Invariant Inference: Part II

İsıl 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 \( v_1 \text{ op } v_2 \) or \( v_1 \text{ op } c \), 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

\[
\text{Candidate Invariants: } \begin{cases} 
(A) & i \geq 0 \\
(B) & i = j \\
(C) & 1 < 1000 \\
(D) & 1 \leq 1000 
\end{cases}
\]

\[
\text{Verify returns refuted annotations}
\]

A Nice Property

- Given a set of candidate loop invariants, Houdini finds the \textit{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 \( I_B = \bigwedge_{b \in B} b \) is inductive
  - This means the algorithm must have eliminated some \( b_i \in B \)
  - But this only happens if either (a) \( \forall \{I_B \land C\}. \text{Body}\{b_i\} \)
  - But neither option is possible since \( I_B \) is inductive.

Example: Finding Loop Invariants

- Consider the following very simple code example:

\[
\begin{align*}
\text{main} & \colon \text{foo}(5, 0); \\
\text{foo} & \colon \{ \text{if}(x \leq 0) \text{z} := y; \text{else} \text{z} := \text{bar}(x,y); \text{return} \text{z}; \} \\
\text{bar} & \colon \{ \text{x} := x-1; \text{y} := y+1; \text{return} \text{foo}(x,y); \}
\end{align*}
\]

- What are the contracts computed for \( \text{foo} \) and \( \text{bar} \)?

Example, cont.

Beyound 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 \( q \in \text{Post}(M) \), remove \( q \) from \( M \)’s post-conditions

Example

- Consider the following procedures:

\[
\begin{align*}
\text{main} & \colon \text{foo}(5, 0); \\
\text{foo} & \colon \{ \text{if}(x = y) \text{z} := y; \text{else} \text{z} := \text{bar}(x,y); \text{return} \text{z}; \} \\
\text{bar} & \colon \{ \text{x} := x-1; \text{y} := y+1; \text{return} \text{foo}(x,y); \}
\end{align*}
\]

- What are the contracts computed for \( \text{foo} \) and \( \text{bar} \)?

- When analyzing \( \text{main} \), we eliminate \( P_3 (x = y) \) for \( \text{foo} \) because \( \text{assert}(x=0) \) fails

- When analyzing \( \text{foo} \), we eliminate \( Q_2 \) (\( \text{ret} = 0 \)) for \( \text{foo} \) because \( \text{assert}(x=0) \) fails

- When analyzing \( \text{bar} \), we eliminate \( P_3 (x = y) \) for \( \text{bar} \) because \( \text{assert}(x=y) \) fails at call site
Example, cont.

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

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!

```c
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:
  
  \[
  \begin{align*}
  &1. \quad i \geq j \land ? = i \geq 100 \\
  &2. \quad SAT(i \geq j \land ?)
  \end{align*}
  \]
▶ 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!

Abductive Reasoning

- Making educated guesses that support some observation is known as abductive reasoning
- Given known facts \( \Gamma \) and desired outcome \( \phi \), abductive inference finds "simple" explanatory hypothesis \( \psi \) such that:
  1. \( \Gamma \land \psi \models \phi \) (i.e., explains conclusion)
  2. \( SAT(\Gamma \land \psi) \) (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, ...

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 $\varphi$, quantifier elimination yields a quantifier-free formula $\varphi'$ 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 $\varphi$ and conclusion $\chi$, and we want a hypothesis containing only variables $V$.
- Then, the logically weakest quantifier-free explanation over variables $V$ is given by:
  $$\psi \equiv QE(\forall V. \varphi \rightarrow \chi)$$
- 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

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

- Our abduction problem:
  1. $i \geq j \land ? \models i \geq 100$
  2. SAT($i \geq j \land ?$)
- Suppose we want solution containing just variable $j$:
  $$QE(\forall i. (i \geq j \rightarrow i \geq 100))$$
  $$\equiv j \geq 100$$

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

- 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

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