8 Bi-Abduction

Bi-Abduction

Bi-abduction and Abstraction

Slide 1

- In the last lecture, we saw how frame inference lets us verify that the pre- and post-conditions and loop invariants of a given program are correct.
- Abstraction lets us infer loop invariants of programs automatically.
- Bi-abduction lets us infer pre- and post-conditions of programs automatically.
- With these techniques, tools are able to analyse millions of lines of code!

Bi-Abduction

Baby bi-abduction example

Slide 94

$\{\mathsf{emp} * ?M\}$
$\{\mathrm{x} \mapsto - * ?F\}$
$[\mathrm{x}] := 1;$
$\{\mathrm{x} \mapsto 1 * ?F\}$
$[\mathrm{y}] := 1;$

$\{???\}$

- Axiom of the current command:

$$\{\mathrm{x} \mapsto -\}\ [\mathrm{x}] := 1\ \{\mathrm{x} \mapsto 1\}$$

- Bi-abduction problem:

$$\mathsf{emp} * ?M \vdash \mathrm{x} \mapsto - * ?F$$

Abductive Inference

Slide 3

From philosophy:

> *"Abduction is the process of forming an explanatory hypothesis.*
> *It is the only logical operation which introduces any new idea."*

Charles Peirce, writing about the scientific process.

The Abduction problem

Slide 4

Given formulas P and Q, the abduction problem between P and Q consists in finding $?M$ such that

$$P * ?M \vdash Q$$

- $M =$ False and $M = Q$ are always solutions
- In general, we look for solutions that are minimal with respect to an ordering $\preceq$.

Slide 5

On the Quality of Abduction Solutions

Consider the abduction problem

$$\mathsf{emp} * {\color{red}?M} \vdash \mathtt{x} \mapsto -$$

- The ordering takes into account spatial minimality:

$$\mathtt{x} \mapsto - \;\preceq\; \mathtt{x} \mapsto - * \mathtt{y} \mapsto -$$

and logical minimality:

$$\mathtt{x} \mapsto - \;\preceq\; \mathsf{False}$$
$$\mathtt{x} \mapsto - \;\preceq\; \mathtt{x} \mapsto 10 \wedge \mathtt{x} = 12$$

- The $\preceq$-minimal solution to this abduction problem is $M = \mathtt{x} \mapsto -$.

Slide 6

Abduction Examples

$$\mathtt{x} \mapsto 1 * {\color{red}?M} \vdash \mathtt{y} \mapsto - * \mathsf{True}$$

$$\mathtt{x} \mapsto a, \mathtt{null} * {\color{red}?M} \vdash \mathsf{list}(\mathtt{x}) * \mathsf{list}(\mathtt{y})$$

Slide 7

The Bi-Abduction problem

Given formulas P and Q, the bi-abduction problem between P and Q consists in finding $?M$ and $?F$ such that

$$P * ?M \vdash Q * ?F$$

- $M =$ False or ($M = Q$ and $F = P$) are always solutions
- Again, we look for solutions that are minimal with respect to an ordering $\preceq$.
- One way to solve bi-abduction problems (used by tools):
  1. Find M such that
     $$P * ?M \vdash Q * \text{True}$$
  2. Find F such that
     $$P * M \vdash Q * ?F$$

Slide 8

Bi-Abduction Examples

$$\text{emp} * ?M \vdash \mathrm{x} \mapsto - * ?F$$

$$\mathrm{x} \mapsto 1 * ?M \vdash \mathrm{y} \mapsto - * ?F$$

Baby bi-abduction example

Slide 101

$\{\mathsf{emp} * ?M\}$
$\{\mathtt{x} \mapsto - * ?F\}$
$[\mathtt{x}] := 1;$
$\{\mathtt{x} \mapsto 1 * ?F\}$
$[\mathtt{y}] := 1;$

$\{???\}$

- Axiom of the current command:

$$\{\mathtt{x} \mapsto -\}\ [\mathtt{x}] := 1\ \{\mathtt{x} \mapsto 1\}$$

- Bi-abduction problem:

$$\mathsf{emp} * ?M \vdash \mathtt{x} \mapsto - * ?F$$

Bi-Abduction along a Path

Slide 10

- In the previous example, we did not need to restart from the top every time new pieces were added to the pre-condition.
- This is thanks to the following rule, derived from sequence, frame, and consequence, when C_1 does not modify variables in M:

$$\frac{\{P\}\ C_1\ \{Q\} \quad Q * M \vdash Q' \quad \{Q'\}\ C_2\ \{R\}}{\{P * M\}\ C_1; C_2\ \{R\}}$$

- Abducing pre-conditions on a path is sound for that path.
- What about non straigtht-line code, i.e., conditionals and loops?

Abducing Unsound Pre-Conditions

Slide 11

```
z := random();
if (z = 0) {
  [y] := 0;
  dispose(x);
} else {
  dispose(x);
  dispose(y);
}
```


Re-Execution

Slide 12

- Abducing pre-conditions inside a path is unsound for other paths in general.
- Bi-abduction yields only candidate pre-conditions.
- A re-execution phase (*à la* Smallfoot) prunes incorrect specifications.

Bi-Abduction and Abstraction: High-Level Overview

Inferred pre-condition:
$\{\}$

Slide 13

```
while(x ≠ null){
  t := x;
  x := [x + 1];
  dispose(t); dispose(t + 1);
}
```


Abstraction in Pre-Conditions and Re-Execution

Slide 14

- Abstraction replaces a candidate pre-condition A with A' such that $A \rightsquigarrow A'$.
- As a tentative rule:

$$\frac{A \rightsquigarrow A' \quad \{A\}\ C\ \{B\}}{\{A'\}\ C\ \{B\}}$$ **Unsound!**

- Abstracted pre-conditions also need to be re-executed.

Summary

Slide 15

Recipe for bi-abductive program analysis:

- Do symbolic execution
- Abduce missing resources
- Abstract to discover loop invariants
- Repeat until the post-condition is reached
- Check the candidate specifications by re-execution if needed

Bi-Abduction

Slide 16

$\{x \doteq x_0\}$

```
y := null;
while(x ≠ null)
{
  z := [x + 1];
  [x + 1] := y;
  y := x;
  x := z;
}
return y;
```

Attacking Large Programs

Slide 17

We can show **memory safety** for large programs:

> *the program does not dereference* `null` *or dangling pointers, and does not leak memory.*

for large programs. This reasoning is possible, due to the compositional reasoning given by bi-abduction.

Examples

OS device drivers ($< 15K$ lines), Apache ($1.7M$), the Linux kernel ($16M$), recently 15 bugs found in OpenSSL ($450K$ lines).

Still, we need to scale to industrial tools ...