Abstraction Refinement: a Model of Software Evolution Discussion by M. Ward

I am writing in response to the paper by B. J. Keller and R. E. Nance entitled 'Abstraction refinement: a model of software evolution', published in JSM, Vol. 5, No. 3, pp. 132–146, September 1993. The paper claims to present a model for the concept of 'abstraction' as a relationship between programs ('a pre-order of relative correctness between system descriptions'). The model is claimed to 'provide a characterization of software evolution that is both descriptive and instructive'. However, I believe that the model (as an attempt to capture the informal notion of abstraction) is seriously flawed, and that therefore the characterization of software evolution derived from the model is at best spurious.

The authors do not give a denotational semantic interpretation of the programs in their various languages (i.e. an interpretation of a program as a function which maps each initial state to the set of possible final states). Had they done so, then the flaws in their model would be clearly revealed. Consider their definition of 'levels of abstraction': The zero level consists of all deterministic programs (i.e. all programs for which each initial state has a single possible final state). The first-level language consists of all pairs of nondeterministic choices of deterministic programs, with all the deterministic programs removed. In other words, at least one initial state has two possible final states, and no initial state has more than two final states. Similarly, each program in the nth level has one or more initial states with $n+1$ final states, and no initial state with more than $n+1$ final states. Some of the implications of this definition are:

1. The program $\text{pick}_{32\ldots}\text{bit}_{-}\text{int}$ which nondeterministically sets the variable $n$ to a value in the set $\{0, 1, 2, \ldots, 2^{32} - 1\}$ is a highly abstract program (according to this definition)—it is in the $2^{32} - 2$ level of abstraction.

2. The refinement process whereby $\text{pick}_{3\ldots2\ldots}\text{bit}_{-}\text{int}$ is refined to $n := 0$ will require $2^{32} - 1$ separate basic (vertical) refine-

3. Consider, the following specification for sorting an array $A$:

$$A := A'.(\text{sorted}(A) \land \text{permutation}_of?(A,A'))$$

This reads 'pick a new value $A'$ for $A$ which is a sorted array and is also a permutation of the original value $A$'. Most people would regard this as a fairly abstract description of a sorting program: it says what the result should look like without giving any indication as to how the result is to be achieved. But this specification happens to be deterministic, so in the model it will appear at the lowest level of abstraction: in fact at the same level as all of its implementations, but far below the program $\text{pick}_{32\ldots}\text{bit}_{-}\text{int}$.

4. The program $\text{abort}$ (in a sense the 'most abstract' program: since anything is a refinement of $\text{abort}$) does not appear anywhere in the hierarchy, since each input state has an infinite set of final states (i.e. all the possible final states, including the nonterminating state $\bot$). More seriously, any program or specification with unbounded nondeterminacy (where an input state has an infinite set of associated final states) will also be excluded. Most seriously of all, a program such as:

$$x := x'.(x' \in \mathbb{N} \land 1 \leq x' \leq n)$$

(which sets $x$ to a random integer between 0 and $n$) will also be excluded, even though this program has bounded nondeterminacy.

5. The discussions on the model discuss 'searching for a common abstraction' of two programs, and mention that 'for most forms of maintenance, a common element exists'. In fact, any pair of programs in
the model has a common abstraction: namely the nondeterministic choice of the two programs. (This program is also the least common abstraction.) The fact that this is a trivial common abstraction is yet another flaw in the model.

In conclusion: it is clear that any attempt to define 'levels of abstraction' cannot depend solely on the (denotational) semantics of programs; since there are many programs, such as the sorting specification and its implementations, which have identical semantics, but which are clearly on different levels of abstraction. A formal definition of the 'abstraction level' of each atomic operation in the program is required: a definition which recognizes that

\[ A := A'.(\text{sorted}(A) \land \text{permutation\_of}(A,A')) \]

is 'more abstract' than

\[ A := A'.(A[i] = x \land A'[0..i-1] = A[0..i-1] \land A'[i+1..\ell(A)] = A[i+1..\ell(A)]) \]

(The latter is a specification for the assignment \( A[i] := x \).)

Dr. Martin Ward
University of Durham
Department of Computer Science
Science Laboratories
Durham
DH1 3LE
UK