A Constraint-based Approach to Program Analysis and Property Inference

Saurabh Srivastava
University of Maryland, College Park

Sumit Gulwani
Ramarathnam Venkatesan
Microsoft Research, Redmond

Jeffrey Foster
Michael Hicks
University of Maryland, College Park
Program Analysis as SAT Solving

• Core Program Analyses:
  – Verification: Inferring Loop Invariants
  – Weakest Precondition Inference
  – Strongest Postcondition Inference

• SAT solving
  – Well engineered with fast SAT solvers available

• This work
  – Program analyses/fix-points as SAT constraints
  – Using off-the-shelf solvers for solutions
  – Solve core analyses in a unified non-iterative framework
Evaluation

• Domains
  – Linear Arithmetic
    • E.g. \( x > 0 \lor y < 0 \)
  – Predicate Abstraction
    • E.g. \( x \geq 0 \land b = A[x] \)

• Implementation
  – Verification, Weakest Precondition, Strongest Postcondition
  – Termination, Loop Bounds, Non-termination, Bug-finding
  – Analyzed difficult program instances of all these problems
Approach Outline

CFG to VC  VC to SAT

\[ a_1 \Rightarrow b_1 \quad \phi_1 \]
\[ a_2 \Rightarrow b_2 \quad \phi_2 \]
\[ \vdots \]
\[ a_n \Rightarrow b_n \quad \phi_n \]

SAT Solver

Invariants

Procedure Summaries

Weakest Precondition

Strongest Postcondition

Very general. Can encode:
- Termination and Loop Bounds
- Non-termination
- Bug-finding

Constraints on (unknown) pre/post
Constraints From Programs

Hoare triple \{\text{pre}\} \text{program} \{\text{post}\}

Verification constraints:
\exists I
\forall X \quad \text{pre} \Rightarrow I
\forall X \quad I \land \neg c \Rightarrow \text{post}
\forall X \quad (I \land c) [S] \Rightarrow I

Note: Because of the \forall X we cannot directly dump these to a solver. That will give \exists X
Example

\{x=0 \land n \geq 0\}

while \(x < n\) do
  \(x++\)
\{x=n\}

\{x=0 \land n \geq 0\}

\exists I

\forall x,n \quad \text{pre} \Rightarrow I

\forall x,n \quad I \land \neg c \Rightarrow \text{post}

\forall x,n \quad (I \land c) [S] \Rightarrow I

\exists I

\forall x,n \quad x=0 \land n \geq 0 \Rightarrow I

\forall x,n \quad I \land \neg x<n \Rightarrow x=n

\forall x,n \quad (I \land x<n) [x++] \Rightarrow I
Discovering Invariants: Tough

- Inferring invariants is difficult
- Iterative fixed-point based:
  - Abstract Interpretation
  - Model checking
  - Probabilistic Inference
- Alternative using constraint solving
  - Encode fixed-point in SAT instance
  - SAT instance generated from local VCs
Approach Outline

CFG to VC  
VC to SAT

CFG

Simple Paths  
Verification Conditions

\[ \begin{align*}
\phi_1 & \Rightarrow a_1 \Rightarrow b_1 \\
\phi_2 & \Rightarrow a_2 \Rightarrow b_2 \\
& \vdots \\
\phi_n & \Rightarrow a_n \Rightarrow b_n
\end{align*} \]

Sat Solver

Invariants

\begin{align*}
x=0 \land n \geq 0 & \Rightarrow I \\
I \land \neg x<n & \Rightarrow x=n \\
(I \land x<n) [x++] & \Rightarrow I
\end{align*}
VC to SAT Constraint

\[
\begin{align*}
a_1 & \Rightarrow b_1 \\
a_2 & \Rightarrow b_2 \\
\vdots \\
a_n & \Rightarrow b_n \\
\end{align*}
\]

Verification Conditions

\[
\begin{align*}
\text{Linear Arithmetic} & \\
\text{Predicate Abstraction} & \\
\end{align*}
\]

SAT Constraint

\[
\begin{align*}
\phi_1 \\
\phi_2 \\
\vdots \\
\phi_n \\
\end{align*}
\]

- Each domain needs a specialized conversion
  - Linear Arithmetic: Farkas’ lemma
  - Predicate Abstraction: Predicate Cover

\[
\begin{align*}
x=0 & \land n \geq 0 \Rightarrow I \\
I \land \neg x<n & \Rightarrow x=n \\
(I \land x<n) [x++] & \Rightarrow I
\end{align*}
\]
Farkas’ Lemma

• Formally:
  – If e, e_i’s are linear in some variables
  – Let U be \( \land_i \{e_i \geq 0\} \implies (e \geq 0) \)
  – Let E be
    \[ \exists \lambda_0, \lambda_i \geq 0 . \ (\lambda_0 + \lambda_1 e_1 + \lambda_2 e_2 \ldots \equiv e) \]
  – Then \( U \iff E \)

• Example:
  – Let U: \( (x \geq 0 \land y \geq 0) \implies (x+2y \geq 0) \)
  – Let E: \( \lambda_0 + \lambda_1 x + \lambda_2 y \equiv x+2y \)
  – U holds iff \( \exists \lambda_0, \lambda_1, \lambda_2 \geq 0 \) satisfying E
Example: Linear Arithmetic

- I comes from the domain of Linear Arithmetic
- Assume a boolean form, lets say one conjunct
- I can be written as \((a_1+a_2.x+a_3.n \geq 0)\)
- Then \(\exists I\) can be translated to \(\exists a_i\)

\[
\exists a_i \forall x,n \begin{cases} 
    x=0 \land n \geq 0 & \implies (a_1+a_2.x+a_3.n \geq 0) \\
    (a_1+a_2.x+a_3.n \geq 0) \land \neg x<n & \implies x=n \\
    ((a_1+a_2.x+a_3.n \geq 0) \land x<n) [x++] & \implies (a_1+a_2.x+a_3.n \geq 0)
\end{cases}
\]

- Next, we apply Farkas’ lemma and convert \(\forall x,n\) to \(\exists \lambda_i\)

\[
\land_i \{e_i \geq 0\} \implies (e \geq 0)
\]

\[
\exists \lambda, \lambda_i \geq 0 \quad e \equiv \lambda + \sum_i \lambda_i e_i
\]
Applying Farkas’ Lemma

∀ x, n → ∃ λ_i

\[ x=0 \land n \geq 0 \implies (a_1 + a_2 \cdot x + a_3 \cdot n \geq 0) \]

\[ \lambda_0 + \sum_{i=1}^3 \lambda_i \cdot x_i \]

∃ λ_i ≥ 0  λ_0 + λ_1 \cdot x - λ_2 \cdot x + λ_3 \cdot n ≡ a_1 + a_2 \cdot x + a_3 \cdot n

- Identity that holds for all x, n
- Collect coefficients of x, n
- (Quadratic) constraints in λ_i, a_i
- Bit-vector modeling to get SAT
Predicate Abstraction

- Given a fixed set of predicates \( P \)
  - E.g. \( \{x \leq n, x \geq n, x \geq 0, n \geq 0\} \)
- Find an Invariant that consists of preds in \( P \)
  - E.g. if invariant is \( (0 \leq x \leq n) \) then \( \{x \leq n, x \geq 0, n \geq 0\} \)
- Boolean indicator \( b_p \) for predicate \( p \in P \)
  - E.g. \( \{b_{x \leq n}, b_{x \geq n}, b_{x \geq 0}, b_{n \geq 0}\} \)
  - \( p \in \text{Invariant} \Leftrightarrow b_p = \text{true} \)
  - Assignment to \( b_p \)'s gives predicates in invariant
Translation to SAT

• Predicate Cover for formula F
  – Weakest formula over predicates that implies F
    • PC(x=n) over \{x\leq n, x\geq n, x\geq 0, n\geq 0\} is \(x\leq n \land x\geq n\)

• Local constraints on \(b_p\)’s using VCs
  – Invariant on LHS needs predicate cover:
    • \((I \land \neg x<n \Rightarrow x=n)\) induces the clause \(b_{x\leq n}\)
  – Invariant on RHS implies constraints directly
    • \((x=0 \land n\geq 0 \Rightarrow I)\) induces the clause \(\neg b_{x\geq n}\)

• Solution to all clauses gives invariant
## Experiments – Linear Arithmetic

<table>
<thead>
<tr>
<th>Type of Analysis</th>
<th>Example*</th>
<th>Formula size (# clauses)</th>
<th>Constraint Generation (secs)</th>
<th>Constraint Solving (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification</td>
<td>x&lt;n Loop</td>
<td>2k</td>
<td>0.09</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Disjunctive</td>
<td>345k</td>
<td>0.30</td>
<td>230</td>
</tr>
<tr>
<td>Weakest Precondition</td>
<td>Two Minima</td>
<td>90k</td>
<td>0.15</td>
<td>0.6-1.5</td>
</tr>
<tr>
<td></td>
<td>x+i, y+j</td>
<td>75k</td>
<td>0.15</td>
<td>0.6-2.9</td>
</tr>
<tr>
<td>Strongest Postcondition</td>
<td>Infinite Loop</td>
<td>86k</td>
<td>0.15</td>
<td>0.3</td>
</tr>
<tr>
<td>Termination</td>
<td>x&lt;z Loop</td>
<td>244k</td>
<td>0.13</td>
<td>0.5</td>
</tr>
<tr>
<td>Procedure Summaries</td>
<td>Recursive</td>
<td>80k</td>
<td>0.09</td>
<td>1.0</td>
</tr>
</tbody>
</table>

# Experiments – Predicate Abstr.

<table>
<thead>
<tr>
<th>Type of Analysis</th>
<th>Example*</th>
<th>Formula size (# clauses)</th>
<th>Constraint Generation (secs)</th>
<th>Constraint Solving (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification</td>
<td>counter</td>
<td>21</td>
<td>0.37</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>lockstep</td>
<td>8</td>
<td>0.34</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>nested</td>
<td>62</td>
<td>0.49</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>twoloop</td>
<td>79</td>
<td>0.59</td>
<td>0.04</td>
</tr>
<tr>
<td>Weakest Precondition</td>
<td>counter</td>
<td>1,345</td>
<td>0.67</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>lockstep</td>
<td>584</td>
<td>0.52</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>nested</td>
<td>2,866</td>
<td>1.75</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>twoloop</td>
<td>3,778</td>
<td>2.09</td>
<td>0.16</td>
</tr>
</tbody>
</table>

* Details at [http://research.microsoft.com/users/sumitg/benchmarks/pa.html](http://research.microsoft.com/users/sumitg/benchmarks/pa.html)
Future Directions…

• Reduction for quantified domain
  – E.g. \( \forall k : 0 \leq k < n \Rightarrow A[k] = 0 \)

• Scalability
  – Disjunctive invariants
    • Case splits not very efficient
    • Symmetry in SAT formulae problematic
  – Larger programs
    • Incorporate facts inferred from simpler analyses
    • Modular verification
Conclusions

- Core Analyses
  - Program Verification
  - Weakest Precondition Inference
  - Strongest Postcondition Inference

- Applications:
  - Termination & Non-termination
  - Most general bug-finding

- Domains:
  - Linear Arithmetic (PLDI’08)
  - Predicate Abstraction (under submission)

- Constraint-based Analysis
  - Program analyses/fix-points as SAT constraints
  - Using off-the-shelf solvers for solutions
  - Core program analyses in a unified non-iterative framework

http://www.cs.umd.edu/~saurabhs/pacs/