Separation Logics for Pointer Programs

James Brotherston

Lorentz Center Workshop on Effective Verification of Pointer Programs

Monday 13th May, 2019

Part I

Introduction to separation logic

Introduction

Verification of imperative programs is classically based on Hoare triples:

$$\{P\}\,C\,\{Q\}$$

where C is a program and P, Q are assertions in some logical language.

These are read, roughly speaking, as

> *for any state σ satisfying P, if C transforms state σ to σ', then σ' satisfies Q.*

(with some wriggle room allowing us to deal with faulting or non-termination in various ways.)

Classical failure of frame rule

The so-called rule of constancy in Hoare logic,

$$\frac{\{P\}\, C\, \{Q\}}{\{F \wedge P\}\, C\, \{F \wedge Q\}}\ (FV(F) \cap mod(C) = \emptyset)$$

becomes unsound when we consider pointers.
E.g.,

$$\frac{\{x \mapsto 0\}\, [x] := 2\, \{x \mapsto 2\}}{\{y \mapsto 0 \wedge x \mapsto 0\}\, [x] := 2\, \{y \mapsto 0 \wedge x \mapsto 2\}}$$

is not valid (because y could alias x).

Assertions, informally

Separation logic lets us abstractly describe heap memory, including data structures such as linked lists and trees.

E.g., binary trees with root pointer x can be defined by:

$$\begin{array}{rcl} x = \mathsf{nil} : \mathsf{emp} & \Rightarrow & \mathsf{tree}(x) \\ x \neq \mathsf{nil} : x \mapsto (y, z) * \mathsf{tree}(y) * \mathsf{tree}(z) & \Rightarrow & \mathsf{tree}(x) \end{array}$$

where

- emp denotes the empty heap;
- $x \mapsto (y, z)$ denotes a single pointer to a pair of data cells;
- $*$ means "and, separately in memory".

Semantics of assertions

- Program states are stack-heap pairs (s, h), where .
  - stacks map variables to values, $s : \mathsf{Var} \rightarrow \mathsf{Val}$;
  - heaps map finitely many locations to values, $h : \mathsf{Loc} \rightharpoonup_{\text{fin}} \mathsf{Val}$.
- Heap composition $h_1 \circ h_2$ is defined to be $h_1 \cup h_2$ if their domains are disjoint, and undefined otherwise.
- Clauses of the forcing relation $s, h \models A$:

$$
\begin{array}{lcl}
s, h \models \mathsf{emp} & \Leftrightarrow & \mathsf{dom}(h) = \emptyset \\
s, h \models x \mapsto \mathbf{t} & \Leftrightarrow & \mathsf{dom}(h) = \{s(x)\} \text{ and } h(s(x)) = s(\mathbf{t}) \\
s, h \models A * B & \Leftrightarrow & \exists h_1, h_2.\ h = h_1 \circ h_2 \text{ and } s, h_1 \models A \\
 & & \text{and } s, h_2 \models B
\end{array}
$$

Semantics of Hoare triples

- The small-step semantics of programs is given by a relation $\rightsquigarrow$ between program-and-state configurations:

$$(C, s, h) \rightsquigarrow (C', s', h')$$

- We take a fault-avoiding interpretation of Hoare triples: $\{P\}\, C\, \{Q\}$ is valid if, whenever $s, h \models P$,
  1. $(C, s, h) \not\rightsquigarrow^* \mathit{fault}$ (i.e. is memory-safe), and
  2. if $(C, s, h) \rightsquigarrow^* (\epsilon, s, h)$, then $s, h \models Q$.

- If we are interested in total correctness, simply replace "safe" by "safe and terminating" in condition 1!

The frame rule

The frame rule of separation logic is:

$$\frac{\{P\}\, C\, \{Q\}}{\{F * P\}\, C\, \{F * Q\}}\ (FV(F) \cap mod(C) = \emptyset)$$

In particular, e.g.,

$$\frac{\{x \mapsto 0\}\, [x] := 2\, \{x \mapsto 2\}}{\{y \mapsto 0 * x \mapsto 0\}\, [x] := 2\, \{y \mapsto 0 * x \mapsto 2\}}$$

is now fine; y cannot alias x because of separation.

Example: proof of recursive tree disposal

$\{\mathsf{tree}(x)\}$

```
deltree(*x) {
    if x=nil then return;  {emp}
    else {    {x ↦ (y, z) ∗ tree(y) ∗ tree(z)}
        l,r := x.left,x.right;
        {x ↦ (l, r) ∗ tree(l) ∗ tree(r)}
        deltree(l);
        {x ↦ (l, r) ∗ emp ∗ tree(r)}
        deltree(r);
        {x ↦ (l, r) ∗ emp ∗ emp}
        free(x);
        {emp ∗ emp ∗ emp}
    }  {emp}
}  {emp}
```

Soundness of frame rule

Soundness of the frame rule depends on the following two operational facts about the programming language:

Lemma (Safety monotonicity)

If $(C, s, h) \not\rightsquigarrow^* \textit{fault}$ *and* $h \circ h'$ *is defined then* $(C, s, h \circ h') \not\rightsquigarrow^* \textit{fault}$.

Lemma (Frame property)

Suppose $(C, s, h_1 \circ h_2) \rightsquigarrow^* \langle s, h \rangle$, *and that* $(C, s, h_1) \not\rightsquigarrow^* \textit{fault}$. *Then* $\exists h'$ *with* $(C, s, h_1) \rightsquigarrow^* \langle s, h' \rangle$ *and* $h = h' \circ h_2$.

Together, these lemmas imply the locality of all commands.

Concurrent separation logic (CSL)

- Concurrent separation logic (CSL) extends vanilla SL with the following concurrent frame rule:

$$\frac{\{A_1\}\, C_1\, \{B_1\} \quad \{A_2\}\, C_2\, \{B_2\}}{\{A_1 * A_2\}\, C_1 \,||\, C_2\, \{B_1 * B_2\}}$$

(provided $FV(A_1) \cap mod(C_2) = FV(A_2) \cap mod(C_1) = \emptyset$)

- The rule says that concurrent threads behave compositionally when run on separate resources.
- However, many interesting concurrent programs do share resources between threads!

Fractional permissions

- Fractional permissions are intended to allow the division of memory into two or more “read-only copies”.
- Standard example of a permissions algebra: rationals in the open interval $(0, 1]$. Heaps are now $h : \mathsf{Loc} \rightharpoonup_{\text{fin}} \mathsf{Val} \times \text{Perm}$.
- Composition of heaps-with-permissions: heaps must agree on their values where they overlap; then one simply adds the permissions at overlapping locations.
- We can then annotate points-to formulas with permissions, e.g. $x \overset{0.5}{\mapsto} d$. Note that

$$x \overset{0.5}{\mapsto} d * x \overset{0.5}{\mapsto} d \equiv x \mapsto d \ .$$

Fractional permission proofs

We can then write program proofs with the following structure.

$$\{x \mapsto d\}$$

$$\{x \overset{0.5}{\mapsto} d * x \overset{0.5}{\mapsto} d\}$$

$\{x \overset{0.5}{\mapsto} d\}$	$\Vert$	$\{x \overset{0.5}{\mapsto} d\}$
`foo();`	$\Vert$	`bar();`
$\{x \overset{0.5}{\mapsto} d * A\}$	$\Vert$	$\{x \overset{0.5}{\mapsto} d * B\}$

$$\{x \overset{0.5}{\mapsto} d * x \overset{0.5}{\mapsto} d * A * B\}$$

$$\{x \mapsto d * A * B\}$$

Selected references

S. Ishtiaq and P. O'Hearn.
BI as an assertion language for mutable data structures. In *Proc. POPL-28*, 2001.
(Winner of *Most Influential POPL Paper 2001* award.)

J.C. Reynolds.
Separation logic: A logic for shared mutable data structures. In *Proc. LICS-17*, 2002.

S. Brookes.
A semantics for concurrent separation logic. In *Theor. Comp. Sci. 375*, 2007.
(Joint winner of 2016 Gödel Prize.)

R. Bornat, C. Calcagno, P. O'Hearn and M. Parkinson.
Permission accounting in separation logic. In *Proc. POPL-32*, 2005.

Part II

Logical problems in SL verification

A feast of fragments

- The difficulty of logical problems associated with verification is heavily influenced by the precise choice of assertion language.
- The main vectors influencing complexity include:
  - Propositional structure; presence of $\wedge$, $\rightarrow$, $\neg$ and $-\!\!*$ (adjoint of $*$) greatly complicates matters.
  - Inductively defined predicates, needed to capture heap data structures.
  - Arithmetic in assertions, sometimes needed to capture data constraints or to account for pointer arithmetic in programs.
  - Quantifiers; alternation increases complexity as usual.

Symbolic heaps

- A widely-used restricted form of SL formulas.
- Terms t are expressions built from variables $x, y, z \ldots$ and function / constant symbols.
- Pure formulas π, spatial formulas F and symbolic heaps Σ:

$$\begin{array}{rcl} \pi & ::= & t = t \mid t \neq t \mid \ldots \mid \pi \wedge \pi \\ F & ::= & \mathsf{emp} \mid x \mapsto \mathbf{t} \mid P\mathbf{t} \mid F * F \\ \Sigma & ::= & \exists \mathbf{x}.\ \pi : F \mid \Sigma \vee \Sigma \end{array}$$

  (where P a predicate symbol, $\mathbf{t}$ a tuple of terms).

- The predicate symbols might be hard-coded, or else user-defined (possibly with restrictions).

Model checking

- Model checking problem: given formula A and state (s, h), decide whether $s, h \models A$.
- Use case: in dynamic verification. Namely,
  - start with an assertion-annotated program;
  - generate concrete memory states satisfying the precondition;
  - run program and dynamically check current memory states against assertions (model checking!).

Results on model checking

- For symbolic heaps with user-defined predicates, complexity ranges from PTIME to EXPTIME depending on definition restrictions.

  J. Brotherston, N. Gorogiannis, M. Kanovich and R. Rowe",
  Model checking for symbolic-heap separation logic with inductive predicates. In *Proc. POPL-43*, 2016.

- Status unknown (AFAIK) for larger fragments.

Satisfiability

- Satisfiability problem: given formula A, decide whether there is a state (s, h) with $s, h \models A$.

- Use cases: speeding up static verification in two ways,
  1. assertions are often large disjunctions, and any unsatisfiable disjunct can be eliminated ($A \vee \mathsf{false} \equiv A$);
  2. because any Hoare triple of the form $\{\mathsf{false}\}\, C\, \{Q\}$ is valid, proof search can be terminated as soon as one generates an unsatisfiable assertion.

Results on satisfiability

- For symbolic heaps with user-defined predicates, complexity is EXPTIME-complete but can become easier (PTIME) depending on definition restrictions.

  J. Brotherston, C. Fuhs, N. Gorogiannis and J. Navarro Pérez",
  A decision procedure for satisfiability in separation logic with inductive predicates. In *Proc. CSL-LICS*, 2014.

- If one adds Presburger arithmetic then satisfiability becomes undecidable (one can encode Peano arithmetic). But in a restricted form of arithmetic, still decidable.

  Q.L. Le, M. Tatsuta, J. Sun and W-N. Chin.
  A decidable fragment in separation logic with inductive predicates and arithmetic. In *Proc. CAV*, 2017.

Entailment

- Entailment problem: given formulas A and B, decide whether $A \models B$, meaning $s, h \models A \Rightarrow s, h \models B$.

- Use cases: in the course of verification proofs, e.g.
  1. to transform an assertion into a form suitable for symbolic execution, e.g.,

$$\dfrac{\dfrac{}{\{\mathsf{tree}(x)\}\,\texttt{deltree(x)}\,\{\mathsf{emp}\}} \qquad x \mapsto (\mathsf{nil}, z) * \mathsf{tree}(z) \models \mathsf{tree}(x)}{\{x \mapsto (\mathsf{nil}, z) * \mathsf{tree}(z)\}\,\texttt{deltree(x)}\,\{\mathsf{emp}\}}\,(\models)$$

  2. to establish loop invariants, e.g. by

$$\dfrac{\dfrac{\{B \wedge P\}\,C\,\{Q\} \quad Q \models P}{\{B \wedge P\}\,C\,\{P\}}\,(\models)}{\{P\}\,\texttt{while}\,B\,\texttt{do}\,C\,\{\neg B \wedge P\}}\,(\text{while})$$

Results on entailment

- For symbolic heaps with user-defined predicates, the problem is undecidable (one can encode CFG inclusion).

T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich and J. Ouaknine.
Foundations for decision problems in separation logic with general inductive predicates. In *Proc. FoSSaCS-17*, 2014.

- Hard-coded linked lists, and arrays with arithmetic, are decidable (PTIME resp. Π_2^P-hard):

B. Cook, C. Haase, J. Ouaknine, M. Parkinson and J. Worrell.
Tractable reasoning in a fragment of separation logic. In *Proc. CONCUR*, 2011.

James Brotherston, Nikos Gorogiannis and Max Kanovich.
Biabduction (and related problems) in array separation logic. In *Proc. CADE-26*, 2017.

More results on entailment

- Various classes of inductively defined predicates for which entailment is decidable have also been identified:

  - Radu Iosif and Adam Rogalewicz and Jiri Simacek.
    The tree width of separation logic with recursive definitions. In *Proc. CADE-24*, 2013.
  - M. Tatsuta and D. Kimura.
    Separation logic with monadic inductive definitions and implicit existentials. In *Proc. APLAS-13*, 2015.
  - X. Gu, T. Chen and Z. Wu.
    A complete decision procedure for linearly compositional separation logic with data constraints. In *Proc. IJCAR*, 2016.

- For anything more complicated, one generally has to use theorem proving.

Example: cyclic entailment proof

Define list segment predicate ls by

$$\begin{array}{rcl} x = y : \mathsf{emp} & \Rightarrow & \mathsf{ls}\,x\,y \\ x \mapsto x' * \mathsf{ls}\,x'\,y & \Rightarrow & \mathsf{ls}\,x\,y \end{array}$$

Cyclic proof of $\mathsf{ls}\,x\,y * \mathsf{ls}\,y\,z \vdash \mathsf{ls}\,x\,z$:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{ls}\,x\,z \vdash \mathsf{ls}\,x\,z}\,(\mathrm{Id})}{\mathsf{emp} * \mathsf{ls}\,x\,z \vdash \mathsf{ls}\,x\,z}\,(\mathsf{emp}) \qquad \dfrac{\dfrac{\dfrac{(\dagger)\ \mathsf{ls}\,x\,y * \mathsf{ls}\,y\,z \vdash \mathsf{ls}\,x\,z}{\mathsf{ls}\,x'\,y * \mathsf{ls}\,y\,z \vdash \mathsf{ls}\,x'\,z}\,(\mathrm{Subst})}{x \mapsto x' * \mathsf{ls}\,x'\,y * \mathsf{ls}\,y\,z \vdash x \mapsto x' * \mathsf{ls}\,x'\,z}\,(*/\mapsto)}{x \mapsto x' * \mathsf{ls}\,x'\,y * \mathsf{ls}\,y\,z \vdash \mathsf{ls}\,x\,z}\,(\mathsf{ls})}{(\dagger)\ \mathsf{ls}\,x\,y * \mathsf{ls}\,y\,z \vdash \mathsf{ls}\,x\,z}\,(\mathrm{Cases})$$

Biabduction

- Biabduction problem: given formulas A and B, find formulas X and Y with

$$A * X \models B * Y \ , \ \text{and } A * X \text{ is satisfiable.}$$

- Use case: Given specs $\{A'\}\, C_1\, \{A\}$ and $\{B\}\, C_2\, \{B'\}$, we can infer a spec for $C_1; C_2$:

$$\dfrac{\dfrac{\dfrac{\{A'\}\, C_1\, \{A\}}{\{A' * X\}\, C_1\, \{A * X\}}\ (\text{Frame})}{\{A' * X\}\, C_1\, \{B * Y\}}\ (\models) \qquad \dfrac{\{B\}\, C_2\, \{B'\}}{\{B * Y\}\, C_2\, \{B' * Y\}}\ (\text{Frame})}{\{A' * X\}\, C_1; C_2\, \{B' * Y\}}\ (;)$$

Results on biabduction

- For lists, biabduction is harder than entailment (NP-complete vs. PTIME):

  N. Gorogiannis, M. Kanovich and P. O'Hearn.
  The complexity of abduction for separated heap abstractions. In *Proc. SAS-18*, 2011.

- For arrays with arithmetic, biabduction is easier than entailment (NP-complete vs. Π_2^P-hard):

  James Brotherston, Nikos Gorogiannis and Max Kanovich.
  Biabduction (and related problems) in array separation logic. In *Proc. CADE-26*, 2017.

- For other fragments, a theorem-proving approach is generally taken (based on matching "missing" parts of entailments). Note that solution quality is an important consideration.

Thanks for listening!