Program Verification using Templates over Predicate Abstraction

Saurabh Srivastava
University of Maryland, College Park
Sumit Gulwani
Microsoft Research, Redmond
What the technique will let you do!

A. Infer invariants with arbitrary quantification and boolean structure
   \( \forall \) : E.g. Sortedness
   \( \forall \exists \) : E.g. Permutation

B. Infer weakest preconditions

Weakest conditions on input:

Selection Sort:
```java
for i = 0..n {
    for j = i..n {
        find min index
    }
    if (min != i)
        swap A[i] with A[min]
}
```

Worst case behavior: swap every time it can
Improves the state-of-art

- Can infer very expressive invariants
  - Quantification
    - E.g. $\forall k \exists j : (k<n) \Rightarrow (A[k]=B[j] \land j<n)$
  - Disjunction
    - E.g. $\forall k : (k<0 \lor k>n \lor A[k]=0)$
    - Previous techniques are too specialized to particular types of quantification or to quantifier-free disjunction

- Can infer weakest preconditions
  - Good for debugging: can discover worst case inputs
  - Good for analysis: can infer missing precondition facts
    - No satisfactory solutions known
Key facilitators

- **Templates**
  \[
  \forall k : (-) \Rightarrow (-) \\
  \forall k_1 \exists k_2 : (-) \Rightarrow (-)
  \]
  - Task of inferring conjunctive facts for the holes remains

- **Predicate Abstraction**
  - Allows us to efficiently compute solutions for the holes
  - E.g., \{i<j, i>j, i≤j, i≥j, i<j-1, i>j+1... \}
Outline

• Three fixed-point inference algorithms
  – Iterative fixed-point
    • Greatest Fixed-Point (GFP)
    • Least Fixed-Point (LFP)
  – Constraint-based (CFP)

• Optimal Solutions
  – Built over a clean theorem proving interface

• Weakest Precondition Generation

• Experimental evaluation
Outline

• Three fixed-point inference algorithms
  – Iterative fixed-point
    • Greatest Fixed-Point (GFP)
    • Least Fixed-Point (LFP)
  – Constraint-based (CFP)

• Optimal Solutions
  – Built over a clean theorem proving interface

• Weakest Precondition Generation

• Experimental evaluation
Analysis Setup

Loop headers (with invariants) split program into simple paths. Simple paths induce program constraints (verification conditions).

E.g. Selection Sort:

- Initial state: $I_1$ (true $\iff i=0 \implies I_1$)
- $i := 0$
- $i < n$ (if $min \neq i$, swap $A[i], A[min]$)
- $j := i$
- $j < n$
- Find min index
- Sorted array
- $I_1 \land i < n \land j = i \implies I_2$
- $I_1 \land i \geq n \implies$ sorted array
Analysis Setup

Loop headers (with invariants) split program into simple paths. Simple paths induce program constraints (verification conditions).

true ∧ i=0 ⇒ I₁

I₁ ∧ i<n ∧ j=i ⇒ I₂

I₁ ∧ i≥n ⇒
∀k₁,k₂ : 0≤k₁<k₂<n ⇒ A[k₁] ≤ A[k₂]

Simple FOL formulae over I₁, I₂!
We will exploit this.
Iterative Fixed-point: Overview

Candidate Solution

Values for invariants

\( <x,y> \rightarrow \{ \text{vc}(\text{pre},I_1),\text{vc}(I_1,I_2) \} \)

VCs that are not satisfied

\[ \text{vc}(I_1,\text{post}) \]

\[ \text{vc}(I_1,I_2) \]

\[ \text{vc}(I_2,\text{post}) \]

\[ \text{vc}(I_2,I_1) \]

\[ \text{vc}(\text{pre},I_1) \]
Iterative Fixed-point: Overview

Candidate satisfies all VCs

Improve candidate
- Pick unsat VC
- Use theorem prover to compute optimal change
- Results in new candidates

Set of candidates
<x,y> → {...}

Compuatable because:
- Templates make V explicit
- Predicate abstraction restricts search to finite space

Iterative Fixed-point: Overview

pre

vc(pre, I₁)

I₁

vc(I₁, post)

vc(I₁, I₂)

vc(I₂, I₂)

I₂

vc(I₂, I₁)

post
Backwards Iterative (GFP)

Backward:
- Always pick the source invariant of unsat constraint to improve
- Start with $<T, ..., T>$

Candidate Sols $<I_1, I_2> ightarrow$ Unsat constraints

$<T, T> \rightarrow \{vc(I_1, post)\}$
Backwards Iterative (GFP)

**Candidate Sols** $<I_1,I_2> \rightarrow \text{Unsat constraints}$

- $<\top, \top> \rightarrow \{vc(I_1,post)\}$
- $<a_1, \top> \rightarrow \{vc(I_2,I_1)\}$

**Backward:**
- Always pick the source invariant of unsat constraint to improve
- Start with $<\top, \ldots, \top>$

Optimally strengthen so $vc(pre,I_1)$ ok unless no soln
Backwards Iterative (GFP)

Backward:
- Always pick the source invariant of unsat constraint to improve
- Start with $<\top, \ldots, \top>$

Candidate Sols $<I_1, I_2> \rightarrow$ Unsat constraints

- $<\top, \top> \rightarrow \{ \text{vc}(I_1, \text{post}) \}$
- $<a_1, \top> \rightarrow \{ \text{vc}(I_2, I_1) \}$
- $<a_1, b_1> \rightarrow \{ \text{vc}(I_2, I_2) \}$
- $<a_1, b_2> \rightarrow \{ \text{vc}(I_2, I_2), \text{vc}(I_1, I_2) \}$

Optimally strengthen so $\text{vc}(\text{pre}, I_1)$ ok unless no sol

Multiple orthogonal optimal sols
Backwards Iterative (GFP)

Candidate Sols $\langle I_1, I_2 \rangle$ → Unsat constraints

- $\langle \top, \top \rangle \rightarrow \{ vc(I_1, post) \}$
- $\langle a_1, \top \rangle \rightarrow \{ vc(I_2, I_1) \}$
- $\langle a_1, b_1 \rangle \rightarrow \{ vc(I_2, I_2) \}$
- $\langle a_1, b_2 \rangle \rightarrow \{ vc(I_2, I_2), vc(I_1, I_2) \}$
- $\langle a_1, b'_1 \rangle \rightarrow \{ vc(I_2, I_2) \}$
- $\langle a_1, b'_2 \rangle \rightarrow \{ vc(I_1, I_2) \}$

Optimally strengthen so $vc(pre, I_1)$ ok unless no sol

Multiple orthogonal optimal sols

Backward:
- Always pick the source invariant of unsat constraint to improve
- Start with $\langle \top, \ldots, \top \rangle$
Backwards Iterative (GFP)

Backward:
- Always pick the source invariant of unsat constraint to improve
- Start with $<\top, ..., \top>$

Candidate Sols $<I_1, I_2>$ → Unsat constraints

- $<\top, \top> \rightarrow \{ vc(I_1, \text{post}) \}$
- $<a_1, \top> \rightarrow \{ vc(I_2, I_1) \}$
- $<a_1, b_1> \rightarrow \{ vc(I_2, I_2) \}$
- $<a_1, b_2> \rightarrow \{ vc(I_2, I_2), vc(I_1, I_2) \}$
- $<a_1, b'_1> \rightarrow \{ vc(I_2, I_2) \}$
- $<a_1, b'_2> \rightarrow \{ vc(I_1, I_2) \}$
- $<a_1, b_2> \rightarrow \{ vc(I_2, I_2), vc(I_1, I_2) \}$
- $<a_1, b'_1> \rightarrow \{ vc(I_2, I_2) \}$
- $<a'_1, b'_2> \rightarrow$ none

Optimally strengthen so $vc(\text{pre}, I_1)$ ok unless no sol

Multiple orthogonal optimal sols

$<a'_1, b'_2>$ : GFP solution
Forward Iterative (LFP)

Forward: Same as GFP except

- Pick the destination invariant of unsat constraint to improve
- Start with $<\bot,\ldots,\bot>$
Constraint-based over Predicate Abstraction

pre

\[ vc(pre,I_1) \]

\[ I_1 \]

\[ vc(I_1,post) \]

\[ I_2 \]

\[ vc(I_1,I_2) \]

\[ post \]

\[ vc(I_2,post) \]

\[ vc(I_2,I_2) \]

\[ vc(I_2,I_1) \]
Constraint-based over Predicate Abstraction

Remember: VCs are FOL formulae over $I_1, I_2$
Constraint-based over Predicate Abstraction

vc(pre, I₁)
vc(I₁, post)
vc(I₁, I₂)
vc(I₂, I₁)
vc(I₂, I₂)
pred(I₁)
pred(A) : A to unknown predicate indicator variables

Remember:
VCs are FOL formulae over I₁, I₂
**VCs are FOL formulae over I<sub>1</sub>, I<sub>2</sub>**

Constraint-based over Predicate Abstraction

SAT formulae over predicate indicators

\[
\begin{align*}
\text{vc}(\text{pre}, I_1) & \rightarrow \text{boolc}(\text{pred}(I_1)) \\
\text{vc}(I_1, \text{post}) & \rightarrow \text{boolc}(\text{pred}(I_1)) \\
\text{vc}(I_1, I_2) & \rightarrow \text{boolc}(\text{pred}(I_1), \text{pred}(I_2)) \\
\text{vc}(I_2, I_1) & \rightarrow \text{boolc}(\text{pred}(I_2), \text{pred}(I_1)) \\
\text{vc}(I_2, I_2) & \rightarrow \text{boolc}(\text{pred}(I_2))
\end{align*}
\]

\(\text{pred}(A) : A \text{ to unknown predicate indicator variables}\)

Optimal solutions from SMT solver to impose minimal constraints

Program constraint to boolean constraint

Invariant sol\textsubscript{n}

Boolean constraint to satisfying sol\textsubscript{n} (SAT Solver)

Local reasoning

Fixed-Point Computation

Remember: VCs are FOL formulae over I<sub>1</sub>, I<sub>2</sub>
Outline

• Three fixed-point inference algorithms
  – Iterative fixed-point
    • Greatest Fixed-Point (GFP)
    • Least Fixed-Point (LFP)
  – Constraint-based (CFP)

• Optimal Solutions
  – Built over a clean theorem proving interface

• Weakest Precondition Generation

• Experimental evaluation
Optimal Solutions

- **Key: Polarity of unknowns in formula $\Phi$**
  - Positive or negative:
    - Value of positive unknown stronger $\Rightarrow \Phi$ stronger
    - Value of negative unknown stronger $\Rightarrow \Phi$ weaker

- **Optimal Soln:** Maximally strong positives, maximally weak negatives

- **Assume theorem prover interface:** $\text{OptNegSol}$
  - Optimal solutions for formula with only negative unknowns
  - Built using a lattice search by querying SMT Solver

Mathematical expression:

$$\Phi = \forall x : \exists y : ( \neg u_1 \lor u_2 ) \land u_3$$
Optimal Solutions using OptNegSol

formula $\Phi$ contains unknowns:

$u_1 \ldots u_p$ positive

$u_{1} \ldots u_{n}$ negative

OptNegSol

Repeat until set stable

Opt

Opt'$\ldots$

Opt''

Merge positive tuples

$\Phi[\alpha_1 \ldots \alpha_p]$ $
\Phi[\alpha'_1 \ldots \alpha'_p]$

$\ldots$

$\Phi[\alpha''_1 \ldots \alpha''_p]$

$\alpha_1 \ldots \alpha_p, S_1 \ldots S_n$

$\alpha'_1 \ldots \alpha'_p, S'_1 \ldots S'_n$

$\ldots$

$\alpha''_1 \ldots \alpha''_p, S''_1 \ldots S''_n$

Optimal Solutions for formula $\Phi$

$P \times$ Size of predicate set

$P$-tuple that assigns a single predicate to each positive unknown

$S_i$ sol$^n$ for the negative unknowns
Outline

• Three fixed-point inference algorithms
  – Iterative fixed-point
    • Greatest Fixed-Point (GFP)
    • Least Fixed-Point (LFP)
  – Constraint-based (CFP)

• Optimal Solutions
  – Built over a clean theorem proving interface

• Weakest Precondition Generation

• Experimental evaluation
Outline

• Three fixed-point inference algorithms
  – Iterative fixed-point
    • Greatest Fixed-Point (GFP)
    • Least Fixed-Point (LFP)
  – Constraint-based (CFP)

• Optimal Solutions
  – Built over a clean theorem proving interface

• Weakest Precondition Generation

• Experimental evaluation
Implementation

C Program
Templates
Predicate Set

Microsoft's Phoenix Compiler

CFG
Verification Conditions

Invariants
Preconditions

Iterative Fixed-point GFP/LFP
Constraint-based Fixed-Point

Candidate Solutions
Z3 SMT Solver

Boolean Constraint

Invariants Preconditions
Verifying Sorting Algorithms

• **Benchmarks**
  – Considered difficult to verify
  – Require invariants with quantifiers
  – Sorting, ideal benchmark programs

• **5 major sorting algorithms**
  – Insertion Sort, Bubble Sort (\(n^2\) version and termination checking version), Selection Sort, Quick Sort and Merge Sort

• **Properties:**
  – Sort elements
    • \(\forall k : 0 \leq k < n \Rightarrow A[k] \leq A[k+1]\) --- output array is non-decreasing
  – Maintain permutation, when elements distinct
    • \(\forall k \exists j : 0 \leq k < n \Rightarrow A[k] = A_\text{old}[j] \& 0 \leq j < n\) --- no elements gained
    • \(\forall k \exists j : 0 \leq k < n \Rightarrow A_\text{old}[k] = A[j] \& 0 \leq j < n\) --- no elements lost
Runtimes: Sortedness

Tool can prove sortedness for all sorting algorithms!
Run times: Permutation

...Permutations too!
Inferring Preconditions

• Given a property (worst case runtime or functional correctness) what is the input required for the property to hold

• Tool automatically infers non-trivial inputs/preconditions

• **Worst case input** (precondition) for sorting:
  – Selection Sort: sorted array except last element is the smallest
  – Insertion, Quick Sort, Bubble Sort (flag): Reverse sorted array

• **Inputs** (precondition) for **functional correctness**:
  – Binary search requires sorted input array
  – Merge in Merge sort requires sorted inputs
  – Missing initializers required in various other programs
Runtimes (GFP): Inferring worst case inputs for sorting

Tool infers worst case inputs for all sorting algorithms!

Nothing to infer as all inputs yield the same behavior.
Conclusions

- Powerful invariant inference over predicate abstraction
  - Can infer quantifier invariants

- Three algorithms with different strengths
  - Iterative: Least fixed-point and Greatest fixed-point
  - Constraint-based

- Extend to maximally-weak precondition inference
  - Worst case inputs
  - Preconditions for functional correctness

- Techniques builds on SMT Solvers, so exploit their power

- Successfully verified/inferred preconditions
  - All major sorting algorithms
  - Other difficult benchmarks