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

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

Loop Invariants

- 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.

Inductive Loop Invariants

```
while(C)
{
  S;
}
assert(Q);
```

- Want to prove Q after the loop

Inductive Loop Invariants

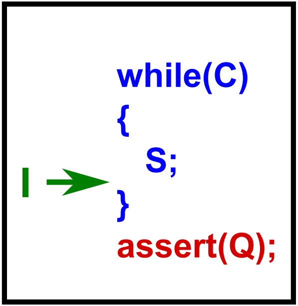
```
while(C)
{
  S;
}
I → assert(Q);
```

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

Inductive Loop Invariants

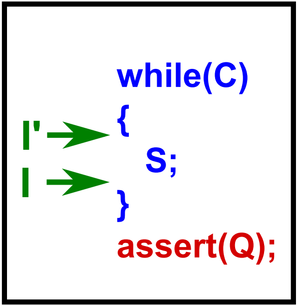
```
    while(C)
    {
      S;
    }
I   assert(Q);
```

- Want to prove Q after the loop
- A loop invariant I must be strong enough to show Q. $I \wedge \neg C \Rightarrow Q$



- 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:

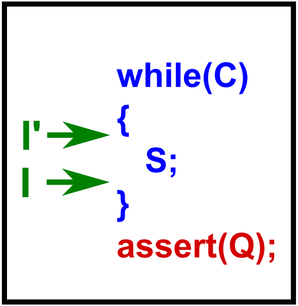
Inductive Loop Invariants



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

$$I \wedge C \Rightarrow I' \text{ where } I' = wp(s, I)$$

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:

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

Only way to prove a loop invariant is to show it is inductive.

Loop Invariant Example

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

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

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

Loop Invariant Example

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

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

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

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

Loop Invariant Example

```
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.

Loop Invariant Example

```
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?

Loop Invariant Example

```
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 $I \wedge x < n \not\Rightarrow (x + 1 \geq n \rightarrow (x + 1) + (y + 2) \geq 3n)$

Loop Invariant Example

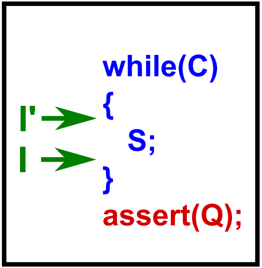
```
int x = 0;
int y = 0;

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

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

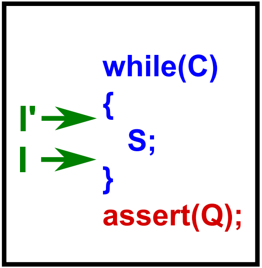
- Postcondition $Q : x + y \geq 3n$
- If assertion holds, $x \geq n \rightarrow x + y \geq 3n$ must be loop invariant.
- But is $I : x \geq n \rightarrow x + y \geq 3n$ inductive?
  - No, because $I \wedge x < n \not\Rightarrow (x + 1 \geq n \rightarrow (x + 1) + (y + 2) \geq 3n)$
  - We need stronger invariant

This Talk



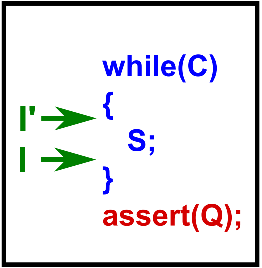
- Finding inductive loop invariants is key challenge in verification

This Talk



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

This Talk



- 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

What is Abduction?

- Abduction: Opposite of deduction

What is Abduction?

- 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 Γ and desired outcome ϕ, abductive inference finds "simple" explanatory hypothesis ψ such that

$$\Gamma \land \psi \models \phi \quad \text{and} \quad \mathrm{SAT}(\Gamma \land \psi)$$

Simple Example

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

Simple Example

- Facts: “If it rains, then it is wet and cloudy”, “If it is wet, then it is slippery”: $R \Rightarrow W \wedge C \;\; \wedge W \Rightarrow S$
- Conclusion: “It is cloudy and slippery”, i.e., $C \wedge S$

Simple Example

- Facts: "If it rains, then it is wet and cloudy", "If it is wet, then it is slippery": $R \Rightarrow W \wedge C \ \wedge W \Rightarrow S$
- Conclusion: "It is cloudy and slippery", i.e., $C \wedge S$
- Abductive explanation: R, i.e., "It is rainy"

Abduction for Loop Invariant Generation

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

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

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

Abduction for Loop Invariant Generation

```
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

Abduction for Loop Invariant Generation

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

Abduction for Loop Invariant Generation

```
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
- Desired conclusion Q: $x + y \geq 3n$
- We want stronger I such that:

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

Abduction for Loop Invariant Generation

```
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
- Desired conclusion Q: $x + y \geq 3n$
- We want stronger I such that:

$$I \wedge C \models Q$$
$$\text{SAT}(I \wedge C)$$

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

Abduction for Loop Invariant Generation

```
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
- Desired conclusion Q: $x + y \geq 3n$
- We want stronger I such that:

$$I \wedge C \models Q$$
$$\text{SAT}(I \wedge C)$$

- Abductive explanation: I: $y \geq 2x$
- Corresponds to missing inductive loop invariant

Properties of Desired Solutions

- In general, the abduction problem $\Gamma \wedge ? \models \phi$ has infinitely many solutions

Properties of Desired Solutions

- In general, the abduction problem $\Gamma \wedge ? \models \phi$ has infinitely many solutions
- Trivial solution: ϕ, but generally not inductive

Properties of Desired Solutions

- In general, the abduction problem $\Gamma \wedge ? \models \phi$ has infinitely many solutions
- Trivial solution: ϕ, but generally not inductive
- So, what kind of solutions do we want to compute?

Which Abductive Explanations Are Good?

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

Using Abduction for Loop Invariant Generation

Key idea: Perform backtracking search combining Hoare logic with abduction

- Starting with true, iteratively strengthen loop invariants

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:

$$\begin{array}{ll} \text{Inductive}: & I \land C \Rightarrow wp(s, I) \\ \text{Sufficient}: & I \land \neg C \Rightarrow Q \end{array}$$

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:

$$\begin{array}{ll} \text{Inductive}: & I \wedge C \Rightarrow wp(s, I) \\ \text{Sufficient}: & I \wedge \neg C \Rightarrow Q \end{array}$$

- 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:

$$
\begin{aligned}
\text{Inductive :} \quad & I \wedge C \Rightarrow wp(s, I) \\
\text{Sufficient :} \quad & I \wedge \neg C \Rightarrow Q
\end{aligned}
$$

- If all VCs are valid, found inductive invariants sufficient to verify program
- Otherwise, strengten LHS using abduction

Using Abduction for Loop Invariant Generation, cont.

- If $I \land \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant ψ such that $I \land \psi$ is strong enough to show Q

Using Abduction for Loop Invariant Generation, cont.

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

Using Abduction for Loop Invariant Generation, cont.

- If $I \land \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant ψ such that $I \land \psi$ is strong enough to show Q
- If $I \land C \Rightarrow wp(s, I)$ is invalid, abduction produces auxiliary invariant ψ such that I is inductive relative to ψ
- In either case, strengthen invariant to $I \land \psi$ and try to prove correctness

Backtracking

- 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

Backtracking

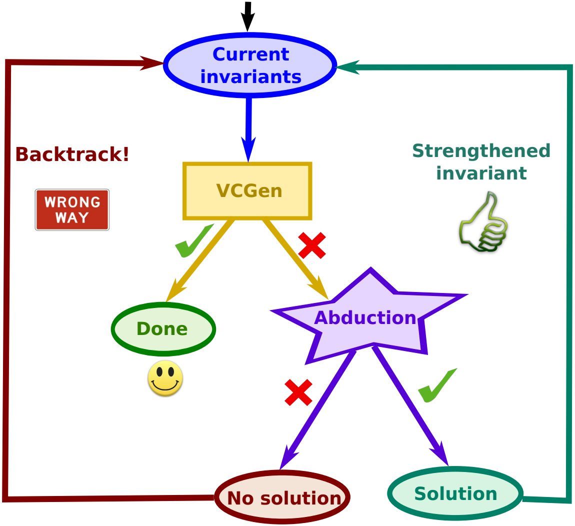
- Since candidate invariant is a speculation, it may be wrong
  - E.g. may contradict loop precondition
- In this case, backtrack and try another solution

Backtracking

- 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



Experimental Results

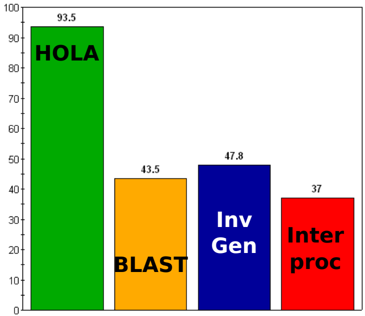
- Evaluated this technique on 46 loop invariant benchmarks

Experimental Results

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

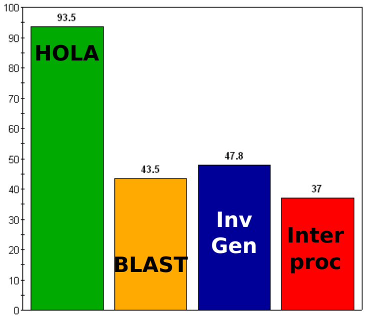
Experimental Results

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



Experimental Results

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



- But not strictly better: cannot prove two benchmarks at least one tool can show

Summary

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

Summary

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

Summary

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

Summary

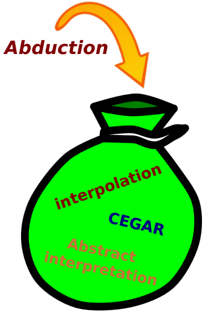
- Lots of work on loop invariant generation (AI, CEGAR, Houdini, . . .).
- Main characteristics of this approach:
  - Demand-driven
  - No templates

Summary

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

Summary

- 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?