Invariant Inference: Part II

Işıl Dillig

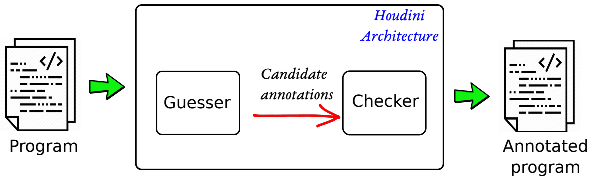
Motivation

- **Previous lecture:** Abstract interpretation
- **This lecture:** Other annotation inference techniques
  - Houdini Algorithm
  - Abduction-based inference

Houdini Overview

- Named after magician Harry Houdini
- Originally proposed as annotation assistant for ESC/Java
- Can generate both loop invariants and method contracts
- **"Guess-and-check" approach:** Guess some annotations, then check if they are correct

Houdini Workflow



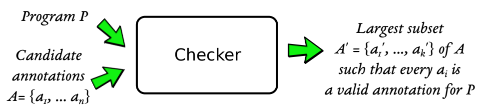
- The annotations produced by Houdini are sound (i.e., true loop invariants and method contracts)
- However, it is not complete ⇒ synthesized annotations may not be sufficient to prove property

Phase I: Guess Invariants

Many different techniques for guessing invariants:

- Mine candidates from source code based on heuristics
  - Expressions of the form $v_1\ op\ v_2$ or $v_1\ op\ c$, where v_1, v_2 are variables used in source code and c is an "interesting" constant
- Use dynamic analysis (Daikon approach)
  - Facts that have been observed while running the program
- All these techniques are heuristic in nature – not our main focus...

Phase II: Check Invariants



- The checker only throws out candidate annotations that are **refuted** by the verifier
- Loop invariant I is refuted if (1) it is not implied by loop precondition or (2) it is not preserved in the loop body
- Method precondition P is refuted if it does not hold at call site
- Method post-condition Q is refuted if $P \not\Rightarrow wp(M, Q)$

The Checking Algorithm

Initialization / *Fixed-point computation*

```
Check(P, Candidates){
   A := Candidates;
   while(true) {
      rft:= Verify(P, A);
      if(rft = ∅) break;
      A := A \ rft;
   }
   return A;
}
```

Verify returns refuted annotations

- **Soundness**: Upon termination, annotations in A are verified
- **Termination**: Terminates after $\leq |Candidates|$ iterations

Example: Finding Loop Invariants

- Consider the following very simple code example :

```
i := 0; j := -1;
while(i<1000) {
   j := i;
   i := i+1    }
```

Candidate invariants:

(A) `i >= 0` (B) `i = j`
(C) `i < 1000` (D) `i <= 1000`

- Candidate (B) is immediately refuted because not implied by pre-condition
- Candidate (C) is also refuted b/c

$$\not\models \{(A) \wedge (C) \wedge (D)\}\ Body\ \{(C)\}$$

- Algorithm terminates with inductive invariant:

$$i \geq 0 \wedge i \leq 1000$$

A Nice Property

- Given a set of candidate loop invariants, Houdini finds the **largest subset** that is inductive!
  - Largest subset $\Rightarrow$ Strongest invariant
- Why is this true?
  - Suppose Houdini returns set A, but there exists a $B \supset A$ such that $I_B = \bigwedge_{b_i \in B} b_i$ is inductive
  - This means the algorithm must have eliminated some $b_i \in B$
  - But this only happens if either (a) $Pre \not\Rightarrow b_i$ or (b) $\not\models \{I_B \wedge C\} Body \{b_i\}$
  - But neither option is possible since I_B is inductive.

Beyond Loops

- Houdini is not just limited to inferring loop invariants; can also infer method contracts
- Suppose we have a set P of candidate pre-conditions and a set Q of candidate post-conditions
- For every method, initialize pre-condition set to be P and post-cost condition set to be Q
- When analyzing method M:
  - If verification fails due to callee's precondition p, remove p from callee's pre-condition set
  - If verification fails because could not establish some $q \in Post(M)$, remove q from M's post-conditions

Example

- Consider the following procedures:

```
main() { foo(5, 0); }

foo(x, y) {                 bar(x, y) {
  if(x<=0) z:= y;             x := x-1;
  else z:= bar(x,y);          y := y+1;
  return z;                   return foo(x,y);
}                           }
```

Candidate pre-conditions:

(P1) `x>=0` (P2) `y>=0`
(P3) `x=y` (P4) `x>0`

Candidate post-conditions:

(Q1) `ret >= 0`
(Q2) `ret = 0`

- What are the contracts computed for `foo` and `bar`?

Example, cont.

```
main() { foo(5, 0); }

foo(x, y) {                 bar(x, y) {
  if(x<=0) z:= y;             x := x-1;
  else z:= bar(x,y);          y := y+1;
  return z;                   return foo(x,y);
}                           }
```

- When analyzing `main`, we eliminate P3 ($x = y$) for `foo` because `assert(5=0)` fails
- When analyzing `foo`, we eliminate Q2 ($ret = 0$) for `foo` because `assert(z=0)` fails
- When analyzing `foo`, we eliminate P3 ($x = y$) for `bar` because `assert(x=y)` fails at call site

Example, cont.

```
main() { foo(5, 0); }
foo(x, y) {                bar(x, y) {
  if(x<=0) z:= y;            x := x-1;
  else z:= bar(x,y);         y := y+1;
  return z;                  return foo(x,y);
}                          }
```

- When analyzing bar, we eliminate P4 ($x > 0$) for foo
- When analyzing bar, we eliminate Q2 ($ret = 0$) for bar
- Inferred contract for foo:

  $$requires(x \geq 0 \wedge y \geq 0)$$
  $$ensures(ret \geq 0)$$

- Inferred contract for bar:

  $$requires(x > 0 \wedge y \geq 0)$$
  $$ensures(ret \geq 0)$$

Discussion: Pros and Cons of the Houdini Approach

- **Pros:**
  - Can infer both loop invariants and method contracts
  - Infers strongest invariants over the candidate set
  - Conceptually simple; easy to implement
- **Cons:**
  - Only infers conjunctions of predicates in the candidate set
  - No guarantee that the inferred invariants are useful for verifying property

Motivation for Being Property-Directed

- Houdini does not leverage the property we are trying to prove
- But the property we are trying to prove gives strong hints about what invariants are useful!

```
while(i<j)
{
   ...
}
assert(i>=100)
```

j>=100 would be useful for proving the assertion!

- **Idea:** Use the property we are trying to prove to **guess** candidate invariants!

Abductive Reasoning

- Making educated guesses that support some observation is known as **abductive reasoning**
- Given known facts Γ and desired outcome ϕ, abductive inference finds "simple" explanatory hypothesis ψ such that:
  1. $\Gamma \wedge \psi \models \phi$ (i.e., explains conclusion)
  2. $\text{SAT}(\Gamma \wedge \psi)$ (i.e., it's consistent with known facts)
- In our case, the "desired outcome" is the property we are trying to prove
- "Known facts" can come from different sources – e.g., pre-condition, proven invariants, ...

Back to Previous Example

```
while(i<j) {...}
assert(i>=100)
```

- From loop condition, we have $i \geq j$ after the loop
- Want invariant that is **strong enough** to prove assertion
- Formulate this as an abduction problem:

  $$(1) \quad i \geq j \ \wedge \ ? \models i \geq 100$$
  $$(2) \quad SAT(i \geq j \wedge ?)$$

- Condition (2) says our guess is non-trivial (i.e., doesn't make assertion unreachable)
- $j \geq 100$ is a solution; so is $i \geq 100$ – not unique!

Desirable Properties

- An abductive reasoning problem has many solutions – what makes a "good" solution?
- **Occam's razor principle**: Want simplest explanation
- Many ways to define "simple", but one option:
  - Uses few variables (intuition: parsimonious invariants)
  - Logically weakest – the weaker the explanation, the less assumptions it makes

Quantifier Elimination

- In some first-order theories, we can automate abduction using **quantifier elimination (QE)**
- Given a quantified formula φ, quantifier elimination yields **quantifier-free** formula φ' such that $\varphi \Leftrightarrow \varphi'$
- Example theories that admit quantifier elimination:
  - Linear rational arithmetic
  - Linear integer arithmetic (extended with mod operator)

Automating Abduction via Quantifier Elimination

- Suppose we have premises φ and conclusion χ, and we want a hypothesis containing only variables V
- Then, the **logically weakest** quantifier-free explanation over variables V is given by:

$$\psi \equiv QE(\forall \overline{V}.\ \varphi \rightarrow \chi)$$

- Why is this a solution?
  - First, observe: $\varphi \land (\varphi \rightarrow \chi) \models \chi$
  - Second, we have $\psi \Rightarrow (\varphi \rightarrow \chi)$
  - Thus, $\varphi \land \psi \models \chi$

Back to Example

```
while(i<j) {...}
assert(i>=100)
```

- Our abduction problem:

$$\begin{array}{ll}(1) & i \geq j \land ? \models i \geq 100 \\ (2) & SAT(i \geq j \land ?)\end{array}$$

- Suppose we want solution containing just variable j:

$$QE(\forall i.\ (i \geq j \rightarrow i \geq 100)) \equiv j \geq 100$$

Back to Invariant Generation

- We can conjecture candidate invariants using abduction; then use Houdini as before
  - See our PLDI'18 paper by Ferles et al.
- **Advantages:**
  - Property-directed; conjectured invariants known to be useful
  - Candidate invariants can have disjunctions; so not limited to conjunctive invariants

Can Do Even Better!

- This approach has some advantages, but it still suffers from one shortcoming of the Houdini algorithm
  - Houdini can discard true loop invariants if they are not inductive
- **Idea:** Use abduction to strengthen loop invariants to make them inductive!

Motivating Example

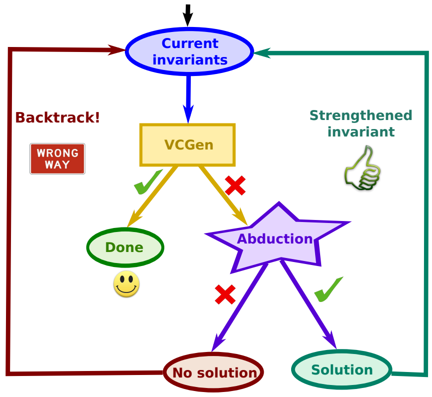
```
i:=1; j:=100;
while(i<j) {
  if(*) j := j+i;
  i:=i*2;
}
assert(i>=100)
```

- Using abduction, we can generate $j \geq 100$ as a candidate invariant
- But since it's not inductive (why?), Houdini will reject it
- But now we can use abduction to figure out how to **strengthen** it!

$$(i < j \ \land \ j \geq 100 \land\ ?) \Rightarrow wp(Body, j \geq 100)$$

- **Solution:** $i \geq 0$
- New candidate invariant is now $j \geq 100 \land i \geq 0$, which **is** inductive!

The Full Algorithm



Dillig et al. OOPSLA'13

Comparison with Houdini

Similarities:

- Also guess-and-check approach
- Uses verifier to check correctness of annotations

Differences:

- Property-directed; guesses generated using abduction
- Generates new candidate invariants on-line rather than statically up-front
- Does not have termination guarantees
  - But can bound number of strengthening steps