred over more general types of refinement, as defined by authors such as Schoett [8] and Nipkow [7].
The extended normal refinements are believed to be complete, in the sense that no sensible examples are known where they have not been able to express a refinement which an intuitive understanding suggests should hold between an abstract and a concrete specification.

The extended normal refinements provides more explicit information about the properties of the actual refinement relation that can be established between an abstract and concrete specification. The reason for this is that the software engineer must define to precisely what extent a refinement is biased and partial. In the opinion of the authors, bias and partiality in a refinement can result from engineering compromises which are negotiated between the specifier and implementer. In more general types of refinements, such compromises can more easily be overseen, because no explicit record of it is made.

Therefore, in our view, the characterization of refinements offered by the extended normal refinement proof obligations better reflects the requirements for specific applications, because more specific information about the problem at hand is reflected directly in the refinement relations that can be established.

As a piece of evidence in support of this thesis we consider three minor, but prototypical examples while we define the extended normal proof obligations.

To further motivate the concept of extended normal refinement, we show how it may be applied to three slightly larger examples. The first, is a fairly typical case where real numbers are approximated in a refinement by discrete values and an upper limit on the values allowed is imposed. The next two examples are interesting non-standard applications of refinement. Both may be regarded in the traditional way as an abstract specification and a concrete “implementation”, but they have another interpretation as well. The abstract specification may be regarded as a definition of the internal state of a system and the representation is some kind of interface to that system. In the examples presented, the concrete specification could define the behaviour of the user interface or a programming interface to an underlying system. Discharging the refinement obligations guarantees that the interface is a “faithful” presentation of the system for which it is a “front end”.

The paper is structured as follows: Sect. 2 outlines some of the terminology and VDM notation which will be used in the rest of the paper. Section 3 describes the normal VDM refinement proof obligations and presents the case for making extensions by way of three simple example refinements which the normal obligations fail to cope with. In Sect. 3.2 extended normal refinements are defined, and are applied to the original examples in Sect. 3.3. The next three sections, Sects. 4, 5 and 6, contain the larger examples. Finally, Sect. 7 summarizes our position and makes some concluding remarks.

We use basically the same notation as Jones did in the VDM textbook [5]. The logic in which we represent specifications and refinement relations is the three-valued logic LPF, The Logic of Partial Functions. LPF is defined in [5] as a first-order logic that is extended to allow partial predicates and functions. For total predicates LPF behaves as ordinary first order logic and all the predicates we introduce can be taken to be total.