A Verification Framework for *MiniMaple* Programs[*]

Muhammad Taimoor Khan and Wolfgang Schreiner
Dokotratskolleg Computational Mathematics (DK) and
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
`muhammad.khan@dk-compmath.jku.at`
`Wolfgang.Schreiner@risc.jku.at`

In this poster, we present an overview of our ongoing work and results on the development of a verification framework for programs written in a (substantial) subset of the language of the computer algebra system Maple, which we call *MiniMaple*. The main goal here is to detect behavioral errors in such programs w.r.t. their specifications by static analysis. However, the task of the formal specification and verification of *MiniMaple* programs is complex as Maple supports various non-standard types of objects such as unevaluated expressions and also requires abstract data types to formalize computer algebra concepts and notions. To approach our goal, we have defined and formalized the syntax, semantics, type system and specification language for *MiniMaple*. For the verification, we translate an annotated *MiniMaple* program into the language Why3ML of the intermediate verification tool Why3 [1] developed at LRI, France. We generate verifi! cation conditions by the corresponding component of Why3 and then prove the correctness of these conditions by various automatic and interactive theorem provers supported by the Why3 back-end. The main test for our verification framework is the Maple package *DifferenceDifferential* [2] developed at our institute to compute bivariate difference-differential polynomials using relative Gröbner bases. All software (lexer, parser, type checker and translator) is open source and freely available from `http://www.risc.jku.at/people/mtkhan/dk10/`.

As a general overview of our verification framework, first any *MiniMaple* program is parsed to generate an abstract syntax tree (AST). Then the AST is type checked and annotated by type information and translated into a (presumably) semantically equivalent Why3ML program. From this program, Why3 generates verification conditions to be proved correct by its various back-end supported provers. All components of the framework may generate errors and information messages.

The syntax of *MiniMaple* [3] covers all the syntactic domains of Maple but supports fewer alternatives in each domain than Maple; in particular, Maple has many built-in expressions which are not supported in our language. We use the type annotations which Maple introduced for runtime checking for the purpose of the static type checking of *MiniMaple* programs; indeed we have defined a formal type system for *MiniMaple* as a decidable logic with various typing judgments. The type system requires that procedure parameters, procedure results and local variables are type annotated. However, global variables in Maple cannot be type annotated, such that values of arbitrary types can be assigned to them. To handle the correct semantics of such variables inside and outside of the body of procedures, we introduced *global* and *local* contexts. In the former, variables can be introduced by assignments and their types can change arbitrarily, while i! n the latter, variables can only be introduced by declarations and their types can only be specialized [3]. Another issue is the handling of dynamic type tests by the *MiniMaple* expression **type**(E, T). The use of a type test in a conditional may result in different type information for the same variable in different branches of the conditional; we use the type information introduced by the corresponding conditional branches to infer

[*]The research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK10.

the possible type of a variable. We have applied the type checker to the package *DifferenceDifferential*; no crucial errors were found but some bad code parts, e.g. duplicate declaration of variables and global variables that are declared but not used.

Furthermore, we have defined a specification language for *MiniMaple* to formally describe mathematical theories (types, functions, axioms), behavior of procedures (pre- and post-conditions and other constraints), loops (invariants and termination terms) and commands (assertions). In addition to basic formulas, our specification language supports various forms of quantifiers, i.e. logical quantifiers (**forall** and **exists**), numerical quantifiers (**add**, **mul**, **max** and **min**) and sequential quantifiers (**seq**) to represent truth values, numeric values and sequences of values respectively. The language slightly extends the Maple syntax, e.g. logical quantifiers use typed variables and numerical quantifiers use logical conditions that filter values from the specified range of a variable. The language supports abstract data types to specify abstract mathematical concepts, e.g. polynomial rings. As an example, we have f! ormally specified a substantial part of the package *DifferenceDifferential*, e.g. difference-differential operators are formalized by a corresponding abstract data type.

To verify a *MiniMaple* program annotated with types and specifications, we translate this program to the language Why3ML of the intermediate verification tool Why3. We use the Why3 verification conditions generator to produce a set of verification conditions: the pre-conditions of called procedures, the post-conditions of defined procedures, the initial establishing of loop invariants, the preservation of loop invariants after every iteration and the decreasing of termination terms. We then prove their correctness by automated provers (e.g. Z3 and CVC3) and proof assistants (e.g. Coq) supported by the Why3 back-end. The wide range of proof support was one the reasons why we chose Why3, as we are, e.g., dealing with non-linear arithmetic which requires in general an interactive prover. For verification, we have defined the translation of *MiniMaple* into semantically equivalent constructs of Why3ML, e.g. the *MiniMaple* return statement is translated using t! he Why3 exception-handling mechanism, union types are translated to algebraic types and the corresponding type tests are translated using pattern matching. Using this approach, we have already verified most of the low level procedures of the package *DifferenceDifferential*, e.g. "gleicheterme" (comparing two difference-differential terms), "sigmamax" (computing a differential term with given constraints) and "ddsub" (subtraction of differential operators).

Currently, we are in the process of verifying higher level procedures with abstract (data type based) specifications: based on an example we experiment with appropriate proof strategies for such specifications using our verification framework. As a next step, a proof of the soundness of translation for selected *MiniMaple* constructs is planned. One of the reason for choosing Why3 was that it provides a formal (originally weakest precondition based, later also operational) semantics. We have correspondingly defined a formal denotational semantics of *MiniMaple* programs [3], e.g. the semantics of command execution is defined as a state relationship between pre- and post-states, i.e. the *MiniMaple* command semantics $[[C]](e)(s, s')$ states that in an environment e the execution of a command C in a pre-state s may result in a post-state s'. Based on these definitions, we plan to prove that our translation preserves the programs' semantics.

References

[1] F. Bobot and et al. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wrocław, Poland, August 2011.

[2] Christian Dönch. Bivariate Difference-Differential Dimension Polynomials and Their Computation in Maple. Technical report, Research Institute for Symbolic Computation, University of Linz, 2009.

[3] M. T. Khan and W. Schreiner. Towards the Formal Specification and Verification of Maple Programs. In J. Jeuring and et al., editors, *Intelligent Computer Mathematics*, volume 7362 of *LNCS*, pages 231–247. Springer, 2012.