Introduction to Separation Logic

Heavily borrows from slides by Cristiano Calcagno, Imperial College London

October 15, 2019

Table of Contents

Table of Contents

Syntax of Separation Logic

- Given a decidable base-theory T, the syntax of separation logic $SL(T)_{Loc,Data}$ is presented
- *Loc* and *Data* represent the type of the address and the values
- E.g Setting *Loc* and *Data* to be *Int*, then our addresses and values are integers

$$
\begin{aligned}
P, Q ::= \; & \mathit{false} \mid P \land Q \mid P \lor Q \mid P \rightarrow Q \\
& \mid P * Q \mid P \mathrel{-\!\!*} Q \\
& \mid E = E' \mid E \hookrightarrow E' \mid \mathit{empty}
\end{aligned}
$$

We use E and E' to denote expressions in the base theory, where pointer indirection is not used.

Semantics of Separation Logic

The model consists of an interpretation (I) and a heap (h)

$$I : \mathrm{Var} \to \mathrm{Loc}$$
$$h : \mathrm{Loc} \to \mathrm{Data}$$

$$
\begin{aligned}
I, h &\models \mathit{false} && \text{never satisfied} \\
I, h &\models P \wedge Q && I, h \models P \text{ and } I, h \models Q \\
I, h &\models P \vee Q && I, h \models P \text{ or } I, h \models Q \\
I, h &\models P \to Q && I, h \models P \text{ implies } I, h \models Q \\
I, h &\models E = E' && [\![E]\!]_I = [\![E']\!]_I
\end{aligned}
$$

We use $[\![E]\!]_I$, to denote the value of E under the interpretation I.

Semantics of Separation Logic

Empty heap

$$I, h \models empty$$
$$\text{iff } h = \phi$$

Separating conjunction

$$I, h \models P * Q$$
$$\text{iff } \exists h_1, h_2.(h_1 \perp h_2) \wedge (h = h1 \circ h2) \wedge I, h_1 \models P \wedge I, h_2 \models Q$$

Where $h_1 \perp h_2$ denotes that the heaps are disjoint and $h_1 \circ h_2$ means their union.

Semantics of Separation Logic

Separating Implication

$$I, h \models P \mathrel{-\!\!*} Q$$
$$\text{iff } \forall h'.(h \perp h') \wedge (I, h' \models P) \rightarrow I, h \circ h' \models Q$$

Interpretation : If we extend the current heap with a disjoint heap satisfying P, then the new heap satisfies Q. In some ways, we can imagine that our current heap is only missing the records of P, to make it satisfy Q.

Points to

$$I, h \models E \hookrightarrow E'$$
$$\text{iff } h([\![E]\!]_I) = [\![E']\!]_I$$

Examples

Points to,

$$
\begin{aligned}
&F : x \hookrightarrow 10\\
&I : \{(x, 0)\}\\
&h : \{(0, 10)\}\\
&I, h \models F
\end{aligned}
$$

Separating conjunction,

$$
\begin{aligned}
&F : x \hookrightarrow 10 * y \hookrightarrow 20\\
&I : \{(x, 0), (y, 1)\}\\
&h : \{(0, 10), (1, 20)\}\\
&I, h \models F
\end{aligned}
$$

Examples

Separating Implication

$$\begin{aligned} & I, h \models P \mathrel{-\!\!*} Q \\ \text{iff } & \forall h'.(h \perp h') \wedge (I, h' \models P) \rightarrow I, h \circ h' \models Q \end{aligned}$$

Example,

$$\begin{aligned} & F : (x \hookrightarrow 10) \mathrel{-\!\!*} (x \hookrightarrow 10 * y \hookrightarrow 20) \\ & I : \{(x, 0), (y, 1)\} \\ & h' : \{(0, 10)\} \\ & h : \{(1, 20)\} \\ & h \circ h' : \{(0, 10), (1, 20)\} \\ & I, h \models F \end{aligned}$$

Table of Contents

Translating Separation Logic into Pointer Logic

Points to,

$$I, h \models x \hookrightarrow v$$

$$\iff$$

$$L, M \models \ ^*x = v$$

Separating conjunction,

$$I, h \models x \hookrightarrow v_1 * y \hookrightarrow v_2$$

$$\iff$$

$$L, M \models \ ^*x = v_1 \ \wedge \ ^*y = v_2 \wedge x \neq y$$

Table of Contents

Need for inductive predicates

- Most interesting data structures in programs are defined as inductive systems
- For example : linked lists, trees, graphs
- Being able to reason about these in SL is useful
- But inductive predicates introduce quantifiers

Example - List

$$\text{list } 0\ x \equiv \mathit{empty} \wedge x = \mathit{nil}$$
$$\text{list } n\ x \equiv \exists y.(x \hookrightarrow n, y) * (\mathit{list}\ (n-1)\ y)$$

- This defines a linked-list rooted at x
- Base case : empty list where heap is empty and root pointer is null
- Inductive case : the root points to a struct which has a value and the pointer for the remaining list
- Separately, the next pointer points to a list. Prevents pointer aliasing, each pointer is different

Comparison with pointers

Begin by defining a list in pointer logic,

$$\text{list } 0\ x \equiv x = nil$$
$$\text{list } n\ x \equiv \exists y.(x \hookrightarrow v, y) \wedge (list\ (n-1)\ y)$$

This is a satisfying assignment for *list* 3 x,

$$I = \{\ (x, 0), (y, 0), (z, 1), (w, nil)\ \}$$
$$h = \{\ (0, (v, 1)), (1, (v, w))\ \}$$

The pointers x and y got aliased to point to z.

$$list\ 3\ x \equiv$$
$$(x = 0) \hookrightarrow v, 1 \wedge$$
$$(y = 0) \hookrightarrow v, 1 \wedge$$
$$(z = 1) \hookrightarrow v, nil \wedge$$