Compositional Safety Verification with Max-SMT

Daniel Larraz, Albert Oliveras, Enric Rodríguez Carbonell, Albert Rubio
(Universitat Politècnica de Catalunya)

Marc Brockschmidt (MSR)
Overview
Compositional Software Analysis

What I mean by it:

- Partial proof can be “plugged” into larger proof
- Not “whole-program” - No/Limited context information
- Clear correspondence between proof parts and code parts

Why it’s desirable:

- Scalable (via parallel/distributed analysis)
- Incremental (continuous integration setting)
- Open programs have clear semantics
Top-down
(AI: forward)

Pros:
+ Can prune infeasible runs
+ Avoids reasoning over unused code

Cons:
- Needs to keep strongest information

Bottom-up
(AI: backward)

Cons:
- Has to analyse all code & cases leading to property

Pros:
+ Can prune unneeded information
+ Avoids reasoning over unused variables
Compositional & Bottom-up: Plan

1. Propagate assertions backwards
   • Straight-line code: Weakest precondition
   • Loops: *Conditional Inductive Invariants* (via MaxSMT)

2. Repeat until
   • Reached program start: Done
   • Failure: Backtrack & Refine with *Program Narrowing*
Examples
Example: Conditional Inductive Invariants

{\textit{Q}_2 \equiv j \geq 0 \land x + 5(i + j) \geq 0}\]
while \ j > 0 \ do
    \ j := j - 1
    \ i := i + 1
done

{\textit{Q}_1 \equiv x + 5i \geq 0}\]
while \ i > 0 \ do
    \ x := x + 5
    \ i := i - 1
done
assert(x \geq 0)

Find \textit{Q}_2 such that
• \textit{Q}_2 \land j \leq 0 \Rightarrow \textit{Q}_1
• \textit{Q}_2 \text{ inductive}

Find \textit{Q}_1 such that
• \textit{Q}_1 \land i \leq 0 \Rightarrow x \geq 0
• \textit{Q}_1 \text{ inductive}
Example: Program Narrowing

... 
\{Q_1 \equiv x > y\} \lor Q_2 \equiv x < y\}
if !(x > y) then
    while nondet() \& \& !(x > y) do
        assert(x \neq y)
        x := x + 1
        y := y + 1
    done
fi

Find $Q_1$ such that
• $Q_1 \implies x \neq y$
• $Q_1$ inductive

$Q_1$ doesn’t always hold
⇒ Add “blocking clause”

Find $Q_2$ such that
• $Q_2 \implies x \neq y$
• $Q_2$ inductive
Technique
Max-SMT

Input: CNF $H_1 \land \cdots \land H_n \land [S_1, \omega_1] \land \cdots \land [S_m, \omega_m]$

Output: Model $\sigma$ such that
- $\sigma \models H_i$ for all $H_i$
- $\sum_{\sigma \models S_i} \omega_i$ is maximal
Programs

Variables: \( V = \{ v_1, \ldots, v_n \} \) (+ post-variables \( V' \) )

Programs: Graphs of Locations \( L \), Transitions \( T \)

States: \((\ell, v) \in L \times (V \rightarrow \mathbb{Z})\)
Current location + variable valuation

Transitions: \((\ell, \tau(V, V'), \ell'), \tau \in QF_LIA\)
Evaluate \((\ell, v)\) to \((\ell', v')\) if \( \tau(v(V), v'(V')) \)
Example: Program Graph

while $i > 0$ do
  $x := x + 5$
  $i := i - 1$
done

assert ($x \geq 0$)
Finding Conditional Inductive Invariants

Input: SCC \( C \), SCC entries \( E_C \), assertion \((\ell, \neg \varphi, \ell_{error})\)

Template per \( \ell \): \( T_\ell(V) \) (e.g. \( 0 \leq a_\ell + \sum_{v \in V} a_{\ell,v} v \))

Constraints:
- Consecution: \( \wedge_{(\ell,\tau,\ell') \in C} T_\ell(V) \land \tau(V, V') \Rightarrow T_{\ell'}(V) \)
- Safety: \( T_\ell(V) \Rightarrow \varphi(V, V') \)
- Initiation: \( \wedge_{(\ell,\tau,\ell') \in E_C} [\tau(V, V') \Rightarrow T_{\ell'}(V'), \omega_i] \)
Proving Safety w/ Conditional Invariants

Input: Assertion $(\ell, \neg \varphi, \ell_{error})$, SCC $C$ of $\ell$, SCC entries $E_C$

1. Find conditional inductive invariant $Q_t$ for $t \in C \cup E_C$
2. Try to prove safety for assertion $(\tilde{\ell}_t, \tau_t \land \neg Q_t, \tilde{\ell}'_t)$
3. If successful for all entries: Done, celebrate
4. Otherwise: Narrow program:
   Replace all $(\ell_t, \tau_t \land \neg Q_t, \ell'_t)$ in $C \cup E_c$ by
   $(\ell_t, \tau_t \land \neg Q_t, \ell'_t)$
5. Restart from 1
Optimisations

1. Add more soft constraints, e.g., trying to disable transitions
2. Memoisation for failed proof attempts
3. Store proven invariants in program
4. Parallelisation:
   • Visit all predecessors in parallel
   • Directly attempt narrowing
... wrapping up
## Experiments: HOLA Benchmarks

<table>
<thead>
<tr>
<th>Tool</th>
<th>Safe</th>
<th>Fail</th>
<th>Timeout</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPAChecker (sv-comp15)</td>
<td>33</td>
<td>3</td>
<td>10</td>
<td>4490</td>
</tr>
<tr>
<td>CPAChecker (predicateAnalysis)</td>
<td>25</td>
<td>11</td>
<td>10</td>
<td>2271</td>
</tr>
<tr>
<td>SeaHorn</td>
<td>32</td>
<td>13</td>
<td>1</td>
<td>212</td>
</tr>
<tr>
<td>HOLA</td>
<td>43</td>
<td>0</td>
<td>3</td>
<td>624</td>
</tr>
<tr>
<td>VeryMax-Seq</td>
<td>44</td>
<td>2</td>
<td>0</td>
<td>344</td>
</tr>
<tr>
<td>VeryMax-Par</td>
<td>45</td>
<td>1</td>
<td>0</td>
<td>151</td>
</tr>
</tbody>
</table>

46 safe examples from safety proving literature
17-71 LOC, 1-4 loops per example, timeout 200s
## Experiments: Numerical Recipes

<table>
<thead>
<tr>
<th>Tool</th>
<th>Safe</th>
<th>Unsafe</th>
<th>Fail</th>
<th>Timeout</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPAChecker (sv-comp15)</td>
<td>5570</td>
<td>251</td>
<td>326</td>
<td>305</td>
<td>735337</td>
</tr>
<tr>
<td>CPAChecker (predicateAnalysis)</td>
<td>5928</td>
<td>170</td>
<td>234</td>
<td>120</td>
<td>64652</td>
</tr>
<tr>
<td>SeaHorn</td>
<td>6077</td>
<td>233</td>
<td>80</td>
<td>62</td>
<td>24167</td>
</tr>
<tr>
<td>VeryMax-Seq</td>
<td>6105</td>
<td>0</td>
<td>326</td>
<td>21</td>
<td>38981</td>
</tr>
<tr>
<td>VeryMax-Par</td>
<td>6106</td>
<td>0</td>
<td>346</td>
<td>0</td>
<td>23668</td>
</tr>
</tbody>
</table>

217 numerical algorithms, array bounds turned into 6452 safety assertions up to ~300 LOC, up to ~35 loops per example, timeout 300s
Conclusion

Present(ed):

- Compositional, bottom-up safety proofs
- Invariant generation from templates with MaxSMT
- **VeryMax** precision & performance competitive

Future:

- Interplay with top-down analysis
- Reachability instead of safety
- Liveness properties: (Non)termination, CTL
- Complexity analysis