

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

Muhammad Taimoor Khan  
Supervisor: Wolfgang Schreiner



Formal Methods Seminar  
Hagenberg, Austria



April 17, 2013

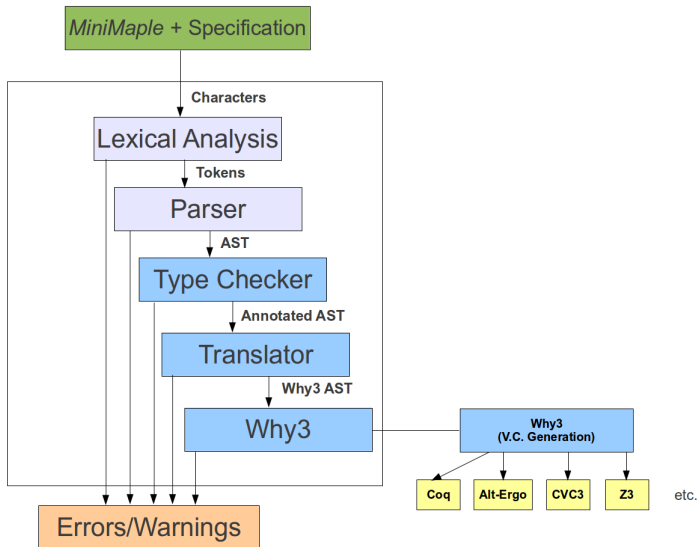
## Introduction

- ▶ 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 Döncb
    - ▶ computes bivariate difference-differential polynomials using relative Gröbner basis

## Achievements

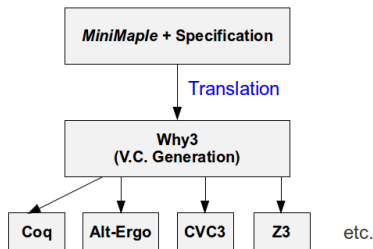
- ▶ *MiniMaple*
  - ▶ a simple but substantial subset (with slight modifications) of Maple
  - ▶ covers all syntactic domains of Maple but fewer expressions
- ▶ A formal type system for *MiniMaple*
  - ▶ typing rules/judgments
  - ▶ auxiliary functions and predicates
- ▶ Implemented a corresponding type 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

```
1. status:=0;
2. 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));
@*)
3. global status;
4. local i::integer, x::Or(integer,float), si::integer:=0, sf::float:=0.0;
5. for i from 1 by 1 to nops(l) do
6.     x:=l[i]; status:=i;
7.     if type(x,integer) then
8.         if (x = 0) then return [si,sf]; end if; si:=si+x;
9.     elif type(x,float) then
10.        if (x < 0.5) then return [si,sf]; end if; sf:=sf+x;
11.    end if;
12. end do;
13. status:=-1; return [si,sf];
14. end proc;
```

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

## MiniMaple program

```
status:=0;
sum := proc(l::list(0r(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::0r(integer,float), si::integer:=1, sf::f]end
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
...
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

The screenshot shows the Why3 Interactive Proof Session interface. The left sidebar contains navigation and tool options:

- Context:** Unproved goals, All goals.
- Provers:** Alt-Ergo (0.94), CVC3 (2.4.1), Coq (8.3p14), 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.

The main area displays a project tree for 'sum\_list000.mlw' with the following structure:

- sum\_list000.mlw (verified)
- SumList (verified)
  - add\_int\_right (verified)
  - add\_int\_right2 (verified)
  - add\_real\_right (verified)
  - add\_real\_right2 (verified)
  - WP SumListImpl (verified)
    - parameter sum (verified)
      - split\_goal (verified)
        - normal postcondition (verified)
        - For loop initialization (verified)
        - for loop preservation (verified)**
        - normal postcondition (verified)
      - parameter main (verified)

The code editor on the right shows the verified code for the 'for loop preservation' goal:

```
614 | Real1 r1 -> r1 <. 0.5
615 end
616 end
617 no_zero l status3
618 si = add_int l status3
619 sf = add_real l status3
620 else forall sf1:real.
621 sf1 = (sf +. r) ->
622 (i + 1) = 0
623 (i + 1) > 0
624 status3 = ((i + 1) - 1)
625 no_zero l (status3 + 1)
626 si = add_int l (status3 + 1)
627 sf1 = add_real l (status3 + 1)
628 end)
629 end
630
631
632 let sum (l: list or _integer_float) : (Int, real) =
633 { true }
634 status := 0;
635 let si = ref 0.0 in
636 let sf = ref 0.0 in
637 try
638 for i = 1 to length l - 1 do
639 invariant (i = 0
640 (i > 0
641 no_zero l (lstatus + 1)
642 lsi = add_int l (lstatus + 1)
643 lsf = add_real l (lstatus + 1))
644 )
645 status := i;
646 match get i with
647 | Integer n -> if n = 0 then raise Break; si := lsi + n
648 | Real r -> if r <. 0.5 then raise Break; sf := lsf +. r
649 end
650 done;
651 status := -1;
652 (lsi, lsf)
653 with Break ->
654 (lsi, lsf)
655 end
656 (let (si, sf) = result in
657 (lstatus == 1
658 no_zero l (length l)
659 si = add_int l (length l)
660 sf = add_real (length l))
661 )
662 (0 <= lstatus < length l
663 zero_at l lstatus
664 no_zero l lstatus
665 si = add_int l lstatus
666 sf = add_real l lstatus)
667 )
668 }
669
670 let main () =
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



## 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.

*lemma add\_int\_right :*

$\forall e : \text{list or\_integer\_float}, j : \text{int}.$

$0 \leq j < \text{length } e \rightarrow \forall n : \text{int}.$

$\text{nth } j \text{ } e = \text{Some } (\text{Integer } n)$

$\rightarrow \text{add\_int } e (j + 1) = \text{add\_int } e j + n$

- ▶ 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.

The screenshot displays the Why3 Interactive Proof Session interface. On the left, there are several panels: 'Context' showing 'Unproved goals' and 'All goals'; 'Provers' with buttons for Alt-Ergo (0.94), CVC3 (2.4.1), Coq (8.3pl4), Gappa (0.16.0), Spass (3.5), and Z3 (2.2); 'Transformations' with 'Split' and 'Inline' buttons; 'Tools' with 'Edit' and 'Replay' buttons; 'Cleaning' with 'Remove' and 'Clean' buttons; and 'Proof monitoring' with 'Interrupt' button.

The main area is a table of goals with columns for 'Theories/Goals', 'Status', and 'Time'. The 'Status' column shows various icons (orange circle with a dot, green checkmark, red X) indicating the verification progress of each goal. The 'Time' column shows the execution time for some goals, such as '0.26 (obsolete)'.

On the right, a detailed view of a goal's proof script is shown, featuring nested quantifiers and logical expressions:

```
( $\exists$  i0 <= 107  $\wedge$  ! (i0 < length g2)  $\wedge$  ! (i0 < n1  $\rightarrow$ 
nth i0 i2 = nth i0 i1 result4)  $\rightarrow$ 
(forall g3: list (symbol, list int, list int, symbol).
g3 = result4  $\rightarrow$  0 <= i1  $\wedge$  ! i1 < length g3)))  $\wedge$ 
(i0 <= 0  $\rightarrow$ 
(forall m2: int.
i0 <= m2 && m2 <= 0  $\rightarrow$  (let result = m2 - 1 in
result <= 0  $\rightarrow$ 
(forall f1: list (symbol, list int, list int, symbol).
forall n1: int.
result <= n1 && n1 <= 0  $\rightarrow$ 
(0 <= m2  $\wedge$  m2 < length f1) &&
(forall result1: symbol, result2: list int, result3: list
int, result4: symbol.
nth m2 f1 = Some (result1, result2, result3, result4)  $\rightarrow$ 
(0 <= n1  $\wedge$  n1 < length f1) &&
(forall result5: symbol, result6: list int, result7:
list int, result8: symbol.
list int, result8: symbol.
nth n1 f1 =
Some (result5, result6, result7, result8)  $\rightarrow$ 
equals_list_integer result2 result6 = True  $\wedge$ 
equals_list_integer result3 result7 = True  $\wedge$ 
match result4 with
| b1  $\rightarrow$  true
end  $\rightarrow$ 
(forall a0: symbol.
a0 = result1  $\rightarrow$ 
(forall f0: symbol.
f0 = result5  $\rightarrow$ 
(forall x0: or_integer_symbol.
x0 = add_symbol a0 f0  $\rightarrow$ 
match x0 with
| integer  $\rightarrow$  true
| symbol c  $\rightarrow$ 
(0 <= n1  $\wedge$  n1 < length f1)  $\wedge$ 
0 = 0
end)))))))))
end)))))))))
end
```

Below the proof script, there are several lines of code defining variables and functions, such as:

```
418
419 let ddsub (c: ddo) (b: ddo) : ddo =
420 | DDO (c) [anzdelta] [anzsigma] [generators] = true  $\wedge$  ! isDDO (b) [anzdelta] [anzsigma] [generators] = true
421 let f = ref (any ddo) in
422 let g = ref (any ddo) in
423 let a1 = ref (any symbol) in
424 let a0 = ref (any symbol) in
425 let x0 = ref (any or_integer_symbol) in
426 let x1 = ref (any symbol) in
427 let y0 = ref (any symbol) in
428 let y1 = ref (any symbol) in
429 let f0 = ref (any symbol) in
430 f := c;
431 g := b;
432 let i0 = ref 0 in
433 for i = i0 to length lg - 1 do
```

The bottom status bar shows 'File: output/./output.mlw'.

## Formal semantics - procedure specification

```
proc_spec = requires exp1;  
           global lseq;  
           ensures exp2;  
           excep;  
           proc(Pseq)::T; S;R end
```

$\llbracket \text{proc\_spec} \rrbracket : \mathbb{P}(\text{Env})$

$\llbracket \text{proc\_spec} \rrbracket (e) \Leftrightarrow$

LET (*iseq*, *Tseq*) = *getIdsAndTypes*(Pseq)

IN

$\forall vseq \in \llbracket Tseq \rrbracket, e_1 \in \text{Env}, s_1, s_2 \in \text{State}, v, r \in \text{Value}, b, b_1 \in \text{Bool} :$

$e_1 = \text{push}(e, \text{iseq}, vseq) \wedge \llbracket \text{exp}_1 \rrbracket (e_1)(s_1, \text{inStateU}(s_1), r, \text{inValueU}(b)) \wedge b = \text{inTrue}()$

$\wedge \exists p \in \text{Proc} : \llbracket \text{proc}(Pseq)::T; S; \text{Rend} \rrbracket (e_1)(s_1, \text{inStateU}(s_1), \text{inValueU}(p))$

$\wedge p(vseq, s_1, \text{inStateU}(s_2), \text{inValueU}(v))$

$\Rightarrow \text{equalsExcept}(s_1, s_2, \text{lseq}) \wedge$

IF *exceptions*(*data*(*s*<sub>2</sub>)) THEN

$\llbracket \text{excep} \rrbracket (e_1)(s_2, \text{inStateU}(s_2), v, \text{inValueU}(b_1)) \wedge b_1 = \text{inTrue}()$

ELSE

$\llbracket \text{exp}_2 \rrbracket (e_1)(s_2, \text{inStateU}(s_2), v, \text{inValueU}(b_1)) \wedge b_1 = \text{inTrue}()$

END

Statement for the correctness of procedure specification

## Example translation function for *Command*

### ► function signatures

$T \llbracket C \rrbracket : Env_m \times Env_w \times DeclU_w \times Thry_w \rightarrow Exp_w \times Env_w \times DeclU_w \times Thry_w$

### ► function definition for for-while-loop command

$T \llbracket \text{for } l \text{ in } E_1 \text{ while } E_2 \text{ do } Cseq \text{ end do} \rrbracket (tenv, wenv, wmdecl, wth) =$   
 $(inWhy3\_ExpU(\text{let } l_0 = \text{ref } 0 \text{ in}$   
    **while**  $l_0 < \text{op\_length}(w\_exp_1) \ \& \ w\_exp_2$  **do**  
        **let**  $l = \text{op\_nth}(l_0, w\_exp_1)$  **in**  
             $w\_exp_3; l_0 := !l_0 + 1$   
    **done**),  $wenv_3, wmdecl_3, wth_3)$

where

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

## Example translation function for *Command*

- ▶ **function signatures**

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

## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**

## Example translation function for *Command*

- ▶ **function signatures**

$$\text{T } \llbracket C \rrbracket: \text{Env}_m \times \dots \rightarrow \text{Exp}_w \times \dots$$

- ▶ **soundness statement for command-translation**

$$\llbracket C \rrbracket(e)(s_1, s_2)$$



## Example translation function for *Command*

- ▶ **function signatures**

$$\text{T } \llbracket C \rrbracket: \text{Env}_m \times \dots \rightarrow \text{Exp}_w \times \dots$$

- ▶ **soundness statement for command-translation**



$$\llbracket C \rrbracket(e)(s_1, s_2)$$

## Example translation function for *Command*

- ▶ **function signatures**

$$\text{T } \llbracket C \rrbracket: \text{Env}_m \times \dots \rightarrow \text{Exp}_w \times \dots$$

- ▶ **soundness statement for command-translation**

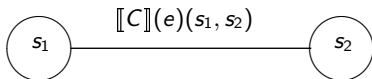
$$\begin{array}{ccc} \textcircled{s_1} & \llbracket C \rrbracket(e)(s_1, s_2) & \textcircled{s_2} \end{array}$$

## Example translation function for *Command*

- ▶ **function signatures**

$$\mathbb{T} \llbracket C \rrbracket: Env_m \times \dots \rightarrow Exp_w \times \dots$$

- ▶ **soundness statement for command-translation**

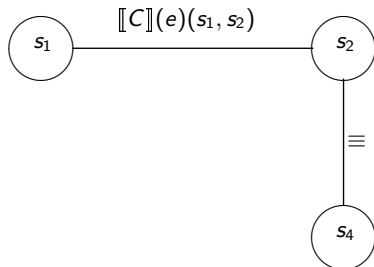


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**

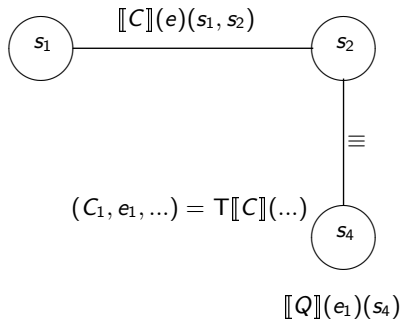


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**

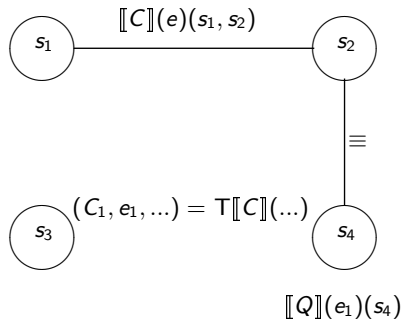


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**

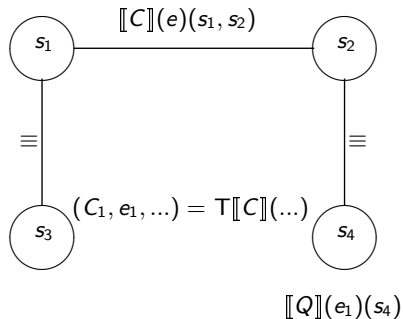


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**

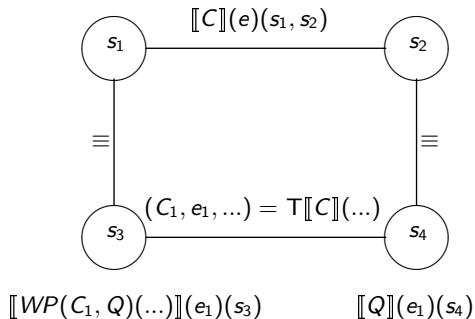


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**



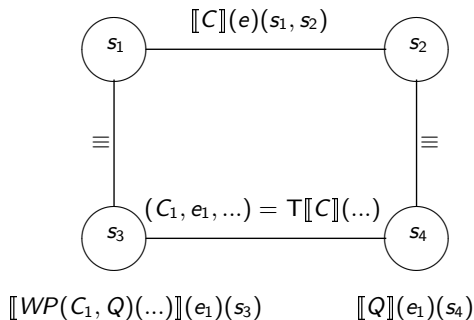


## Example translation function for *Command*

- ▶ **function signatures**

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

- ▶ **soundness statement for command-translation**



$$\begin{aligned}
 &\forall C \in \text{Command}, C_1, P, Q \in Exp_w, \\
 &s_1, s_2 \in State_m, s_3, s_4 \in State_w, \\
 &e \in Env_m, e_1 \in Env_w \dots : \\
 &(C_1, e_1, \dots) = T \llbracket C \rrbracket(e, \dots) \wedge P = WP(C_1, Q)(e_1, \dots) \wedge \\
 &s_1 \equiv s_3 \wedge s_2 \equiv s_4 \\
 &\Rightarrow \llbracket C \rrbracket(e)(s_1, s_2) \wedge \llbracket P \rrbracket(e_1)(s_3) \Rightarrow \llbracket Q \rrbracket(e_1)(s_4)
 \end{aligned}$$

## 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>

### ► 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