Modular Refinement and Model Building

Martin de Groot and Ken Robinson
School of Computer Science and Engineering
martindg@cse.unsw.edu.au, k.robinson@unsw.edu.au
University of New South Wales
Sydney NSW 2052 Australia

Abstract. In this paper we show that formal program development can be viewed as a process of model building. Refinement diagrams are introduced and formally defined in terms of refinement developments. Hierarchical models are shown to be equivalent to modular refinement developments. Modular refinement developments are a subset of refinement developments and refinement diagrams. A function is defined to extract the corresponding model from any refinement development.

1 Introduction

We propose that formal program derivation effectively produces a hierarchical model of the system being derived. A specification can be seen as representing any one level, while refinement (⊆) defines the ordering of the levels within the hierarchy. The specification defining any one level will be very complex for a realistically large system. Thus it is useful to advance formal program developments by decomposing larger specifications into a collection of smaller specifications, along with a program constructor to hold the whole system specification together. This decomposition process not only makes refinement manageable, it also defines the ultimate structure of the implemented system in terms of its constituent components. A layered description of a system’s structure is a hierarchical model of that system.

All the information necessary to define the final structure is contained within the record of the refinement development. However, as sub-specifications are sometimes (re-)composed during refinement, the refinement development itself does not immediately give the ultimate system structure. This structure must be extracted from the refinement development. Below we will formally define a total function that performs hierarchical model extraction from refinement developments.

Although a program derivation is a formal process, program derivations themselves have not been formally defined. The clearest proposal of how such derivations are structured is given by Ralph Back in his paper Refinement Diagrams [1]. Refinement diagrams were the starting point for this investigation of the formal structure of refinement developments. For completeness, we show that refinement diagrams can also be formally defined in terms of refinement developments. Hence, models can also be extracted from refinement diagrams.
2. Notation

Two different, but widely used, formalisms are employed in this paper. We will use the Z notation of Abrial [7] to define refinement developments and diagrams. However, individual specifications will be given using the wide spectrum language of the refinement calculus of Morgan [6,4]. For a more detailed explanation please consult the references.

2.1 Z

Z is a specification language based on set theory and logic. The basic entity of a Z specification is the schema. A schema defines a named set of objects, in which each object has a set of named components. There are two equivalent forms of a schema definition

```
Name
Signature
Predicate
```

or

```
Name \equiv [Signature \mid Predicate]
```

where Signature is a set of variable declarations and Predicate is a predicate that expresses a constraint on those variables. For example:

```
add
in1, in2, sum : \mathbb{N}
sum = in1 + in2
```

A schema calculus supports the building of new schemas using schema inclusion, conjunction, hiding, etc. For example,

```
arithmetic
add
diff : Z
diff = in1 - in2
```

Abbreviations are defined using new == old, and constants are defined using an axiomatic definition, for example:

```
double : Z \rightarrow Z
\forall n : Z . double(n) = n \times 2
```

The following definition of a linked list of integers demonstrates how disjoint sets, or free types, are defined.

```
list ::= end
| link(Z \times list)
```