TYPELAB: An environment for modular program development *

F.W. von Henke, M. Luther, M. Strecker

Universität Ulm
D-89069 Ulm, Germany

1 Introduction

TYPELAB is an experimental specification and verification environment. Its specification language and its tool support provide assistance for a modular design and development methodology.

The specification language of TYPELAB is based on a type theory, the *Extended Calculus of Constructions (ECC)* [Luo94], which gives the system a sound semantic foundation. The pure type theory has been augmented by constructs partly to be found in algebraic specification formalisms [Wir86, Gog84, Bid91] and other verification environments [OSR93, SJ94]. Particular language support is offered for axiomatizing theories and specifications, for stating theorems, for defining (even incomplete) morphisms between theories, for parameterization over theories, and for operators on theories. The novelty of the language lies in the combination of its features, which altogether yield a very expressive formalism, rather than in each aspect taken separately. Some aspects of the language will be illustrated in more detail in Section 2.

TYPELAB comprises a proof assistant which is primarily thought to be used as an interactive proof checker. A sequent-style theorem prover has been developed for automatically solving medium-sized problems in restricted fragments of the logic. The integration of a rewriting system for equality proofs is currently under way. During a specification development activity or during a proof, a knowledge base of previous program developments and proofs can be consulted. The knowledge base mainly consists of specifications and is structured by theory morphisms between specifications. The system support of TYPELAB is further described in Section 3.

2 The Specification Language of TYPELAB

The TYPELAB specification language has properties that make it suitable both for small-scale and large-scale development and verification tasks. As an example of a typical specification, consider an incomplete definition of the theory of lists, as shown in Figure 1.

* This research has partly been supported by the "Deutsche Forschungsgemeinschaft" within the "Schwerpunktprogramm Deduktion"

```
defn LIST := fun (E:ELEM)
SPEC
   List:Type,
   % constructors
   nil: List,
   cons: E.T -> List -> List,

   % selectors
   first: {l:List | not (l = nil)} -> E.T,
   rest: {l:List | not (l = nil)} -> List,

   % application of constructors to selectors
   AXIOM first_selector:
         all(e:E.T,l:List) (first (cons e l)) = e,

   % induction
   AXIOM list_ind:
         all(P:List ->Prop)
            (P nil) ->
            (all(e:E.T,l:List) ((P l) -> (P (cons e l)))) ->
            (all(l:List) (P l)),

   % constructor completeness
   THEOREM constr_compl:
     all(l:List) (l = nil) or (exists(e:E.T, l1:List) l = cons e l1)
END-SPEC;
```

Fig. 1. (Incomplete) Specification of Lists

A specification, enclosed in `SPEC .. END-SPEC`, consists of a sequence of declarations and definitions, which can logically be interpreted in the following way:

- A specification can roughly be understood as a dependent record type of the underlying logic *ECC*. Its elements can be regarded as algebras of the corresponding signature that satisfy the axioms stated in the specification.
- Logically, there is no difference between declarations marked as `AXIOM` and declarations of sorts and elements of the signature. There is a pragmatic distinction insomuch as axioms, together with theorems, receive special treatment from the knowledge base and theorem prover (see section 3). Note that a specification is "flat" in the sense that it is not formally split into a computational and a propositional part, as in the case of deliverables [BM92].
- Theorems give rise to proof obligations, which can be solved using the prover integrated into TYPELAB.

Specification types are first-class objects in TYPELAB. Parameterization over specifications can be expressed by functions taking an element of a specification type (such as **ELEM**, the specification of a non-empty carrier set, in Figure 1 above) and yielding a specification type.

In a similar vein, theory morphisms and refinements of specifications can be expressed by functions such as the one depicted in Figure 2, mapping lists to monoids.

```
defn LIST_to_MONOID := fun(E:ELEM, L:(LIST E))
        (# T := L.List,  op := L.append,  unit := L.nil #) :: MONOID ;
```

Fig. 2. Interpretation of Lists as Monoids

Here, the monoid carrier is taken to be the carrier set of the lists, the binary operation `op` is the `append` function, and the unit is the empty list. By coercing this (partial) realization to the type of monoid specifications, proof obligations corresponding to the axioms of monoids are generated, such as the associativity of the operation (here: `append`).

3 System Support

TYPELAB [vH+96] consists of a type checker, a partly automated proof assistant and, as an experimental feature, a knowledge base of developments and proofs. TYPELAB aims at integrating different currents of system development. Firstly, it is related to systems implementing particular type theories, as for example Alf [MN94], Coq [Cor95], Lego [LP92] and Nuprl [Con86]. With these systems, it shares the logical foundations and the constructive aspect, in that carrying out a proof essentially requires the construction of an appropriate term of the logic.

However, TYPELAB tries to go beyond that by providing high-level proof tactics resembling those found in systems like PVS [OSR93], thus hiding the constructive aspect whenever it is not essential. In particular, a Tableaux-style theorem prover [Wag95] that can handle medium-size proof obligations automatically has been developed. The integration of a rewriting system for dealing with equality proofs is under way [Sor96].

Currently, a knowledge-based component is under construction, which aims at organizing mathematical entities and components of the software development process. These objects are arranged in a taxonomy which is structured by a subsumption relation (see [SLW96] and [Lut95] for details). In the case of parameterized specifications, the subsumption relation is a covariant refinement relation. In a style partly inspired by the IMPS system [FGT93], theorems can be inherited from more general to more specific theories along theory morphisms.

Acknowledgments

The design of the TYPELAB language and system has to a great extent been influenced by Holger Pfeifer, Harald Rueß and Detlef Schwier. Matthias Wagner

has contributed a lot to the infrastructure and has implemented most of the Tableaux style theorem prover. Maria Sorea is currently working on integrating an equality prover into TYPELAB.

References

[Bid91] Michel Bidoit. Development of modular specifications by stepwise refinements using the PLUSS specification language. *Proc. of the Unified Computation Laboratory, Oxford University Press*, 1991.

[BM92] Rod Burstall and James McKinna. Deliverables: a categorical approach to program development in type theory. Technical Report ECS-LFCS-92-242, University of Edinburgh, October 1992.

[Con86] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[Cor95] Cristina Cornes et al. *The Coq Proof Assistant Reference Manual.* INRIA Rocquencourt and CNRS-ENS Lyon, 1995.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. *J. of Automated Reasoning*, 11:213–248, 1993.

[Gog84] J.A. Goguen. Parameterized programming. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984.

[LP92] Zhaohui Luo and Robert Pollack. *LEGO Proof Development System: User's Manual.* University of Edinburgh, Department of Computer Science, 1992.

[Luo94] Zhaohui Luo. *Computation and Reasoning.* Oxford University Press, 1994.

[Lut95] Marko Luther. Wissensbasierte Methoden zur Beweisunterstützung in Typentheorie. Master's thesis, Universität Ulm, 1995. Available at URL http://www.informatik.uni-ulm.de/ki/Forschung/Deduktion /ml-dipl.html.

[MN94] Lena Magnusson and Bengt Nordström. The ALF proof editor and its proof engine. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *Springer LNCS*, pages 213–237, 1994.

[OSR93] S. Owre, N. Shankar, and J.M. Rushby. *The PVS Specification Language.* Computer Science Lab, SRI International, Menlo Park CA 94025, March 1993.

[SJ94] Y. V. Srinivas and R. Jüllig. Specware: Formal support for composing software. Technical Report KES.U.94.5, Kestrel Institute, 1994.

[SLW96] M. Strecker, M. Luther, and M. Wagner. Structuring and using a knowledge base of mathematical concepts: A type-theoretic approach. In *ECAI-96 Workshop on Representation of mathematical knowledge*, pages 23–26, 1996.

[Sor96] Maria Sorea. Integration von Gleichheitsbeweisen in einen typentheoretischen Beweiser. Master's thesis, Universität Ulm, 1996. Forthcoming.

[vH+96] F.W. von Henke, M. Luther, M. Strecker, and M. Wagner. The TYPELAB specification and verification environment. In M. Nivat M. Wirsing, editor, *Proceedings AMAST'96*, pages 604–607. Springer LNCS 1101, 1996.

[Wag95] Matthias Wagner. Entwicklung und Implementierung eines Beweisers für konstruktive Logik. Master's thesis, Universität Ulm, 1995. http://www.informatik.uni-ulm.de/ki/Forschung/Deduktion/mw-dipl.html.

[Wir86] Martin Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42:123–249, 1986.

This article was processed using the LaTeX macro package with LLNCS style