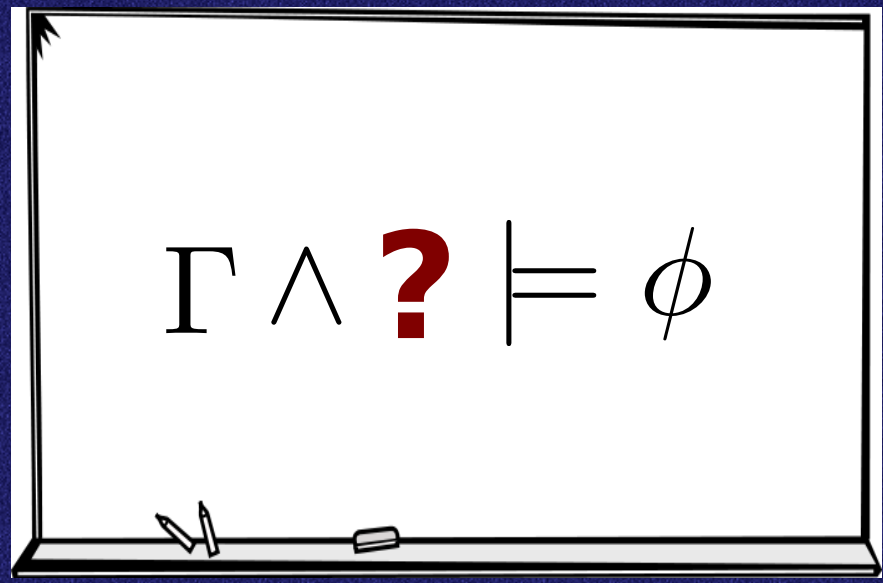
Logical Abduction and its Applications in Program Verification



Isil Dillig
MSR Cambridge

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

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

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”

Arithmetic Example

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

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

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

Arithmetic Example

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

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

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

- Suppose we know $x \geq n$
  - e.g., from loop termination condition

Arithmetic Example

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

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

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

- Suppose we know $x \geq n$
  - e.g., from loop termination condition
- Desired conclusion $x + y \geq 3n$
  - property we want to prove

Arithmetic Example

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

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

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

- Suppose we know $x \geq n$
  - e.g., from loop termination condition
- Desired conclusion $x + y \geq 3n$
  - property we want to prove
- Abductive explanation: $y \geq 2x$
  - corresponds to missing loop invariant

Abduction vs. Interpolation

Abduction vs. Interpolation

- Abduction is flip-side of Craig interpolation
  - Abduction explains why a formula is invalid; interpolation explains why it is valid

Abduction vs. Interpolation

- Abduction is flip-side of Craig interpolation
  - Abduction explains why a formula is invalid; interpolation explains why it is valid
- In abduction, given invalid $\Gamma \Rightarrow \phi$, find ψ that explains why.

Abduction vs. Interpolation

- Abduction is flip-side of Craig interpolation
  - Abduction explains why a formula is invalid; interpolation explains why it is valid
- In abduction, given invalid $\Gamma \Rightarrow \phi$, find ψ that explains why.
- In interpolation, given valid $\Gamma \Rightarrow \phi$, if we can find ψ s.t. $\Gamma \Rightarrow \psi$ and $\psi \Rightarrow \phi$, then ψ constitutes proof of validity.

Abduction vs. Interpolation

- Abduction is flip-side of Craig interpolation
  - Abduction explains why a formula is invalid; interpolation explains why it is valid
- In abduction, given invalid $\Gamma \Rightarrow \phi$, find ψ that explains why.
- In interpolation, given valid $\Gamma \Rightarrow \phi$, if we can find ψ s.t. $\Gamma \Rightarrow \psi$ and $\psi \Rightarrow \phi$, then ψ constitutes proof of validity.
- Abduction + Interpolation: Yin-and-Yang of logical inference

Road Map

1. Properties of desired solutions

Road Map

1. Properties of desired solutions
2. Algorithm for performing abduction in logics that admit QE

Road Map

1. Properties of desired solutions
2. Algorithm for performing abduction in logics that admit QE
3. Applications of abduction in program analysis/verification

Road Map

1. Properties of desired solutions
2. Algorithm for performing abduction in logics that admit QE
3. Applications of abduction in program analysis/verification
4. Thoughts on abduction vs. interpolation in verification

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 not useful because does not take into account what we know

Properties of Desired Solutions

- In general, the abduction problem $\Gamma \wedge ? \models \phi$ has infinitely many solutions
- Trivial solution: ϕ, but not useful because does not take into account what we know
- So, what kind of solutions do 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: Not clear-cut, but we use 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: Not clear-cut, but we use number of variables
- This simplicity criterion makes sense in verification because we want proof subgoals to be local and refer to few variables

Solutions We Will Compute

Want to compute logically weakest solutions with fewest variables

Solutions We Will Compute

Want to compute logically weakest solutions with fewest variables

- First talk about how to compute solutions with fewest variables

Solutions We Will Compute

Want to compute logically weakest solutions with fewest variables

- First talk about how to compute solutions with fewest variables
- Then talk about how to obtain most general solution containing these variables

Minimum Satisfying Assignments

To find solutions with fewest variables, we use minimum satisfying assignments of formulas

Minimum Satisfying Assignments

To find solutions with fewest variables, we use minimum satisfying assignments of formulas

Minimum satisfying assignment (MSA):

- ✓ assigns values to a subset of variables in formula
- ✓ sufficient to make formula true
- ✓ Among all other partial satisfying assignments, contains fewest variables

Example

- Consider the following formula in linear integer arithmetic:

$$x + y + w > 0 \vee x + y + z + w < 5$$

Example

- Consider the following formula in linear integer arithmetic:

$$x + y + w > 0 \vee x + y + z + w < 5$$

- Minimum satisfying assignment: $z = 0$

Example

- Consider the following formula in linear integer arithmetic:

$$x + y + w > 0 \vee x + y + z + w < 5$$

- Minimum satisfying assignment: $z = 0$
- Note: Algorithm for computing MSAs given in our CAV'12 paper, "Minimum Satisfying Assignments for SMT" by Dillig & McMillan

Why Are MSAs Useful for Abduction?

- Recall: Want to find ψ such that $\Gamma \wedge \psi \models \phi$

Why Are MSAs Useful for Abduction?

- Recall: Want to find ψ such that $\Gamma \wedge \psi \models \phi$
- This entailment can be rewritten as:

$$\psi \models \Gamma \Rightarrow \phi$$

Why Are MSAs Useful for Abduction?

- Recall: Want to find ψ such that $\Gamma \wedge \psi \models \phi$
- This entailment can be rewritten as:

$$\psi \models \Gamma \Rightarrow \phi$$

- Hence, MSA of $\Gamma \Rightarrow \phi$ consistent with Γ is a solution to abduction problem

Why Are MSAs Useful for Abduction?

- Recall: Want to find ψ such that $\Gamma \wedge \psi \models \phi$
- This entailment can be rewritten as:

$$\psi \models \Gamma \Rightarrow \phi$$

- Hence, MSA of $\Gamma \Rightarrow \phi$ consistent with Γ is a solution to abduction problem
- Furthermore, it makes assumptions about as few variables as possible

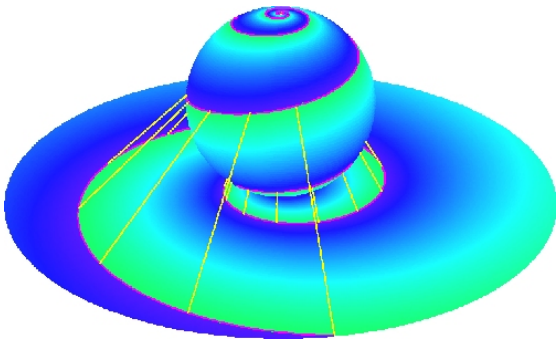
Why Are MSAs Useful for Abduction?

- Recall: Want to find ψ such that $\Gamma \wedge \psi \models \phi$
- This entailment can be rewritten as:

$$\psi \models \Gamma \Rightarrow \phi$$

- Hence, MSA of $\Gamma \Rightarrow \phi$ consistent with Γ is a solution to abduction problem
- Furthermore, it makes assumptions about as few variables as possible
- But it is not the most general solution

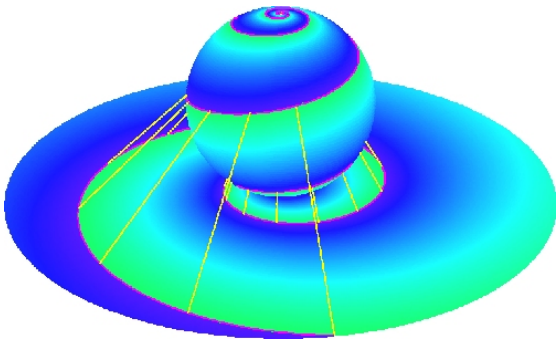
Finding Most General Solutions



Key idea:
Quantifier elimination

- To find most general solution containing variables in the MSA, universally quantify all other variables $\overline{V}$ and apply quantifier elimination to $\forall \overline{V}.\ \Gamma \Rightarrow \phi$

Finding Most General Solutions



Key idea:
Quantifier elimination

- To find most general solution containing variables in the MSA, universally quantify all other variables $\overline{V}$ and apply quantifier elimination to $\forall \overline{V}.\ \Gamma \Rightarrow \phi$

- The resulting formula captures all satisfying assignments containing only MSA variables

Abduction Algorithm

- abduce yields formula E such that

$$I \wedge E \models \phi$$

and E is consistent with I

$\text{abduce}(I, \phi)$ {

}

Abduction Algorithm

- abduce yields formula E such that

$$I \wedge E \models \phi$$

  and E is consistent with I

- First, compute all variables in MSA of $I \Rightarrow \phi$ consistent with I, θ

$\text{abduce}(I, \phi)$ {

$V = \text{msa}(I \Rightarrow \phi,\ I)$

}

Abduction Algorithm

- abduce yields formula E such that

$$I \wedge E \models \phi$$

  and E is consistent with I

- First, compute all variables in MSA of $I \Rightarrow \phi$ consistent with I, θ

- $\forall$-quantify variables not in the MSA and apply quantifier elimination

$\text{abduce}(I, \phi)$ {

$V = \text{msa}(I \Rightarrow \phi,\ I)$

$\psi = \text{QE}(\forall \overline{V}.(I \Rightarrow \phi))$

}

Abduction Algorithm

- abduce yields formula E such that

$$I \wedge E \models \phi$$

  and E is consistent with I

- First, compute all variables in MSA of $I \Rightarrow \phi$ consistent with I, θ

- $\forall$-quantify variables not in the MSA and apply quantifier elimination

- Remove subparts of ψ implied by I

$\text{abduce}(I, \phi)$ {

$V = \text{msa}(I \Rightarrow \phi,\ I)$

$\psi = \text{QE}(\forall \overline{V}.(I \Rightarrow \phi))$

$\psi' = \text{simplify}(\psi, I)$

}

Abduction Algorithm

- abduce yields formula E such that

$$I \wedge E \models \phi$$

  and E is consistent with I

- First, compute all variables in MSA of $I \Rightarrow \phi$ consistent with I, θ

- $\forall$-quantify variables not in the MSA and apply quantifier elimination

- Remove subparts of ψ implied by I

  - uses algorithm from SAS'10 paper "Small Formulas for Large Programs: ..."

$\text{abduce}(I, \phi)$ {

$V = \text{msa}(I \Rightarrow \phi,\ I)$

$\psi = \text{QE}(\forall \overline{V}.(I \Rightarrow \phi))$

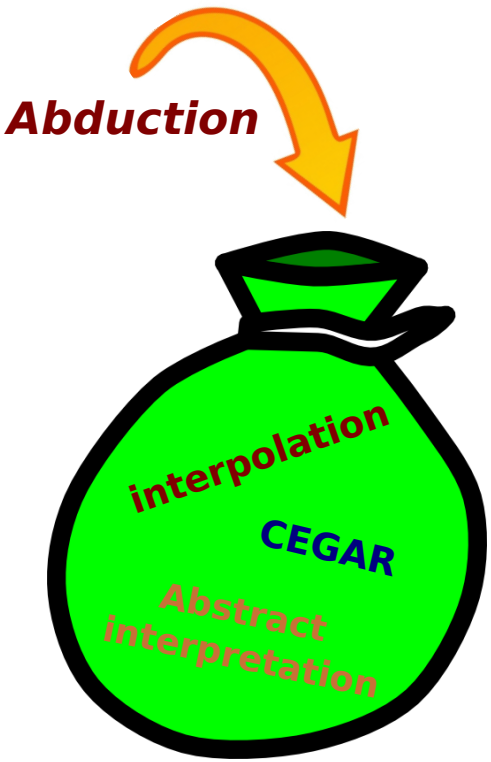
$\psi' = \text{simplify}(\psi, I)$

return ψ'

}

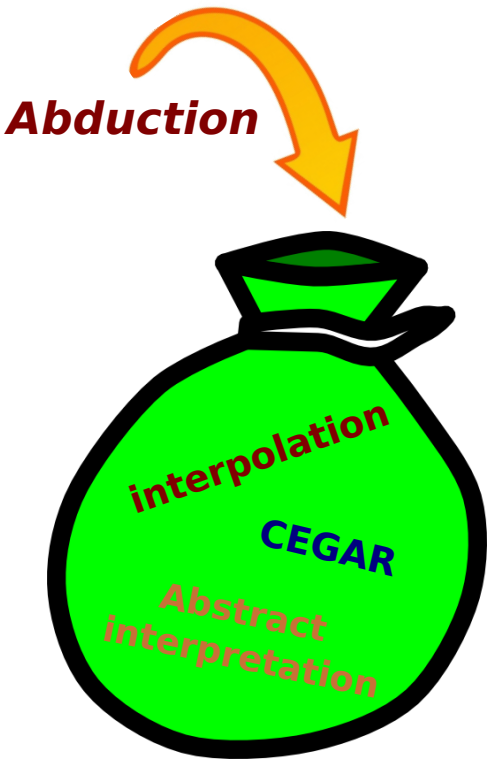
Abduction in Program Analysis

Useful technique to add to our bag of tricks; lots of applications!



Abduction in Program Analysis

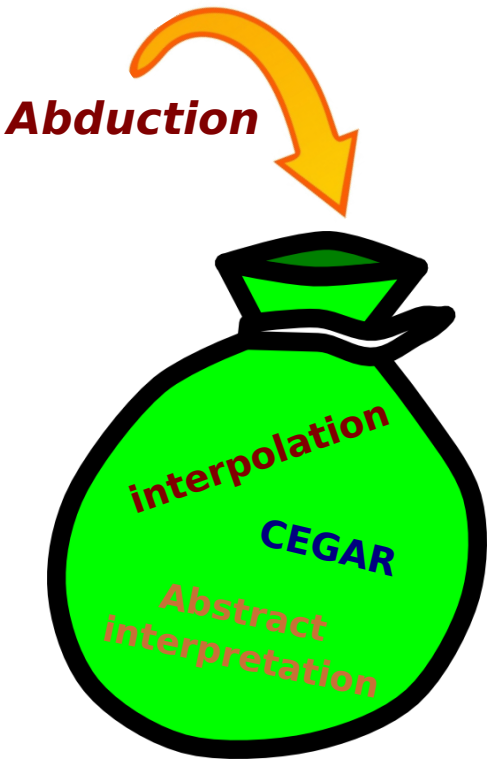
Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation

Abduction in Program Analysis

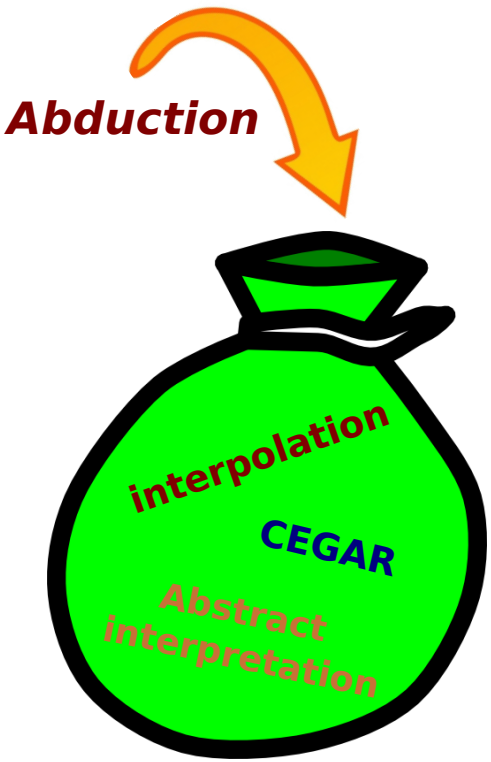
Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation
- Synthesis of compositional program proofs

Abduction in Program Analysis

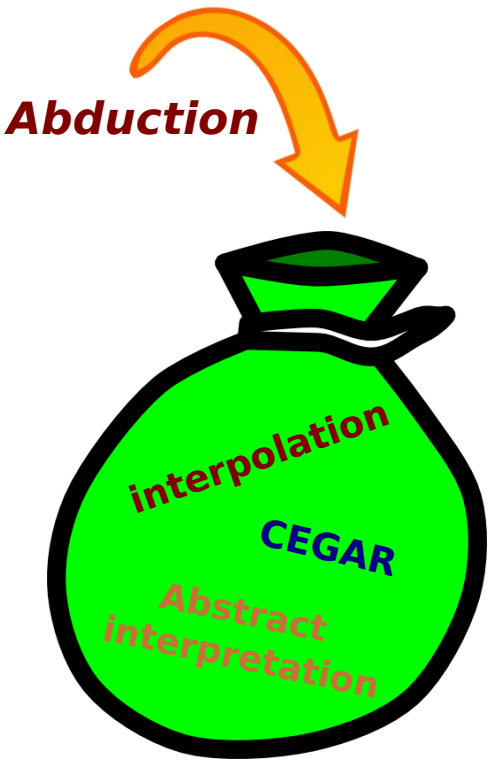
Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation
- Synthesis of compositional program proofs
- Inference of missing library specifications

Abduction in Program Analysis

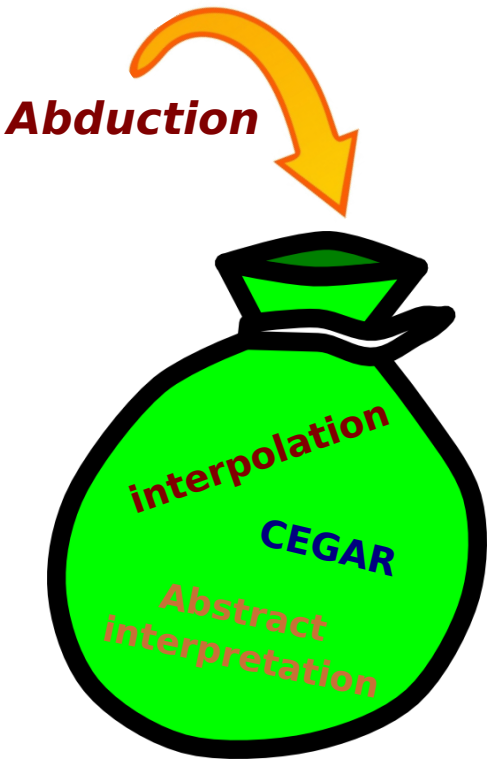
Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation
- Synthesis of compositional program proofs
- Inference of missing library specifications
- Explaining static analysis warnings to programmers

Abduction in Program Analysis

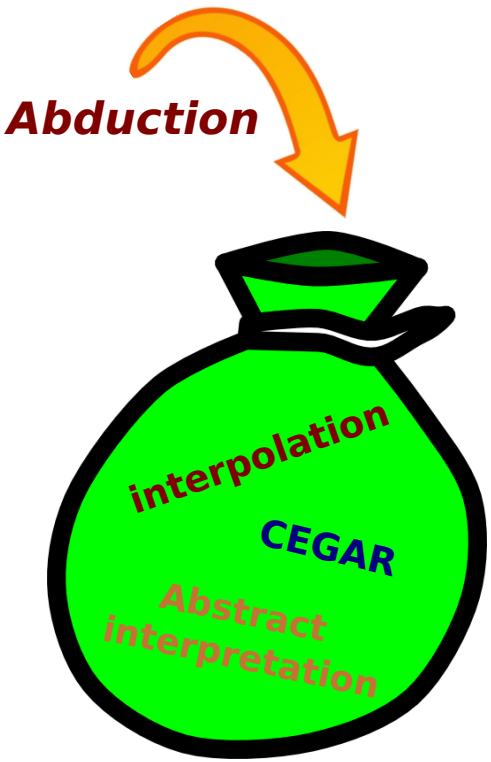
Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation
- Synthesis of compositional program proofs
- Inference of missing library specifications
- Explaining static analysis warnings to programmers
- Modular shape analysis using SL

Abduction in Program Analysis

Useful technique to add to our bag of tricks; lots of applications!



- Loop invariant generation
- Synthesis of compositional program proofs
- Inference of missing library specifications
- Explaining static analysis warnings to programmers
- Modular shape analysis using SL

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:

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

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:

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

- If all VCs are valid, found inductive invariants sufficient to verify program

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

Using Abduction for Loop Invariant Generation, cont.

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

Using Abduction for Loop Invariant Generation, cont.

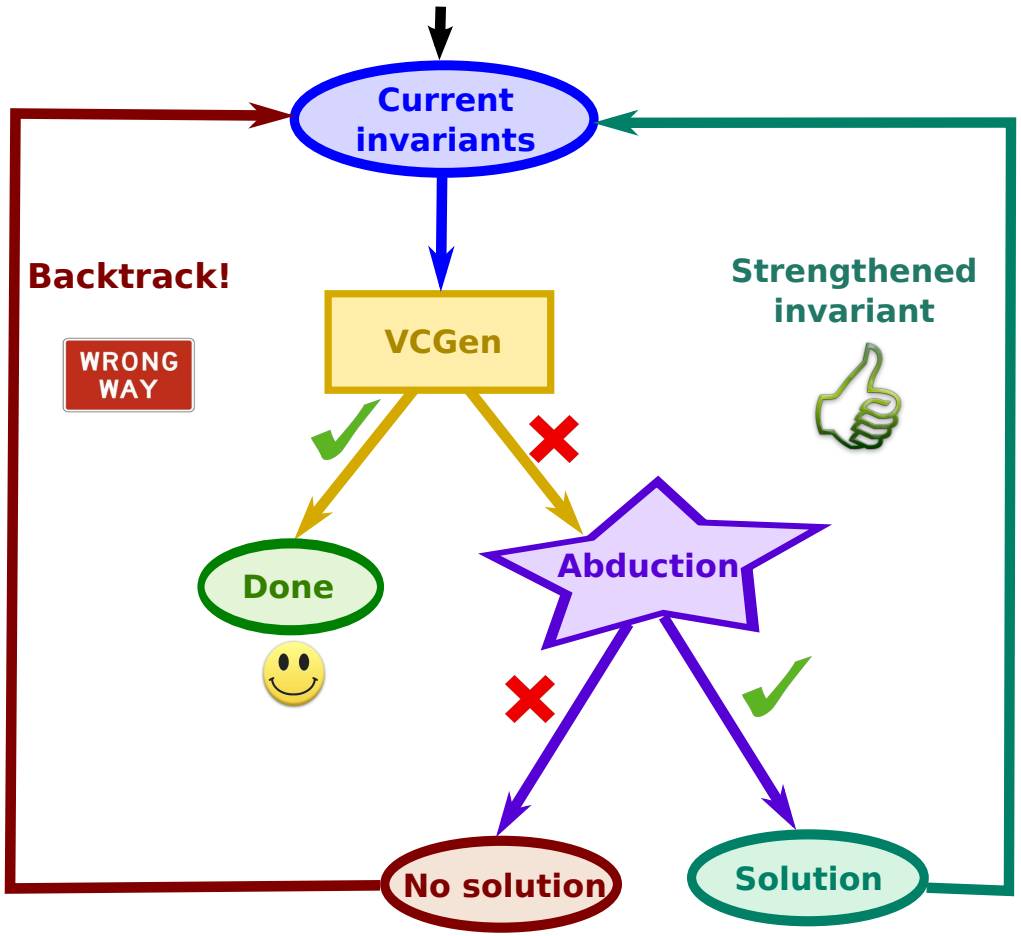
- If $I \wedge \neg C \Rightarrow Q$ is invalid, abduction produces auxiliary invariant ψ such that $I \wedge \psi$ is strong enough to show Q
- If $I \wedge C \Rightarrow wp(s, I)$ is invalid, abduction produces auxiliary invariant ψ such that I is inductive relative to ψ
- In either case, strengthen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

Using Abduction for Loop Invariant Generation, cont.

Since new invariant is a speculation, need to re-check VCs and might have to backtrack.

Using Abduction for Loop Invariant Generation, cont.

Since new invariant is a speculation, need to re-check VCs and might have to backtrack.



Some Experimental Results

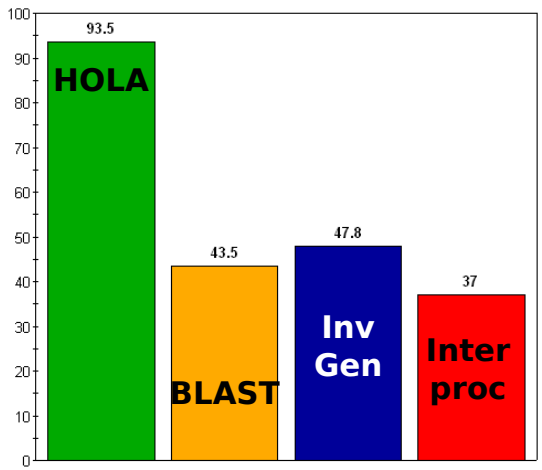
- Evaluated this technique on 46 loop invariant benchmarks

Some Experimental Results

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

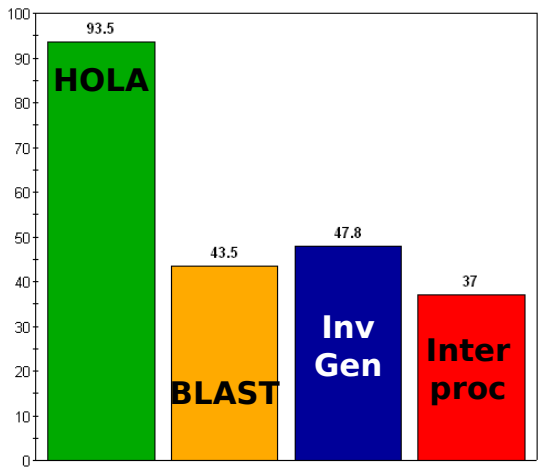
Some Experimental Results

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



Some 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

Abduction vs. Interpolation in Program Verification

Abduction vs. Interpolation in Program Verification

- Interpolation approaches utilize underapproximations
  - generalize from concrete traces
  - speculate invariants implied by underapproximations

Abduction vs. Interpolation in Program Verification

- Interpolation approaches utilize underapproximations
  - generalize from concrete traces
  - speculate invariants implied by underapproximations
- Abduction-based approach uses only overapproximations
  - speculate invariants that are consistent with overapproximations and sufficient to show desired goal

Abduction vs. Interpolation in Program Verification

- Interpolation approaches utilize underapproximations
  - generalize from concrete traces
  - speculate invariants implied by underapproximations
- Abduction-based approach uses only overapproximations
  - speculate invariants that are consistent with overapproximations and sufficient to show desired goal
- Dual concepts, so their combination could be synergistic
  - combine model checking with Hoare-style reasoning?

Questions?