Doctoral Program
Computational Mathematics
Numerical Analysis and Symbolic Computation

FWF
Der Wissenschaftsfonds.

A Progress Report on PhD Thesis (Formal Verification of MiniMaple Programs)

Muhammad Taimoor Khan
Supervisor: Wolfgang Schreiner

LAND OBERÖSTERREICH

Formal Methods Seminar
Hagenberg, Austria

JOHANNES KEPLER UNIVERSITY LINZ | JKU

April 17, 2013

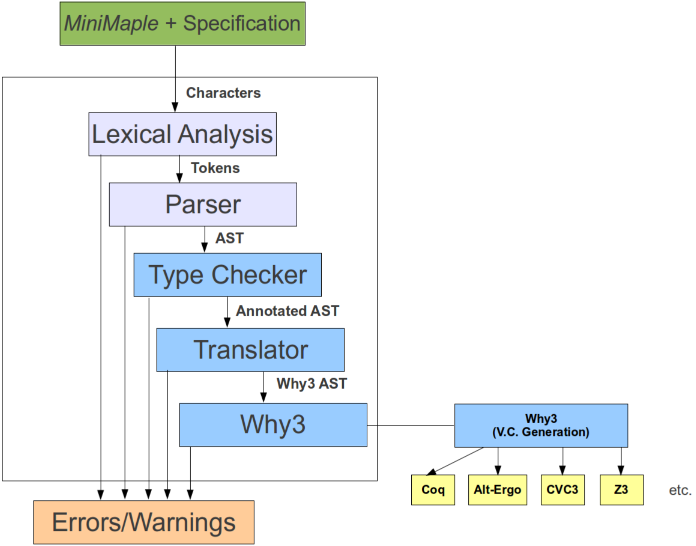
Introduction

- 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 basis

Achievements

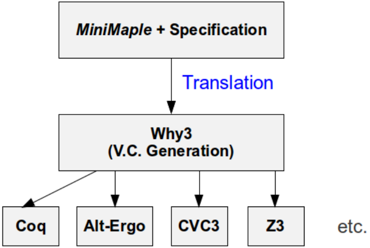
- *MiniMaple*
  - a simple but substantial subset (with slight modifications) of Maple
  - covers all syntactic domains of Maple but fewer expressions
- A formal type system for *MiniMaple*
  - typing rules/judgments
  - auxiliary functions and predicates
- Implemented a corresponding type checker
  - applied type checker to package *DifferenceDifferential*
  - no crucial errors found but some bad code parts are identified
- A specification language for *MiniMaple*
  - 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 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
  - implemented a corresponding type checker
  - 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)

High level overview of our verification framework



Verification calculus for *MiniMaple*

1. **Translation of annotated MiniMaple to Why3**
   - automatic and semantically equivalent
2. **Verification conditions generation** by using existing framework Why3 developed 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

A *MiniMaple* procedure formally specified

```
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) implies l[i]<>0)
            and forall(i::integer, 1<=i and i<=nops(l) and type(l[i],float) implies l[i]>=0.5))
        or (1<=status and status<=nops(l)
            and RESULT[1] = add(l[i], i=1..status-1, type(l[i],integer))
            and RESULT[2] = add(l[i], i=1..status-1, type(l[i],float))
            and ((type(l[status],integer) and l[status]=0)
                    or (type(l[status],float) and l[status]<0.5))
            and forall(i::integer, 1<=i and i<status and type(l[i],integer) implies l[i]<>0)
            and forall(i::integer, 1<=i and i<status and type(l[i],float) implies l[i]>=0.5));
@*)
      global status;
      local i::integer, x::Or(integer,float), si::integer:=0, sf::float:=0.0;
      for i from 1 by 1 to nops(l) do
         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;
```

An example translation of annotated *MiniMaple* to *Why3*

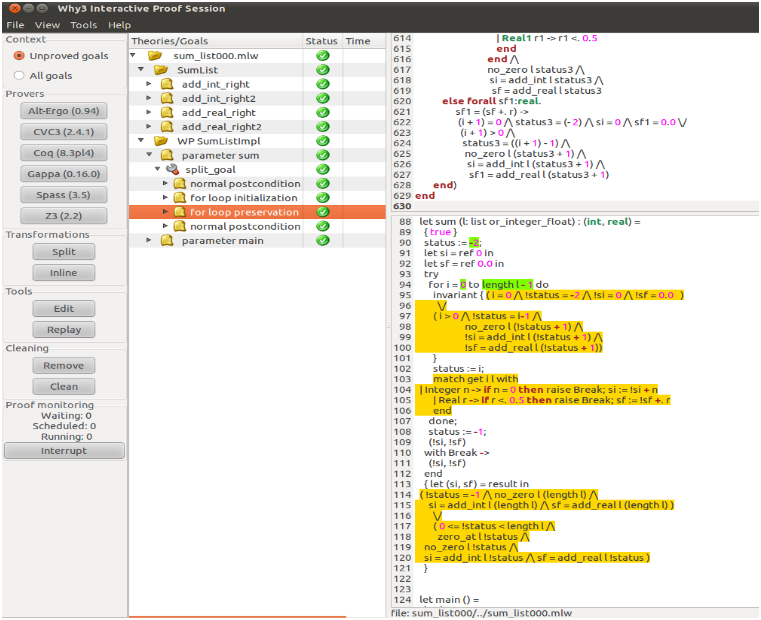
MiniMaple program

```
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],int
and (forall(i::integer, 1<=i and i<=nops(l) and type(l[i],flo
or
...
@*)
global status;
local i::integer, x::Or(integer,float), si::integer:=1, sf::f
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)) an
        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
```

Translated example verified



Verification of example program

1. Four verification conditions (proved by Z3 and Alt-Ergo)
   - a normal postcondition (parameter list is empty)
   - the for-loop (invariant) initialization
   - the for-loop (invariant) preservation and
   - a normal postcondition (parameter list is not empty)
2. Four lemmas (proved by Coq)
   - state that the functions add_int and add_real correctly handle the *reals* and *integers* in the list
   - e.g.

     $$\begin{aligned}&\mathit{lemma}\ \mathit{add\_int\_right}:\\ &\ \ \forall\ e:\mathit{list}\ \mathit{or\_integer\_float}, j:\mathit{int}.\\ &\ \ 0 \le j < \mathit{length}\ e \rightarrow \forall n:\mathit{int}.\\ &\ \ \mathit{nth}\ j\ e = \mathit{Some}\ (\mathit{Integer}\ n)\\ &\qquad\quad \rightarrow \mathit{add\_int}\ e\ (j+1) = \mathit{add\_int}\ e\ j + n\end{aligned}$$
   - prove by induction on e (list)
   - instantiations of universal quantifiers
   - case analysis on elements of e (integer and real) and
   - some Coq tactics, e.g. intros, rewrite and simpl

Verification of main test package *DifferenceDifferential*

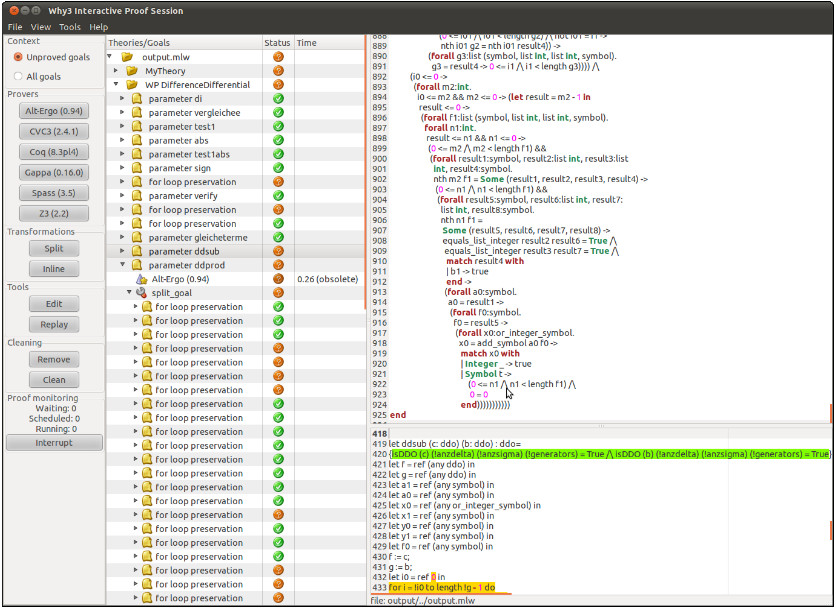
Here the focus is to verify

1. (a substantial part of the package with) concrete specifications
   - low level procedures, e.g. vergleichee, gleicheterme, ddsub, ddprod
2. (some parts of the package with) abstract specifications
   - high level procedures (using low level procedures), e.g. SP, lt2, lt
   - e.g. difference-differential operator (ddo) as list(...) and abstract ddo (addo) as abstract data type

Some gaps w.r.t. full verification are

- no formal documentation of the implementation
- only source is a technical report
- no adequate information for formal specification of some procedures
- manual production of loop invariants is a time taking job
- identified some bugs for the translation of Why3 goals to Coq
- requires advanced skills of proving with Coq

Verification of the package *DifferenceDifferential* - contd.



Formal semantics - procedure specification

$$
\begin{aligned}
\text{proc\_spec} = \; & \textbf{requires } \mathit{exp}_1; \\
& \textbf{global } \mathit{Iseq}; \\
& \textbf{ensures } \mathit{exp}_2; \\
& \mathit{excep}; \\
& \textbf{proc}(\text{Pseq})\text{::T; S;R } \textbf{end}
\end{aligned}
$$

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

$$
\begin{aligned}
& [\![\text{proc\_spec}]\!](e) \Leftrightarrow \\
& \;\text{LET } (\mathit{iseq}, \mathit{Tseq}) = \mathit{getIdsAndTypes}(\mathit{Pseq}) \\
& \;\text{IN} \\
& \;\forall \mathit{vseq} \in [\![\mathit{Tseq}]\!], e_1 \in \mathit{Env}, s_1, s_2 \in \mathit{State}, v, r \in \mathit{Value}, b, b_1 \in \mathit{Bool} : \\
& \quad e_1 = \mathit{push}(e, \mathit{iseq}, \mathit{vseq}) \wedge [\![\mathit{exp}_1]\!](e_1)(s_1, \mathit{inStateU}(s_1), r, \mathit{inValueU}(b)) \wedge b = \mathit{inTrue}() \\
& \quad \wedge \exists p \in \mathit{Proc} : [\![\textbf{proc}(\mathit{Pseq})\text{::}T; S; R\textbf{end}]\!](e_1)(s_1, \mathit{inStateU}(s_1), \mathit{inValueU}(p)) \\
& \quad \wedge p(\mathit{vseq}, s_1, \mathit{inStateU}(s_2), \mathit{inValueU}(v)) \\
& \;\Rightarrow \mathit{equalsExcept}(s_1, s_2, \mathit{Iseq}) \wedge \\
& \quad \text{IF } \mathit{exceptions}(\mathit{data}(s_2)) \text{ THEN} \\
& \qquad [\![\mathit{excep}]\!](e_1)(s_2, \mathit{inStateU}(s_2), v, \mathit{inValueU}(b_1)) \wedge b_1 = \mathit{inTrue}() \\
& \quad \text{ELSE} \\
& \qquad [\![\mathit{exp}_2]\!](e_1)(s_2, \mathit{inStateU}(s_2), v, \mathit{inValueU}(b_1)) \wedge b_1 = \mathit{inTrue}() \\
& \;\text{END}
\end{aligned}
$$

Statement for the correctness of procedure specification

Example translation function for *Command*

- **function signatures**

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

- **function definition for for-while-loop command**

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

where

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

Example translation function for *Command*

- **function signatures**

$$\mathrm{T}\ [\![C]\!]: \mathit{Env}_m \times \ldots \rightarrow \mathit{Exp}_w \times \ldots$$

Example translation function for *Command*

- **function signatures**

$$\mathsf{T}\ [\![C]\!]\colon Env_m \times \ldots \to Exp_w \times \ldots$$

- **soundness statement for command-translation**

Example translation function for *Command*

- **function signatures**

$$\mathrm{T}\ [\![C]\!]\colon Env_m \times \ldots \to Exp_w \times \ldots$$

- **soundness statement for command-translation**

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

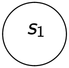
Example translation function for *Command*

- **function signatures**

$$T\ [\![C]\!]: Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

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



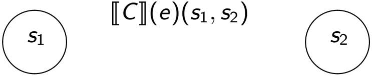
Example translation function for *Command*

- **function signatures**

$$T\ [\![C]\!]: Env_m \times ... \rightarrow Exp_w \times ...$$

- **soundness statement for command-translation**

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

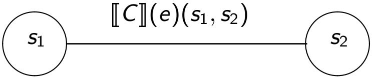


Example translation function for *Command*

- **function signatures**

$$\mathsf{T}\,[\![C]\!]\colon Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

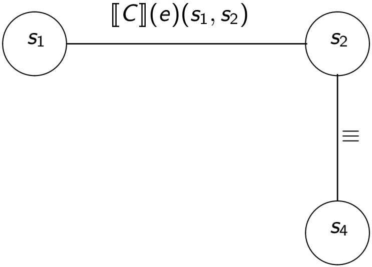


Example translation function for *Command*

- **function signatures**

$$\mathrm{T}\ [\![C]\!]: Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

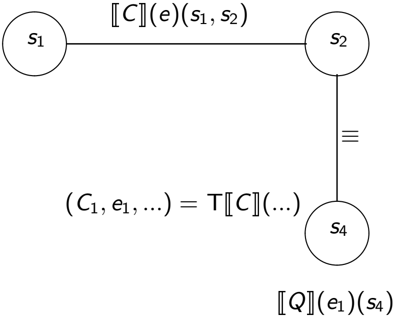


Example translation function for *Command*

- **function signatures**

$$T \llbracket C \rrbracket : Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

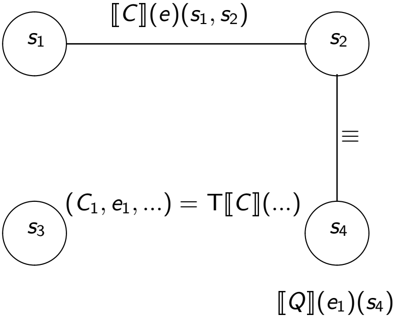


Example translation function for *Command*

- **function signatures**

$$\mathsf{T}\ [\![C]\!]\colon Env_m \times \ldots \to Exp_w \times \ldots$$

- **soundness statement for command-translation**

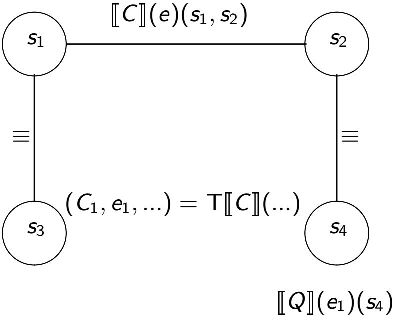


Example translation function for *Command*

- **function signatures**

$$\mathrm{T}\ [\![C]\!]\colon Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

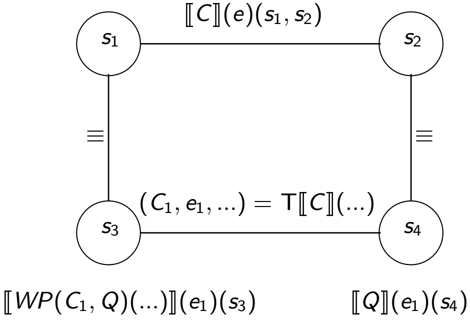


Example translation function for *Command*

- **function signatures**

$$\mathsf{T}\ [\![C]\!]: Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**

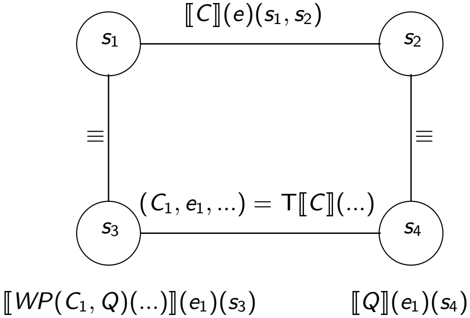


Example translation function for *Command*

- **function signatures**

$$\mathrm{T}\ [\![C]\!]: Env_m \times \ldots \rightarrow Exp_w \times \ldots$$

- **soundness statement for command-translation**



$$\begin{aligned}
&\forall\ C \in Command, C_1, P, Q \in Exp_w,\\
&\quad s_1, s_2 \in State_m, s_3, s_4 \in State_w,\\
&\quad e \in Env_m, e_1 \in Env_w \ldots :\\
&\quad (C_1, e_1, \ldots) = \mathrm{T}\ [\![C]\!](e, \ldots) \wedge P = WP(C_1,\ Q)(e_1, \ldots) \wedge\\
&\quad s_1 \equiv s_3 \wedge s_2 \equiv s_4\\
&\quad \Rightarrow [\![C]\!](e)(s_1, s_2) \wedge [\![P]\!](e_1)(s_3) \Rightarrow [\![Q]\!](e_1)(s_4)
\end{aligned}$$

Current status and activities

Current Work

- Example verification for abstract specification
  - abstract (stack) with representation as (array, int) (concrete)
- Verification of test example
  - mainly detection of violations of methods preconditions
  - verified some parts of the package
  - verification of concrete and abstract specifications

Finally

- Proof for the soundness of translation (parts) w.r.t. the semantics of *MiniMaple* and Why3ML

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

Publications

- **Conference/workshop proceedings**
  1. M.T. Khan, *On the Formal Semantics of MiniMaple and its Specification Language*, In: Proc. of FIT, 2012, IEEE library, Islamabad, December 2012
  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 the 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
- **Technical reports**
  1. M.T. Khan, *Translation of MiniMaple to Why3ML*, Technical report no. 2013-02 in DK Report Series, February 2013
  2. M.T. Khan, *Formal Semantics of a Specification Language for MiniMaple*, Technical report no. 2012-06 in DK Report Series, April 2012
  3. M.T. Khan, *Formal Semantics of MiniMaple*, Technical report no. 2012-06 in DK Report Series, January 2012
  4. M.T. Khan, *Towards a Behavioral Analysis of Computer Algebra Programs*, Technical report no. 2011-13 in DK Report Series, November 2011