Verifying Array Manipulating Programs with Full-Program Induction

TACAS Artifact Evaluation 2020 Accepted

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 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] == k³);
```

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]

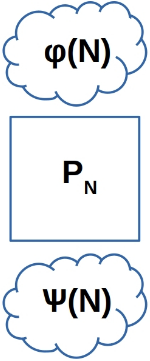
Challenges Faced by State-of-the-Art Tools & Techniques

- 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

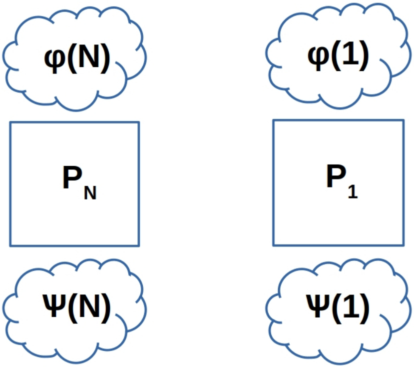
- Quantified invariants with non-linear terms difficult to synthesize
  - Loop invariants required by the respective loops in the program:
  - $\forall i \in [0...x\text{-}1]\ (A[i] = 6i + 6)$
  - $\forall j \in [0...y\text{-}1]\ (B[j] = 3j^2 + 3j + 1 \wedge A[j] = 6j + 6)$
  - $\forall k \in [0...z\text{-}1]\ (C[k] = k^3 \wedge B[k] = 3k^2 + 3k + 1)$
  - **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]
- Difficult to solve (non-linear) recurrences when data flows across loops and loop iterations; difficult to find fix-points
  - **VIAP**[VSTTE'18], **Booster**[ATVA'14]

Full-Program Induction



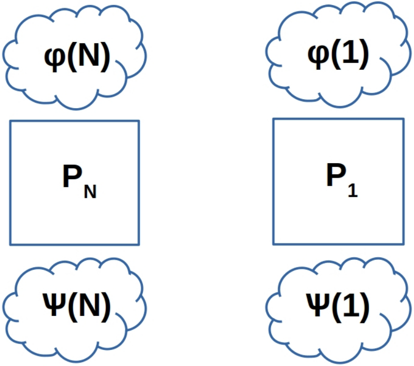
Holds?

Full-Program Induction



Holds? Base Case

Full-Program Induction

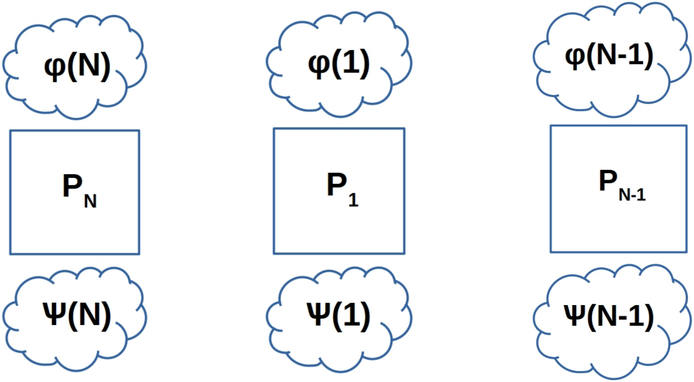


Holds?

Base Case

✗ Property Violation

Full-Program Induction



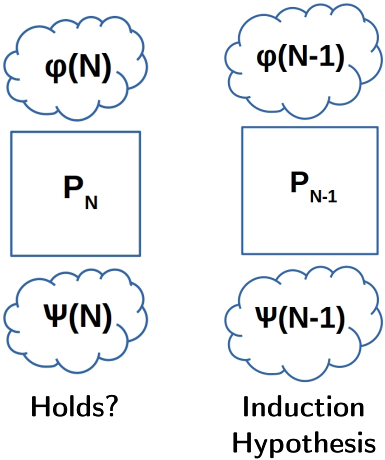
Holds?

Base Case

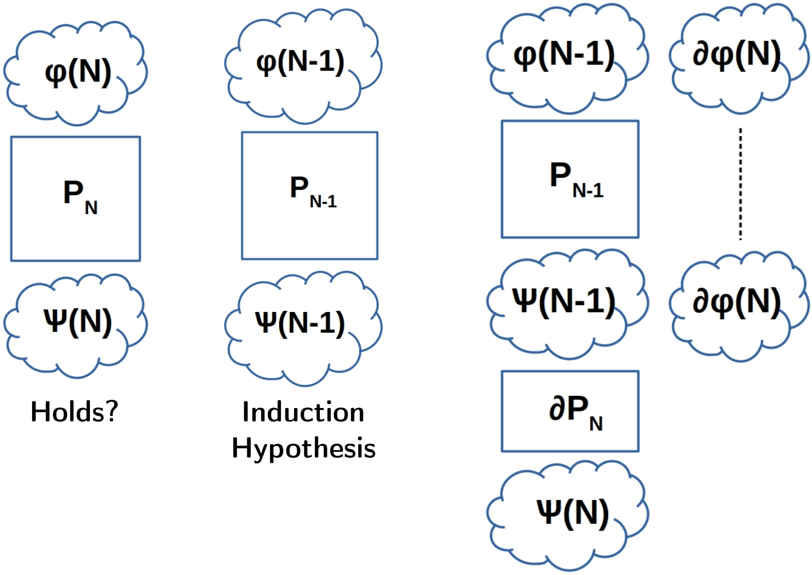
Induction Hypothesis

✗ Property Violation

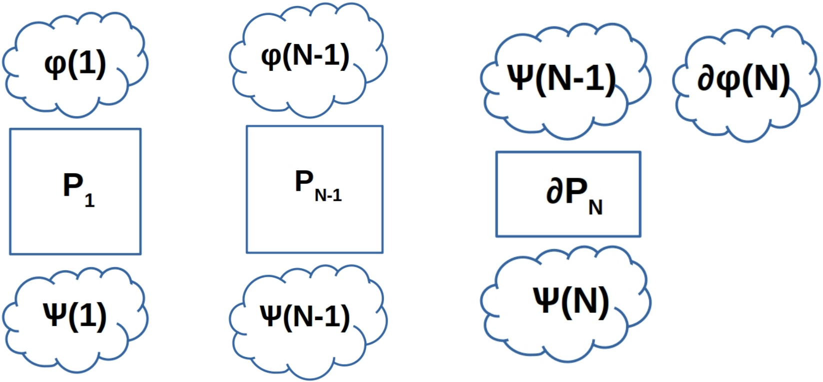
Full-Program Induction



Full-Program Induction



Full-Program Induction

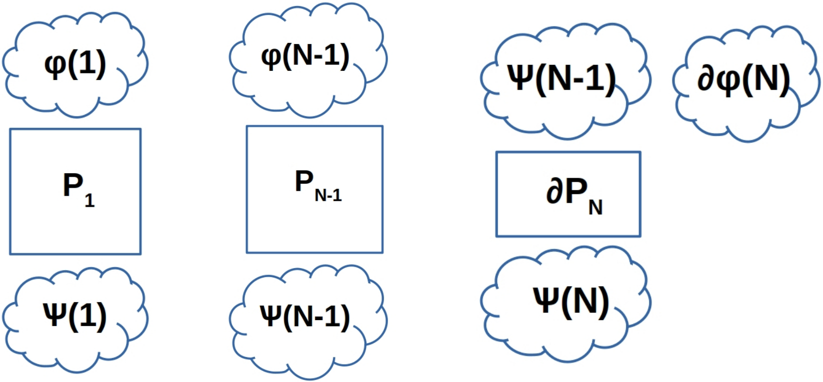


Base Case

Induction Hypothesis

Inductive Step

Full-Program Induction



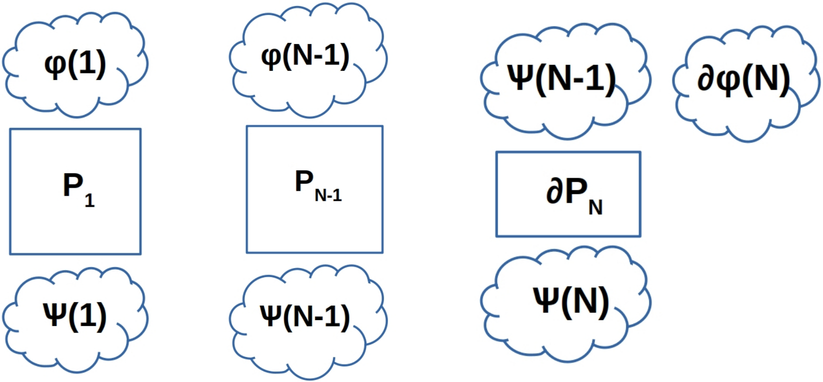
Base Case

Induction Hypothesis

Inductive Step

✓ Proved

Full-Program Induction



Base Case

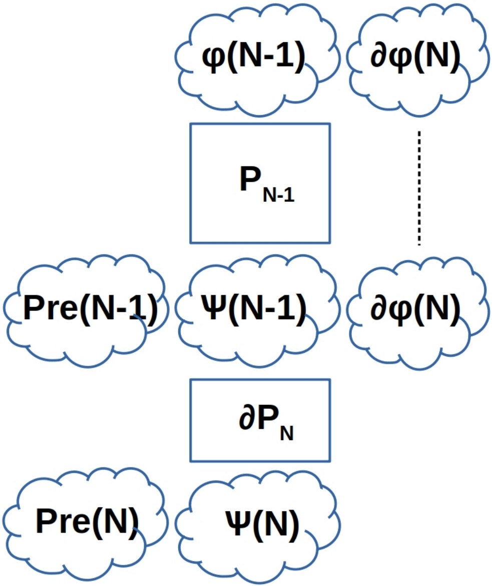
Induction Hypothesis

Inductive Step

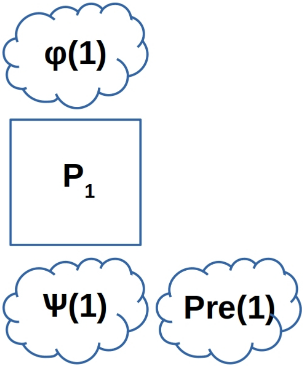
✓ Proved

✗ Infer New Sub-goals

Full-Program Induction

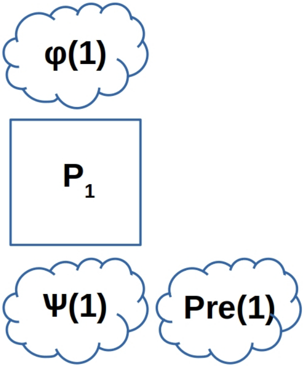


Full-Program Induction



Base Case

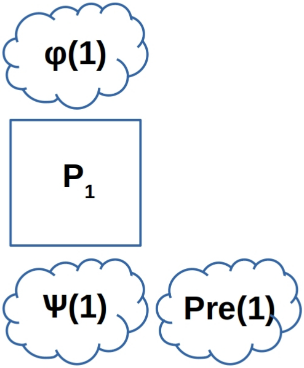
Full-Program Induction



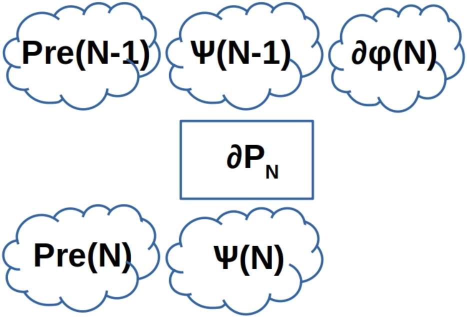
Base Case

✗ Infer New Sub-goals
or
Report unable to prove

Full-Program Induction



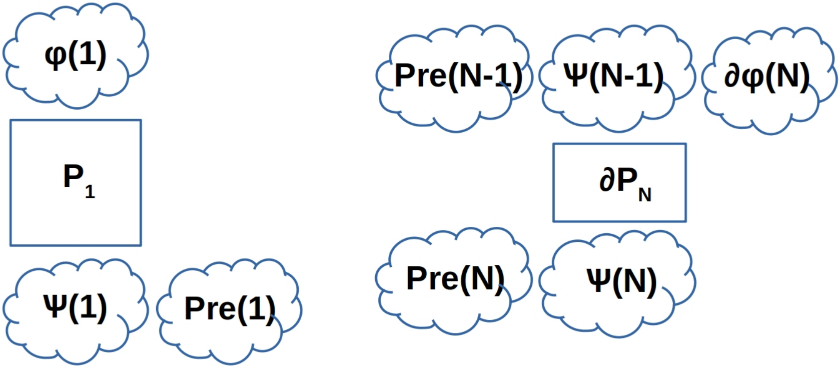
Base Case



Inductive Step

✗ Infer New Sub-goals
or
Report unable to prove

Full-Program Induction



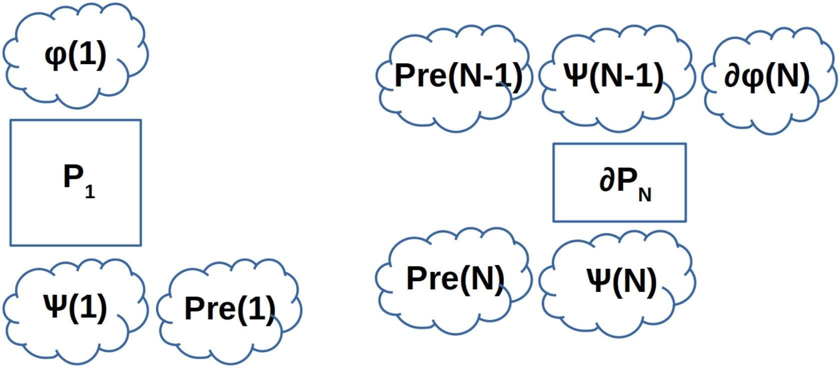
Base Case

Inductive Step

✗ Infer New Sub-goals
or
Report unable to prove

✓ Proved

Full-Program Induction



Base Case

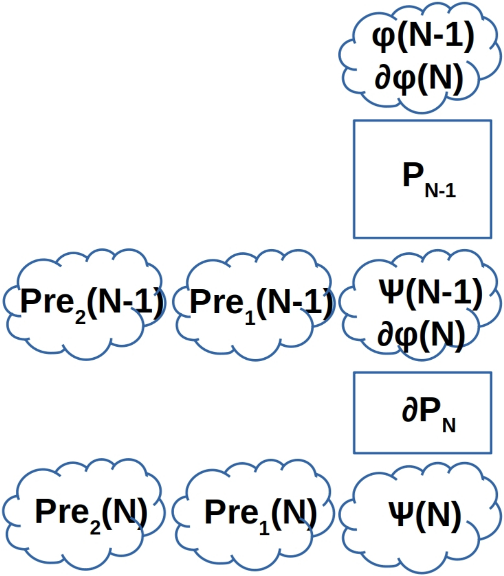
✗ Infer New Sub-goals
or
Report unable to prove

Inductive Step

✓ Proved

✗ Infer New Sub-goals

Full-Program Induction



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

```
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);

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(∀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);
```

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);
```

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);

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_1

```
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);
```

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);
```

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);

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(∀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);
```

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] == k³);
```

Quantify Inferred Pre_2

```
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);
```

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);
```

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);
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] == k^3);
```

Inductive Step

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

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);
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 shape of $\varphi(N)$
  - **If** $\varphi(N) := \forall i\ (0 \leq i \leq N) \rightarrow \hat{\varphi}(i)$ **then** $\partial\varphi(N) := \hat{\varphi}(N)$
    - ★ $\varphi(N) := \forall i\ (0 \leq i \leq N) \rightarrow A[i] > 0$ $\quad$ $\partial\varphi(N) := A[N] > 0$
  - **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 $\partial\varphi(N) := \mathrm{True}$

Computing the "Difference" Program - ∂P_N

- Peel all the loops in the input program P_N
- Replace assignments in the peeled loops with "difference" statements
  - `A[i] = C;`

    is transformed to

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

    is transformed to

    `A[i] = A_Nm1[i] + (B[i] - B_Nm1[i]) + (v - v_Nm1);`
- "Simplify" generated difference terms, "Accelerate" loops
- Slice loops that simply copy values from N-1^{th} version 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) $\varphi(N) \rightarrow \varphi(N-1) \wedge \partial\varphi(N)$
   (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* $N > M$

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

Implemented in a prototype tool - **Vajra**

TACAS Artifact Evaluation 2020

Accepted

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 Benchmarking

- Performance compared with the following tools:
  - VIAP v1.0 - Inductive encoding with arrays as uninterpreted functions
  - VeriAbs v1.3.10 - Loop shrinking/pruning and Output abstraction
  - Booster v0.2 - Acceleration and Lazy Abstraction for Arrays
  - Vaphor v1.2 - Distinguished Cell Abstraction for Arrays
  - FreqHorn v3 - Solving CHC's using Syntax Guided Synthesis
- Benchmarks manually translated to input format of these tools
- Time limit - 100s

Vajra in Action

Benchmark	#L	Vajra	VIAP	VeriAbs	Booster	Vaphor	FreqHorn
pcomp	3	✓0.68	TO	TO	?0.23	TO	?0.58
ncomp	3	✓0.68	TO	TO	?0.41	TO	?0.68
eqnm2	2	✓0.52	TO	TO	?0.07	TO	?0.59
eqnm3	2	✓0.53	TO	TO	?0.07	TO	?0.56
eqnm4	2	✓0.51	TO	TO	?0.07	TO	?0.60
eqnm5	2	✓0.55	TO	TO	?0.07	TO	?0.58
sqm	2	✓0.51	✓69.7	TO	?0.11	TO	?0.57
res1	4	✓0.17	TO	TO	TO	TO	TO
res1o	4	✓0.18	TO	TO	TO	TO	TO
res2	6	✓0.20	TO	TO	TO	TO	TO
res2o	6	✓0.22	TO	TO	TO	TO	TO
ss1	4	✓0.40	TO	TO	✗0.13	?19.2	?1.7
ss2	6	✓0.46	TO	TO	✗0.13	TO	?9.7
ss3	5	✓0.35	TO	TO	✗0.13	TO	?2.1
ss4	4	✓0.29	TO	TO	✗0.13	TO	?1.6
ssina	5	✓0.41	✓72.5	TO	TO	TO	?2.0
sina1	2	✓0.56	✓65.4	TO	TO	TO	TO
sina2	3	✓0.69	✓66.5	TO	TO	TO	TO
sina3	4	✓0.83	TO	TO	TO	TO	TO
sina4	4	✓0.85	TO	TO	TO	TO	TO
sina5	5	✓0.93	TO	TO	TO	TO	TO

Vajra in Action

Benchmark	#L	Vajra	VIAP	VeriAbs	Booster	Vaphor	FreqHorn
pcomp	3	✓0.68	TO	TO	?0.23	TO	?0.58
ncomp	3	✓0.68	TO	TO	?0.41	TO	?0.68
eqnm2	2	✓0.52	TO	TO	?0.07	TO	?0.59
eqnm3	2	✓0.53	TO	TO	?0.07	TO	?0.56
eqnm4	2	✓0.51	TO	TO	?0.07	TO	?0.60
eqnm5	2	✓0.55	TO	TO	?0.07	TO	?0.58
sqm	2	✓0.51	✓69.7	TO	?0.11	TO	?0.57
res1	4	✓0.17	TO	TO	TO	TO	TO
res1o	4	✓0.18	TO	TO	TO	TO	TO
res2	6	✓0.20	TO	TO	TO	TO	TO
res2o	6	✓0.22	TO	TO	TO	TO	TO
ss1	4	✓0.40	TO	TO	✗0.13	?19.2	?1.7
ss2	6	✓0.46	TO	TO	✗0.13	TO	?9.7
ss3	5	✓0.35	TO	TO	✗0.13	TO	?2.1
ss4	4	✓0.29	TO	TO	✗0.13	TO	?1.6
ssina	5	✓0.41	✓72.5	TO	TO	TO	?2.0
sina1	2	✓0.56	✓65.4	TO	TO	TO	TO
sina2	3	✓0.69	✓66.5	TO	TO	TO	TO
sina3	4	✓0.83	TO	TO	TO	TO	TO
sina4	4	✓0.85	TO	TO	TO	TO	TO
sina5	5	✓0.93	TO	TO	TO	TO	TO

Vajra in Action

Benchmark	#L	Vajra	VIAP	VeriAbs	Booster	Vaphor	FreqHorn
zerosum1	2	✓0.33	✓62.0	✓11	✓0.77	✗0.29	TO
zerosum2	4	✓0.46	✓75.8	✓18	TO	✗1.64	TO
zerosum3	6	✓0.59	✓73.1	✓39	TO	✗3.13	TO
zerosum4	8	✓0.76	✓76.1	TO	?18.2	✗6.85	TO
zerosum5	10	✓0.97	✓80.6	TO	?16.5	✗10.4	TO
zerosumm2	4	✓0.46	✓71.5	✓24	TO	✗1.22	TO
zerosumm3	6	✓0.59	✓70.9	TO	TO	✗5.22	TO
zerosumm4	8	✓0.77	✓76.4	TO	?16.7	✗12.39	TO
zerosumm5	10	✓0.98	✓81.7	TO	?18.7	✗22.8	TO
zerosumm6	12	✓1.29	✓86.8	TO	?16.1	TO	TO
copy9	9	✓0.69	✓86.8	✓3.91	✓18.8	TO	✓0.67
min	1	✓0.48	✓23.6	✓3.82	✓0.52	✓0.14	✓0.13
max	1	✓0.46	✓25.4	✓4.70	✓1.0	✓0.28	✓0.18
compare	1	✓0.82	✓18.8	✓17.9	✓0.06	✓0.84	✓0.31
conda	3	✓0.72	✓13.9	TO	✓0.07	✓0.09	TO
condn	1	?0.51	✓14.7	✓18.9	✓0.02	✓0.15	✓0.20
condm	2	?0.59	✓20.5	✓16.7	✓0.04	TO	-
condg	3	?0.52	TO	TO	TO	TO	TO
modn	2	?0.63	✓22.6	TO	-	TO	TO
mods	4	?0.61	TO	✓18.2	-	-	-
modp	2	?0.71	✓17.3	✓40	-	?32	-

Vajra in Action

Benchmark	#L	Vajra	VIAP	VeriAbs	Booster	Vaphor	FreqHorn
zerosum1	2	✓0.33	✓62.0	✓11	✓0.77	✗0.29	TO
zerosum2	4	✓0.46	✓75.8	✓18	TO	✗1.64	TO
zerosum3	6	✓0.59	✓73.1	✓39	TO	✗3.13	TO
zerosum4	8	✓0.76	✓76.1	TO	?18.2	✗6.85	TO
zerosum5	10	✓0.97	✓80.6	TO	?16.5	✗10.4	TO
zerosumm2	4	✓0.46	✓71.5	✓24	TO	✗1.22	TO
zerosumm3	6	✓0.59	✓70.9	TO	TO	✗5.22	TO
zerosumm4	8	✓0.77	✓76.4	TO	?16.7	✗12.39	TO
zerosumm5	10	✓0.98	✓81.7	TO	?18.7	✗22.8	TO
zerosumm6	12	✓1.29	✓86.8	TO	?16.1	TO	TO
copy9	9	✓0.69	✓86.8	✓3.91	✓18.8	TO	✓0.67
min	1	✓0.48	✓23.6	✓3.82	✓0.52	✓0.14	✓0.13
max	1	✓0.46	✓25.4	✓4.70	✓1.0	✓0.28	✓0.18
compare	1	✓0.82	✓18.8	✓17.9	✓0.06	✓0.84	✓0.31
conda	3	✓0.72	✓13.9	TO	✓0.07	✓0.09	TO
condn	1	?0.51	✓14.7	✓18.9	✓0.02	✓0.15	✓0.20
condm	2	?0.59	✓20.5	✓16.7	✓0.04	TO	-
condg	3	?0.52	TO	TO	TO	TO	TO
modn	2	?0.63	✓22.6	TO	-	TO	TO
mods	4	?0.61	TO	✓18.2	-	-	-
modp	2	?0.71	✓17.3	✓40	-	?32	-

Vajra in Action

Benchmark	#L	Vajra	VIAP	VeriAbs	Booster	Vaphor	FreqHorn
zerosum1	2	✓0.33	✓62.0	✓11	✓0.77	✗0.29	TO
zerosum2	4	✓0.46	✓75.8	✓18	TO	✗1.64	TO
zerosum3	6	✓0.59	✓73.1	✓39	TO	✗3.13	TO
zerosum4	8	✓0.76	✓76.1	TO	?18.2	✗6.85	TO
zerosum5	10	✓0.97	✓80.6	TO	?16.5	✗10.4	TO
zerosumm2	4	✓0.46	✓71.5	✓24	TO	✗1.22	TO
zerosumm3	6	✓0.59	✓70.9	TO	TO	✗5.22	TO
zerosumm4	8	✓0.77	✓76.4	TO	?16.7	✗12.39	TO
zerosumm5	10	✓0.98	✓81.7	TO	?18.7	✗22.8	TO
zerosumm6	12	✓1.29	✓86.8	TO	?16.1	TO	TO
copy9	9	✓0.69	✓86.8	✓3.91	✓18.8	TO	✓0.67
min	1	✓0.48	✓23.6	✓3.82	✓0.52	✓0.14	✓0.13
max	1	✓0.46	✓25.4	✓4.70	✓1.0	✓0.28	✓0.18
compare	1	✓0.82	✓18.8	✓17.9	✓0.06	✓0.84	✓0.31
conda	3	✓0.72	✓13.9	TO	✓0.07	✓0.09	TO
condn	1	?0.51	✓14.7	✓18.9	✓0.02	✓0.15	✓0.20
condm	2	?0.59	✓20.5	✓16.7	✓0.04	TO	-
condg	3	?0.52	TO	TO	TO	TO	TO
modn	2	?0.63	✓22.6	TO	-	TO	TO
mods	4	?0.61	TO	✓18.2	-	-	-
modp	2	?0.71	✓17.3	✓40	-	?32	-

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