Compositional Satisfiability Solving in Separation Logic

Quang Loc Le

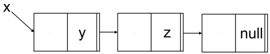
University College London

VMCAI - January 19, 2021

Verification to Satisfiability

Generating verification conditions for heap-based programs.

```
struct node {int val;node next};
int main(node x, node y) {
  // PRE: odd(x,m) * odd(y,n)
  int i = get_size(x);
  int j = get_size(y);
  if (i+j % 2 == 1) ERROR();
  return 1;
}
int get_size(node x) // SPEC only
// PRE:  odd(x,k)
// POST: odd(x,k) & res=k
  ;
```

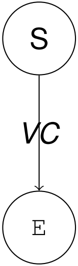


`odd`(x, m): singly-linked list whose length m is an odd number.

Verification Condition:
Is ERROR() called?

Verification to Satisfiability

```
struct node {int val;node next};
int main(node x, node y) {
  // PRE: odd(x,m) * odd(y,n)
  int i = get_size(x);
  int j = get_size(y);
  if (i+j % 2 == 1) ERROR();
  return 1;
}
int get_size(node x) // SPEC only
// PRE:  odd(x,k)
// POST: odd(x,k) & res=k
  ;
```



VC: separation logic with inductive definitions and arithmetic.

$$\begin{array}{lcl} \exists x_1.\, x \mapsto node(\_, x_1) * \mathtt{even}(x_1, n-1) & \Rightarrow & \mathtt{odd}(x, n) \\ \mathtt{emp} \land x = \mathtt{null} \land n = 0 & \Rightarrow & \mathtt{even}(x, n) \\ \exists x_1 . x \mapsto node(\_, x_1) * \mathtt{odd}(x_1, n-1) & \Rightarrow & \mathtt{even}(x, n) \end{array}$$

$$VC \equiv \mathtt{odd}(x,m) * \mathtt{odd}(y,n) \land (\exists k.\ m + n = 2k + 1)$$

We need a satisfiability solver

Our Approach

Compositional Satisfiability Solver

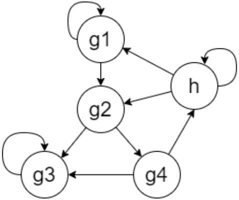
Compositional Satisfiability Solver

The analysis result of a composite program is defined in terms of the analysis results of its parts[a].

---

[a] *compositional shape analysis*. C. Calcagno et. al. POPL'09.

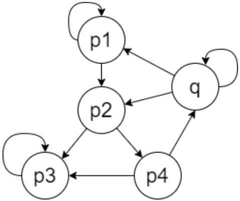
1. summaries of the calling procedures are inferred;
2. summaries of the composite program is computed from the summaries of its callees.



Dependency
Call Graph

Compositional Satisfiability Solver

The satisfiability result of a composite formula is defined in terms of the satisfiability results of its parts.



Dependency Predicate Graph

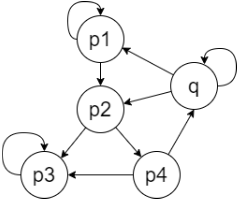
satisfiability result $=$ bases

- a base is a formula without any inductive predicate
- base of a formula precisely characterises its satisfiability
- satisfiability of bases is decidable

Compositional Satisfiability Solver

Given a formula,

1. bases of the inductive predicates are inferred;
2. a base of the formula is computed from the bases of its inductive predicates.



Dependency Predicate Graph

Compositional Satisfiability Solver

Decision algorithm

1. Infer its base via bases of inductive predicates;
2. Transform the base to SMT formulas;
3. Discharge the SMT formulas.

Compositional Satisfiability Solver: Example

$$VC = \mathtt{odd}(x,m) * \mathtt{odd}(y,n) \wedge (\exists k.\ m+n = 2k+1)$$

1. Infer base:
   - a base of predicate $\mathtt{odd}(x, n)$ is inferred as:

     $$\{\ x{\mapsto}node(\_) \wedge (\exists i.\ n = 2i+1 \wedge i{\geq}0)\ \}$$

   - a base of VC is computed as:

     $$\begin{aligned} VC'{\equiv}\quad & x{\mapsto}node(\_){*}y{\mapsto}node(\_){\wedge} \\ & (\exists k.m{+}n{=}2k{+}1){\wedge}(\exists i.m{=}2i{+}1{\wedge}i{\geq}0){\wedge}(\exists i.n{=}2i{+}1{\wedge}i{\geq}0) \end{aligned}$$

2. Transform VC' into an equi-sat SMT formula:

   $$\begin{aligned} \pi{\equiv}\quad & x{\neq}\mathtt{null}{\wedge}y{\neq}\mathtt{null}{\wedge}x{\neq}y{\wedge} \\ & \underline{(\exists k.m{+}n{=}2k{+}1){\wedge}(\exists i.m{=}2i{+}1{\wedge}i{\geq}0){\wedge}(\exists i.n{=}2i{+}1{\wedge}i{\geq}0)} \end{aligned}$$

3. Discharge π: as π is unsatisfiable, so is VC' and then VC.

Base Inference via Regular Unfolding Trees

method of infinite descent: a standard approach to Diophantine equations

To show that an equation P has a solution.

First, we need to hypothesize a simpler equation Q and we show that:

- $Q(a)$ and $P(a)$ hold for some natural number constant a,

- and whenever $Q(n)$ and $P(n)$ hold, there exists a positive integer m such that $m < n$ and both $Q(m)$ and $P(m)$ hold.

Then, P has the same set of solutions with Q.

Our analogy

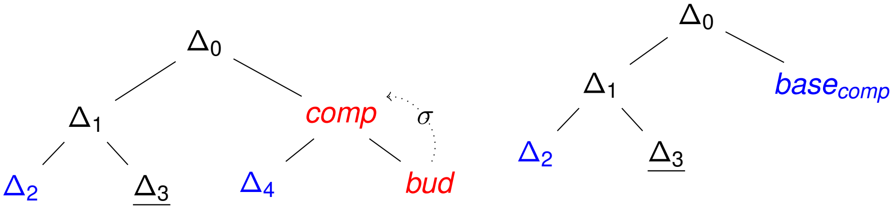
P is an inductive predicate; Q is its base

We propose an algorithm to infer the base

Base Inference via Regular Unfolding Trees

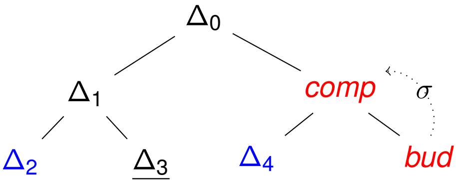
Given an inductive predicate $P(\bar{x})$,

1. Construct a regular unfolding tree for $\Delta_0 \equiv P(\bar{x})$
2. Infer the base for the cycles in a bottom-up manner



$$\mathtt{base}^{\mathcal{P}}(P(\bar{x}))\equiv\{\Delta_2, base_{comp}\}$$

Base Inference via Regular Unfolding Trees



Sound Condition

exist an unfolding of some predicate P between *comp* and *bud*.

Our algorithm finds a *base* of the *comp* such that:

1. *base* and Δ_4 are both sat when P has been unfolded a constant a times.
2. if *base* and *bud* are both sat when P has been unfolded n times, then *base* and *comp* are both sat when P has been unfolded $m < n$ times.

Evaluation: with/without Compositionality

4, 368 queries: 473 `UNSAT` and 3, 895`SAT`

Data Structure	#query	*without*		*with*	
		#Z3	Time	#Z3	Time
Singly llist	666	3,173	1.01	762	0.40
Sorted llist	217	796	0.55	336	0.36
Doubly llist	452	1,803	0.79	552	0.46
Heap trees	386	3,732	6.03	865	2.61
AVL	881	9,051	23.06	2,026	10.85
RBT	1,741	3,491,730	74,158	1,767	2.81
rose-tree	25	300	0.34	153	0.25
	4,368	3,510,585	74,189.78	6461	17.74

with/without

- 0.024% in time
- 0.184% in the numbers of Z3 invocations

More in the paper

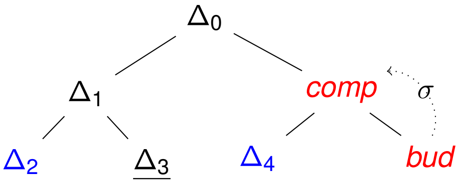
- Correctness
- Decidable Fragment
- More experiments on SL-COMP benchmarks

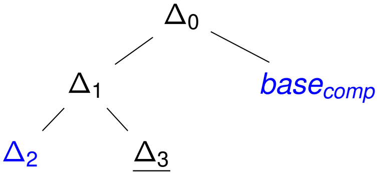
Conclusion

Compositional Satisfiability Solving

Given an inductive predicate $P(\bar{x})$,

1. Construct a regular unfolding tree for $\Delta_0 \equiv P(\bar{x})$
2. Flatten the tree into a disjunctive set of base formulas





$$\mathtt{base}^{\mathcal{P}}(P(\bar{x})) \equiv \{\Delta_2, base_{comp}\}$$