Axiomatic Language

Walter W. Wilson

Axiomatic language is proposed as a tool for greater programmer productivity and software reliability.

Goals – Axiomatic language has the following goals: (1) pure specification – you tell the computer what to do without telling it how to do it, (2) minimal, but extensible – as small as possible without sacrificing expressiveness, (3) a metalanguage – able to incorporate the capabilities and advantages of other languages, and (4) beauty.

Idea – Axiomatic language is based on the idea that the external behavior of a function or program – even an interactive program – can be represented by a static, infinite set of symbolic expressions. These expressions enumerate all possible inputs – or sequences of inputs – along with the corresponding outputs. For an interactive program each expression would record the inputs/outputs of a particular execution history as seen by an external observer. The set of expressions would cover all possible execution histories. The language is just a formal system for defining these infinite sets.

Recipe – Axiomatic language can be described as pure, definite Prolog with Lisp syntax, HiLog higher-order generalization, and "string variables", which match a string of expressions in a sequence. (See details here.)

Example – In axiomatic language a finite set of "axioms" generates a (usually) infinite set of "valid expressions". For example, the following two axioms,

```
(a b).
((%) $ $)< (% $).
```

generate the valid expressions (a b), ((a) b b), (((a)) b b b b), etc. The symbols % and $ are expression and string variables, respectively.

Benefits – Specifications should be smaller and more readable than algorithms. (Specifications just define external behavior while implementation code defines both external behavior and internal processing.) Specification definitions should also be more reusable than code constrained by efficiency. Thus, smaller code size should provide increased programmer productivity.

The purity and small size of axiomatic language should make it well-suited to proof. Proof would guarantee equivalence between the user's specification and the generated program. One may also be able to prove assertions about specifications to validate them and this could be more powerful than just testing.

Implementation Challenge – The problem of automatically transforming specifications to efficient programs can be considered a grand challenge of computer science. If the target is a parallel machine, it subsumes the problem of automatic parallelization. This transformation problem is a form of the old automatic programming problem, except that here we are not trying to understand natural language requirements and the system is not expected to have knowledge about any particular application domain. Also, the specifications are complete – the system doesn't have to infer an input/output function from examples.

In summary, axiomatic language can be seen as idealistic in its goals, intriguing in its potential, and formidable in its realization.