http://ie.technion.ac.il/~ofers/frontend/

Regression Verification for unbalanced recursive functions

OFER STRICHMAN MAOR VEITSMAN

TECHNION, HAIFA, ISRAEL

Submitted to FM16'

Regression Verification

Develop a method for formally verifying the equivalence of two similar programs.

Selling points:

- Specification: not needed
- Complexity: depends on the semantic difference between the programs, and not on their size.

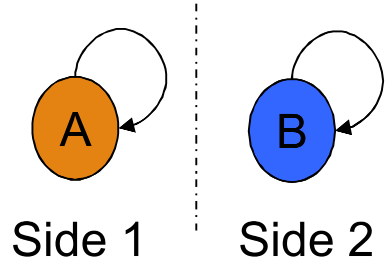
Partial Equivalence

- There are many definitions of equivalence.
- We will focus on partial equivalence:
- Executions of P1 and P2 on equal inputs
  - …which terminate,
  - result in equal outputs.

- Undecidable

Partial Equivalence for Recursive Functions

Consider the call graphs:



- … where A,B have:
  - same prototype
  - no loops

Prove partial equivalence of A, B

- How shall we handle the recursion ?

Hoare's Rule for Recursion

Let A be a recursive function.

$$\frac{\{p\}\ \texttt{call}\ A\ \{q\}\ \vdash\ \{p\}\ A\ \texttt{body}\ \{q\}}{\{p\}\ \texttt{call}\ A\ \{q\}}\ \text{(REC)}$$

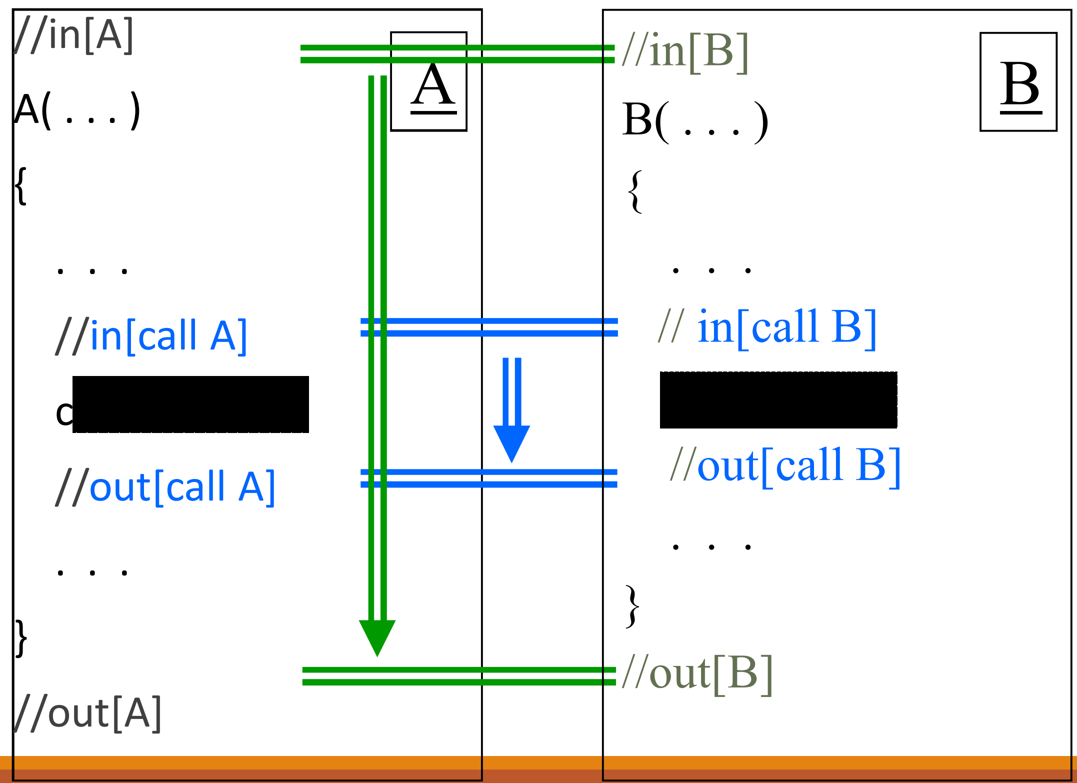
Hoare's Rule for Recursion

$$\frac{\{p\}\ \texttt{call}\ A\ \{q\}\ \vdash\ \{p\}\ A\ \texttt{body}\ \{q\}}{\{p\}\ \texttt{call}\ A\ \{q\}}\ \text{(REC)}$$

// {p}
A(. . .)
{
    . . .
    // {p}
    [redacted]
    // {q}
    . . .
}
// {q}

Proving Partial Equivalence

$$\frac{\text{partial-equiv}(\textbf{call}\ A, \textbf{call}\ B) \vdash \text{partial-equiv}(A\ \textbf{body}, B\ \textbf{body})}{\text{partial-equiv}(\textbf{call}\ A, \textbf{call}\ B)}\ (\text{PART-EQ-1})$$



Proving Partial Equivalence

$$\frac{\text{partial-equiv}(\textbf{call } A, \textbf{call } B) \vdash \text{partial-equiv}(A \textbf{ body}, B \textbf{ body})}{\text{partial-equiv}(\textbf{call } A, \textbf{call } B)} \quad (\text{PART-EQ-1})$$

Q: How can a verification condition for the premise look like?

A: Replace the recursive calls with calls to functions that

- over-approximate A, B, and
- are partially equivalent by construction

Natural candidates: Uninterpreted Functions

Proving Partial Equivalence

Let A^{UF}, B^{UF} be A,B, after replacing the recursive call with a call to (the same) uninterpreted function.

We can now rewrite the rule:

The premise is decidable

$$\frac{\text{partial-equiv}(A^{UF}, B^{UF})}{\text{partial-equiv}(A, B)} \quad \text{(PART-EQ-1)}$$

What (PART-EQ) cannot prove (1)

Calling under different base conditions:

```
int fact1(int n){
        if (n <= 1) return 1;
        return n * fact_uf(n-1);
}
```

```
int fact2(int n){
        if (n <= 0) return 1;
        return n * fact_uf(n-1);
}
```

when $n = 1$:

returns 1

returns $1 * nondet()$

The verification condition

```
int main(){
        int n = non_det();  // suppose n = 1
        int ret1, ret2;
        ret1 = fact1(n);  // returns 1
        ret2 = fact2(n);  // returns nondet
        assert(ret1 = ret2);  // fails !
}
```

- We check this program with a bounded model checker (i.e. CBMC).

What (PART-EQ) cannot prove (2)

Unbalanced recursive functions lead to function calls with different arguments:

```
int sum1(int n){
 if (n <= 1){
  return n;
 }
 return n + n-1 +sum1(n-2);
}
```

```
int sum2(int n){
 if (n <= 1){
  return n;
 }
 return n + sum2(n-1);
}
```

returns n + n -1 + nondet()

returns n + nondet()

Our strategy

1. For the same input: f invokes base-case, g does not.

Therefor, we will prove equivalence separately for:

- Inputs that invoke the base-case in *at-least* one of f, g
- All the rest

2. f, g are not in lock-step!

Therefor: unroll them separately to their least-common multiplier.

- But: Unrolling changes what we mean by base-case. Previous solution must be adapted.

New Proof Rule

Our new proof rule contains two premises:

- Base cases are equivalent: $base{-}equiv(f, g)$
- Step is equivalent: $step{-}equiv(f, g)$

$$\frac{base{-}equiv(f, g) \qquad step{-}equiv(f, g)}{partial{-}equiv(f, g)}$$

$base{-}equiv(f, g)$

- Let in_B be the set of inputs driving f **or** g to a base case.
- Let $partial{-}equiv(f, g)|_{in_B}$ be partial-equivalence under inputs in_B.
- We now define:

$$base{-}equiv(f, g) \doteq partial{-}equiv(f, g)\Big|_{in_B}$$

$step{-}equiv(f, g)$

- Let in be the full set of possible inputs.
- Let in_S = $in \ - in_B$
  - (the set of inputs **not** driving f **or** g to a base case).
- We now define:

$$step{-}equiv(f, g) = partial{-}equiv(f, g)\Big|_{in_S}$$

Non Balanced Recursive Step - Solution

- We perform unbalanced unrolling.
- By applying $unroll(sum2, 1)$ we get:

```
int sum1(int n){
 if (n <= 1){
  return n;
 }
 return n + n-1 + uf_sum(n-2);
}
```

8 + 7 + uf_sum(6)

$n = 8$

```
int sum2_1(int n){
 if (n <= 1){
  return n;
 }
 return n + uf_sum(n-1);
}
```

7 + uf_sum(6)

```
int sum2(int n){
 if (n <= 1){
  return n;
 }
 return n + sum2_1(n-1);
}
```

8

Non Balanced Recursive Step - Problem

- For $n = 2$:

```
int sum1(int n){
 if (n <= 1){
  return n;
 }
 return n + n-1 + uf_sum(n-2);
}
```

returns $\mathbf{2 + 1 +} \boldsymbol{nondet}\mathbf{()}$

```
int sum2_1(int n){
 if (n <= 1){
  return n;
 }
 return n + uf_sum(n-1);
}

int sum2(int n){
 if (n <= 1){
  return n;
 }
 return n + sum2_1(n-1);
}
```

returns $\mathbf{2 + 1}$

Proof Rule for Unbalanced Recursion Base Cases

- Let us define now the proof rule for unbalanced recursions:

$$\frac{base\text{-}equiv_{n,m}(f,g) \qquad step\text{-}equiv_{n,m}(f,g)}{partial\text{-}equiv(f,g)}$$

- n, m: unrolling factors for sides 1 and 2, respectively.

Can software verifiers prove equivalence?

- Seahorn [GKKN'15]
  - Based on Horn-clauses representation of the program and rules
  - Invariants are searched-for with μZ (PDR-based)
- HSF [GGLPR'12]
  - Based on Horn-clauses representation of the program and rules
  - based on predicate abstraction and refinement using CEGAR
- REVE [FGKRU'14]
  - Based on Horn-clauses representation of the program and uninterpreted predicates.

Questions?