Logical Abduction and its Applications in Program Verification

\[ \Gamma \land ? \models \phi \]

Isil Dillig
MSR Cambridge
Abduction: Opposite of deduction
What is Abduction?

- **Abduction**: Opposite of deduction
- **Deduction**: Infers valid conclusion from premises
Abduction: Opposite of deduction

Deduction: Infers valid conclusion from premises

Abduction: Infers missing premise to explain a given conclusion
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

$$\Gamma \land \psi \models \phi \quad \text{and} \quad \text{SAT}(\Gamma \land \psi)$$
Facts: “If it rains, then it is wet and cloudy”, “If it is wet, then it is slippery”:

$R \Rightarrow W \land C \land W \Rightarrow S$
Simple Example

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

- **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);
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
Suppose we know $x \geq n$

- e.g., from loop termination condition

Desired conclusion $x + y \geq 3n$

- property we want to prove
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 is the flip-side of Craig interpolation. Abduction explains why a formula is invalid; interpolation explains why it is valid.

In abduction, given an invalid \( \Gamma \Rightarrow \phi \), find \( \psi \) that explains why.

In interpolation, given a valid \( \Gamma \Rightarrow \phi \), if we can find \( \psi \) such that \( \Gamma \Rightarrow \psi \) and \( \psi \Rightarrow \phi \), then \( \psi \) 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
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 $\psi$ 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 $\psi$ that explains why.

- In interpolation, given valid $\Gamma \Rightarrow \phi$, if we can find $\psi$ s.t. $\Gamma \Rightarrow \psi$ and $\psi \Rightarrow \phi$, then $\psi$ 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 $\psi$ that explains why.

- In interpolation, given valid $\Gamma \Rightarrow \phi$, if we can find $\psi$ s.t. $\Gamma \Rightarrow \psi$ and $\psi \Rightarrow \phi$, then $\psi$ 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 \( \Gamma \land \exists \models \phi \) has infinitely many solutions.
In general, the abduction problem $\Gamma \land \phi \models \phi$ has infinitely many solutions.

**Trivial solution:** $\phi$, but not useful because does not take into account what we know.
Properties of Desired Solutions

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

- **Trivial solution:** $\phi$, but not useful because does not take into account what we know.

- 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:** 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
Want to compute logically weakest solutions with fewest variables
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
To find solutions with fewest variables, we use **minimum satisfying assignments** of formulas.
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.
Consider the following formula in linear integer arithmetic:

\[ x + y + w > 0 \lor x + y + z + w < 5 \]
Consider the following formula in linear integer arithmetic:

\[ x + y + w > 0 \lor x + y + z + w < 5 \]

Minimum satisfying assignment: \( z = 0 \)
Consider the following formula in linear integer arithmetic:

\[ x + y + w > 0 \lor x + y + z + w < 5 \]

- 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 $\psi$ such that $\Gamma \land \psi \models \phi$
Why Are MSAs Useful for Abduction?

- **Recall:** Want to find $\psi$ such that $\Gamma \land \psi \models \phi$

- This entailment can be rewritten as:

  $$\psi \models \Gamma \Rightarrow \phi$$
Why Are MSAs Useful for Abduction?

- **Recall:** Want to find $\psi$ such that $\Gamma \land \psi \models \phi$

- This entailment can be rewritten as:

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

- Hence, **MSA** of $\Gamma \Rightarrow \phi$ consistent with $\Gamma$ is a solution to abduction problem
Why Are MSAs Useful for Abduction?

- **Recall:** Want to find $\psi$ such that $\Gamma \land \psi \models \phi$

- This entailment can be rewritten as:

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

- Hence, **MSA** of $\Gamma \Rightarrow \phi$ consistent with $\Gamma$ 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 $\psi$ such that $\Gamma \land \psi \models \phi$

- This entailment can be rewritten as:

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

- Hence, **MSA** of $\Gamma \Rightarrow \phi$ consistent with $\Gamma$ 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

$$I \land E \models \phi$$

and $E$ is consistent with $I$
Abduction Algorithm

- **abduce** yields formula $E$ such that

$$I \land E \models \phi$$

and $E$ is consistent with $I$

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

```
abduce(I, \phi) {
    V = msa(I \Rightarrow \phi, I)
}
```
Abduction Algorithm

- **abduce** yields formula $E$ such that

  $$I \land E \models \phi$$

  and $E$ is consistent with $I$

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

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

```plaintext
abduce(I, \phi)  
\{ 
  V = msa(I \Rightarrow \phi, I) 
  \psi = QE(\forall V.(I \Rightarrow \phi)) 
\} 
```
Abduction Algorithm

- **abduce** yields formula $E$ such that

$$I \land E \models \phi$$

and $E$ is consistent with $I$

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

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

- Remove subparts of $\psi$ implied by $I$

```plaintext
abduce(I, \phi) {
    V = msa(I \Rightarrow \phi, I )
    \psi = QE(\forall V.(I \Rightarrow \phi))
    \psi' = simplify(\psi, I)
}
```
Abduction Algorithm

- **abduce** yields formula $E$ such that

$$I \land E \models \phi$$

and $E$ is consistent with $I$.

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

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

- Remove subparts of $\psi$ implied by $I$.

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

```plaintext
abduce(I, \phi)  {
    V = msa(I \Rightarrow \phi, I)
    \psi = QE(\forall V. (I \Rightarrow \phi))
    \psi' = simplify(\psi, I)
    return \psi'
}
```
Useful technique to add to our bag of tricks; lots of applications!
Useful technique to add to our bag of tricks; lots of applications!

- Loop invariant generation
Useful technique to add to our bag of tricks; lots of applications!

- Loop invariant generation
- Synthesis of compositional program proofs
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
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
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
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
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
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:  \( I \land \neg C \Rightarrow Q \)
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{align*}
  \text{Inductive : } & \quad I \land C \Rightarrow wp(s, I) \\
  \text{Sufficient : } & \quad I \land \neg C \Rightarrow Q
  \end{align*}

- 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:** \( I \land C \Rightarrow w_p(s, I) \)
  - **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 $I \land C \Rightarrow wp(s, I)$ is invalid, abduction produces auxiliary invariant $\psi$ such that $I$ is inductive relative to $\psi$. 
Using Abduction for Loop Invariant Generation, cont.

- 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 $I \land C \Rightarrow wp(s, I)$ 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 new invariant is a speculation, need to re-check VCs and might have to backtrack.
Since new invariant is a speculation, need to re-check VCs and might have to backtrack.
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:
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

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