Synthesizing Shape Predicate via Second-Order Bi-Abduction

Quang-Loc Le, Cristian Gherghina, Shengchao Qin and Wei-Ngan Chin

Motivation

- Shape analysis is crucial for proving memory safety and is a precursor to supporting functional correctness on heap-manipulating programs.
- Inferring shapes describing abstractions for data structures used by each method is challenging to automatic program verifiers.
- Shape synthesis for arbitrary data structures to support recursive procedures is even more challenging.

Main Ideas

- Grounded on abduction rather than deduction.
- Distinguish unknown pre-predicates in pre-conditions, from unknown post-predicates in post-condition, since the former may be strengthened, while the latter may be weakened.
- Support heap guard mechanism for more precise preconditions on heap specification.
- Normalize inferred shape predicates to facilitate the reuse of existing predicates.

Second-Order Bi-Abductive Entailment

Given an antecedent \( \Delta_{ant} \) and a consequent \( \Delta_{cons} \), the second-order bi-abductive entailment checker constructs both the frame residue \( \Delta_{fr} \) and a set of relational assumptions of the form \( R = \Delta_{fr} \land (\Delta_{cons} \vee (R \land \Delta_{fr})) \) such that:

\[ \Delta_{ant} \vdash \Delta_{cons} \land \neg(\Delta_{fr} \land \neg \Delta_{cons}) \]

Seminotics

\( R \land \Delta_{ant} \vdash \Delta_{cons} \land \neg(\Delta_{fr} \land \neg \Delta_{cons}) \)

Example: \( H \) is an unknown predicate and \( \neg\text{size}(p.r,t) \) is a constant that ensures a points to an allocated heap, as required for field access.

Derivation of Shape Predicates

**Function** \( \text{Pred}\_\text{Syn}(R) \)

**Derivation**

\[ \Gamma \vdash \square \]

while \( \neg \text{size}(p.r,t) \) do

**Haro Rules**

\[ \{ (\Delta_{fr}) \} \vdash (R \land \Delta_{ant}) \]

end while

end function

Normalization of Shape Predicates

**Function** \( \text{Pred}\_\text{Norm}(\Gamma) \)

**Derivation**

\[ \Gamma \vdash \square \]

while \( \neg \text{size}(p.r,t) \) do

end while

end function

Implementation and Experiments

**Example**

- **SLL (delete)**
  - Size: 9
  - Normal: 2.3
  - Verify: 0.2

- **SLL (reverse)**
  - Size: 20
  - Normal: 0.12
  - Verify: 0.23

- **SLL (insert)**
  - Size: 13
  - Normal: 0.08
  - Verify: 0.21

- **SLL (delete)**
  - Size: 13
  - Normal: 0.08
  - Verify: 0.21

- **SLL (append)**
  - Size: 9
  - Normal: 0.2
  - Verify: 0.19

- **SLL (delete)**
  - Size: 23
  - Normal: 0.18
  - Verify: 0.16

- **SLL (append)**
  - Size: 9
  - Normal: 0.18
  - Verify: 0.19

- **SLL (delete)**
  - Size: 23
  - Normal: 0.18
  - Verify: 0.19

- **SLL (append)**
  - Size: 9
  - Normal: 0.18
  - Verify: 0.19

- **SLL (delete)**
  - Size: 23
  - Normal: 0.18
  - Verify: 0.19

- **SLL (append)**
  - Size: 9
  - Normal: 0.18
  - Verify: 0.19

**Example**

- **SLL (delete)**
  - Size: 9
  - Normal: 2.3
  - Verify: 0.2

- **SLL (reverse)**
  - Size: 20
  - Normal: 0.12
  - Verify: 0.23

- **SLL (insert)**
  - Size: 13
  - Normal: 0.08
  - Verify: 0.21

- **SLL (delete)**
  - Size: 23
  - Normal: 0.18
  - Verify: 0.19

Implementation:

- CIL infrastructure for C programs.
- Omega and Z3 to discharge numerical proof obligations.

Experimental results:

- synthesis time < 1s (Syn. column)
- normalization (w/ norm column) reduces 40% (288/482) the size of synthesized predicates with an overhead of 1% time.

More experiment on CUDA library.