Verifying Array Manipulating Program with Full-Program Induction

Supratik Chakraborty[1], Ashutosh Gupta[1], Divyesh Unadkat[1,2]

Indian Institute of Technology Bombay[1]
TCS Research[2]

TACAS 2020

Verify Properties of Programs with Arrays

- Arrays of parametric size N
- Compute values data dependent on values from previous iterations
- No trivial translation of loops to parallel assignments
- Quantified as well as quantifier-free properties, with possibly non-linear terms

Does $\{\varphi_N\}\ \mathsf{P}_N\ \{\psi_N\}$ hold?

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Challenges Faced by State-of-the-Art Tools & Techniques

Challenges Faced by State-of-the-Art Tools & Techniques

- Quantified invariants with non-linear terms difficult to synthesize
  - Loop invariants required by the respective loops in the program:
  - $\forall i \in [0...x\text{-}1]\ (A[i] = 6i + 6)$
  - $\forall j \in [0...y\text{-}1]\ (B[j] = 3j^2 + 3j + 1 \wedge A[j] = 6j + 6)$
  - $\forall k \in [0...z\text{-}1]\ (C[k] = k^3 \wedge B[k] = 3k^2 + 3k + 1)$
  - **FreqHorn**[CAV'19], **Tiler**[SAS'17]

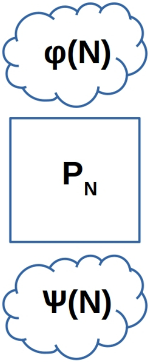
Challenges Faced by State-of-the-Art Tools & Techniques

- Quantified invariants with non-linear terms difficult to synthesize
  - Loop invariants required by the respective loops in the program:
  - $\forall i \in [0...x\text{-}1]\ (A[i] = 6i + 6)$
  - $\forall j \in [0...y\text{-}1]\ (B[j] = 3j^2 + 3j + 1 \wedge A[j] = 6j + 6)$
  - $\forall k \in [0...z\text{-}1]\ (C[k] = k^3 \wedge B[k] = 3k^2 + 3k + 1)$
  - **FreqHorn**[CAV'19], **Tiler**[SAS'17]
- Abstraction-based techniques are imprecise in presence of data dependence across loop iterations
  - **VeriAbs**[ASE'19], **Vaphor**[SAS'16]

Challenges Faced by State-of-the-Art Tools & Techniques

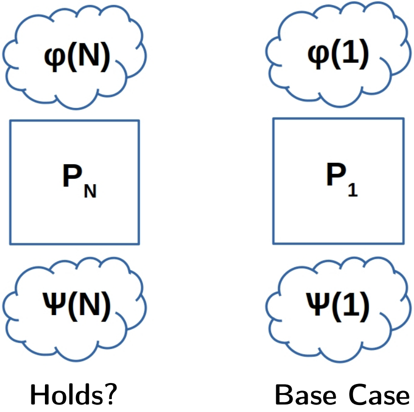
- Quantified invariants with non-linear terms difficult to synthesize
  - Loop invariants required by the respective loops in the program:
  - $\forall i \in [0 \ldots x\text{-}1]\ (A[i] = 6i + 6)$
  - $\forall j \in [0 \ldots y\text{-}1]\ (B[j] = 3j^2 + 3j + 1 \land A[j] = 6j + 6)$
  - $\forall k \in [0 \ldots z\text{-}1]\ (C[k] = k^3 \land B[k] = 3k^2 + 3k + 1)$
  - **FreqHorn**[CAV'19], **Tiler**[SAS'17]
- Abstraction-based techniques are imprecise in presence of data dependence across loop iterations
  - **VeriAbs**[ASE'19], **Vaphor**[SAS'16]
- Difficult to solve (non-linear) recurrences when data flows across loops and loop iterations as well as difficult to find fix-points
  - **VIAP**[VSTTE'18], **Booster**[ATVA'14]

Full-Program Induction - Pictorial Overview

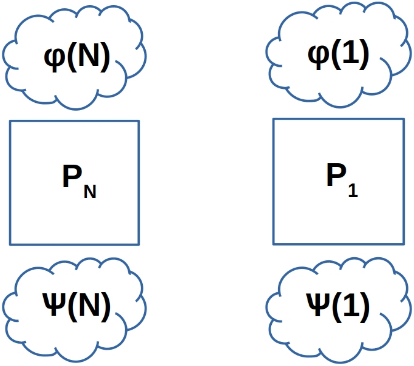


Holds?

Full-Program Induction - Pictorial Overview



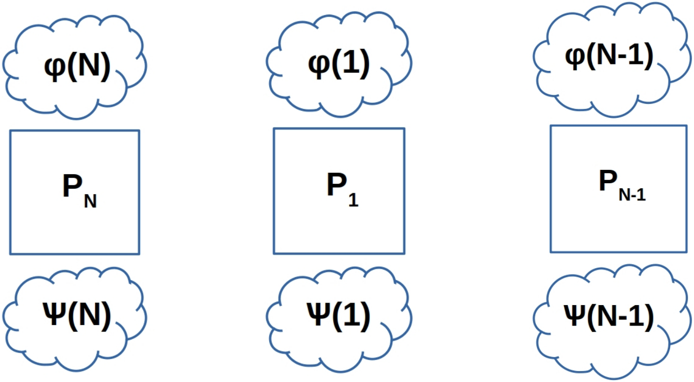
Full-Program Induction - Pictorial Overview



Holds? Base Case

✗ Property Violation

Full-Program Induction - Pictorial Overview



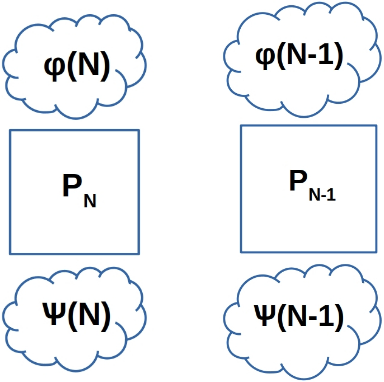
Holds?

Base Case

✗ Property Violation

Induction Hypothesis

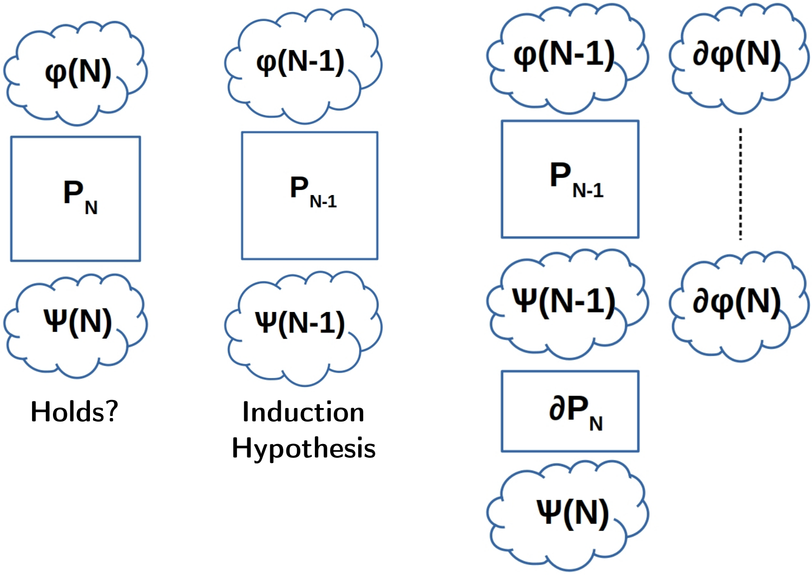
Full-Program Induction - Pictorial Overview



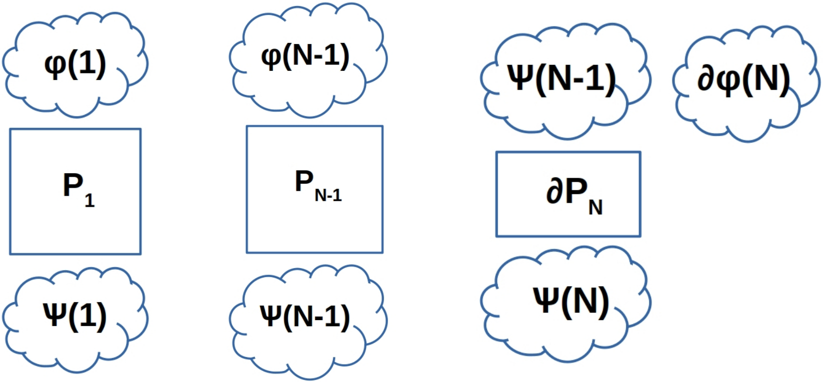
Holds?

Induction Hypothesis

Full-Program Induction - Pictorial Overview

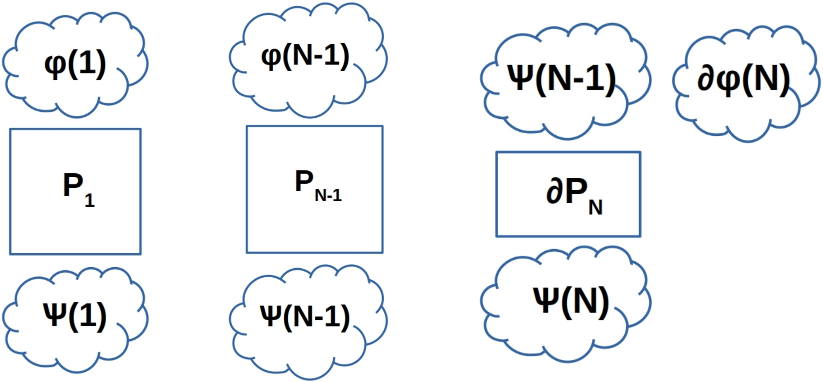


Full-Program Induction - Pictorial Overview



Base Case **Induction Hypothesis** **Inductive Step**

Full-Program Induction - Pictorial Overview



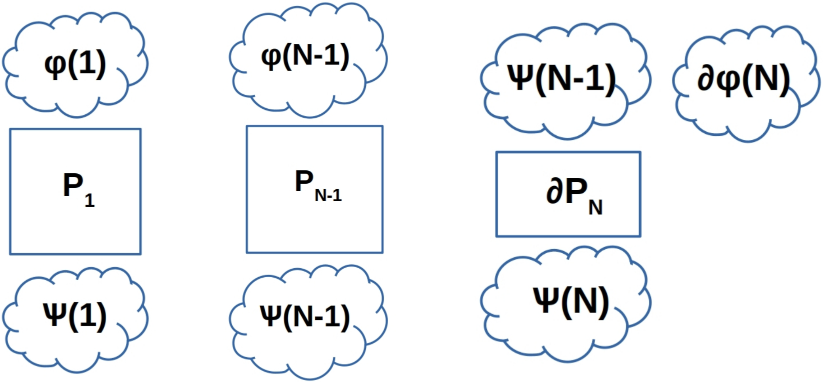
Base Case

Induction Hypothesis

Inductive Step

✓ Proved

Full-Program Induction - Pictorial Overview



Base Case

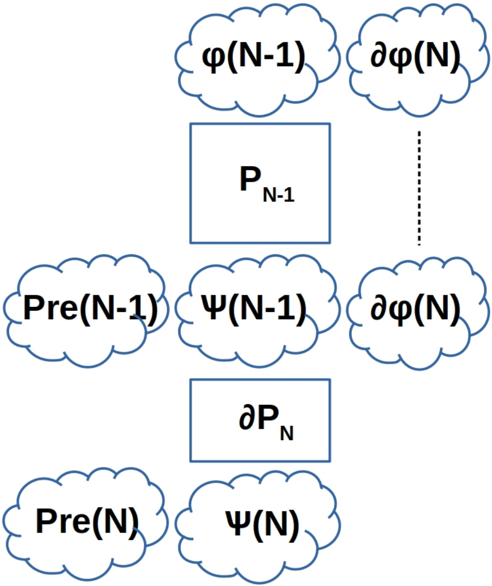
Induction Hypothesis

Inductive Step

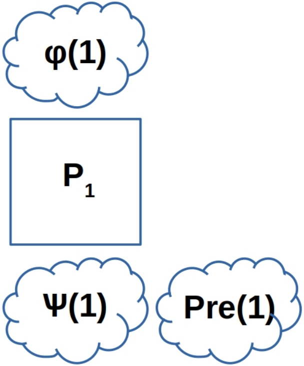
✓ Proved

✗ Infer New Sub-goals

Full-Program Induction - Pictorial Overview

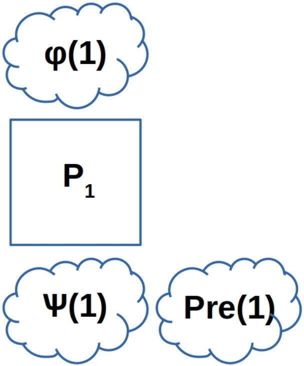


Full-Program Induction - Pictorial Overview



Base Case

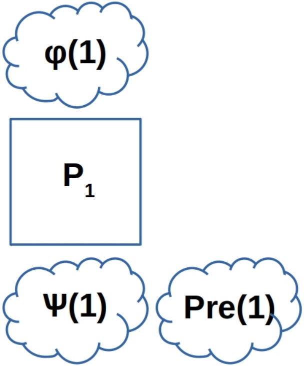
Full-Program Induction - Pictorial Overview



Base Case

✗ **Infer New Pre'(N)**
or

Full-Program Induction - Pictorial Overview



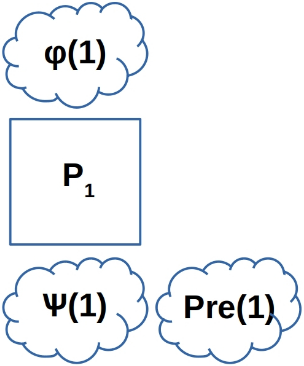
Base Case

✗ **Infer New Pre'(N)**

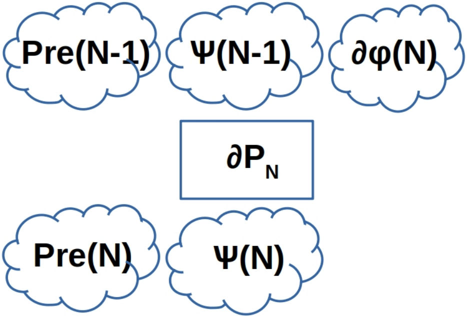
or

Split Pre(N) and/or $\partial\varphi(N)$

Full-Program Induction - Pictorial Overview



Base Case



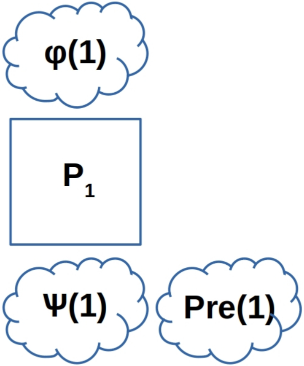
Inductive Step

✗ **Infer New Pre'(N)**

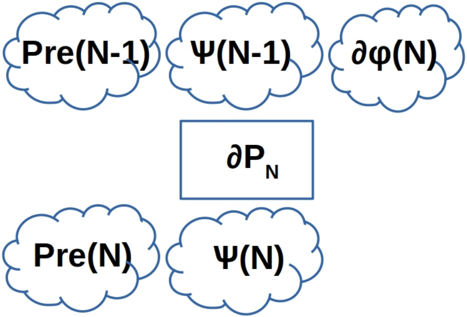
or

Split Pre(N) and/or $\partial\varphi(N)$

Full-Program Induction - Pictorial Overview



Base Case

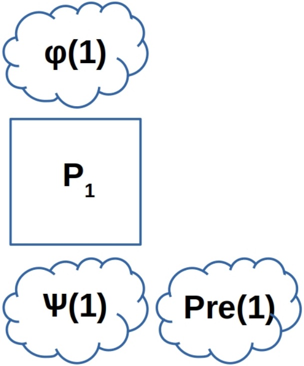


Inductive Step

✗ **Infer New Pre'(N)**
or
Split Pre(N) and/or $\partial\varphi(N)$

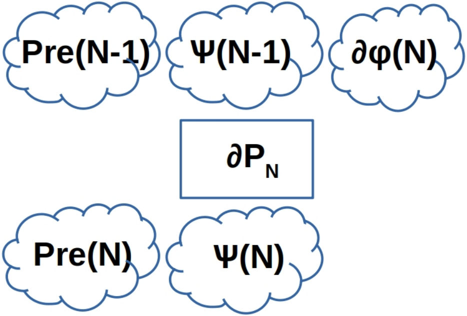
✓ **Proved**

Full-Program Induction - Pictorial Overview



Base Case

✗ Infer New Pre'(N)
or
Split Pre(N) and/or $\partial\varphi(N)$

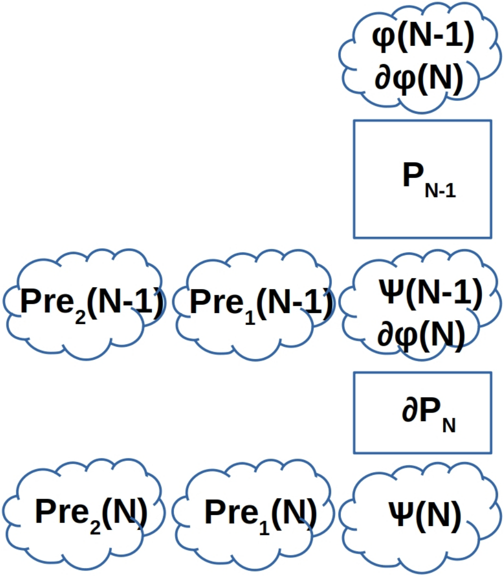


Inductive Step

✓ Proved

✗ Infer New Sub-goals

Full-Program Induction - Pictorial Overview



Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Base Case: Substitute N=1

```
assume(true);

void PolyCompute(int N) {
  int A[1], B[1], C[1];
  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<1; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<1; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<1; z++)
    C[z] = C[z-1] + B[z-1];
}

assert(∀k∈[0,1),C[k]==k^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Inductive Step

```
assume(∀k∈[0,N-1),C[k]==k^3);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(∀k∈[0,N),C[k]==k^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Inductive Step

```
assume(∀k∈[0,N-1),C[k]==k^3);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Inferred Pre_1

```
assume(B[N-2]==(N-1)^3-(N-2)^3);
assume(∀k∈[0,N-1),C[k]==k^3);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}$ P_N $\{\psi(N)\}$

```
assume(true);

1. void PolyCompute(int N) {

2.    int A[N], B[N], C[N];

3.    A[0]=6;  B[0]=1;  C[0]=0;

4.    for (int x=1; x<N; x++)
5.      A[x] = A[x-1] + 6;

6.    for (int y=1; y<N; y++)
7.      B[y] = B[y-1] + A[y-1];

8.    for (int z=1; z<N; z++)
9.      C[z] = C[z-1] + B[z-1];

10. }

assert(∀k∈[0,N), C[k] == k³);
```

Quantify Inferred Pre_1

```
assume(∀j∈[0,N-1),B[j]==(j+1)³-j³);
assume(∀k∈[0,N-1),C[k]==k³);

1.  A[N-1] = A[N-2] + 6;

2.  B[N-1] = B[N-2] + A[N-2];

3.  C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)³);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ P_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k³);
```

Base Case: Substitute N=1

```
assume(true);

void PolyCompute(int N) {
  int A[1], B[1], C[1];
  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<1; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<1; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<1; z++)
    C[z] = C[z-1] + B[z-1];
}

assert(∀k∈[0,1),C[k]==k³);
assert(∀j∈[0,1),B[j]==(j+1)³-j³);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

1. void PolyCompute(int N) {

2.   int A[N], B[N], C[N];

3.   A[0]=6;  B[0]=1;  C[0]=0;

4.   for (int x=1; x<N; x++)
5.     A[x] = A[x-1] + 6;

6.   for (int y=1; y<N; y++)
7.     B[y] = B[y-1] + A[y-1];

8.   for (int z=1; z<N; z++)
9.     C[z] = C[z-1] + B[z-1];

10. }

assert(∀k∈[0,N), C[k] == k^3);
```

Inductive Step

```
assume(∀j∈[0,N-1),B[j]==(j+1)^3-j^3);
assume(∀k∈[0,N-1),C[k]==k^3);

1.  A[N-1] = A[N-2] + 6;

2.  B[N-1] = B[N-2] + A[N-2];

3.  C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
assert(B[N-1]==N^3-(N-1)^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Inferred Pre_2

```
assume(A[N-2]==N^3-2*(N-1)^3+(N-2)^3);
assume(∀j∈[0,N-1),B[j]==(j+1)^3-j^3);
assume(∀k∈[0,N-1),C[k]==k^3);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
assert(B[N-1]==N^3-(N-1)^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Quantify Inferred Pre_2

```
assume(∀i∈[0,N-1),A[i]==(i+2)^3-2*(i+1)^3+i^3);
assume(∀j∈[0,N-1),B[j]==(j+1)^3-j^3);
assume(∀k∈[0,N-1),C[k]==k^3);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
assert(B[N-1]==N^3-(N-1)^3);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

1. void PolyCompute(int N) {

2.   int A[N], B[N], C[N];

3.   A[0]=6;  B[0]=1;  C[0]=0;

4.   for (int x=1; x<N; x++)
5.     A[x] = A[x-1] + 6;

6.   for (int y=1; y<N; y++)
7.     B[y] = B[y-1] + A[y-1];

8.   for (int z=1; z<N; z++)
9.     C[z] = C[z-1] + B[z-1];

10. }

assert(∀k∈[0,N), C[k] == k³);
```

Base case: Substitute N=1

```
assume(true);

1. void PolyCompute(int N) {
2.   int A[1], B[1], C[1];
3.   A[0]=6;  B[0]=1;  C[0]=0;

4.   for (int x=1; x<1; x++)
5.     A[x] = A[x-1] + 6;

6.   for (int y=1; y<1; y++)
7.     B[y] = B[y-1] + A[y-1];

8.   for (int z=1; z<1; z++)
9.     C[z] = C[z-1] + B[z-1];
10. }

assert(∀k∈[0,1),C[k]==k³);
assert(∀j∈[0,1),B[j]==(j+1)³-j³);
assert(∀i∈[0,1),A[i]==(i+2)³-2*(i+1)³+i³);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k³);
```

Inductive Step

```
assume(∀i∈[0,N-1),A[i]==(i+2)³-2*(i+1)³+i³);
assume(∀j∈[0,N-1),B[j]==(j+1)³-j³);
assume(∀k∈[0,N-1),C[k]==k³);

A[N-1] = A[N-2] + 6;

B[N-1] = B[N-2] + A[N-2];

C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)³);
assert(B[N-1]==N³-(N-1)³);
assert(A[N-1]==(N+1)³-2*N³+(N-1)³);
```

Full-Program Induction - Concrete Example

Verify $\{\varphi(N)\}$ P_N $\{\psi(N)\}$

```
assume(true);

void PolyCompute(int N) {

  int A[N], B[N], C[N];

  A[0]=6;  B[0]=1;  C[0]=0;

  for (int x=1; x<N; x++)
    A[x] = A[x-1] + 6;

  for (int y=1; y<N; y++)
    B[y] = B[y-1] + A[y-1];

  for (int z=1; z<N; z++)
    C[z] = C[z-1] + B[z-1];

}

assert(∀k∈[0,N), C[k] == k^3);
```

Eliminate Quantifiers in Pre

```
assume(A[N-2]==N^3-2*(N-1)^3+(N-2)^3);
assume(B[N-2]=(N-1)^3-(N-2)^3);
assume(C[N-2]=(N-2)^3);

  A[N-1] = A[N-2] + 6;

  B[N-1] = B[N-2] + A[N-2];

  C[N-1] = C[N-2] + B[N-2];

assert(C[N-1]==(N-1)^3);
assert(B[N-1]==N^3-(N-1)^3);
assert(A[N-1]==(N+1)^3-2*N^3+(N-1)^3);
```

Validity proved by Z3

Computing the "Difference" Pre-Condition - $\partial\varphi(N)$

- Need to compute $\partial\varphi(N)$ such that
  (a) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$ holds
  (b) $\partial\varphi(N)$ does not refer to scalars and array elements modified in P_{N-1}

Computing the "Difference" Pre-Condition - $\partial\varphi(N)$

- Need to compute $\partial\varphi(N)$ such that
  (a) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$ holds
  (b) $\partial\varphi(N)$ does not refer to scalars and array elements modified in P_{N-1}
- Test for existence of $\partial\varphi(N)$
  - Validity of $\varphi(N) \rightarrow \varphi(N-1)$
  - Difference cannot be computed if above formula is invalid!

Computing the "Difference" Pre-Condition - $\partial\varphi(N)$

- Need to compute $\partial\varphi(N)$ such that
  (a) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$ holds
  (b) $\partial\varphi(N)$ does not refer to scalars and array elements modified in P_{N-1}

- Test for existence of $\partial\varphi(N)$
  - Validity of $\varphi(N) \rightarrow \varphi(N-1)$
  - Difference cannot be computed if above formula is invalid!

- Computed based on the pattern of $\varphi(N)$
  - **If** $\varphi(N) := \forall i\ (0 \leq i \leq N) \rightarrow \widehat{\varphi(i)}$ **then** $\partial\varphi(N) := \widehat{\varphi(N)}$
    - ★ $\varphi(N) := \forall i\ (0 \leq i \leq N) \rightarrow A[i] > 0$ $\quad\quad \partial\varphi(N) := A[N] > 0$
  - **If** $\varphi(N) := \varphi^1(N) \wedge \cdots \wedge \varphi^k(N)$ **then**
    $\partial\varphi(N) := \partial\varphi^1(N) \wedge \cdots \wedge \partial\varphi^k(N)$
  - Otherwise $\partial\varphi(N) :=$ True

Computing the "Difference" Program - ∂P_N

- ∂P_N := PeelLoops(P_N);
- Replace assignments in the peeled loops with "difference" statements
  - `A[i] = C;`

    is replaced with

    `A[i] = A_Nm1[i] + (C - C) ;`
  - `A[i] = B[i] + v;`

    is replaced with

    `A[i] = A_Nm1[i] + (B[i] - B_Nm1[i]) + (v - v_Nm1);`
- "Simplify" generated difference terms, "Accelerate" loops
- Remove loops that simply copy values from N-1^{th} to N^{th} version

Soundness Guarantee

Theorem

Soundness Guarantee

Theorem

Suppose

1) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\} \iff \{\varphi(N)\}\ \mathsf{P}_{N-1}; \partial\mathsf{P}_N\ \{\psi(N)\}$

Soundness Guarantee

Theorem

Suppose

1) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\} \iff \{\varphi(N)\}\ \mathsf{P}_{N-1}; \partial\mathsf{P}_N\ \{\psi(N)\}$

2) *Formula* $\partial\varphi(N)$ *exists such that*
   (a) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$
   (b) $\{\partial\varphi(N)\}\ \mathsf{P}_{N-1}\ \{\partial\varphi(N)\}$

Soundness Guarantee

Theorem

Suppose

1) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\} \iff \{\varphi(N)\}\ \mathsf{P}_{N-1}; \partial\mathsf{P}_N\ \{\psi(N)\}$
2) *Formula* $\partial\varphi(N)$ *exists such that*
   (a) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$
   (b) $\{\partial\varphi(N)\}\ \mathsf{P}_{N-1}\ \{\partial\varphi(N)\}$
3) *Formula* $\mathsf{Pre}(M)$ *exists such that for* $M \geq 1$
   (a) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$ *for* $0 < N \leq M$
   (b) $\{\varphi(M)\}\ \mathsf{P}_M\ \{\psi(M) \wedge \mathsf{Pre}(M)\}$
   (c) $\{\partial\varphi(N) \wedge \psi(N-1) \wedge \mathsf{Pre}(N-1)\}\ \partial\mathsf{P}_N\ \{\psi(N) \wedge \mathsf{Pre}(N)\}$ *for* $N > M$

Soundness Guarantee

Theorem

Suppose

1) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\} \iff \{\varphi(N)\}\ \mathsf{P}_{N-1}; \partial\mathsf{P}_N\ \{\psi(N)\}$

2) *Formula* $\partial\varphi(N)$ *exists such that*
   - (a) $\varphi(N) \to \varphi(N-1) \wedge \partial\varphi(N)$
   - (b) $\{\partial\varphi(N)\}\ \mathsf{P}_{N-1}\ \{\partial\varphi(N)\}$

3) *Formula* $\mathsf{Pre}(M)$ *exists such that for* $M \geq 1$
   - (a) $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$ *for* $0 < N \leq M$
   - (b) $\{\varphi(M)\}\ \mathsf{P}_M\ \{\psi(M) \wedge \mathsf{Pre}(M)\}$
   - (c) $\{\partial\varphi(N) \wedge \psi(N-1) \wedge \mathsf{Pre}(N-1)\}\ \partial\mathsf{P}_N\ \{\psi(N) \wedge \mathsf{Pre}(N)\}$ *for* $N > M$

Then $\{\varphi(N)\}\ \mathsf{P}_N\ \{\psi(N)\}$ *holds for all* $N \geq 1$.

Implemented in a prototype tool - **Vajra**

Permanent Archive

https://doi.org/10.6084/m9.figshare.11875428.v1

- Evaluated on 231 challenging array benchmarks
- Proved 110/121 safe, 108/110 unsafe and inconclusive on 13 programs

Vajra Participated in SV-COMP 2020

- **Vajra** integrated into TCS verification tool VeriAbs
  - Bundled with VeriAbs **v1.4** as a part of its SV-COMP 2020 archive

Vajra Participated in SV-COMP 2020

- **Vajra** integrated into TCS verification tool VeriAbs
  - Bundled with VeriAbs **v1.4** as a part of its SV-COMP 2020 archive

- "**Gold** ● **Medal**" in the Reach-safety category
  - Vajra improved the score in *Arrays sub-category*
  - **1$^{\text{st}}$** place in 2020, with 694/759 points, solved 410/436 programs
    - Map2Check - 2nd place in 2020, with 379/759 points
  - **2$^{\text{nd}}$** place in 2019, with 365/418 points, solved 196/231 programs
  - Vajra solved **158** additional programs in *Arrays sub-category*

Vajra Participated in SV-COMP 2020

- **Vajra** integrated into TCS verification tool VeriAbs
  - Bundled with VeriAbs **v1.4** as a part of its SV-COMP 2020 archive
- "**Gold** ● **Medal**" in the Reach-safety category
  - Vajra improved the score in *Arrays sub-category*
  - **1st** place in 2020, with 694/759 points, solved 410/436 programs
    - Map2Check - 2*nd* place in 2020, with 379/759 points
  - **2nd** place in 2019, with 365/418 points, solved 196/231 programs
  - Vajra solved **158** additional programs in *Arrays sub-category*
- *Publication* in SV-COMP/TACAS 2020 - "VeriAbs: Verification by Abstraction and Test Generation (Competition Contribution)"
  - *Main novelty*: Full-Program Induction using Vajra

Conclusion

- Presented the novel *Full-Program Induction* technique that
  - proves quantified as well as quantifier-free assertions of programs
  - computes the "difference" of program and property in the inductive step
  - uses weakest-pre computation to infer new facts that aid induction
  - is property driven and efficient
- Vajra verifies a large class of challenging array benchmarks

Thank You