Program Analysis as Constraint Solving

Sumit Gulwani  
(MSR Redmond)

Saurabh Srivastava  
(Univ. of Maryland)  

Ramarathnam Venkatesan  
(MSR Redmond)
Introduction

- Last decade has witnessed an engineering revolution in constraint solving.
- We have developed techniques to reduce classic program analysis problems to constraint solving (in the context of linear arithmetic properties).
  - Program Verification
  - Inter-procedural Program Analysis
  - Weakest Precondition Generation
  - Strongest Postcondition Generation
- We show applications of these techniques to
  - Proving termination/Bounds Analysis
  - Finding preconditions for non-termination
  - Most-general counterexamples
Introduction

• Last decade has witnessed an engineering revolution in constraint solving.
• We have developed techniques to reduce classic program analysis problems to constraint solving (in the context of linear arithmetic properties).
  - Program Verification
  - Inter-procedural Program Analysis
  - Weakest Precondition Generation
  - Strongest Postcondition Generation
• We show applications of these techniques to
  - Proving termination/Bounds Analysis
  - Finding preconditions for non-termination
  - Most-general counterexamples
Program Verification

Verify a Hoare triple \((\text{Pre}, \text{Program}, \text{Post})\)

\[
\begin{align*}
\text{Pre} & \implies \text{I} \\
\text{I} \land \neg c & \implies \text{Post} \\
(I \land c)[S] & \implies \text{I}
\end{align*}
\]

Second-order satisfiability constraint
Solving second-order satisfiability constraints

\[ \exists I \forall X \phi_1(I,X) \]

- **Second-order to First-order**
  - Assume \( I \) has some form, e.g., \( \sum_j a_j x_j \geq 0 \)
  - \( \exists I \forall X \phi_1(I,X) \) translates to \( \exists a_j \forall X \phi_2(a_j,X) \)

- **First-order to “only existentially quantified”**
  - Farkas Lemma helps translate \( \forall \) to \( \exists \)
  - \( \forall X (\wedge_k (e_k \geq 0) \implies e \geq 0) \) iff \( \exists \lambda_k \geq 0 \forall X (e \equiv \lambda + \sum_k \lambda_k e_k) \)
    - Eliminate \( X \) from polynomial equality by equating coefficients.
  - \( \exists a_j \forall X \phi_2(a_j,X) \) translates to \( \exists a_j \exists \lambda_k \phi_3(a_j,\lambda_k) \)

- “only existentially quantified” to SAT
  - Bit-vector modeling for integer variables
Program Verification: Example

\[ n=1 \land m=1 \]

\[
x := 0; \ y := 0;
while (x < 100)

\[
\begin{align*}
x &:= x+n; \\
y &:= y+m;
\end{align*}
\]

\[ y \geq 100 \]

Invariant Template

\[
\begin{align*}
a_0 + a_1 x + a_2 y + a_3 n + a_4 m &\geq 0 \\
b_0 + b_1 x + b_2 y + b_3 n + b_4 m &\geq 0 \\
c_0 + c_1 x + c_2 y + c_3 n + c_4 m &\geq 0
\end{align*}
\]

Satisfying Solution

\[ a_2=b_0=c_4=1, \ a_1=b_3=c_0=-1 \]

Loop Invariant

\[ y \geq x \]
\[ m \geq 1 \]
\[ n \leq 1 \]

Invalid triple or Imprecise Template

\[ UNSAT \]
Outline

• Program Verification

- Weakest Precondition Generation

• Termination/Bounds Analysis

• Most-general Counterexamples
Weakest Precondition

Find “weakest” $\text{Pre}$ such that the Hoare triple $(\text{Pre}, \text{Program}, \text{Post})$ is valid.

$m \geq n$

$x := 0; y := 0;$

while $(x < 100)$

\[
\begin{align*}
x & := x + n; \\
y & := y + m;
\end{align*}
\]

$y \geq 100$
Weakest Precondition: Attempt 1

Find “weakest” Pre such that the Hoare triple \((\text{Pre}, \text{Program}, \text{Post})\) is valid.

\[
\begin{align*}
\text{Pre} & \quad \text{Precondition Encoding} \\
\text{I} \quad \text{while} (c) & \quad \exists \text{Pre}, \text{I} \quad \forall X \left( \text{I} \land \neg c \Rightarrow \text{Post} \right) \left( \text{I} \land c[S] \Rightarrow \text{I} \right) \\
\text{S} & \\
\text{Post} &
\end{align*}
\]

“Weakest” Precondition Encoding

\[
\exists \text{Pre}, \text{I} \left( \text{VC(Pre)} \right) \\
\forall R: \text{If R is weaker than Pre, then } \neg \text{VC(R)}
\]

Unfortunately, Farkas lemma is no longer applicable since the formula is non-linear in universally quantified variables.
Weakest Precondition: Attempt 2

Find “weakest” Pre such that the Hoare triple \((\text{Pre}, \text{Program}, \text{Post})\) is valid.

\[
\text{Pre} \quad \text{Precondition Encoding} \quad \exists \text{Pre}, I \quad \forall X \begin{cases} \text{Pre} \Rightarrow I \\
I \land \neg c \Rightarrow \text{Post} \\
(I \land c)[S] \Rightarrow I
\end{cases}
\]

VC(Pre)

I \quad \text{while (c)} \quad S

Post

\[
\exists \text{Pre}, I \left\{ \begin{align*}
\text{Pre is weaker than false} \\
\text{VC(Pre)}
\end{align*} \right. 
\]

Non-
false Precondition Encoding

\[\exists \text{Pre}, I \left\{ \begin{align*}
\text{Pre \ is \ weaker \ than \ false} \\
\text{VC(Pre)}
\end{align*} \right. \]

\[
\begin{align*}
\text{Pre} & \quad x := 0; \ y := 0; \\
& \quad \text{while (x < 100)} \\
& \quad \begin{cases} x := x + n; \\
y := y + m; \\
[y \geq 100]
\end{cases}
\end{align*}
\]

• \(m \geq n + 127\) is a satisfying assignment for Pre.
• However, this is still not “Weakest” Pre.
Weakest Precondition: Attempt 3

Find “weakest” Pre such that the Hoare triple \((\text{Pre}, \text{Program}, \text{Post})\) is valid.

\[
\begin{align*}
\text{Pre} & \quad \text{Precondition Encoding} \\
I \quad \text{while (c)} & \quad \exists \text{Pre, I} \quad \forall X \\
S & \quad \begin{aligned}
I \land \neg c \Rightarrow \text{Post} \\
(I \land c)[S] \Rightarrow I
\end{aligned}
\end{align*}
\]

“Weaker than Previous” Precondition Encoding

\[
\exists \text{Pre, I} \begin{cases}
\text{VC(Pre)} \\
\text{Pre is weaker than Previous Pre}
\end{cases}
\]

- Seq. \(m \geq n+127, m \geq n+126, \ldots, m \geq n\) is admissible.
- This iterative refinement is too slow.

\[
\begin{align*}
\text{Pre} & \\
& \quad x := 0; y := 0; \\
& \quad \text{while (x < 100)} \\
& \quad \hspace{1cm} x := x + n; \\
& \quad \hspace{1cm} y := y + m; \\
& \quad \hspace{1cm} [y \geq 100]
\end{align*}
\]
Weakest Precondition: Solution 1

Find "weakest" Pre such that the Hoare triple \((\text{Pre}, \text{Program}, \text{Post})\) is valid.

Pre

\[ \text{while (c)} \]

\[ \text{S} \]

Post

Precondition Encoding

\[ \exists \text{Pre, I } \forall X \begin{cases} \text{Pre} \Rightarrow \text{I} \\ \text{I} \land \neg c \Rightarrow \text{Post} \\ (\text{I} \land c)[\text{S}] \Rightarrow \text{I} \end{cases} \]

Locally-weakest Precondition Encoding

\[ \exists \text{Pre, I } \begin{cases} \text{VC(Pre)} \\ \text{Pre is weaker then Previous Pre} \\ \neg \text{VC(R}_1) \land \neg \text{VC(R}_2) \ldots \end{cases} \]

where \(R_1, R_2, \ldots\) are weaker neighbors of Pre.

Pre

\[ \begin{align*} x &:= 0; \ y := 0; \\ \text{while (x < 100)} & \quad x := x+n; \\ & \quad y := y+m; \\ & \quad [y \geq 100] \end{align*} \]
Weaker Neighborhood Structure

Weaker neighbors of \( e_1 \geq 0 \wedge e_2 \geq 0 \) are:

- \( e_1 + 1 \geq 0 \wedge e_2 \geq 0 \)  
  Shift plane \( e_i \) parallel to itself.

- \( e_1 \geq 0 \wedge e_2 + 1 \geq 0 \)  
  Rotate plane \( e_i \) along its intersection with another plane.

- \( e_1 + e_2 \geq 0 \wedge e_2 \geq 0 \)

- \( e_1 \geq 0 \wedge e_2 + e_1 \geq 0 \)
Weakest Precondition: Solution 1

Find “weakest” Pre such that the Hoare triple \((\text{Pre, Program, Post})\) is valid.

**Precondition Encoding**

- \(\exists \text{Pre, I} \ \forall \text{X} \ (\text{I} \land \neg \text{c} \Rightarrow \text{Post})\)

- \((\text{I} \land \text{c})[\text{S}] \Rightarrow \text{I}\)

**Locally-weakest Precondition Encoding**

- \(\exists \text{Pre, I} \ \left(\begin{array}{c}
\text{VC(Pre)} \land \neg \text{VC(R}_1) \land \neg \text{VC(R}_2) \ldots \\
\text{Pre is weaker than Previous Pre}
\end{array}\right)\)

- We directly obtain \(m \geq n\).
  - \(m \geq n+c\) is no longer locally-weakest for \(c>0\).
- In fact, we obtain one more solution: \(n \leq 0\).
- Generally, locally weakest \(\neq\) weakest. Iteration may be required.

\[x := 0; \ y := 0;\]
\[\text{while } (x < 100)\]
\[x := x+n;\]
\[y := y+m;\]
\[\text{[y} \geq 100]\]
Outline

• Program Verification

• Weakest Precondition Generation
  ➢ Termination/Bounds Analysis

• Most-general Counterexample
Termination/Bounds Analysis

while (c)
S

\[ i := 0; \]
while (c)
\[ i := i + 1; \]
\[ \text{Assert}(i \leq F(\text{inputs})); \]
S

• Transformation introduces a counter to track loop iterations.
• Then we run “Program Verification” Algorithm.
  - Besides loop invariant, part F of assertion is also unknown.
  - Existence of a solution yields a termination proof.
  - Solution for F gives upper bound on # of loop iterations.
Outline

• Program Verification

• Weakest Precondition Generation

• Termination/Bounds Analysis
  
  ➢ Most-general Counterexample
Most-general Counterexample

\begin{align*}
x & := 0; \\
\text{while } (x < n) & \\
& \quad \text{Assert}(x<200); \\
x & := x+y;
\end{align*}

\begin{align*}
\text{Pre} \\
x & := 0; \ err := 0; \ i := 0; \\
\text{while } (x < n) & \\
& \quad i := i+1; \text{ Assert}(i \leq F(n,y)); \\
& \quad \text{if } (x \geq 200) \ err := 1; \\
x & := x+y; \\
& \text{Assert} (\ err=1)
\end{align*}

- “n=356 and y=12” leads to a bug. However, it is more useful to say “y>0 \land n\geq200+y” leads to a bug. We generate this by:

- 2-step Transformation
  - Introduce error variable \textit{err} to track assertion violation.
  - Introduce counter \textit{i} to track loop iterations/termination.
- Then, we run “weakest” Precondition algorithm.
Experiments: Methodology

• We ran our tool against academic/small benchmarks used by 10 earlier pieces of work that address a wide variety of program analyses related to linear arithmetic properties.
  - Inter-procedural analysis, Disjunctive invariant inference, termination/bounds analysis, non-termination, strongest Postcondition generation.

• Parameters:
  - Templates: 2 conjuncts/1 disjunct
  - Bit-vector modeling: 3 bits for unknown coefficients, 6 bits for unknown constants, 1 bit for multipliers $\lambda$ in Farkas lemma
  - Iteratively increased these quantities.
Experiments: Results

- **Clauses generated**: 5K to 560K (Most examples < 200K)
- **Time taken**: 0.1 sec to 80 sec (Most examples < 5 sec)
  - comparable; however, demonstrates versatility
Related Work: Constraint based techniques

Invariant discovery

- Linear [Colon, Sankaranarayanan, Sipma, CAV '03]
- Non-linear [Sankaranarayanan et al, POPL '04]; [Kapur, '05]
- Linear arithmetic + uninterpreted fns [Beyer et.al., VMCAI '07]

This work only addresses linear invariants, but
- With arbitrary (pre-specified) boolean structure.
- In an inter-procedural setting.
- In a weakest precondition generation mode too.

Bug Finding

- SATURN [Xie, Aiken, CAV '05]: works with loop-free programs

This work only addresses programs with linear assignments, but
- Finds most-general bugs in programs with loops
Conclusion

- Constraint-based techniques offer 2 advantages over fixed-point computation based techniques:
  - Goal-directed (may buy efficiency)
  - Do not require widening (may buy precision)

- This work showed how to reduce a wide variety of program analysis problems to constraint solving over the domain of linear arithmetic.

- Future work includes extending these ideas to other domains such as pointers, quantifiers.