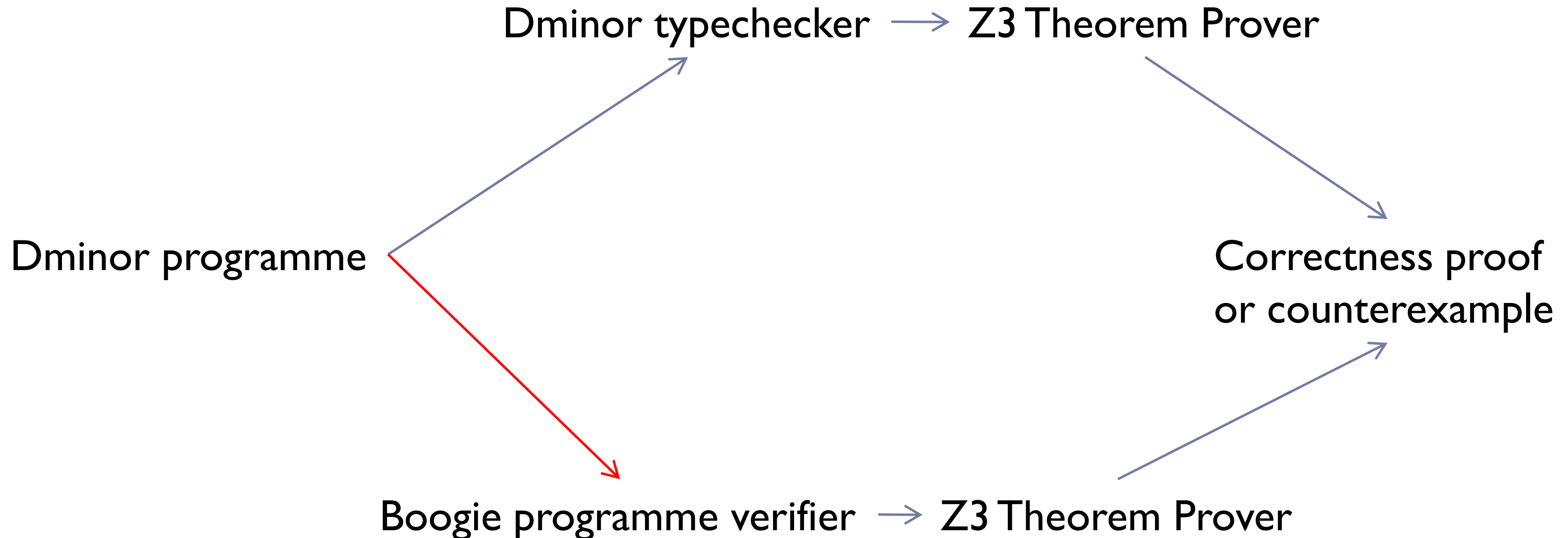
Automatically Verifying "M" Modelling Language Constraints

Supervisor: Prof. Dr. Michael Backes
Advisor: Cătălin Hrițcu, M.Sc.

What we are trying to do

We try to build a tool to verify the correctness of Dminor programmes



Outline

- Introduction of the languages involved
  - M / Dminor
  - Boogie
- How we translate Dminor to Boogie
  - Axiomatisation and translation
  - Demo
- Goals of the thesis
  - Implementation and proofs
  - Comparing verification with typechecking
  - Improvement over the existing typechecker

M / Dminor

- M is a modelling language under development by Microsoft
- First-order functional (stateless) language for manipulating tree-shaped data
- Dminor is a subset of M
  - Ambiguity in the grammar removed
  - Operational + denotational semantics is formalised
- Typechecked statically (unlike M)
  - Typechecker is bidirectional (strongest postcondition algorithm)
  - It is sound but incomplete with respect to declarative type system
- Rich refinement types
  - Subtying checked by an SMT-Solver

```
factorial(n:Integer32 where value > -1) : Integer32 where value > 0
    {
        (n==0)? 1 : (let n2 = (factorial(n - 1)) in n * n2)
    }
```

Dminor sample

```
module Constraints
{
  type Person : { Name:Text; Age:Integer32; };
  type EligiblePerson : Person where value.Age > 17;
  type Marriage : { SpouseA: EligiblePerson;
                    SpouseB: EligiblePerson; };

  PatChris(): Marriage
  {
      {SpouseA => {Name => "Pat", Age => 24},
      SpouseB => {Name => "Chris", Age => 32}}
  }
}
```

Boogie

- Boogie intermediate language for programme verification
- Spec#, VCC translate code to Boogie
- Boogie has mathematical and imperative components
- Boogie generates verification conditions
  - Weakest precondition algorithm
  - Axioms and types are translated directly
- Boogie can infer loop invariants
  - Otherwise they need to be provided by hand

Boogie sample

```
procedure factorial(n:int) returns (o:int)
requires n >= 0;
ensures o > 0;
{
  if (n == 0)
  {
      o := 1;
  }
  else
  {
      call o := factorial(n-1);
      o := o*n;
  }
}
```

Outline

- Introduction of the languages involved
  - M / Dminor
  - Boogie
- How we translate Dminor to Boogie
  - Axiomatisation and translation
  - Demo
- Goals of the thesis
  - Implementation and proofs
  - Comparing verification with typechecking
  - Improvement over the existing typechecker

Translation

- We translate Dminor code into Boogie code
  - Translation will be proven sound
- A basic library with axioms was written by us
  - Contains values plus basic operations on them

```
type String;
type Value;
type VOption;
(...)
function E([String]VOption) returns (Value);

function of_V_Entity(Value) returns ([String]VOption);
axiom (forall e : [String]VOption :: of_V_Entity(E(e)) == e);
(...)
const emptyEntity : [String]VOption;
axiom (forall s : String :: emptyEntity[s] == NoValue);
```

Example translation: Strings

```
module Constraints
{
    type Person : { Name:Text; Age:Integer32; };
    type EligiblePerson : Person where value.Age > 17;
    type Marriage : { SpouseA: EligiblePerson;
                      SpouseB: EligiblePerson; };
```

```
const unique Name:String;
const unique Age:String;

const unique Pat:String;
const unique Chris:String;

const unique SpouseA:String;
const unique SpouseB:String;
```

Example translation: Types

```
module Constraints
{
    type Person : { Name:Text; Age:Integer32; };
    type EligiblePerson : Person where value.Age > 17;
    type Marriage : { SpouseA: EligiblePerson;
                      SpouseB: EligiblePerson; };
```

```
function Person(v:Value) returns (bool) {
  is_Entity(v) && has_field(v,Name) && has_field(v,Age) &&
  Text(dot(v,Name)) && Integer(dot(v,Age))
}

function EligiblePerson(v:Value) returns (bool) {
  Person(v) && (O_GT(dot(v,Age), v_Int(17)) == v_true)
}
```

Example translation: Entity creation

```
PatChris(): Marriage
  {
   {SpouseA => {Name => "Pat", Age => 24},
    SpouseB => {Name => "Chris", Age => 32}}
  }
```

```
var Spouse’:[String]VOption;
var Spouse:Value;
Spouse’ :=
  emptyEntity[Name:=SomeValue(v_Text(Pat))][Age:
  =SomeValue(v_Int(24))];
Spouse := E(Spouse’);
```

Outline

- Introduction of the languages involved
  - M / Dminor
  - Boogie
- How we translate Dminor to Boogie
  - Axiomatisation and translation
  - Demo
- Goals of the thesis
  - Implementation and proofs
  - Comparing verification with typechecking
  - Improvement over the existing typechecker

Outline

- Introduction of the languages involved
  - M / Dminor
  - Boogie
- How we translate Dminor to Boogie
  - Axiomatisation and translation
  - Demo
- Goals of the thesis
  - Implementation and proofs
  - Comparing verification with typechecking
  - Improvement over the existing typechecker

Goals of the thesis

1. Finish the implementation
   - Axiomatise operations on lists and maps
   - We can already translate a large set of samples
   - Performance tuning
     - Quantifier patterns
     - Efficiency vs. Precision
2. Prove the translation to be sound with respect to declarative type system
   - In the end we hope to provide a better alternative to the Dminor typechecker

Goals of the thesis: Comparison

Area	Verification approach (our)	Typechecking approach (Dminor)
Axiomatisation	Library.bpl	DminorFoundation
Error reporting	Path + possibly counterexamples	Counterexample
Backend	SMT-Solver (Z3)	SMT-Solver (Z3)
Formula discharged	One per procedure	Small ones for each subtyping
Loop invariants	Boogie can infer some	All have to be specified
Verification cond. generation	Weakest precondition	Bidirectional typechecking (similar to strongest postcondition)
Performance	?	?
Precision	?	?

Goals of the thesis: Comparison

```
 1 procedure factorial(n:int) returns (o:int)
 2 requires n >= 0;
 3 ensures o > 1;
 4 {
 5     if (n == 0)
 6     {
 7             o := 1;
 8     }
 9     else
10     {
11             call o := factorial(n-1);
12             o := o*n;
13     }
14 }

counterexample-trace.bpl(8,2): Error BP5003: A postcondition
might not hold at this return statement.
Execution trace:
    counterexample-trace.bpl(5,2): anon0
    counterexample-trace.bpl(7,5): anon3_Then
```

g

Goals of the thesis: Comparison

```
module M {

    type Person : closed { Name:Text; Age: Integer32 where
                                               value > 0; };
    type EligiblePerson : Person where value.Age > 17;

    foo (arg:Person) : EligiblePerson {
        arg
    }
}

Type-checking failed
Function definition foo failed to typecheck 181:12. Can't
convert arg to type EligiblePerson. For instance if
arg->{Name=>"Freshval!3"; Age=>1; } expression evaluates to
{Name=>"Freshval!3"; Age=>1; } that does not have type
EligiblePerson.
```

Goals of the thesis: Comparison

Area	Verification approach (our)	Typechecking approach (Dminor)
Axiomatisation	Library.bpl	DminorFoundation
Error reporting	Path + possibly counterexamples	Counterexample
Backend	SMT-Solver (Z3)	SMT-Solver (Z3)
Formula discharged	One per procedure	Small ones for each subtyping
Loop invariants	Boogie can infer some	All have to be specified
Verification cond. generation	Weakest precondition	Bidirectional typechecking (similar to strongest postcondition)
Performance	?	?
Precision	?	?

Improve over the existing typechecker

- Accept correct programmes Dminor currently rejects
- Possible require less loop invariant annotation
- Generate better error messages
  - Output traces that lead to the error
  - Generate counter-examples
- Support features of M not currently in Dminor
  - State
    - Extents
    - Mutable State (not yet in M)
  - Modules

Even more ideas / References

- Proving termination of Dminor expressions (for purity)
- Supporting concurrency
  - M runs in a database → highly concurrent

- References

  - Bierman, Gordon, Hrițcu, Langworthy: Semantic Subtyping with an SMT Solver (Unpublished)
  - Leino: This is Boogie 2 (2008)

Questions