Microsoft Research

Microsoft

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

$\{Q_2 \equiv j \geq 0 \wedge x + 5(i + j) \geq 0\}$

```
while j > 0 do
  j := j - 1
  i := i + 1
done
```

Find Q_2 such that

- $Q_2 \wedge j \leq 0 \implies Q_1$
- Q_2 inductive

$\{Q_1 \equiv x + 5i \geq 0\}$

```
while i > 0 do
  x := x + 5
  i := i - 1
done
assert(x >= 0)
```

Find Q_1 such that

- $Q_1 \wedge i \leq 0 \implies x \geq 0$
- Q_1 inductive

Example: Program Narrowing

…

$\{Q_1 \equiv x > y \vee Q_2 \equiv x < y\}$

```
if !(x > y) then
  while nondet() && !(x > y) do
    assert(x != 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 $H_1 \wedge \cdots \wedge H_n \wedge [S_1, \omega_1] \wedge \cdots \wedge [S_m, \omega_m]$

Output: Model σ such that

- $\sigma \vDash H_i$ for all H_i
- $\sum_{\sigma \vDash S_i} \omega\_i$ is maximal

Programs

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

Programs: Graphs of Locations $\boldsymbol{L}$, Transitions $\boldsymbol{T}$

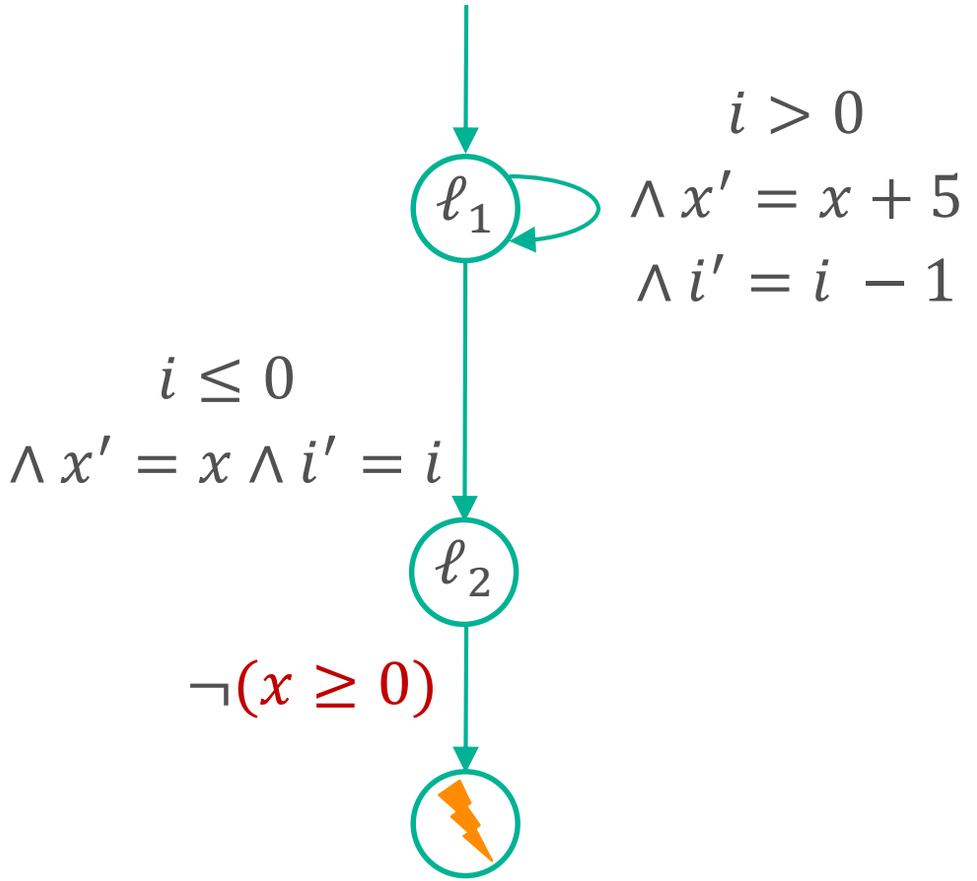
States: $(\ell, \boldsymbol{v}) \in \boldsymbol{L} \times (\boldsymbol{V} \to \mathbb{Z})$
Current location + variable valuation

Transitions: $(\ell, \tau(\boldsymbol{V}, \boldsymbol{V}'), \ell')$, $\tau \in QF\_LIA$
Evaluate $(\ell, \boldsymbol{v})$ to $(\ell', \boldsymbol{v}')$ if $\tau(\boldsymbol{v}(\boldsymbol{V}), \boldsymbol{v}'(\boldsymbol{V}'))$

Example: Program Graph

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

assert (x >= 0)
```



Finding Conditional Inductive Invariants

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

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

Constraints:

- Consecution: $\bigwedge_{(\ell,\tau,\ell') \in C} \boldsymbol{T_\ell(V)} \wedge \tau(\boldsymbol{V}, \boldsymbol{V'}) \Rightarrow \boldsymbol{T_{\ell'}(V)}$
- Safety: $\boldsymbol{T_\ell(V)} \Rightarrow \boldsymbol{\varphi(V, V')}$
- Initiation: $\bigwedge_{(\ell,\tau,\ell') \in E_C} [\tau(\boldsymbol{V}, \boldsymbol{V'}) \Rightarrow \boldsymbol{T_{\ell'}(V')}, \ \omega\_i]$

Proving Safety w/ Conditional Invariants

Input: Assertion $(\ell, \neg\varphi, \ell_{error})$, SCC C of ℓ, 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 $(\widetilde{\ell_t}, \tau_t \wedge \neg Q_t, \widetilde{\ell'_t})$
3. If successful for all entries: Done, celebrate
4. Otherwise: Narrow program:

   Replace all $(\ell_t, \tau_t \quad\quad\quad, \ell'_t)$ in $C \cup E_c$ by

   $(\ell_t, \tau_t \wedge \neg Q_t, \ell'_t)$
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

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

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

Experiments: Numerical Recipes

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

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