ETH Zürich
Chair of Software Engineering

Automatic Verification of Advanced Object-Oriented Features: The AutoProof Approach

Julian Tschannen, Carlo A. Furia,
Martin Nordio, Bertrand Meyer
ETH Zürich, Switzerland

AutoProof

- Automatic verifier for Eiffel
- Integrated in EVE, the Eiffel Verification Environment

http://se.inf.ethz.ch/research/eve/

AutoProof

Prove | 8 Successful | 3 Failed | Filter:

Class	Feature	Information	Po...	Tim...
ACCOUNT	make	Verification successful.		0
ACCOUNT	deposit	Verification successful.		0
ACCOUNT	withdraw	Postcondition balance_decreased may fail.	41	0
ACCOUNT	transfer	Verification successful.		0
ACCOUNT	split_account	Verification successful.		0
ALGORITH...	sum_and_max	Loop invariant might not be maintained.		0
ALGORITH...	max_in_array	Verification successful.		0
APPLICATIO...	make	Check instruction may fail (unnamed assertion).	24	0
CONST	make	Verification successful.		0
CONST	evaluate	Verification successful.		0

Motivation

- Eiffel
  - Pure object-oriented language with advanced features
  - Built-in contracts
  - Large code-base equipped with contracts
- Boogie
  - Intermediate verification language
  - Allows high-level reasoning
  - Boogie is actively used by different projects

AutoProof advanced features

- Agents (function objects)
- **Exceptions**
- Generics
- **Polymorphic calls**
- Simple frame inference
- Simple purity inference

Eiffel's exception mechanism

- One exception handler per routine: the **rescue** clause
- Set **Retry** to **True** to recover and re-execute body

Eiffel

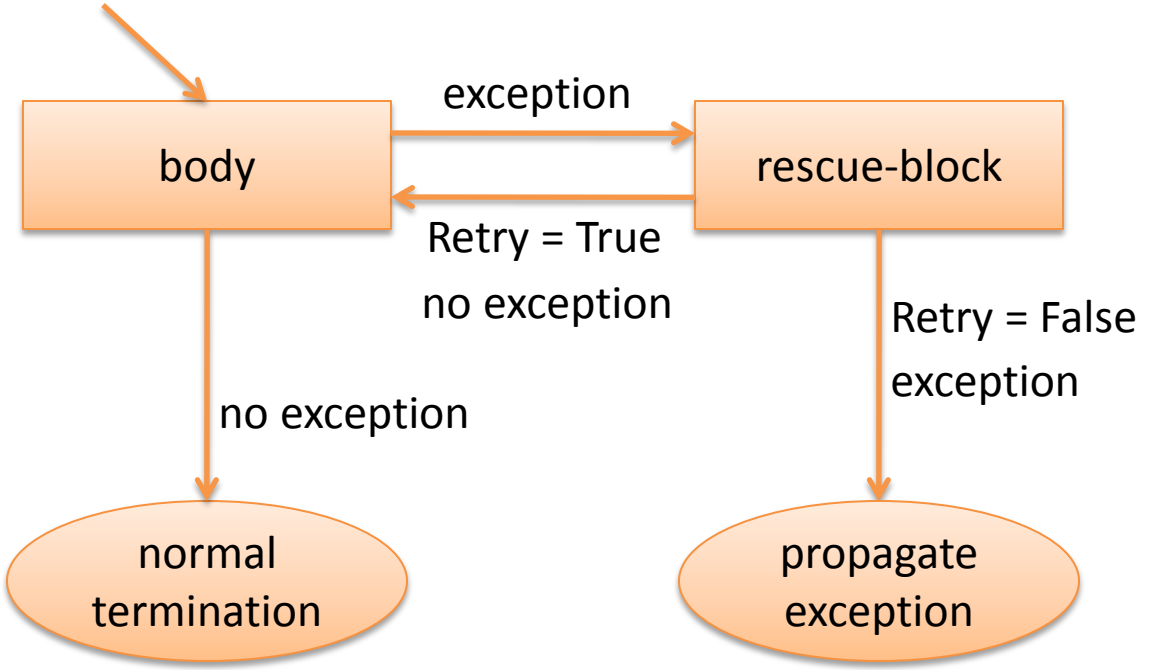
```
attempt_transmission (m: STRING)
   local
      failures: INTEGER
   do
      failed := False
      unsafe_transmit (m)
   rescue
      failures := failures + 1
      if failures < max_attempts then
         Retry := True
      else
         failed := True
      end
   end

failed: BOOLEAN
```

Initial values
failures = 0
Retry = **False**

Rescue loop

- Repeated execution of body and rescue clause is an *implicit loop* with two possible exits

```
Eiffel
do
   body
rescue
   rescue-block
end
```

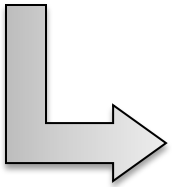


Specifying exceptional behavior

- Exception flag for conditional specification

Eiffel

```
attempt_transmission (m: STRING)
  ensure
    Exception implies failed
    not Exception implies not failed
  end
```



Boogie

```
var Exception: bool;

procedure unsafe_transmit (c: ref, m: ref);
  free requires Exception = false;
  modifies Exception, Heap;
  ensures Exception ==> Heap[current, failed];
  ensures !Exception ==> !Heap[current, failed];
```

Rescue invariants

Eiffel

```
attempt_transmission (m: STRING)
  local
    failures : INTEGER
  do
    failed := False
    unsafe_transmit (m)
  rescue
    failures := failures + 1
    if failures < max_attempts then
      Retry := True
    else
      failed := True
    end
  rescue invariant
    not Exception implies not failed
    (failures < max_attempts) implies not failed
  ensure
    Exception implies failed
    not Exception implies not failed
  end
```

Translating routines to Boogie

∇(*s*): translation of statement *s*

∇(*s*, *l*): translation of statement *s* followed by jump to label

Boogie

```
∇(s); if (Exception) { goto l; }
```

Eiffel

```
do
  body
rescue
  rescue-block
rescue invariant
  I_rescue
end
```

Boogie

```
  ∇(body, excLoopHead)
excLoopHead:
  while (Exception)
  invariant ∇(I_rescue);
  {
    Exception := false;
    Retry := false;
    ∇(rescue-block, excLoopEnd)
    if (!Retry) {
      Exception := true;
      goto excLoopEnd;
    }
    ∇(body, excLoopHead)
  }
excLoopEnd:
```

Polymorphism

Eiffel

```
deferred class EXP
feature
  last_value: INTEGER

  eval
    deferred
    ensure
      last_value >= 0
    end
end
```

Eiffel

```
class CONST
inherit EXP
feature

  value: INTEGER

  eval
    do
      last_value := value
    ensure then
      last_value = value
    end

end
```

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Translating routine signatures

- *Uninterpreted functions* for pre- and post-conditions

Boogie

```
function post.EXP.eval(h1, h2: HeapType; o: ref)
                                          returns (bool);
procedure EXP.eval(current: ref);
  free ensures post.EXP.eval(Heap, old(Heap), current);
  ensures Heap[current, last value] >= 0;
```

- Axioms to link predicates with actual contracts

Boogie

```
axiom (forall h1, h2: HeapType; o: ref ::
  $type(o) <: EXP ==>
  (post.EXP.eval(h1, h2, o) ==> (h1[o, last_value] >= 0)) );

axiom (forall h1, h2: HeapType; o: ref ::
  $type(o) <: CONST ==>
  (post.EXP.eval(h1, h2, o) ==>
                        (h1[o, last_value] = h1[o, value]) );
```

Polymorphic routine calls

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Boogie

```
implementation main (Current: ref) {
  var e: ref;
 entry:




}
```

Polymorphic routine calls

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Boogie

```
implementation main (Current: ref) {
  var e: ref;
 entry:

  havoc e;
  assume Heap[e, $allocated] = false;
  Heap[e, $allocated] := true;
  assume $type(e) = CONST;
  call CONST.make(e, 5);

}
```

Polymorphic routine calls

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Boogie

```
implementation main (Current: ref) {
  var e: ref;
 entry:

  havoc e;
  assume Heap[e, $allocated] = false;
  Heap[e, $allocated] := true;
  assume $type(e) = CONST;
  call CONST.make(e, 5);

  call EXP.eval(e);



}
```

Polymorphic routine calls

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Boogie

```
implementation main (Current: ref) {
  var e: ref;
 entry:

  havoc e;
  assume Heap[e, $allocated] = false;
  Heap[e, $allocated] := true;
  assume $type(e) = CONST;
  call CONST.make(e, 5);

  call EXP.eval(e);


  assert Heap[e, last_value] = 5;

}
```

Polymorphic routine calls

Eiffel

```
main
  local
    e: EXP
  do
    create {CONST} e.make (5)
    e.eval
    check e.last_value = 5 end
  end
```

Boogie

```
implementation main (Current: ref) {
  var e: ref;
 entry:

  havoc e;
  assume Heap[e, $allocated] = false;
  Heap[e, $allocated] := true;
  assume $type(e) = CONST;
  assume Heap[e, value] = 5;

  H1 := Heap;
  assume post.EXP.eval(Heap, H1, e);

  assert Heap[e, last_value] = 5;

}
```

Boogie

```
axiom (forall h1, h2: HeapType; o: ref ::
  $type(o) <: CONST ==>
  (post.EXP.eval(h1, h2, o) ==>
                        (h1[o, last_value] = h1[o, value]) );
```

Summary

- Automatic verifier for Eiffel integrated in EVE
- Translation for exception handling based on *exceptions flag* and *rescue invariants*
- Translation for dynamic contract selection of polymorphic calls based on *uninterpreted functions*

http://se.inf.ethz.ch/research/eve/