Design for Proof: An Approach to the Design of Domain-Specific Languages

Stuart Anderson and Konstantinos Tourlas
Laboratory for the Foundations of Computer Science, Division of Informatics,
The University of Edinburgh, Edinburgh UK

Keywords: Domain-specific languages; Programmable controllers; Programming language design; Industrial critical systems

Abstract. We propose that the domain of a Domain-Specific Language (DSL) can be characterised by:

1. the class of environments in which systems developed in the language are expected to operate; and
2. the class of properties which such systems are expected to possess.

The design of DSLs should therefore include the development of a proof system that eases the task of proving the properties in the class identified for the anticipated operating environments.

We develop these ideas in the context of industrial computing systems by presenting a semantics and proof system for a language based on IEC 1131-3, the international standard programming language for programmable controllers.

Of particular significance in this example is the use of a diagrammatic representation and the development of a proof system for a class of invariance properties that requires only local knowledge of the structure of diagrams.

1. Introduction

Recently there has been increased interest in the so-called domain-specific languages (DSLs), i.e. programming languages designed specifically to support particular kinds of application. According to most advocates of the DSL approach, a successful DSL is one that captures the native vocabulary and semantics of
the application domain [Bru97, Hud96], thus making the language immediately accessible to the experts in the domain. DSLs are therefore believed to contribute to higher user productivity by relieving the application experts from the need to be proficient programmers as well.

The particular domain for which a DSL is developed will therefore determine characteristically:

1. The kind of models that programs in the language denote (i.e. the semantics); and
2. The kind of representations that programs admit (i.e., the form(s) in which programs are presented to the user).

For instance, signal processing programs in Signal [GaG87] have an intuitive interpretation as digital, synchronous circuits. On the other hand the same language admits, in addition to textual syntax, a graphical representation of programs resembling the block diagrams used by the signal processing community.

Our view is that application domains can also be characterised by some class of properties that users want easily proven of programs in the domain. The degree to which the presence of such important properties is made evident in the representation of programs should therefore be a major quality requirement in many domains. Programs written in languages which are ‘designed for proof’ should require little or even no extra effort to exhibit the requisite properties. To this end, the present paper advocates the contributing role that formal modelling and reasoning techniques can play in the design of better DSLs.

Languages for industrial critical systems provide excellent material upon which the above ideas can be applied. Firstly, the importance of certain classes of properties is apparent: industrial software is often subjected to strict certification based on various safety and other quality requirements [HaF98]. Secondly, software representation in industrial systems is often diagrammatic and derived from existing design practices.

Despite being generally appealing, many of the diagrammatic notations used for programming are often described as ‘confusing’. Most commonly these are notations of which the design and vague semantics permit spatially dispersed and seemingly unrelated parts in a diagram to engage in subtle interaction. Consequently, reasoning about programs requires arguments about the global structure of potentially large diagrams.

In this paper, the problem of inferring properties based on diagrams is addressed in a domain-specific context. In particular, it is shown how properties may be proved locally by:

1. Eliminating features in the notation which introduce non-local behaviour; and
2. Reducing the complexity of the reasoning tasks for which basic support is sought. More specifically, the complexity aspect to be minimised in our case is the dependency on computational history of the formulae in which properties are expressed.

1.1. Industrial Process Control

The particular domain we choose for illustrating our approach is that of industrial process control and programmable logic controllers (PLCs). The latter are reactive