Doctoral Program
Computational Mathematics
Numerical Analysis and Symbolic Computation

FWF
Der Wissenschaftsfonds.

On the Formal Specification and Verification of MiniMaple Programs (progress report DK10)

Muhammad Taimoor Khan
Supervisor: Wolfgang Schreiner

LAND OBERÖSTERREICH

Formal Methods Seminar
Hagenberg, Austria

JOHANNES KEPLER UNIVERSITY LINZ | JKU

October 10, 2012

Introduction

- Formal specification respectively 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

Past activities (Mar. 2010 to Sep. 2011)

- *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*
  - no crucial errors found but some bad code parts are identified
- Formal semantics of *MiniMaple* - partially completed
  - defined as a state relationship between pre and post-states
  - also a pre-requisite of our verification calculus

A *MiniMaple* program type checked

```
status:=0;
sum := proc(l::list(Or(integer,float)))::[integer,float];
         global status;
         local i::integer, x::Or(integer,float), si::integer:=0, sf::float:=0.0;
         # π={..., status:anything, i:integer, x:Or(integer,float), si:integer, sf:float}
         for i from 1 by 1 to nops(l) do
             x:=l[i]; status:=i;
             # π={..., i:integer, x:Or(integer,float),..., status:integer}
             if type(x,integer) then
             # π={..., i:integer, x:integer, si:integer, ..., status:integer}
                  if (x = 0) then return [si,sf]; end if; si:=si+x;
             elif type(x,float) then
             # π={..., i:integer, x:float, ..., sf:float, status:integer}
                  if (x < 0.5) then return [si,sf]; end if; sf:=sf+x;
             end if;
             # π={..., i:integer, x:Or(integer,float), si:integer, sf:float, status:integer}
         end do;
         # π={..., status:anything, i:integer, x:Or(integer,float), si:integer, sf:float}
         status:=-1; return [si,sf];
         end proc;
         ...
```

Past activities - continued

- A specification language for *MiniMaple*
  - Formula language that supports
    - basic formulas and expressions
    - logical quantifiers (**exists** and **forall**) over typed variables
    - numerical quantifiers (**add, mul, min** and **max**) with logical condition
    - sequence quantifier (**seq**)
  - Elements of the Specification Language
    1. mathematical theories (**types, functions and axioms**)
    2. procedure specifications (**pre-post conditions, exceptions** and **global variables**)
    3. loop specifications (**invariants** and **termination terms**)
    4. assertions

Example(2) - An example *DifferenceDifferential* utility procedure

VGB := **proc** (z::**integer**, a::DDO, b::DDO)::**list**([**symbol**,**list**(**symbol**),**list**(**symbol**)]);
...
return v;
end proc;

Input: z, a, b
Output: v - a list of tuples [e,f,g] where

- e is a generator
- $f = \mathrm{lt}_{<'_z}(a)$
- $g = \mathrm{lt}_{<'_z}(b)$

Computes generators w.r.t. leading terms of the given difference differential operators

An example utility procedure of *DifferenceDifferential* formally specified

```
(*@
    'type/ADDO';
    define(terms, terms(ad::ADDO)=...);
    define(getTerm, getTerm(ad::ADDO,i::nat, j::nat)=...);
    isADDO(d);
    isADDOTerm(c,n,z,e);
    ...
    assume(isADDO(d) equivalent forall(i::integer, 1<=i and i<=terms(d) implies
             isADDOTerm(getTerm(d,i,1), getTerm(d,i,2), getTerm(d,i,3), getTerm(d,i,4])));
    assume(isADDOTerm(c,n,z,e) equivalent inField(c) and isGenerator(e));
   ...
    define(power, power(a::integer,0)=1, power(a::integer,b::integer)= mul(a,1...b));
    define(maps, maps(d::DDO)=...);
   @*)
global noauto, generators, ...;
   ...
(*@
    requires 1 <= z and z <= power(2,length(noauto)) and
             forall(i::integer, 1<=i and i<=terms(maps(a)) implies isGenerator(getTerm(maps(a),i,4))) and
             forall(i::integer, 1<=i and i<=terms(maps(b)) implies isGenerator(getTerm(maps(b),i,4)));
    global EMPTY;
    ensures
       ( forall(j::integer, 1<=j and j<=nops(RESULT) implies isGenerator(RESULT[j][1],maps(a),maps(b)) and
                  RESULT[j][2] = isLT(maps(a),z) and RESULT[j][3] = isLT(maps(b),z)) )
       or
       (nops(RESULT) = 0 and ...);
    @*)

VGB := proc (z::integer, a::DDO, b::DDO)::list([symbol,list(symbol),list(symbol)]) ... return v; end proc;
```

A type checker for specification language - Demo(1)

```
/home/taimoor/antlr3/Test66.m parsed with no errors.
Generating Annotated AST...
#prog#
#decl#
#FuncDef-decl#
define(
#expression#
#idexp#
fac
,
#rule-sequence#
#rule1#

....

*************PROGRAM-ANNOTATION START*************
PI -> [
result:[integer,float]
sum:procedure[[integer,float]](list(Or(integer,float)))
status:integer
]
RetTypeSet -> {}
ThrownExceptionSet -> {}
RetFlag -> not_aret
*************PROGRAM-ANNOTATION END***************

Annotated AST generated.
The program type-checked correctly.
```

Recent activities (Oct. 2011 to Date)

- Visit to ENSIIE, France (Sep. 15 to Dec. 15, 2011)
  - collaboration with **Why3** and **FoCaLiZe** teams
- Defined formal semantics of
  - *MiniMaple* and
  - its specification language
- Development of verification calculus for *MiniMaple*
  - automatic translation of annotated *MiniMaple* to Why3
  - verification conditions generation
  - proving correctness of generated verification conditions

Why3 is used as an intermediate framework for our verification calculus

Formal semantics - procedure specification

$$
\begin{array}{ll}
\text{proc\_spec} = & \textbf{requires}\ exp_1;\\
& \textbf{global}\ Iseq;\\
& \textbf{ensures}\ exp_2;\\
& excep;\\
& \textbf{proc}(\text{Pseq})\text{::T; S;R}\ \textbf{end}
\end{array}
$$

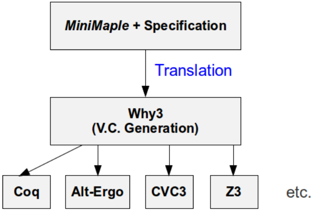
$[\![\text{proc\_spec}]\!]$: $\mathbb{P}(Env)$

$$
\begin{array}{l}
[\![\text{proc\_spec}]\!](e) \Leftrightarrow\\
\ \text{LET}\ (iseq, Tseq) = getIdsAndTypes(Pseq)\\
\ \text{IN}\\
\ \forall vseq \in [\![Tseq]\!], e_1 \in Env, s_1, s_2 \in State, v, r \in Value, b, b_1 \in Bool:\\
\quad e_1 = push(e, iseq, vseq) \wedge [\![exp_1]\!](e_1)(s_1, inStateU(s_1), r, inValueU(b)) \wedge b = inTrue()\\
\quad \wedge \exists p \in Proc: [\![\textbf{proc}(Pseq)::T; S; R\textbf{end}]\!](e_1)(s_1, inStateU(s_1), inValueU(p))\\
\quad \wedge p(vseq, s_1, inStateU(s_2), inValueU(v))\\
\ \Rightarrow equalsExcept(s_1, s_2, Iseq) \wedge\\
\quad \text{IF}\ exceptions(data(s_2))\ \text{THEN}\\
\qquad [\![excep]\!](e_1)(s_2, inStateU(s_2), v, inValueU(b_1)) \wedge b_1 = inTrue()\\
\quad \text{ELSE}\\
\qquad [\![exp_2]\!](e_1)(s_2, inStateU(s_2), v, inValueU(b_1)) \wedge b_1 = inTrue()\\
\ \text{END}
\end{array}
$$

Soundness statement for the correctness of procedure specification

Verification calculus for *MiniMaple*

1. **Translation of annotated MiniMaple to Why3**
   - automatic and semantically equivalent
2. **Verification conditions generation** by using existing framework Why3 by LRI, France (`http://why3.lri.fr/`)
   - verification conditions generated must be sound w.r.t. formal semantics
3. **Proving correctness of conditions** by Why3 back-end provers
   - in particular methods preconditions



- Some features of Why3 (influenced by ML)
  - supports algebraic and abstract data types
  - also supports pattern matching
  - has WP-based semantics
  - provides collaborative proofs by both automatic and interactive provers

An example (manual) translation of annotated *MiniMaple* to *Why3*

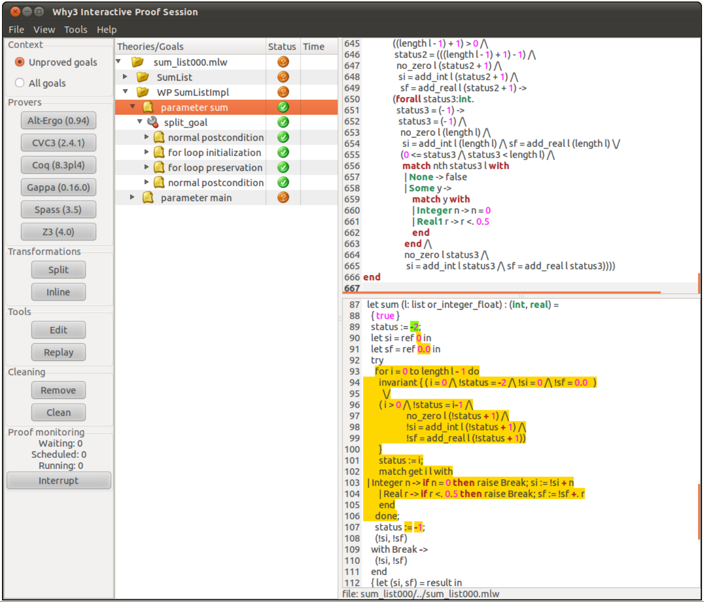
MiniMaple program

```
status:=0;
sum := proc(l::list(Or(integer,float)))::[integer,float];
(*@
requires true;
global status;
ensures
(status = -1 and RESULT[1] = add(e, e in l, type(e,integer))
and RESULT[2] = add(e, e in l, type(e,float))
and (forall(i::integer, 1<=i and i<=nops(l) and type(l[i],integer) ...
and (forall(i::integer, 1<=i and i<=nops(l) and type(l[i],float) ...
or
...
@*)
global status;
local i::integer, x::Or(integer,float), si::integer:=1, sf::float:=1.0;
for i from 1 by 1 to nops(l) do
(*@
invariant status <= i and
        (si = add(l[j], j=1..status-1, type(l[j],integer)) and
        sf = add(l[j], j=1..status-1, type(l[j],float)) and
        forall(i0::integer, 0 <= i0 and i0 <= status ...
        forall(i0::integer, 0 <= i0 and i0 <= status ...
        )
        or
        ... );
decreases nops(l) +1 - i;
*@)
        x:=l[i];
        status:=i;
        if type(x,integer) then
                if (x = 0) then
                        return [si,sf];
                end if;
                si:=si+x;
        elif type(x,float) then
                if (x < 0.5) then
                        return [si,sf];
                end if;
                sf:=sf+x;
        end if;
end do;
status:=-1;
return [si,sf];
end proc;
```

Why3 program

```
theory SumList

  use export int.Int
  ...

  type or_integer_float = Integer int | Real real
  ...
end

module SumListImpl

  use import SumList
  use import module ref.Ref

  val status: ref int

  exception Break

  val get (n: int) (l: list 'a) :
    { 0 <= n < length l } 'a { nth n l = Some result }

  let sum (l: list or_integer_float) : (int, real) =
    { true }
    status := -2;
    let si = ref 0 in
    let sf = ref 0.0 in
    try
      for i = 0 to length l - 1 do
        invariant { ( i = 0 /\ !status = -2 /\ !si = 0 /\ !sf = 0.0   ) ... }
        status := i;
        match get i l with
        | Integer n -> if n = 0 then raise Break; si := !si + n
        | Real r -> if r <. 0.5 then raise Break; sf := !sf +. r
        end
      done;
      status := -1;
      (!si, !sf)
    with Break ->
      (!si, !sf)
    end
    { let (si, sf) = result in
        ( !status = -1 /\ no_zero l (length l) /\
          si = add_int l (length l) /\ sf = add_real l (length l) ) ...
    }
end
```

Manually translated (to Why3) exampled verified

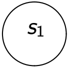


Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation

$$[\![C]\!](e)(s_1, s_2)$$

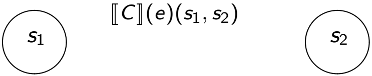
- Soundness of *Command* translation



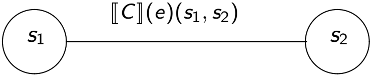
$[\![C]\!](e)(s_1, s_2)$

Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation

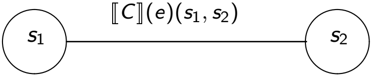


- Soundness of *Command* translation



Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

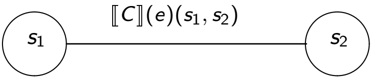
- Soundness of *Command* translation



$$(w\_exp_1, ...) = \mathrm{T}[\![C]\!](...)$$

Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation

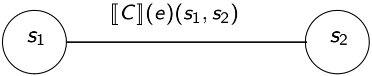


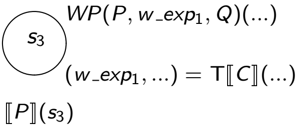
$$WP(P, w\_exp_1, Q)(...)$$

$$(w\_exp_1, ...) = \mathrm{T}[\![C]\!](...)$$

Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

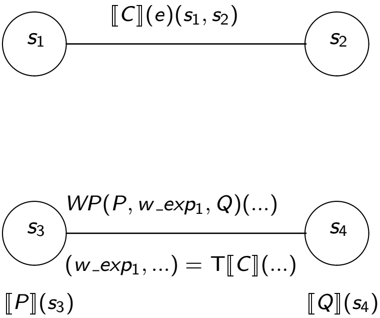
- Soundness of *Command* translation





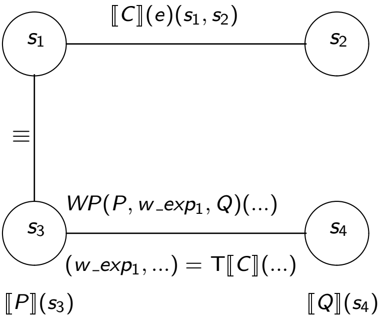
Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation



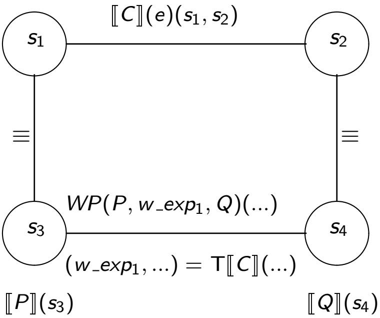
Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation



Automatic translation of annotated *MiniMaple* to Why3 - *Command* example

- Soundness of *Command* translation



Soundness of *Command* translation example

- Soundness statement for translation

$$
\begin{array}{l}
\forall\ C \in Command, s_1, s_2 \in State, e \in Env, wenv, wenv_1 \in Why3\_Env \ldots :\\
\quad [\![C]\!](e)(s_1, inStateU(s_2)) \wedge\\
\quad (wenv, wmdecl, wth) = D[\![C]\!](e) \wedge\\
\quad (inWhy3\_ExpU(w\_exp_1), wenv_1, wmdecl_1, wth_1) = \mathsf{T}\ [\![C]\!](e, wenv, wmdecl, wth)\\
\quad \Rightarrow\\
\qquad \forall P, Q \in Why3\_Exp, s_3, s_4 \in State :\\
\qquad\qquad s_1 \equiv s_3 \wedge s_2 \equiv s_4 \wedge WP(P,\ w\_exp_1,\ Q)(wenv_1, wmdecl_1, wth_1)\\
\qquad\qquad \Rightarrow\\
\qquad\qquad\quad [\![P]\!](s_3) \Rightarrow [\![Q]\!](s_4)
\end{array}
$$

- The formal definition of translator has
  - 40 valuation functions and approx. 50 auxiliary functions/predicates
  - 45 pages

Example translation function for *Command*

- **function signatures**

  $$\mathrm{T}\ [\![C]\!]: Env_m \times Env_w \times DeclU_w \times Thry_w \rightarrow Exp_w \times Env_w \times DeclU_w \times Thry_w$$

- **function definition for for-whlile-loop command**

  $$\begin{array}{l}\mathrm{T}[\![\textbf{for } I \textbf{ in } E_1 \textbf{ while } E_2 \textbf{ do } Cseq \textbf{ end do}]\!](tenv, wenv, wmdecl, wth) = \\ \quad (inWhy3\_ExpU(\textbf{let } I_0 = \textbf{ref } 0 \textbf{ in} \\ \qquad\qquad \textbf{while } I_0 < \mathrm{op\_length}(w\_exp_1)\ \&\ w\_exp_2 \textbf{ do} \\ \qquad\qquad\quad \textbf{let } I = \mathrm{op\_nth}(I_0, w\_exp_1) \textbf{ in} \\ \qquad\qquad\qquad w\_exp_3;\ I_0 := !I_0 + 1 \\ \qquad\qquad \textbf{done}), wenv_3, wmdecl_3, wth_3)\end{array}$$

where

$$\begin{array}{l}(w\_exp_1, wenv_1, wmdecl_1, wth_1) = \mathrm{T}\ [\![E_1]\!](tenv, wenv, wmdecl, wth), \\ (w\_exp_2, wenv_2, wmdecl_2, wth_2) = \mathrm{T}\ [\![E_2]\!](tenv, wenv_1, wmdecl_1, wth_1), \\ (w\_exp_3, wenv_3, wmdecl_3, wth_3) = \mathrm{T}\ [\![Cseq]\!](tenv, wenv_2, wmdecl_2, wth_2), \\ exp\_type_1 = getExpType(w\_exp_1, wenv_1), \\ op\_length = access(length, exp\_type_1, wth_1), \\ op\_nth = access(select, exp\_type_1, wth_1)\end{array}$$

Current status and activities

Achievements

- Defined *MiniMaple* and its specification language
  - Formal grammars
    - implemented parsers correspondingly
  - Type systems
    - implemented type checkers correspondingly
  - Formal semantics
    - as a state relationship between pre- and post-states
    - also as a pre-requisite of our verification calculus
- Typed and formally specified Maple package **DifferenceDifferential**
  - abstract computer algebraic concepts and theories, e.g. Gröbner basis
- Verification calculus for *MiniMaple*
  - Defined translation of annotated *MiniMaple* to Why3
    - semantically equivalent translation
    - implementation of corresponding translator (partially complete)

Next

- To finish implementation of *MiniMaple* to Why3 translator (October 2012)
- Application of translator to test examples
- Application of translator to package *DifferenceDifferential*
  - generation of verification conditions and proving methods preconditions

Recent publications (since October 2011)

- **Technical reports**
  1. M.T. Khan, *Formal Semantics of a Specification Language for MiniMaple*, Technical report no. 2012-06 in DK Report Series, April 2012
  2. M.T. Khan, *Formal Semantics of MiniMaple*, Technical report no. 2012-06 in DK Report Series, January 2012
  3. M.T. Khan, *Towards a Behavioral Analysis of Computer Algebra Programs*, Technical report no. 2011-13 in DK Report Series, November 2011
- **Conference/workshop proceedings**
  1. M.T. Khan, *On the Formal Semantics of MiniMaple and its Specification Language*, In: Proc. of the 10th International Conference on Frontiers of Information Technology (FIT 2012), IEEE digital library, Islamabad, December 2012 (to appear)
  2. 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**)
  3. M.T. Khan, W. Schreiner, *On Formal Specification of Maple Programs*, In: Intelligent Computer Mathematics, LNAI 7362, pp. 443-447, Germany, July 2012
  4. 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