A Semantics for NewSpeak in VDM-SL

Paul Mukherjee*

School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT

Abstract. NewSpeak is a language for programming real-time safety-critical systems. Its distinguishing feature is the property of unexceptionality – the property that any successfully compiled program will not raise a runtime exception. In this paper we describe a denotational semantics for NewSpeak written in VDM-SL, which ensures that it is unexceptional. We also discuss the implications of using VDM-SL for the semantic definitions, from the point of view of development of specifications.

1 Introduction

A prime concern in the construction of a real-time safety-critical system is dependability – the justifiable reliance placed on the service that the system delivers. In hardware, dependability is achieved via fault tolerance using techniques such as the provision of multiple component redundancy. However, dependability in software is more problematic: formal techniques for the development of software remain unpopular (as evidenced in a recent study [8]) while most programming languages currently in use are riddled with insecurities [4].

Requirements of computer-based safety-critical systems have been focussed recently by the publication of United Kingdom Ministry of Defence Standard 00-55 (DefStan 00-55) concerned with the procurement of software for safety-critical systems [15]. This discourages the use of programming techniques such as recursion and dynamic memory management whilst encouraging programming languages which are amenable to static analysis.

One way in which we can contribute to dependability is by the use of unexceptional languages. An unexceptional programming language is one in which programs that successfully compile for a particular target processor never raise a runtime exception on that processor. By using an unexceptional language we remove a major imponderable from the dependability equation.

NewSpeak [5, 6] is such an unexceptional language. The property of unexceptionality is achieved through Orwellian programming – by restricting the programmer’s freedom of expression we remove the possibility of “heretical” techniques such as recursion. A policy of strong typing ensures that all computations are finite: every identifier and expression has an associated finite set (known as its type) delimiting the values it might assume. An identifier is only

* Work partially supported by Science and Engineering Research Council and National Physical Laboratory.
allowed to be assigned an expression if the type of the expression is a sub-type of the type of identifier.

In this paper, we describe a denotational semantics for NewSpeak, written in the language VDM-SL [7]. Our objectives in giving such a semantics are:

- To satisfy the requirements of DefStan 00-55 that programming languages to be used in safety-critical systems should have a formally defined semantics.
- To provide formal justification for the claim of unexceptionality for the language, which has been hypothesized previously.
- To give criteria which enable us to judge whether a NewSpeak program satisfies a formal specification or not.

We use VDM-SL to express the semantics as this allows mechanical analysis of the specification 1. Also, since NewSpeak is to be used as part of a formal development method for safety-critical systems [17]. The specifications in this development method are written in a language similar to VDM-SL. As we shall see in section 5, the use of VDM-SL for the semantic definition of NewSpeak eases development of specifications into NewSpeak programs.

This paper is organised as follows: in the next section we give an outline of the language and after it describe the semantic functions used to give meaning to NewSpeak programs. We then describe the implications of using VDM-SL to express these semantic definitions and finally we discuss what we have achieved. A complete definition of NewSpeak may be found in [16].

2 Features of NewSpeak

All constructs in NewSpeak are value-delivering expressions. To cater for constructs which do not explicitly deliver a value (such as assignment), a special value called \textit{void} is introduced. We begin by describing the type system in NewSpeak.

2.1 Types

In NewSpeak, types define finite sets of values that variables of that type might assume, reflecting the finite nature of the target architecture. The price of unexceptionality is that all types must be inferable at compile-time. Hence bounds on variables must be declared at compile-time. However this need not be a disadvantage of the language as it makes the software designer aware of the physical limitations of the target architecture, for instance in terms of \textit{MAXINT}, the greatest integer that can be represented on the target architecture.

We have five primitive types: Void, Bit, Byte, Word, and Float. \textit{Void} has one value \textit{nil}. However, in conjunction with flavours (see 2.1), \textit{Void} values can be used to simulate finite sets. \textit{Bit} has two values \textit{0} and \textit{1} which are typically used in logical expressions. \textit{Byte} and \textit{Word} respectively represent those values which

---

1 The specification has been analysed using the IFAD VDM-SL toolbox [9]