Chapter1:Introduction

1 Introduction

In software engineering, a formal specification of what a system has to do plays a crucial role in the software development process. Formality is required for verification and validation purposes. Different forms of formal specifications have been developed. This thesis concentrates on the algebraic approach to system specification. In essence, this approach can be characterized by:

- axiomatic specifications which consist of a signature together with a set of axioms in a specification logic which any implementation of the system with that specification has to satisfy.
- operations to combine and modify specifications from subspecifications with an algebraic semantics.
- An abstract presentation of the semantics as independent as possible of the specification logic or institution [GB92].

Another crucial step in the software development process is software design. Some of the usual tasks which are performed during software design are the following: deduction of properties from specifications, refinement of specifications and verification of a program with respect to a specification. Assuming the existence of a model-theoretic semantics of the algebraic specification language by which a class of models is associated to any specification expressible in the language, usually the three main software design tasks are formally represented as propositions with the following informal reading:

- $SP \models \phi$ which can be read as a proposition which is true if and only if all models of the specification SP satisfy the formula ϕ.
- $SP_a \rightsquigarrow SP_i$ which can be read as a proposition which is true iff the specification SP_i implements the more abstract specification SP_a
- $P_L \models SP$ which can be read as a proposition which is true iff the program P written in the language L satisfies the specification SP. This makes sense if the model-theoretic semantics of the specification language is related to the the semantics of the programming language. See [ST96] for some problems to establish this relation.

We understand by an algebraic design framework a formalism which includes a formal definition of an algebraic specification language including a model-theoretic semantics and the implicit or explicit formal definition of at least the two first software design tasks. The definition of algebraic design frameworks should be for several specification logics or even better for an arbitrary but fixed institution ([GB92]), and if it is the case for an arbitrary class of programming languages. In the literature, there exists a large amount of algebraic specification languages and algebraic design frameworks. See [SW] for a survey.

An interesting example of algebraic specification language which will be also used in this thesis is ASL([SW83]). ASL is a kernel specification language which was not originally designed to be used directly but as a basis to define the semantics of higher-level specification languages. *ASL* includes different operators to structure and build specifications from argument specifications. Since it has a loose semantics, there is no reason to restrict the expressibility of the logic to equations and therefore it can be defined with first-order and higher-order logics as specification logic and additionally it is also possible to include a programming language in its associated algebraic design frameworks.

In [BHW95] and [Hen97] it is presented a version of *ASL* with structuring operators, behavioural operators and a reachability operator with many-sorted first-order logic as specification logic. In [Hen97] it is also presented different notions of refinement: the standard notion defined as model inclusion between the model-theoretic semantics of the concrete and abstract specification of the refinement, and additionally two more notions of refinement. These two new notions are also defined by model inclusion but this time between the model-theoretic semantics of the concrete specification and two different notions of behavioural abstraction of the class of models of the abstract specification. One notion of refinement requires a partial congruence between the carrier sets of an algebra (behavioural refinement) and the other notion requires an equivalence relation between algebras (abstract refinement). Behavioural refinement requires that the model-theoretic semantics of the concrete specification is included in the class of models whose quotients by the given partial congruence belong to the class of models of the abstract specification and abstract refinement requires that the model-theoretic semantics of the concrete specification is included in the class of models which are equivalent to some of the models of the abstract specification by the given equivalence relation.

In [HWB97] and in [Hen97] proof systems for the deduction of properties from *ASL* specifications and for the refinement of *ASL* specifications are also presented, and in [BCH] the proof systems for the deduction of properties from structured specifications (without the reachability and behavioural operators) are generalised for a fixed but arbitrary institution. Some of the proof systems are presented as infinitary proof systems in order to achieve completeness.

Another interesting example of an algebraic specification language which has been recently developed in the Common Framework Initiative (CoFI) [San] is CASL ([CoF98]).

The model-theoretic semantics of *CASL* specifications is defined as a class of structures including subsorts, partial operations and predicates. CASL also extends ASL with architectural specifications on top of its specification building operators which are similar to the ones of ASL.

An architectural specification consists of a list of units and a unit term with the following basic syntax:

$$\begin{array}{ll}
\textbf{units} & P_1 : SP_1; \\
 & \vdots \\
 & P_n : SP_n; \\
\textbf{result} & LINK_{BR}(P_1, \dots, P_n)
\end{array}$$

where a unit P_i can be seen as a structure which implements the specification SP_i and the unit term $LINK_{BR}(P_1, \dots, P_n)$ can be seen as the linking operation to build an implementation of the architectural specification from the given structures. This structuring mechanism is useful to fix the architecture of the system or a part of the system under development at a certain stage or branching point BR. Thus, we can always refine a CASL specification by an architectural specification even if it is a unit of another architectural specification. This improves the model of refinement from a linear sequence of refinements of the form

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow \dots \rightsquigarrow SP_N$$

to a modular tree structure of refinements. At any step i of the refinement a specification $SP_{i,j}$ can be decomposed as follows:

$$SP_{i,j} \rightsquigarrow BR \begin{cases} SP_{i+1,k} \\ \vdots \\ SP_{i+1,k+l} \end{cases}$$

With just building operations for specifications, this would not be possible since at any refinement step we can always define a refined specification with a completely different structure than the structure of the abstract specification. This is not the case of architectural specifications because we can not refine architectural specifications to architectural specifications but CASL specifications to architectural specifications. See [BST99] for more motivations on architectural specifications, a formal semantics of architectural specifications including the formal semantics of the different operators of unit terms which are closely related to the *CASL* operators for building specifications and the formal semantics of the refinement of architectural specifications.

For most of the algebraic specification languages and algebraic design frameworks presented in [SW], specific tools to assist different tasks of software design have been developed.

Since the design of CASL, a policy to reuse existing and powerful theorem provers for the development of CASL tools has been established. More precisely, the technical problems to reuse the theorem prover Isabelle([Pau]) (which is based on higher-order logic) for CASL has been solved.

2 Objective and main results of the thesis

The main objective of this thesis is to give an approach to reuse theorem provers for type theories with dependent and inductive types for the development of theorem provers for algebraic design frameworks. The problem can be considered interesting for several reasons. First, we believe that the development of CoFI tools in type-theoretic theorem provers will help to spread the use of algebraic techniques. Note that these kind of theorem provers like for example Coq ([CJJ$^+$]) or Lego ([LP92]) have been widely used and applied. Second, the work to be done is completely different to the work done in Isabelle and finally this work requires a thorough understanding of a non-standard family of formalisms which are type theories.

Because of simplicity and without loss of generality, we will present our generic implementation strategy just for the algebraic design framework of ASL of [HWB97] and [Hen97]. Note that since *ASL* has been used to give semantics of higher-level algebraic specification languages, we believe that it would not be difficult to adapt the presented generic implementation strategy to give proof support to algebraic design frameworks including higher level algebraic specification languages.

Type theories were initially used as a logical language for the foundations of mathematics. Since they also include a computational language (in particular a functional language), most of them have also been used as a framework for program development. Some type theories have also been used as logical frameworks like for example the LF type theory [HHP93].

LF can be seen as a pure type system, that is a three-level typed lambda calculus (level of elements, types and kinds) with dependent Π-types. LF has been used to make adequate encoding of different logics. The principle of encoding is based on the idea of judgements as types, where judgements are seen as families of types of their proofs [HHP93]. Some of the limitations of this principle of encoding are the following: first, it is not possible to develop metatheory of the encoded logical systems. Thus, we can not define properties of the logical systems and prove them using for example induction. Second, the encoding are not very readable and finally there exist important restrictions in the kind of logical systems which can be represented.

UTT [Luo94][Gog94] (Uniform Theory of dependent types) is a type theory which adds to the Extended Calculus of Constructions (ECC [Luo94]) the possibility to define inductive types. The whole type theory is encoded in the Martin-Löf Logical Framework [BNS90]. This type theory has been applied to define an algebraic design design framework including a higher-order logic and a restricted functional programming language as we briefly explain in the section of related work. On the other hand, the increase of the expressive power of the type theory with respect to *LF* is useful to define a new principle of encoding, which, as we will explain in chapter 2 and 3 of this thesis, does not have the main problems of the principle of encoding of *LF*.

In order to give a generic presentation of the framework of ASL presented in [HWB97] and in [Hen97] for different specification logics, we give an abstract semantic framework in which the semantics of the behavioural operators of ASL can be uniformly instantiated in first-order and higher-order logic. Additionally, we relate a different semantics of the behavioural operators given in [HS96] in higher-order logic with ours generalizing also the semantics of [HS96]. We define normal forms for both different semantics in the same abstract framework to have a better understanding of both semantics and their relationship. The abstract semantic framework is referred as behavioural algebraic institutions.

We also redesign the different proof systems for deduction and refinement when it is not possible to give adequate encodings in *UTT* because they are infinitary proof systems. In some cases, different solutions are given for first and higher-order logics.

Finally, the main proof systems of the algebraic design framework for *ASL* have been encoded in *UTT* using the new principle of encoding presented in the thesis. This allows us to take advantage of the current implementation of theorem provers for UTT to implement proof checkers for the proof systems of our chosen algebraic design framework.

3 Related work

The main related works to this thesis are the following:

- In [Luo94] and in the third chapter of this thesis an algebraic design framework for the development of functional programs from modular algebraic specifications is presented. This framework is defined using type-theoretic constructions of UTT.

  The type of signatures of specifications can be any type of the type theory. Since the type theory is quite expressible, it is of course possible to define many-sorted first-order signatures in the algebraic style. Inhabitants of the type of signatures are referred as structures and axioms of specifications are defined as functions which given a structure returns a proposition in the logic of the type theory which is higher-order intuitionistic logic. The consequence relation between specifications and formulas can be easily defined and in [Luo94] different operations on specifications are defined which can be instantiated to operations which are similar to the structuring operators of *ASL*.

  If the type of signatures are many-sorted first-order signatures then the inhabitants of signatures can be seen as functional programs (which can be defined using primitive recursive operators but with no general recursion operator) and therefore it can be defined a satisfaction relation between functional programs and specifications.

  Additionally, in [Luo94] a notion of refinement is defined with a correctness condition which relates the structures which satisfy the concrete specification and the abstract specification.

  Finally, parameterised specifications and refinement of parameterised specifications are also defined.

  The main drawbacks of this framework are the following;

  - There exists an expressibility constraint at the level of functional programming in order to achieve decidability of the type checking. Thus, it is not possible to define functional programs using general recursion and not all the computable functions are representable.
  - It is not generic enough in the sense that it is not independent of the specification logic and of the functional programming language.

  - Although its semantics is based on the semantics of the algebraic design framework for ASL presented in this thesis, it is difficult to relate both semantics as we will explain in chapter 3.

- The proof support developed for CASL in [MKKB97] and in [Mos00]. They use higher-order logic as metalanguage instead of UTT and their aim is to reuse the proof support of the theorem prover Isabelle [Pau]. The representation technique is totally different to the one which we present and it is based on map of institutions [CM97]. They basically present a map between the CASL logic and higher-order logic and technically it requires the definition of the following categorical structores;

  - a functor from the category of theories of the CASL logic to the category of theories of higher-order logic.
  - a natural transformation from the functor of sentences of the CASL logic to the functor of sentences of higher-order logic.
  - a natural transformation from the model functor of the higher-order logic to the model functor of the CASL logic.

  In our case, we would have to define a proof system with a finite set of rules for the CASL logic and then give an adequate encoding of the proof sytem in *UTT*.

  It is out of the scope of this thesis to compare the efficiency of the theorem provers associated to a given algebraic design framework which can be obtained following both approaches.

4 Summary of the thesis

The organization of the thesis is as follow: in chapter 2, we give a general view of the different type theories which have appeared in the literature and how they have been used as metalanguages of different formalisms. We will also present the type theory which we are going to use (UTT [Luo94]) and in the next chapter we will present the two main different ways to use this type theory in software design. We present a new principle of encoding proof systems and higher-order calculi but similar to the one of *LF*. The main advantages of the new principle with respect to the one of *LF* is that it allows to encode a wider range of proof systems, it is possible to develop metatheory of the encoded proof systems in the type theory and the encoded formalisms are more readable.

In chapter 4, we will present the abstract semantics for the algebraic design framework presented in [BHW95], [BCH] and [Hen97] relating the semantics of its behavioural operators for many-sorted first-order logic with the semantics of the behavioural operators presented in [HS96] for higher-order logic. This is achieved by defining both semantics and their normal forms in a new kind of institutions which are referred to as behavioural algebraic institutions. The normal form of specifications is also useful to define certain kind of proof systems which are presented in [Hen97] just for many sorted first-order logic and in this thesis for an arbitrary but fixed behavioural algebraic institution.

In chapter 5, we will present the most significant proof systems which appear in [Hen97]. The main novelty of this chapter is that the infinitary proof systems are redesigned using a finite presentation to be able to represent them adequately in type theory and they are presented for first-order and higher-order logic.

In chapter 6, we will present how to give adequate encodings of the most interesting proof systems presented in the previous chapter in UTT and in the final chapter we will briefly explain the assistance which proof checkers similar to the one of *UTT* offers and we will raise some conclusions and future work.

References

[BCH] Michel Bidoit, María Victoria Cengarle, and Rolf Hennicker. Proof systems for structured specifications and their refinements. Chapter 11 of the book Algebraic Foundations of Systems Specification.

[BHW95] Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2-3):149–186, December 1995.

[BNS90] Kent Petersson Bengt Nordström and Jan Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[BST99] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Architectural specifications in CASL. Technical Report ECS-LFCS-99-407, University of Edinburgh, 1999.

[CJJ$^+$] C.Cornes, J.Courant, J.F.Fillaitre, G.Huet, and et al. The coq proof assistant reference manual v.5. Inria-Rocquencourt and CNRS-ENS Lyon, France.

[CM97] Maura Cerioli and José Meseguer. May i borrow your logic? (transporting logical structures along maps). *Theoretical Computer Science*, 173:311–347, 1997.

[CoF98] CoFi LD Task. CASL the CoFi algebraic specification language. http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary-v1.0/index.html, October 1998.

[GB92] Joseph A Goguen and Rod Burstall. INSTITUTIONS: Abstract model theory for specification and programming. *Journal of the Assoc. for Computing Machinery*, 39(1):95–146, 1992.

[Gog94] Healfdene Goguen. *A Typed Operational Semantics for Type Theory.* PhD thesis, University of Edinburgh, September 1994. Also published as Technical Report CST-110-94, Department of Computer Science.

[Hen97] Rolf Hennicker. *Structured Specifications with Behavioural Operators: Semantics, Proof Methods and Applications.* Habilitationsschrift, Institut für Informatik, Ludwig-Maximilians-Universität München, June 1997.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HS96] Martin Hofmann and Donald Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science*, 167:3–45, 1996.

[HWB97] Rolf Hennicker, Martin Wirsing, and Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173, February 1997.

[LP92] Zhaohui Luo and Randy Pollack. LEGO proof development system: User's manual. Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, May 1992.

[Luo94] Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Clarendon Press Oxford, 1994.

[MKKB97] T. Mossakowski, Kolyang, and B. Krieg-Bruckner. Static semantic analysis and theorem proving for casl. In *12th International Workshop WADT97 (LNCS 1376)*, 1997.

[Mos00] Till Mossakowski. Casl: From semantics to tools. In *TACAS00 (LNCS)*, 2000.

[Pau] Lawrence C. Paulson. Introduction to isabelle. 25 October 1998 (Computer laboratory of University of Cambridge).

[San] Donald Sannella. The common framework initiative for algebraic specifications and development of software. www.dcs.ed.ac.uk/ dts/.

[ST96] D. Sannella and A. Tarlecki. Mind the gap! abstract versus concrete models of specifications. In *Proc. 21st Intl. Symp. on Mathematical Foundations of Computer Science*, 1996.

[SW] Donald Sannella and Martin Wirsing. Specification languages. Chapter 8 of the book Algebraic Foundations of Systems Specification.

[SW83] Don Sannella and Martin Wirsing. A kernel language for algebraic specification and implementation. In *Proc. Intl. Conf. on Foundations of Computation Theory, Borgholm, Sweden*, number 158 in Springer LNCS, pages 413–27, 1983.