Abstract of M.Sc. thesis **Jo Erskine Hannay; August 1, 1995**

The thesis deals with topics in the field of formal abstract specification and verification of programs, particularly within the framework of algebraic methods restricted to equational logic. The discussion views equational specification and the closely related topic of simple term-rewriting as more concrete and nearer to implementation than specification in general. Therefore initial and final models are considered, rather than complete model-classes. Also mostly constructive, "executable" equational specifications are considered giving an inclination towards viewing specifications as abstract programs. The thesis is structured around the following two topics:

1. The method of equational algebraic specification is *generalized* in a manner allowing a certain mode of *modular* specification. The generalization has two forms accommodating initial and final algebra semantics respectively. The generalization is modular in the sense that complex specifications may be constructed stepwise from simpler *kernel* specifications. Each construction step provides the choice of initial or final generalized form. The resulting complex equational specifications differ from general case predicate logic *hierarchical* specifications, in the sense that the polarization into initial and final semantics gives inhomogeneous specifications; a complex specification cannot in principle be viewed as a non-hierarchical simple specification.

   A concept of *consistency relative to kernel specifications* is developed and it is possible to reason about such relative consistency without specific knowledge about the kernel specification.

   The possibility of analogously constructing complex formal-mechanical proof methods from simpler methods is briefly discussed.

2. A variant of (initial) algebraic specification called *indirect specification* is developed. Motivation and a foundation for indirect specification is given by *syntactical specifier functions*; i.e. terms are to be understood identical iff the their values under some given function (with syntactical codomain) are identical. A special and interesting case of syntactical specifier functions are functions giving *canonical representatives* in some well-defined sense.

   Besides providing the specification language with additional means of specifying equality, indirect specification expands the class of congruences (over ground terms) decidable by simple term-rewriting. In addition, although the class of initial congruences equationally specifiable is identical to the class specifiable by syntactical specifier functions with canonical representatives as codomain, there exist syntactical specifier functions which do not give canonical representatives. (Such a function may for instance give a representative in some other class than in which the argument belongs.) It may therefore be the fact that the class of congruences indirectly specifiable is greater than the class equationally specifiable. Characteristically, indirect specification represents a more operational mode of specification than does usual algebraic specification.

   Indirect specification introduces further inhomogeneousity to equational algebraic specification. The modularity of the generalized specification strategy discussed in part 1 is used to encapsulate indirect specification in the hierarchical framework.

   It is then shown that a trivial augmentation of indirect specification can be reduced to standard non-generalized initial or final semantics specification; under certain interesting circumstances. It is also shown that Knuth-Bendix completion of an augmentation of indirect specification, if successful and under a certain congruency condition, will give an equivalent standard algebraic specification. Viewing equational specifications as abstract programs, this might be seen as a *program transformation*. (A more applicable strategy for automatic transformation in this sense is sketched just as an idea without any sort of further proof.

The overall structure sees part 1 primarily as giving a framework for the discussion in part 2.

The subject of *consistency* permeates the entire discussion. Consistency is viewed as related to basic ideas of some semantic domain which are expressed as untamperable *presuppositions* in the act of specifying. For example, the basic idea that a mathematical proposition cannot simultaneously be true and false is presupposed in predicate calculus by predefined interpretations of the symbols true and false, and by referring to a set of axioms as *inconsistent* iff the predicate true = false is deducible from the axioms. The concept of (in)consistency seen always *relatively* to such presuppositions allows a generalized notion of consistency in hierarchical specification *relative* to kernel specifications as discussed in part 1. (This also generalizes Guttag's notion of consistency.) Some simple methods for detecting (generalized) inconsistency and for establishing consistency are briefly presented.

The notion of *artificial inconsistency* is introduced, as inconsistency due to *auxiliary functions*; i.e. functions helpful or necessary during constructive definition or implementation, but otherwise really not belonging to the semantical objects under implementation. Formalisms and results are developed showing that auxiliary functions and hence artificial inconsistency can be *hidden* from the formal reasoning, still allowing for the full implementatory benefits. This goes beyond the model-theoretical notions of *hidden sorts* and *symbols*. The theory developed can probably be used and implemented by modifying existing proof methods (based on the concepts of *proof by consistency* and *inductive completion*).

Finally an extension of Knuth-Bendix completion is presented as a step towards mechanical generation of constructive proofs *in the input theory* given to the process. Constructive proofs may give deeper insight into a theory. It also turns out that the extension of Knuth-Bendix completion under certain circumstances can be used in establishing consistency in conjunction with indirect specification (part 2). The extension also turns out to be vital in proving that hiding of auxiliary functions and artificial inconsistency may be implemented in proof methods based on inductive completion.