Program Verification

Plus: “Program Verification with Probabilistic Inference” by Sumit Gulwani and Nebojsa Jojic
What is Program Verification?

• Simple idea: Prove that a program behaves correctly, given some specification

• What kinds of specifications?
  
  Invariants: precondition and postcondition

• **Hoare Triple** – \{A\} P \{B\}
What is Program Verification?

• Simple idea: Prove that a program behaves correctly, given some specification

• What kinds of specifications?
  
  Invariants: precondition and postcondition

• Hoare Triple – \{A\} P \{B\}
What is Program Verification?

- Simple idea: Prove that a program behaves correctly, given some specification
- What kinds of specifications?
  - Invariants: precondition and postcondition
- Hoare Triple – \{A\} P \{B\}
What is Program Verification?

• Simple idea: Prove that a program behaves correctly, given some specification

• What kinds of specifications?

  Invariants: precondition and postcondition

• **Hoare Triple** – \{A\} P \{B\}
Invariants and Program State

- Precondition and postcondition are special cases of program invariants, denoted $\phi$.
- A program state $\sigma$ is a mapping of variables in the program to values.
- Invariants restrict the set of valid program states at a specific point in execution.

$\sigma \models \phi$ means “$\sigma$ is a valid state given $\phi$” or “$\sigma$ satisfies $\phi$.”
Validity of Hoare Triple

• A program is correct w.r.t. invariants if the Hoare triple is valid

\[ \models \{A\} \mathbf{P} \{B\} \]

“For all \( \sigma \), if \( \sigma \models A \) then \( \sigma' \) is the state after executing \( \mathbf{P} \), and \( \sigma' \models B \).”

• Not feasible to look at every state. Can we prove this another way?
Proof of Validity

- Find $\varphi$s such that each individual statement + invariants forms a valid Hoare triple
- Require $pre \Rightarrow \varphi_0$ and $\varphi_4 \Rightarrow post$
- How can we find these invariants?
- One option: backwards analysis “pushes” invariants backwards past statements.

$pre: x = 0$

\[\varphi_0\]

\[\varphi_1\]

\[\varphi_3\]

while $(x < 5)$

\[\varphi_3\]

$\varphi_4$

$\varpost: x = 5$

$pre: x = 0$

$\varphi_0$

$\varphi_1$

$\varphi_3$

$x++$

$\varphi_4$

$\varpost: x = 5$
Pushing Invariants
(the first one’s always free)

• Given postcondition(s) for a statement $s$, find an invariant s.t. all states satisfying the invariant prior $s$ satisfy the postcondition(s) after $s$

• Many possible invariants (e.g. $\textbf{false}$ trivially suffices). Choose the \textit{weakest} one

What does it mean for an invariant to be “\textbf{weak}” or “\textbf{strong}”?
Strengths and Weaknesses

• \( \varphi' \) is weaker than \( \varphi \) if \( \varphi \Rightarrow \varphi' \)

• What does \( \varphi \Rightarrow \varphi' \) mean?
  - For all \( \sigma \models \varphi \), \( \sigma \models \varphi' \)
  - Matches our natural understanding of \( \Rightarrow \)

• Intuition: the more valid program states an invariant allows, the weaker it is.
Example

• $\varphi_0$ must be chosen so $\varphi_1$ is valid after assignment

• Many options (e.g. \{x \geq 3\}). We choose the one which has the most valid states:

  $\varphi_0: x \geq 1$

• Note, for all other $\varphi$ that work, $\varphi \Rightarrow \varphi_0$

• We call $\varphi_0$ the “weakest precondition”
Backwards analysis

• Initialize all invariants to true
• Push invariants back until convergence
• Produces $\varphi_0$ at beginning of program. Must prove that precondition $\implies \varphi_0$

• This is undecidable! (Thanks, Gödel...)
• Solution: restrict domain of invariants
Underapproximation

- When domain of invariants is restricted, we must **underapproximate** invariant
  - Precondition we *want* may not be expressible in domain
  - **We must choose stronger invariant** (i.e. fewer valid states)
    - This may preclude finding proof
      - Precondition may not imply $\varphi_0$
Example

- Domain: conjunctions of inequalities (i.e. convex polyhedra)

\[ \varphi_0 \]

\[ \text{if } (x > 0) \]

\[ \varphi_1 \]

\[ \varphi_2 \]

\[ \varphi_2: y \geq 2 \land y \leq 4 \]

\[ \varphi_1: y \geq 0 \land y \leq 2 \]
Example

- Weakest $\varphi_0$: $((x > 0) \land \varphi_1) \lor (x \leq 0) \land \varphi_2)$
Example

\[ \varphi_0: (x > 0 \land y \geq 0 \land y \leq 2) \lor (x \leq 0 \land y \geq 2 \land y \leq 4) \]

- This can’t be expressed in abstract domain! Must choose different invariant
- Underapproximation sound, but loses precision
- Some valid preconditions can’t be verified
  - e.g. \( \{x = 1 \land y = 1\} \)
Wrapping up

• Similar procedure for forward analysis
  • Initialize to false, push forward using strongest postcondition
  • Show that final $\varphi \Rightarrow$ program’s postcondition
  • May overapproximate

• Analysis produces correctness proof: $\vdash \{A\} P \{B\}$
• This is sound, but not complete:

  $\vdash \{A\} P \{B\} \Rightarrow \vdash \{A\} P \{B\}$
On to the Paper!
Program Verification: Rethought

• Recall: a program is verified when a proof is found establishing the postconditions given the preconditions

• This is a global condition

• Alternate formulation: a proof is valid when all $\varphi$s are locally consistent
Local Consistency

• Consider a program point \( \pi_k \)

• Weakest precondition of successors: \( \text{pre}(\pi_k) \)

• Strongest postcondition of predecessors: \( \text{post}(\pi_k) \)

• Define \( \text{pre}(\pi_{\text{exit}}) \) to be postcondition of program and \( \text{post}(\pi_{\text{entry}}) \) to be its precondition.

• \( \varphi_k \) is **locally consistent** when:

\[
\text{post}(\pi_k) \Rightarrow \varphi_k \land \varphi_k \Rightarrow \text{pre}(\pi_k)
\]
Main Idea of Paper

- Randomly choose \( \varphi \)s until all are locally consistent!
- Deciding if \( \varphi \) is locally consistent does not require global knowledge
- But may take unbounded time
- Apply *probabilistic inference* to converge on \( \varphi \)s faster!
Quick Detour:
Need to Climb a Hill
Probabilistic Inference

• Given a probability density function (pdf) of K variables:

\[ p(x_1, x_2, \ldots, x_K) \]

Can we find values for all \( x_i \)s such that \( p \) is maximized?
Gibbs Sampling

• Pick arbitrary \( x_i \) and consider \textbf{conditional distribution function} (cdf):

\[
p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)
\]

• Choose a value for \( x_i \) according to probabilities of cdf ("Draw a sample from cdf")

• Choose another \( x_i \) and continue

• Will converge to optimal values for variables
Analogy with Hill Climbing

• Classic AI search technique:
  • Pick a variable $x_i$ and change it to improve target function
  • With some (small) probability, choose something other than best value for $x_i$
  • Avoids local maxima
Now back to your regularly scheduled program verification
Inconsistency Measure

• Define an inconsistency measure, $M$ for invariants $\varphi$ and $\varphi'$

  • Intuition: The closer $\varphi$ is to being stronger than $\varphi'$, the more consistent the two invariants are

  • $M(\varphi, \varphi') = 0$ iff $\varphi \Rightarrow \varphi'$ (no inconsistency)

  • As $\varphi$ gets stronger, consistency increases

  • As $\varphi'$ gets stronger, consistency decreases
Local Consistency as a Function

• Local inconsistency for a given \( \varphi \) at program point \( \pi_k \)

\[
L(\varphi, \pi_k) = M(\text{post}(\pi_k), \varphi) + M(\varphi, \text{pre}(\pi_k))
\]

• Note that when \( L(\varphi, \pi_k) = 0 \)

\[
\text{post}(\pi_k) \Rightarrow \varphi \land \varphi \Rightarrow \text{pre}(\pi_k)
\]

so \( \varphi \) is locally consistent
Verification as Optimization

• Now have a real-valued measure of local consistency at each program point

• Construct function $f$

  $$f(\varphi_0, \varphi_1, \ldots \varphi_K)$$

  using $L(\varphi_i, \pi_i)$ such that $f$ is maximized when all $\varphi$s are locally consistent

• Can apply Gibbs sampling to this function!
Operation of algorithm

• Initialize all $\varphi$s to $\bot$

• Pick a random program point $\pi_k$ whose invariant $\varphi_k$ is not locally consistent

• Choose $\varphi$ to minimize inconsistency at $\pi_k$
  • But with some probability, choose other $\varphi$

• Update $\varphi_k = \varphi$

• Continue until no local inconsistency
Key Algorithm Features

- Only local decisions made at any point
  - Local inconsistency only related to small number of program points
- Uses both forward and backward information
  - $L$ involves both predecessors and successors
- Avoids precision issues of standard analyses
Example, take two

• Consider choosing appropriate invariant for $\varphi_0$

\[
\text{if } (x > 0) \quad \begin{align*}
\varphi_1 & : y \geq 0 \land y \leq 2 \\
\varphi_2 & : y \geq 2 \land y \leq 4 \\
\varphi_p & : y = 1 \land x = 1
\end{align*}
\]
Example, take two

- Consider choosing appropriate invariant for \( \varphi_0 \)

\[
\text{post}(\pi_0) = \varphi_p
\]

\[
\text{pre}(\pi_0) = ((x > 0) \land \varphi_1) \lor (x \leq 0) \land \varphi_2)
\]
Example, take two

- Consider choosing appropriate invariant for $\varphi_0$
  
  $\text{post}(\pi_0) = \varphi_p$
  
  $\text{pre}(\pi_0) = ((x > 0) \land \varphi_1) \lor (x \leq 0) \land \varphi_2$)

- Desire to minimize inconsistency with both post and pre leads to correct choice of $\varphi_0$

\[
\begin{align*}
\varphi_p & : y = 1 \land x = 1 \\
\varphi_0 & : (x > 0 \land y \geq 0 \land y \leq 2) \\
\varphi_1 & : y \geq 0 \land y \leq 2 \\
\varphi_2 & : y \geq 2 \land y \leq 4
\end{align*}
\]
Forward + Backward > Standing Still

• Essentially, analysis uses information from predecessors to “guide” its underapproximation (equivalently, uses information from successors to guide overapproximation)

• Produces better results than many existing analyses
Random Choices are Good

- Random choices

- Which program point to update: Finding the proper invariants may require very specific sequence of updates. This is almost impossible to determine normally.

- What invariant to use: Given a set of equally inconsistent choices, random selection will eventually choose the right invariant.

- Upshot: Randomness leads to proper result when there is no clear strategy.
Some Results

• Abstract domain: Boolean combinations of difference constraints with \((m \times n)\) template
  
  • \(m\) conjuncts, each with at most \(n\) disjuncts

• \(M(\varphi, \varphi')\) where \(\varphi'\) is the conjunction of several clauses:

\[
M(\phi, \bigwedge_{i=1}^{m} C_i) = \sum_{i=1}^{m} \frac{1}{m} \times M(\phi, C_i) \quad M(\bigvee_{j=1}^{k} D_j, C_i) = \sum_{j=1}^{k} \frac{1}{k} \times M(D_j, C_i)
\]
Test Program and Proof

\[ \phi_{pre}: x = 0 \]

\[ \pi_{entry} \]

\[ y := 50; \]

\[ \pi_1 \]

\[ x < 100 \]

\[ \pi_2 \]

\[ \pi_{exit} \]

\[ \pi_3 \]

\[ x < 50 \]

\[ \phi_{post}: y = 100 \]

\[ x := x+1; \]

\[ \pi_4 \]

\[ \pi_5 \]

\[ \pi_6 \]

\[ x := x+1; \]

\[ y := y+1; \]

\[ \pi_7 \]

\[ \pi_8 \]

<table>
<thead>
<tr>
<th>Program Point</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi_0 )</td>
<td>( x = 0 )</td>
</tr>
<tr>
<td>( \pi_1 )</td>
<td>((y = 50) \land (x = 0))</td>
</tr>
<tr>
<td>( \pi_2 )</td>
<td>((y = 50 \lor x \geq 50) \land (y = x \lor x &lt; 50) \land (y = 100 \lor x &lt; 100))</td>
</tr>
<tr>
<td>( \pi_3 )</td>
<td>((y = 50 \lor x \geq 50) \land (y = x \lor x &lt; 50) \land (y = 99 \lor x &lt; 99))</td>
</tr>
<tr>
<td>( \pi_4 )</td>
<td>((y = 50) \land (x &lt; 50))</td>
</tr>
<tr>
<td>( \pi_5 )</td>
<td>((y = 50) \land (x &lt; 51))</td>
</tr>
<tr>
<td>( \pi_6 )</td>
<td>((x \geq 50) \land (y = x \lor x &lt; 50) \land (y = 99 \lor x &lt; 99))</td>
</tr>
<tr>
<td>( \pi_7 )</td>
<td>((x &gt; 50) \land (y = x \lor x &lt; 51) \land (y = 100 \lor x &lt; 100))</td>
</tr>
<tr>
<td>( \pi_8 )</td>
<td>((y = 50 \lor x \geq 50) \land (y = x \lor x &lt; 50) \land (y = 100 \lor x &lt; 100))</td>
</tr>
<tr>
<td>( \pi_9 )</td>
<td>(y = 100)</td>
</tr>
</tbody>
</table>

Existing techniques unable to verify this program!
How long does it take?

- Performed multiple runs of prover
- Histogram of tests which took a certain number of updates per $\pi$
- Black bar: all $\pi$s initialized to $\perp$
- Gray bar: use previously found proof on slightly modified program
Discussion

• Could there be some benefit to a more directed search? (e.g. choosing which program point to update in a more systematic way)

• Is this randomized approach useful in other domains? Can it be applied to any dataflow/abstract interpretation problem?