Inductive Invariant Generation via Abductive Inference

Işıl Dillig  
MSR Cambridge

Thomas Dillig  
University College London

Ken McMillan  
MSR Redmond

Boyang Li  
College of William & Mary
Loop Invariants

- When proving correctness of software, **finding loop invariants** is a fundamental challenge.
When proving correctness of software, finding loop invariants is a fundamental challenge.

Intuitively, a loop invariant summarizes the behavior of an unbounded number of computations in one formula.
Want to prove $Q$ after the loop

```
while(C)
{
    S;
}
assert(Q);
```
Inductive Loop Invariants

- Want to prove $Q$ after the loop
- A loop invariant $I$ must be strong enough to show $Q$.

```
while(C)
{
    S;
}
I \rightarrow assert(Q);
```
Inductive Loop Invariants

Want to prove $Q$ after the loop

A loop invariant $I$ must be strong enough to show $Q$. $I \land \neg C \Rightarrow Q$

```plaintext
while(C)
{
    S;
}
I → assert(Q);
```
Want to prove $Q$ after the loop.

A loop invariant $I$ must be strong enough to show $Q$. $I \land \neg C \Rightarrow Q$

Invariant $I$ is inductive if assuming it holds at the beginning, it must hold at the end of iteration:
Want to prove $Q$ after the loop

A loop invariant $I$ must be strong enough to show $Q$. $I \land \neg C \Rightarrow Q$

Invariant $I$ is inductive if assuming it holds at the beginning, it must hold at the end of iteration:

$$I \land C \Rightarrow I' \text{ where } I' = \wp(s, I)$$
Inductive Loop Invariants

- Want to prove $Q$ after the loop
- A loop invariant $I$ must be strong enough to show $Q$. $I \land \neg C \Rightarrow Q$
- Invariant $I$ is inductive if assuming it holds at the beginning, it must hold at the end of iteration:

\[
I \land C \Rightarrow I' \text{ where } I' = \text{wp}(s, I)
\]

Only way to prove a loop invariant is to show it is inductive.
int x = 0;
int y = 0;

while(x < n)
{
    x = x+1;
    y = y+2;
}

assert( x + y >= 3*n);
Loop Invariant Example

Postcondition $Q : x + y \geq 3n$

```c
int x = 0;
int y = 0;
while(x < n)
{
    x = x+1;
    y = y+2;
}
assert( x + y >= 3*n);
```
Loop Invariant Example

- Postcondition: \( Q : x + y \geq 3n \)

- If assertion holds, \( x \geq n \rightarrow x + y \geq 3n \) must be loop invariant.

```c
int x = 0;
int y = 0;

while(x < n)
{
    x = x+1;
    y = y+2;
}

assert(x + y >= 3*n);
```
Loop Invariant Example

- Postcondition  \( Q : x + y \geq 3n \)
- If assertion holds,  \( x \geq n \rightarrow x + y \geq 3n \) must be loop invariant.
- But is  \( I : x \geq n \rightarrow x + y \geq 3n \) inductive?

```c
int x = 0;
int y = 0;
while(x < n) {
    x = x+1;
    y = y+2;
}
assert( x + y >= 3*n);
```
Loop Invariant Example

```c
int x = 0;
int y = 0;
while(x < n) {
    x = x+1;
    y = y+2;
}
assert( x + y >= 3*n);
```

- **Postcondition** $Q : x + y \geq 3n$
- If assertion holds, $x \geq n \rightarrow x + y \geq 3n$ must be loop invariant.
- But is $I : x \geq n \rightarrow x + y \geq 3n$ inductive?
  - No, because $I \wedge x < n \nRightarrow (x + 1 \geq n \rightarrow (x + 1) + (y + 2) \geq 3n)$
int x = 0;
int y = 0;

while(x < n) {
    x = x+1;
    y = y+2;
}

assert( x + y >= 3*n);

Postcondition $Q : x + y \geq 3n$

If assertion holds, $x \geq n \rightarrow x + y \geq 3n$

must be loop invariant.

But is $I : x \geq n \rightarrow x + y \geq 3n$ inductive?

No, because $I \land x < n \not\Rightarrow (x + 1 \geq n \rightarrow (x + 1) + (y + 2) \geq 3n)$

We need stronger invariant
Finding inductive loop invariants is key challenge in verification
Finding inductive loop invariants is key challenge in verification

A new approach for strengthening candidate invariants to discover inductive loop invariants
Finding inductive loop invariants is key challenge in verification.

A new approach for strengthening candidate invariants to discover inductive loop invariants.

Key Insight:
Use logical abduction to find inductive invariants.
Abduction: Opposite of deduction
Abduction: Opposite of deduction

Deduction: Infers valid conclusion from premises
What is Abduction?

- **Abduction**: Opposite of deduction
- **Deduction**: Infers valid conclusion from premises
- **Abduction**: Infers missing premise to explain a given conclusion
What is Abduction?

- **Abduction**: Opposite of deduction
- **Deduction**: Infers valid conclusion from premises
- **Abduction**: Infers missing premise to explain a given conclusion

Given known facts $\Gamma$ and desired outcome $\phi$, **abductive inference** finds “simple” **explanatory hypothesis** $\psi$ such that

$$\Gamma \land \psi \models \phi \text{ and } \text{SAT}(\Gamma \land \psi)$$
Simple Example

- **Facts:** “If it rains, then it is wet and cloudy”, “If it is wet, then it is slippery”:
  \[ R \Rightarrow W \land C \land W \Rightarrow S \]
Facts: “If it rains, then it is wet and cloudy”, “If it is wet, then it is slippery”:
\[ R \Rightarrow W \land C \land W \Rightarrow S \]

Conclusion: “It is cloudy and slippery”, i.e., \( C \land S \)
Simple Example

- Facts: “If it rains, then it is wet and cloudy”, “If it is wet, then it is slippery”:
  \[ R \Rightarrow W \land C \land W \Rightarrow S \]

- Conclusion: “It is cloudy and slippery”, i.e., \[ C \land S \]

- Abductive explanation: \( R \), i.e., “It is rainy”
int x = 0;
int y = 0;

while(x < n)
{
    x = x+1;
    y = y+2;
}

assert( x + y >= 3*n);
Here we have $C : x \geq n$ from loop termination condition.
Here we have $C : x \geq n$ from loop termination condition

Desired conclusion $Q: x + y \geq 3n$
Here we have $C : x \geq n$ from loop termination condition

Desired conclusion $Q : x + y \geq 3n$

We want stronger $I$ such that:

\[ I \land C \models Q \]

\[ \text{SAT}(I \land C) \]
Here we have $C : x \geq n$ from loop termination condition

Desired conclusion $Q: x + y \geq 3n$

We want stronger $I$ such that:

$I \land C \models Q$

$\text{SAT}(I \land C)$

Abductive explanation: $I: y \geq 2x$
Here we have $C : x \geq n$ from loop termination condition

Desired conclusion $Q: x + y \geq 3n$

We want stronger $I$ such that:

$$I \land C \models Q$$

$$\text{SAT}(I \land C)$$

Abductive explanation: $I: y \geq 2x$

Corresponds to missing inductive loop invariant
In general, the abduction problem \( \Gamma \land ? \models \phi \) has infinitely many solutions
In general, the abduction problem $\Gamma \land ? \models \phi$ has infinitely many solutions

**Trivial solution:** $\phi$, but generally not inductive
In general, the abduction problem $\Gamma \land ? \models \phi$ has infinitely many solutions.

- **Trivial solution:** $\phi$, but generally not inductive.

- So, what kind of solutions do we want to compute?
Guiding Principle:
Occam’s Razor
Which Abductive Explanations Are Good?

Guiding Principle:
Occam’s Razor

- If there are multiple competing hypotheses, select the one that makes fewest assumptions
Which Abductive Explanations Are Good?

Guiding Principle: 
Occam’s Razor

- If there are multiple competing hypotheses, select the one that makes fewest assumptions

- **Generality:** If explanation $A$ is logically weaker than explanation $B$, always prefer $A$
Which Abductive Explanations Are Good?

Guiding Principle: Occam’s Razor

- If there are multiple competing hypotheses, select the one that makes fewest assumptions.

- **Generality**: If explanation $A$ is logically weaker than explanation $B$, always prefer $A$.

- **Simplicity**: Prefer solutions with fewest number of variables.
Which Abductive Explanations Are Good?

Guiding Principle: Occam’s Razor

- If there are multiple competing hypotheses, select the one that makes fewest assumptions
- **Generality:** If explanation $A$ is logically weaker than explanation $B$, always prefer $A$
- **Simplicity:** Prefer solutions with fewest number of variables
- **Intuition:** Most likely to generalize behavior of a loop
Using Abduction for Loop Invariant Generation

**Key idea:** Perform backtracking search combining Hoare logic with abduction.

1. Starting with true, iteratively strengthen loop invariants.
2. At every step, use current set of invariants to generate VCs:
   - **Inductive:** $I \land C \Rightarrow \text{wp}(s, I)$
   - **Sufficient:** $I \land \neg C \Rightarrow Q$
3. If all VCs are valid, found inductive invariants sufficient to verify program.
4. Otherwise, strengthen LHS using abduction.
Key idea: Perform backtracking search combining Hoare logic with abduction

- Starting with true, iteratively strengthen loop invariants
Key idea: Perform backtracking search combining Hoare logic with abduction

- Starting with true, iteratively strengthen loop invariants
- At every step, use current set of invariants to generate VCs:

  Inductive: $I \land C \Rightarrow \text{wp}(s, I)$
  Sufficient: $I \land \neg C \Rightarrow Q$
Using Abduction for Loop Invariant Generation

- **Key idea:** Perform backtracking search combining Hoare logic with abduction

- Starting with true, iteratively strengthen loop invariants

- At every step, use current set of invariants to generate VCs:
  - **Inductive:** \( I \land C \Rightarrow wp(s, I) \)
  - **Sufficient:** \( I \land \neg C \Rightarrow Q \)

- If all VCs are valid, found inductive invariants sufficient to verify program
Using Abduction for Loop Invariant Generation

Key idea: Perform backtracking search combining Hoare logic with abduction

- Starting with true, iteratively strengthen loop invariants
- At every step, use current set of invariants to generate VCs:
  - Inductive: \( I \land C \Rightarrow wp(s, I) \)
  - Sufficient: \( I \land \neg C \Rightarrow Q \)
- If all VCs are valid, found inductive invariants sufficient to verify program
- Otherwise, strengthen LHS using abduction
If $I \land \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I \land \psi$ is strong enough to show $Q$. 
If $I \land \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I \land \psi$ is strong enough to show $Q$.

If $I \land C \Rightarrow \text{wp}(s, I)$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I$ is inductive relative to $\psi$. 
If $I \land \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I \land \psi$ is strong enough to show $Q$.

If $I \land C \Rightarrow \text{wp}(s, I)$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I$ is inductive relative to $\psi$.

In either case, strengthen invariant to $I \land \psi$ and try to prove correctness.
Since candidate invariant is a speculation, it may be wrong
Backtracking

Since candidate invariant is a speculation, it may be wrong
- E.g. may contradict loop precondition
Since candidate invariant is a speculation, it may be wrong
  - E.g. may contradict loop precondition

In this case, backtrack and try another solution
Since candidate invariant is a speculation, it may be wrong
  - E.g. may contradict loop precondition

In this case, backtrack and try another solution

Therefore, generate sequence of abductive solutions with increasing number of variables

\[ I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow I_3 \ldots \]
Full Algorithm

- Current invariants
- VCGen
- Done
- Abduction
- No solution
- Solution
- Strengthened invariant
- Backtrack!
- Wrong way
- Right way
Experimental Results

- Evaluated this technique on 46 loop invariant benchmarks
Experimental Results

- Evaluated this technique on 46 loop invariant benchmarks
- Compared our results against BLAST, InvGen, and Interproc:
Experimental Results

- Evaluated this technique on 46 loop invariant benchmarks
- Compared our results against BLAST, InvGen, and Interproc:

![Bar Chart]

<table>
<thead>
<tr>
<th>Tool</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLA</td>
<td>93.5</td>
</tr>
<tr>
<td>BLAST</td>
<td>43.5</td>
</tr>
<tr>
<td>InvGen</td>
<td>47.8</td>
</tr>
<tr>
<td>InterProc</td>
<td>37</td>
</tr>
</tbody>
</table>
Experimental Results

- Evaluated this technique on 46 loop invariant benchmarks
- Compared our results against BLAST, InvGen, and Interproc:

```
HOLA  BLAST  InvGen  Interproc
93.5  43.5   47.8   37
```

- But not strictly better: cannot prove two benchmarks at least one tool can show
Lots of work on loop invariant generation (AI, CEGAR, Houdini, ...).
Lots of work on loop invariant generation (AI, CEGAR, Houdini, ...).

Main characteristics of this approach:
Lots of work on loop invariant generation (AI, CEGAR, Houdini, . . .).

Main characteristics of this approach:

- Demand-driven
Lots of work on loop invariant generation (AI, CEGAR, Houdini, . . .).

Main characteristics of this approach:

- Demand-driven
- No templates
Lots of work on loop invariant generation (AI, CEGAR, Houdini, ...).

Main characteristics of this approach:

- Demand-driven
- No templates
- Can naturally derive disjunctive invariants
Lots of work on loop invariant generation (AI, CEGAR, Houdini, ...).

Main characteristics of this approach:

- Demand-driven
- No templates
- Can naturally derive disjunctive invariants

Abduction-based approach useful addition to known techniques for loop invariant generation
Questions?