Doctoral Program
Computational Mathematics
Numerical Analysis and Symbolic Computation

FWF
Der Wissenschaftsfonds.

Formal Specification and Verification of Computer Algebra Software (DK10)

Muhammad Taimoor Khan
Supervisor: Wolfgang Schreiner

LAND OBERÖSTERREICH

JOHANNES KEPLER UNIVERSITY LINZ | JKU

Formal Methods Seminar
October 9, 2013

Introduction

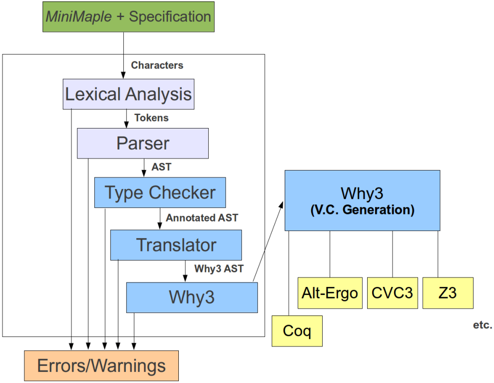
- Behavioral analysis (formal specification and verification) of programs written in (the most widely used) untyped computer algebra languages
  - Mathematica and Maple
- Develop a tool to find errors by static analysis
  - for example type inconsistencies
  - and violations of methods preconditions
- Also
  - to bridge the gap between the example computer algebra algorithm and its implementation
  - to formalize properties of computer algebra
- Demonstration example
  - Maple package *DifferenceDifferential* developed by Christian Dönch
    - computes bivariate difference-differential polynomials using relative Gröbner bases

Achievements

- *MiniMaple*
  - a simple but substantial subset (with slight modifications) of Maple
  - covers all syntactic domains of Maple but fewer expressions
- A formal type system for *MiniMaple*
  - typing rules/judgments
  - auxiliary functions and predicates
  - implemented a corresponding type checker
  - applied type checker to package *DifferenceDifferential*
- A specification language for *MiniMaple*
  - basic formulas and expressions
  - logical quantifiers over typed variables
  - numerical quantifiers with logical conditions
  - sequence quantifier
  - elements of the language
    1. mathematical theories
    2. procedure specifications
    3. loop specifications
    4. assertions
  - formally specified a substantial part of package *DifferenceDifferential*
- Formal semantics of *MiniMaple* and its specification language
  - defined as a state relationship between pre and post-states
  - also a pre-requisite of our translation (to Why3ML)
- Verification framework for *MiniMaple* and specification language

High level overview of our verification framework

- scanning and parsing
- type checking
- translation to Why3ML
  - automatic and semantically equivalent
- generation of verification conditions
- proving of verification conditions
  - in particular methods preconditions
  - with automatic and interactive theorem provers, e.g. Z3, Coq etc.



Example: Package *DifferenceDifferential*

The procedure SP(z,s,t,v,s1,t1)

- is a high-level procedure (calls low-level procedures, e.g. ddprod, ddsub)
- computes s-polynomial of the given two difference-differential operators (DDO) s and t, where
  - z is a positive integer
  - s, t $\in K[\Delta, \Sigma]E$
  - v $\in [\Delta, \Sigma]E$
  - s1, t1 $\in [\Delta, \Sigma]$

```
SP := proc (z::integer, s::list(ddo_term), t::list(ddo_term), v::list(ddo_term),
        s1::[list(integer),list(integer)], t1::[list(integer),list(integer)])::list(ddo_term);
...
for i from 1 by 1 to anzdelta do
b1[2][i] := b1[2][i]-s1[2][i];
...
end do;
...
c1 := ddprod(d1, f);
c2 := ddprod(d2, g);
sp := ddsub(c1, c2);
return sp;
end proc;
```

Formal specification of the procedure SP

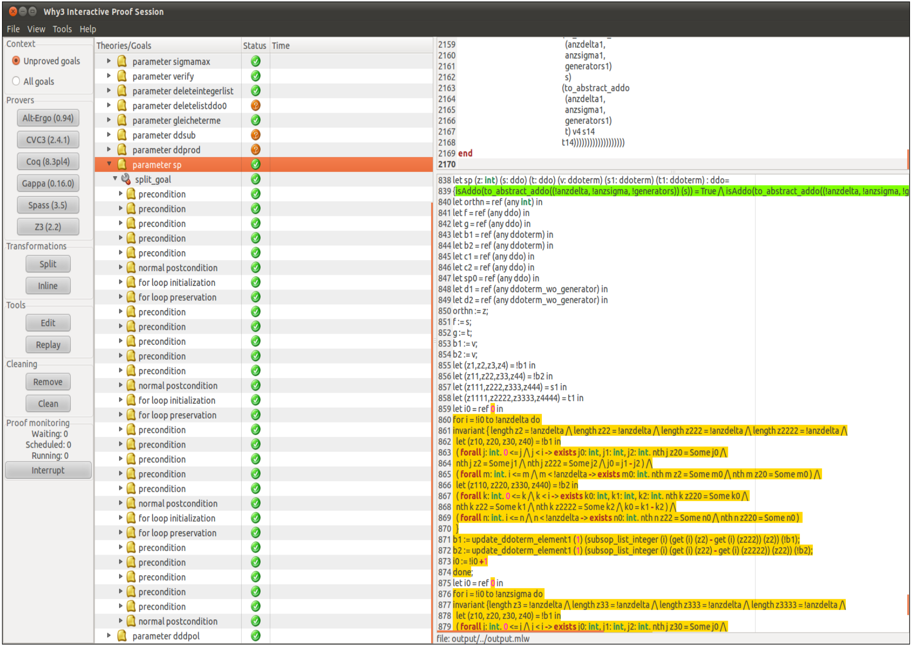
```
(*@ `type/addo`;
`type/ddo_term`:=[symbol, list(integer), list(integer), symbol];
`type/addo_data`:=[integer, integer, list(symbol)];
define(create_addo()::addo);
define(add_term_addo(d:: addo_data, a:: addo, t:: ddo_term)::addo);
define(isAddo(a:: addo)::boolean,
    isAddo(a:: addo) = forall(n:: integer, 1 <= n and n <= length_addo(a) implies
    isAddo_term(get_addo_data(a), get_addo_term(a)));
define(sPol(z::integer, s::addo, t::addo, v::list(ddo_term),...)::addo, ...)
`type/addo_rep`:=list(ddo_term);
define(to_abstract_ddo(d:: addo_data, m:: addo_rep)::addo,
    to_abstract_addo(d:: addo_data, m:: add_rep) =
     'if'(nops(m) = 0, create_addo(), add_addo_term(d, [op(1..nops(m)-1, m)], m[nops(m)]) ));
define(concrete_addo(d:: addo_data, m:: addo_rep, a:: addo)::boolean, ...
...
@*)
...

SP := proc (z::integer, s::list(ddo_term), t::list(ddo_term), v::list(ddo_term),
          s1::[list(integer),list(integer)], t1::[list(integer),list(integer)])::list(ddo_term);
(*@
requires isAddo(to_abstract_addo([anzdelta, anzsigma, generators], s)) = true and
         isAddo(to_abstract_addo([anzdelta, anzsigma, generators], t)) = true and ...;
global EMPTY;
ensures LET ad = to_abstract_addo([anzdelta, anzsigma, generators], RESULT) IN
          isAddo(ad) = true and
          ad = sPol(z, to_abstract_addo([anzdelta, anzsigma, generators], s),
                 to_abstract_addo([anzdelta, anzsigma,generators], t), v, s1, t1);
@*)
...
end proc;
```

Verification of the procedure SP

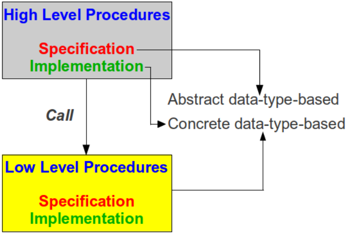
- Translation to Why3ML
  - some manual modifications in the translation to ease reasoning
- Verification conditions generation
  - pre-conditions of called procedures, e.g. ddprod, ddsub
  - post-condition of SP
  - initialization and preservation of loop invariants
- Proving verification conditions
  - mostly proved by automatic decision procedures, e.g. Z3, Alt-Ergo
  - further conditions resulted in two lemmas
    - $\forall a : addo, r : addo\_rep, d : addo\_data.$
      $invariant\_addo(d)(r) \Rightarrow$
      $a = to\_abstract\_addo(d)(r) \Leftrightarrow concrete\_addo(d)(r)(a)$
    - $\forall d : addo\_data, r : addo\_rep, a : addo.$
      $let\ (z1, z2, z3) = d\ in$
      $a = to\_abstract\_addo(d)(r) \Rightarrow$
      $isDDO(r)(z1)(z2)(z3) = isAddo(to\_abstract\_addo(d)(r))$
    - proof by induction on addo_rep (i.e. list of tuples)
    - case analysis on addo's constructors and
    - some Coq tactics, e.g. intros, rewrite, simpl, ring
  - with the lemmas, the rest gets proved automatically

Demo

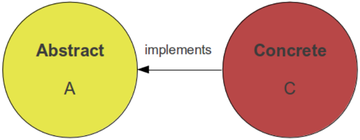

Why3 Interactive Proof Session
File View Tools Help
Context
Unproved goals
All goals
Provers
Alt-Ergo (0.94)
CVC3 (2.4.1)
Coq (8.3pl4)
Gappa (0.16.0)
Spass (3.5)
Z3 (2.2)
Transformations
Split
Inline
Tools
Edit
Replay
Cleaning
Remove
Clean
Proof monitoring
Waiting: 0
Scheduled: 0
Running: 0
Interrupt
Theories/Goals
Status
Time
parameter sigmamax
parameter verify
parameter deleteintegerlist
parameter deletelistddo0
parameter gleicheterme
parameter ddsub
parameter ddprod
parameter sp
split_goal
precondition
normal postcondition
for loop initialization
for loop preservation
parameter dddpol
end
838 let sp (z: int) (s: ddo) (t: ddo) (v: ddoterm) (s1: ddoterm) (t1: ddoterm) : ddo=
840 let orthn = ref (any int) in
841 let f = ref (any ddo) in
842 let g = ref (any ddo) in
843 let b1 = ref (any ddoterm) in
844 let b2 = ref (any ddoterm) in
845 let c1 = ref (any ddo) in
846 let c2 = ref (any ddo) in
847 let sp0 = ref (any ddo) in
848 let d1 = ref (any ddoterm_wo_generator) in
849 let d2 = ref (any ddoterm_wo_generator) in
850 orthn := z;
851 f := s;
852 g := t;
853 b1 := v;
854 b2 := v;
855 let (z1,z2,z3,z4) = !b1 in
856 let (z11,z22,z33,z44) = !b2 in
857 let (z111,z222,z333,z444) = s1 in
858 let (z1111,z2222,z3333,z4444) = t1 in
859 let i0 = ref 0 in
860 for i = !i0 to !anzdelta do
862 let (z10, z20, z30, z40) = !b1 in
866 let (z110, z220, z330, z440) = !b2 in
870 }
871 b1 := update_ddoterm_element1 (1) (subsop_list_integer (i) (get (i) (z2) - get (i) (z222)) (z2)) (!b1);
872 b2 := update_ddoterm_element1 (1) (subsop_list_integer (i) (get (i) (z22) - get (i) (z2222)) (z22)) (!b2);
873 i0 := !i0 +1
874 done;
875 let i0 = ref 0 in
876 for i = !i0 to !anzsigma do
878 let (z10, z20, z30, z40) = !b1 in
file: output/../output.mlw

Verification of package *DifferenceDifferential*

- 15 low level procedures
  - almost 100% verified
  - verification of concrete specifications
  - includes 80% automatic and 20% interactive proofs (including lemmas)
- 13 high level procedures
  - 6 proofs (all interactive)
  - verification of abstract specifications, where such procedures
    - have implementation based on concrete data types, e.g. DDO is implemented as a list of its terms as tuples and
    - are specified by abstract data type, e.g. the DDO is specified by an abstract data type "addo" with corresponding operations and mathematical properties
  - $>$ 70% formally specified



Logical Formulation for the Verification of High-Level Procedures



1. An *abstract model* (ADT-based) defines A
2. A concrete type *representation* C
3. A *mapping* function "$abstract : C \rightarrow A$"
4. A *concretization relationship* between C and A "$concrete \subseteq C \times A$"
5. An *invariant* predicate "$invariant \subseteq C$"
6. A *lemma*

$$\forall c : C, a : A, invariant(c) \Rightarrow (a = abstract(c) \Leftrightarrow concrete(a, c))$$

With the help of *concrete*, the relationship between given abstract (a) and concrete (c) becomes more direct usable in the proof

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$Soundness\_cseq(Cseq) \Leftrightarrow$

$\langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw)$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression:$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$Soundness\_cseq(Cseq) \Leftrightarrow$

$\langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw)$
$\Rightarrow$

$$\langle t, cw \rangle \rightarrow \langle t', vw \rangle$$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$Soundness\_cseq(Cseq) \Leftrightarrow$

$$\langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw)$$
$$\Rightarrow$$

$$\langle t, cw \rangle \rightarrow \langle t', vw \rangle$$
$$\Rightarrow \quad equals(s, t) \wedge [\![Cseq]\!](em)(s, s')$$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$Soundness\_cseq(Cseq) \Leftrightarrow$

$\langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw)$
$\Rightarrow$

$$\Rightarrow \quad equals(s, t) \wedge \frac{\langle t, cw \rangle \rightarrow \langle t', vw \rangle}{[\![Cseq]\!](em)(s, s')}$$

$\Rightarrow equals(s', t') \wedge equals(dm, vw)$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$$
\begin{aligned}
&Soundness\_cseq(Cseq) \Leftrightarrow \\
&\forall em \in Environment, cw \in Expressionw, ew, ew' \in Environmentw, dw, dw' \in Declw, \\
&\quad tw, tw' \in Theoryw : wellTyped(em, Cseq) \wedge consistent(em, ew, dw, tw) \wedge \\
&\quad \langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw) \\
&\quad \Rightarrow wellTyped(cw, ew', dw', tw') \wedge extendsEnv(ew', cw, ew) \wedge extendsDecl(dw', cw, dw) \wedge \\
&\qquad extendsTheory(tw', cw, tw) \wedge \\
&\qquad \forall t, t' \in Statew, vw \in Valuew : \langle t, cw \rangle \rightarrow \langle t', vw \rangle \\
&\qquad\quad \Rightarrow \exists s, s' \in State : equals(s, t) \wedge [\![Cseq]\!](em)(s, s') \wedge \\
&\qquad\qquad \forall s, s' \in State, dm \in InfoData : equals(s, t) \wedge [\![Cseq]\!](em)(s, s') \wedge dm = infoData(s') \\
&\qquad\qquad\quad \Rightarrow equals(s', t') \wedge equals(dm, vw)
\end{aligned}
$$

Soundness of *MiniMaple* to Why3ML Translation

- Soundness statement is:
  $\forall Cseq \in Command, C \in Command, E \in Expression :$
  $Soundness\_cseq(Cseq) \wedge Soundness\_c(C) \wedge Soundness\_e(E)$

where the main goal is as follows:

$$
\begin{array}{l}
Soundness\_cseq(Cseq) \Leftrightarrow \\
\forall em \in Environment, cw \in Expressionw, ew, ew' \in Environmentw, dw, dw' \in Declw, \\
\quad tw, tw' \in Theoryw : wellTyped(em, Cseq) \wedge consistent(em, ew, dw, tw) \wedge \\
\quad \langle cw, ew', dw', tw' \rangle = T[\![Cseq]\!](em, ew, dw, tw) \\
\quad \Rightarrow wellTyped(cw, ew', dw', tw') \wedge extendsEnv(ew', cw, ew) \wedge extendsDecl(dw', cw, dw) \wedge \\
\qquad extendsTheory(tw', cw, tw) \wedge \\
\qquad \forall t, t' \in Statew, vw \in Valuew : \langle t, cw \rangle \rightarrow \langle t', vw \rangle \\
\qquad\quad \Rightarrow \exists s, s' \in State : equals(s, t) \wedge [\![Cseq]\!](em)(s, s') \wedge \\
\qquad\qquad \forall s, s' \in State, dm \in InfoData : equals(s, t) \wedge [\![Cseq]\!](em)(s, s') \wedge dm = infoData(s') \\
\qquad\qquad\quad \Rightarrow equals(s', t') \wedge equals(dm, vw)
\end{array}
$$

$Soundness\_c(C) \Leftrightarrow \ldots$

$Soundness\_c(E) \Leftrightarrow \ldots$

Proof for the soundness of translation

- We prove the soundness for selected constructs of *MiniMaple*
  - command sequence
  - conditional command
  - assignment command
  - while-loop command (partially complete)
- Proof is based on
  - formal semantics of *MiniMaple* and its specification language and
  - available definition of operational semantics of Why3ML
- We prove by induction on *MiniMaple* syntactic domains
  - with various lemmas

Current status and activities

Current Work

- To complete the soundness proof
- Writing of thesis

Finally

- Complete the writing of thesis and final defense (Nov)

https://www.dk-compmath.jku.at/people/mtkhan

Publications

- **Conference/workshop proceedings**
  1. M.T. Khan, W. Schreiner, *A Verification Framework for MiniMaple Programs*, In: ACM Communications in Computer Algebra, 47(3):?-?, September 2013 (accepted poster at ISSAC 2013)
  2. M.T. Khan, *On the Formal Semantics of MiniMaple and its Specification Language*, In: Proc. of FIT, 2012, IEEE library, Islamabad, December 2012
  3. M.T. Khan, W. Schreiner, *Towards the Formal Specification and Verification of Maple Programs*, In: Intelligent Computer Mathematics, LNAI 7362, Springer, pp. 231-247, Germany, July 2012 (**Best Student Paper Award**)
  4. M.T. Khan, W. Schreiner, *On the Formal Specification of Maple Programs*, In: Intelligent Computer Mathematics, LNAI 7362, pp. 443-447, Germany
  5. M.T. Khan, W. Schreiner, *Towards a Behavioral Analysis of Computer Algebra Programs*, In: Proc. of the 23rd Nordic Workshop on Programming Theory (NWPT'11), pp. 42-44, Vasteras, Sweden, October 2011
- **Technical reports**
  1. M.T. Khan, *On the Formal Verification of Maple Programs*, Technical report no. 2013-06 in DK Report Series, July 2013
  2. M.T. Khan, *Translation of MiniMaple to Why3ML*, Technical report no. 2013-02 in DK Report Series, February 2013
  3. M.T. Khan, *Formal Semantics of a Specification Language for MiniMaple*, Technical report no. 2012-06 in DK Report Series, April 2012
  4. M.T. Khan, *Formal Semantics of MiniMaple*, Technical report no. 2012-06 in DK Report Series, January 2012
  5. M.T. Khan, *Towards a Behavioral Analysis of Computer Algebra Programs*, Technical report no. 2011-13 in DK Report Series, November 2011
- M.T. Khan, *Formal Specification and Verification of Computer Algebra Software*, PhD Thesis, November 2013 (to appear)