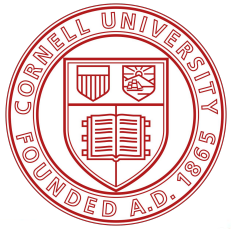

Evaluating Soundness of a Gradual Verifier with Property Based Testing

Jan-Paul Ramos-Dávila

Cornell University

Advised by: Jenna Wise, Jonathan Aldrich, Joshua Sunshine

Carnegie Mellon University

Overview
Why Gradual Verification?

Static Verification

```
int findMax(Node l)
{
  int m = l→val;
  Node curr = l→next;
  while(curr ≠ null) {
    if(curr→val > m) {
      m = curr→val;
    }
    curr = curr→next;
  }
  return m;
}
```

S3D

Static Verification: Does not support incrementality

```
int findMax(Node l)
  requires l ≠ NULL
  ensures max(result,l) && contains(result,l)
{
  int m = l→val;
  Node curr = l→next;
  FOLDS/UNFOLDS
  while(curr ≠ null) { LOOP INVARIANTS
    if(curr→val > m) {
      m = curr→val;
      FOLDS/UNFOLDS
      LEMMAS
    }
    FOLDS/UNFOLDS
    curr = curr→next;
  }
  return m;
```

S3D

Dynamic Verification

```
int findMax(Node l)
  ensures max(result,l) && contains(result,l)
{
  int m = l→val;
  Node curr = l→next;
  while(curr ≠ null) {
    if(curr→val > m) {
      m = curr→val;
    }
    curr = curr→next;
  }
  return m;
}
```

S3D

Dynamic Verification: Runtime overhead is too much

```
int findMax(Node l)
  ens
{
  int
  Nod
  whi
    i

    }
    c
  }
  ret
}
```

yarnpkg/berry

#1817 **Benchmark PnP runtime overhead**

43 comments

remorses opened on September 8, 2020

Gradual Verification supports **incrementality**.

Allows users to specify as much as they want.

Provides a formal guarantee of verifiability.

```
int findMax(Node l)
  requires ?
  ensures max(result,l) &&
  contains(result,l)
{
  int m = l→val;
  Node curr = l→next;
  while(curr ≠ NULL) ? {
    if(curr→val > m) {
      m = curr→val;
    }
    curr = curr→next;
  }

  return m;
}
```

Gradual Verification

```
int findMax(Node l)
  requires ? && l ≠ NULL
  ensures max(result,l) && contains(result,l)
{
  int m = l→val;
  Node curr = l→next;
  while(curr ≠ NULL) ? && LOOP INVARIANTS {
    if(curr→val > m) {
      m = curr→val;
    }
    curr = curr→next;
  }

  return m;
}
```

S3D

Gradual Verification: Formal guarantee of verifiability

```
int findMax(Node l)
  requires ? && l ≠ NULL
  ensures max(result,l) && contains(result,l)
{
  int m = l→val;
  Node curr = l→next;
  FOLDS/UNFOLDS
  while(curr ≠ NULL) FOLDS/UNFOLDS {
    if(curr→val > m) {
      m = curr→val;
      FOLDS/UNFOLDS
      LEMMAS
    }
    FOLDS/UNFOLDS
    curr = curr→next;
  }

  return m;
}
```

S3D

Gradual Guarantee: Verifiability and reducibility are **monotone with respect to precision**.

Conservative extension: Anything provable in the statically should be provable in the gradually.

Gradual C0: Design has been proven sound.

Extended with Gradual Formulas

Static Verifier with Implicit Dynamic Frames and recursive Abstract predicates

Gradual Guarantee: Verifiability and reducibility are **monotone with respect to precision**.

Conservative extension: Anything provable in the statically should be provable in the gradually.

Gradual C0: Design has been proven sound.

Extended with Gradual Formulas

Static Verifier with Implicit Dynamic Frames and recursive Abstract predicates

Lifting (Garcia et al. '16)

Optimistic Static Verification System

Gradual Guarantee: Verifiability and reducibility are **monotone with respect to precision**.

Conservative extension: Anything provable in the statically should be provable in the gradually.

Gradual C0: Design has been proven sound.

Extended with Gradual Formulas

Static Verifier with Implicit Dynamic Frames and recursive Abstract predicates

Lifting (Garcia et al. '16)

Optimistic Static Verification System

Dynamic Verification System

Goal

Ensure Gradual C0's implementation is sound and allow for scalable bug fixes.

S3D

Why Property Based Testing?

A number of bugs had been caught by hand, in which *Gradual C0*'s design was unsound

S3D

Why Property Based Testing?

A number of bugs had been **caught by hand**, in which *Gradual C0*'s design was unsound

There are no **lightweight** techniques available

S3D

Why Property Based Testing?

A number of bugs had been **caught by hand**, in which *Gradual C0*'s design was unsound

There are no **lightweight** techniques available

Capturing the *truthiness* of a property's result provides **good coverage** for finding these implementation bugs.

S3D

Three-stage Pipeline

Reference model language
Uses Gradual C0's specification language

S3D

Three-stage Pipeline

Reference model language
Uses Gradual C0's specification language

WIP: Input Generator
Test suite of examples that are not supposed to verify correctly

S3D

Three-stage Pipeline

Reference model language
Uses Gradual C0's specification language

WIP: Input Generator
Test suite of examples that are not supposed to verify correctly

Checker: Dynamic C0
Asserts runtime checks everywhere.
The ground truth

Checker: Gradual C0

S3D

Checker Architecture

Input
C0 program **p**

Intermediate Representation for *Gradual* **p**

Gradually Verify *Gradual* **p**

Compile & Exec *Gradual* **p**

success String **S1** OR **error** String **S1**

Wait for **S2** output

Wait for **S1** output

IR for *Dynamic* **p**

Dynamically Verify *Dynamic* **p**

success String **S2** OR **error** String **S2**

Output
if **S1** ≠ **S2** and
S1 = **success**
then : **error**
else : **success**

S3D

By Example
Evaluating Soundness of Gradual C0 using PBT

S3D

To prevent trivial failure, we must **avoid specifying preconditions and folds/unfolds** that won't be met while running

The truthiness for all programs consists of a pair of file executions

Binary Search Tree breaking order:
`left node > right node`

```
-     if (x < v) {
+     if (v < x) {
        if (l ≠ NULL) {
          root→left = tree_add_helper(l,
x, min, v-1);
        } else {
          root→left =
create_tree_helper(x, min, v-1);
        }
      } else {
-       if (v < x) {
+       if (x < v) {
          if (r ≠ NULL) {
            root→right =
tree_add_helper(r, x, v+1, max);
          } else {
            root→right =
create_tree_helper(x, v+1, max);
          }
        }
    }
```

Input Generator Architecture

We caught **4 soundness bugs** at different implementation phases of *Gradual C0*

```
/*@
predicate list(struct Node *l) =
  ? && (l != NULL ? acc(l->value)
&& list(l->next) : true);
@*/
void append(Node *root, int value)
  //@requires ?;
  //@ensures ? && list(root);
  {
    Node *n = root;
    while (n->next != NULL)
      //@loop_invariant ?;
      n = n->next;
    n->next = alloc(Node);
    n->next->value = value;
}
```

S3D

Future Work

Carnegie Mellon University
School of Computer Science

S3D

Software and Societal
Systems Department

Check out our website to
learn more about our work:
S3D.cmu.edu