MLCS 2020

Separation Logic

Symposium - Mathematical Logic for Computer Science

Student:

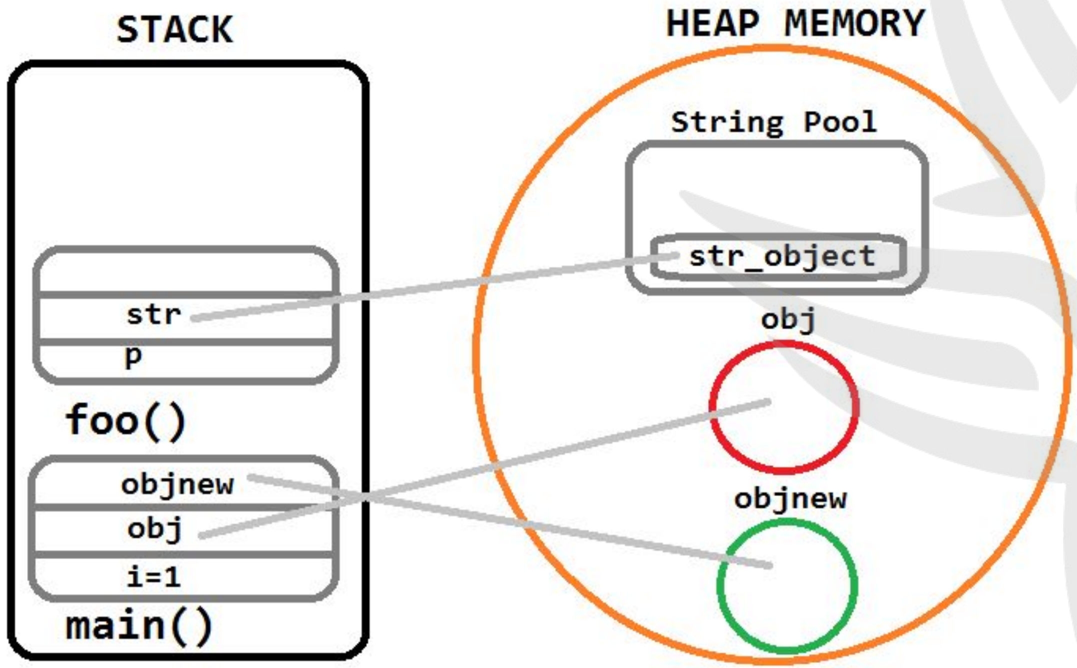
- Riccardo Taiello - 1914000

Professor:

- Lorenzo Carlucci

Sapienza University of Rome | A.Y. 2019-2020

Recap - Stack & Heap



Introduction

Separation Logic (a.k.a. S.L.) is a logic framework that allow us to reason over an abstract machine which its “state” is made by store and heap.

In other words, S.L. handles the issue regarding the pointer in a language such as C, because the standard Hoare Logic (a.k.a. HL) was designed to deal just with a static memory, the dynamic memory (heap) wasn’t considered.

Hoare Logic

To be more precise S.L. is an extension of H.L.

<u>What is Hoare Logic?</u>

is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs.

SAPIENZA

IMP - A Small Programming Language

IMP syntax, the grammar in BNF[1]

For **Aexp**: Arithmetic expression

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1. \qquad n \in \mathbb{Z}$$

For **Bexp**: Boolean expression

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

For **Com**: C

$$c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$$

[1] **Backus-Naur Form**

IMP - Aexp Semantic

IMP Semantic **Aexp**:

- Let define the set of states as $\mathbf{State} = \mathbf{Var} \Rightarrow \mathbf{Val}$ $\quad \mathrm{Val} \in \mathbb{Z}$
- Let define the evaluation function for the arithmetic expression as

$$aval : \mathbf{Aexp} \times \mathbf{State} \Rightarrow \mathbf{Val}$$

$$\begin{array}{lcl} aval(\underline{n}, s) & = & n \\ aval(X, s) & = & s(X) \\ aval(a_1 + a_2, s) & = & aval(a_1, s) + aval(a_2, s) \\ aval(a_1 - a_2, s) & = & aval(a_1, s) - aval(a_2, s) \\ aval(a_1 * a_2, s) & = & aval(a_1, s) \cdot aval(a_2, s) \end{array}$$

IMP - Bexp Semantic

IMP Semantic **Bexp**:

- Let define the evaluation function for the boolean expression as

$$bval : \mathbf{Bexp} \times \mathbf{State} \Rightarrow \mathbf{Bool}$$

$$\begin{array}{lcl}
bval(\mathbf{true}, s) & = & true \\
bval(\mathbf{false}, s) & = & false \\
bval(a_1 = a_2, s) & = & aval(a_1, s) = aval(a_2, s) \\
bval(a_1 < a_2, s) & = & aval(a_1, s) < aval(a_2, s) \\
bval(\neg\, b) & = & \neg\, bval(b, s) \\
bval(b_1 \wedge b_2, s) & = & bval(b_1, s) \wedge bval(b_2, s) \\
bval(b_1 \vee b_2, s) & = & bval(b_1, s) \vee bval(b_2, s)
\end{array}$$

IMP - Structural Operational Semantic of Com

IMP Semantic **Com**:

Let:

- $c \in \mathbf{Com}$
- $s, s' \in \mathbf{State}$
- $\langle c, s \rangle \Rightarrow s'$

$$\frac{}{\langle \mathbf{skip}, s \rangle \Rightarrow s}\ \mathbf{skip}, \Rightarrow$$

$$\frac{}{\langle X := a, s \rangle \Rightarrow s[X := aval(a, s)]}\ \text{Loc}, \Rightarrow$$

$$\frac{\langle c_1, s \rangle \Rightarrow s'' \qquad \langle c_2, s'' \rangle \Rightarrow s'}{\langle c_1; c_2, s \rangle \Rightarrow s'}\ \text{Comp}, \Rightarrow$$

$$\frac{bval(b, s) = \mathbf{true} \qquad \langle c_1, s \rangle \Rightarrow s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ \{c_1\}\ \mathbf{else}\ \{c_2\}, s \rangle \Rightarrow s'}\ \text{If-}\mathbf{true}, \Rightarrow$$

$$\frac{bval(b, s) = \mathbf{false} \qquad \langle c_2, s \rangle \Rightarrow s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ \{c_1\}\ \mathbf{else}\ \{c_2\}, s \rangle \Rightarrow s'}\ \text{If-}\mathbf{false}, \Rightarrow$$

$$\frac{bval(b, s) = \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ \{c\}, s \rangle \Rightarrow s}\ \text{While-}\mathbf{false}, \Rightarrow$$

$$\frac{bval(b, s) = \mathbf{true} \quad \langle c, s \rangle \Rightarrow s'' \quad \langle \mathbf{while}\ b\ \mathbf{do}\ \{c\}, s'' \rangle \Rightarrow s'}{\langle \mathbf{while}\ b\ \mathbf{do}\ \{c\}, s \rangle \Rightarrow s'}\ \text{While-}\mathbf{true}, \Rightarrow$$

Hoare Logic - Assertion

Just for convention we renamed the **State s = σ** and the set of **States as Σ.**

In order to reason over the correctness of the program, we are going to use **FOL**

Ghost variable

$$\textbf{Aexpv}: \quad a \quad ::= \quad \underline{n} \mid x \mid X \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$$

$$\textbf{Assn}: \quad A \quad ::= \quad \textbf{true} \mid \textbf{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \rightarrow A_2 \mid \forall x A \mid \exists x A$$

Hoare Logic - Semantic of FOL in Hoare Logic

Let **a** ∈ **Aexp**, **σ** ∈ **Σ** e *I* : Var →**Z.**

$$
\begin{array}{rcl}
[\![\underline{n}]\!]^I_\sigma & = & n \\
[\![x]\!]^I_\sigma & = & I(x) \\
[\![X]\!]^I_\sigma & = & \tilde{\sigma}(X) \\
[\![a_1 + a_2]\!]^I_\sigma & = & [\![a_1]\!]^I_\sigma + [\![a_2]\!]^I_\sigma \\
[\![a_1 - a_2]\!]^I_\sigma & = & [\![a_1]\!]^I_\sigma - [\![a_2]\!]^I_\sigma \\
[\![a_1 \times a_2]\!]^I_\sigma & = & [\![a_1]\!]^I_\sigma \times [\![a_2]\!]^I_\sigma
\end{array}
$$

Hoare Logic - Semantic of FOL in Hoare Logic

Let **A** ∈ **Assn**, **σ** ∈ **Σ** e *I* : Var →**Z.**
We define the relation **true** in **σ** w.r.t. ***I*** **i.i.f** $\sigma \models^I A$
by structural induction on **A:**

$$\sigma \models^I \mathbf{true}$$

$$\sigma \models^I a_1 = a_2 \quad \Leftrightarrow \quad [\![a]\!]^I_\sigma = [\![a]\!]^I_\sigma$$

$$\sigma \models^I a_1 \leq a_2 \quad \Leftrightarrow \quad [\![a]\!]^I_\sigma \leq [\![a]\!]^I_\sigma$$

$$\sigma \models^I \neg A \quad \Leftrightarrow \quad \sigma \not\models^I A$$

$$\sigma \models^I A_1 \wedge A_2 \quad \Leftrightarrow \quad \sigma \models^I A_1 \text{ e } \sigma \models^I A_2$$

$$\sigma \models^I A_1 \vee A_2 \quad \Leftrightarrow \quad \sigma \models^I A_1 \text{ o } \sigma \models^I A_2$$

$$\sigma \models^I A_1 \rightarrow A_2 \quad \Leftrightarrow \quad \sigma \models^I A_1 \text{ implies } \sigma \models^I A_2$$

$$\sigma \models^I \forall x A \quad \Leftrightarrow \quad \sigma \models^{I[n/x]} A \text{ for every } n \in \mathbb{Z}$$

$$\sigma \models^I \exists x A \quad \Leftrightarrow \quad \sigma \models^{I[n/x]} A \text{ for some } n \in \mathbb{Z}$$

Hoare Logic

Hoare triple

$\{A\}\,c\,\{B\}$ $\qquad A, B \in$ **Assn** , $c \in \mathbf{Com}.$

Just for convention we renamed the **State s = σ** and the set of **States as Σ**

These are read, roughly speaking, as for any state **σ** satisfying A, if c transforms state **σ** to $\boldsymbol{\sigma}^1$, then $\boldsymbol{\sigma}^1$ satisfies B.

Hoare Logic - Inference Rules

General rule HL:

The following inference rules are called general because they apply to any program C

Note: $A \supset B \equiv A \rightarrow B$

$$\frac{}{\{P\}\ C\ \{true\}}\ (true)$$

$$\frac{}{\{false\}\ C\ \{P\}}\ (false)$$

$$\frac{P \supset Q \qquad \{Q\}\ C\ \{R\}}{\{P\}\ C\ \{R\}}\ (str)$$

$$\frac{\{P\}\ C\ \{Q\} \qquad Q \supset R}{\{P\}\ C\ \{R\}}\ (weak)$$

$$\frac{\{P\}\ C\ \{Q_0\}\ \ldots\ \{P\}\ C\ \{Q_n\}}{\{P\}\ C\ \{Q_0 \wedge \cdots \wedge Q_n\}}\ (and)$$

$$\frac{\{P_0\}\ C\ \{Q\}\ \ldots\ \{P_n\}\ C\ \{Q\}}{\{P_0 \vee \cdots \vee P_n\}\ C\ \{Q\}}\ (or)$$

Hoare Logic - Inference Rules

Special rules HL:

The following inference rules are called special because they are specific of each program construct.

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}}\ (\mathit{skip})$$

$$\frac{}{\{[E/x]P\}\ x := E\ \{P\}}\ (\mathit{assign})$$

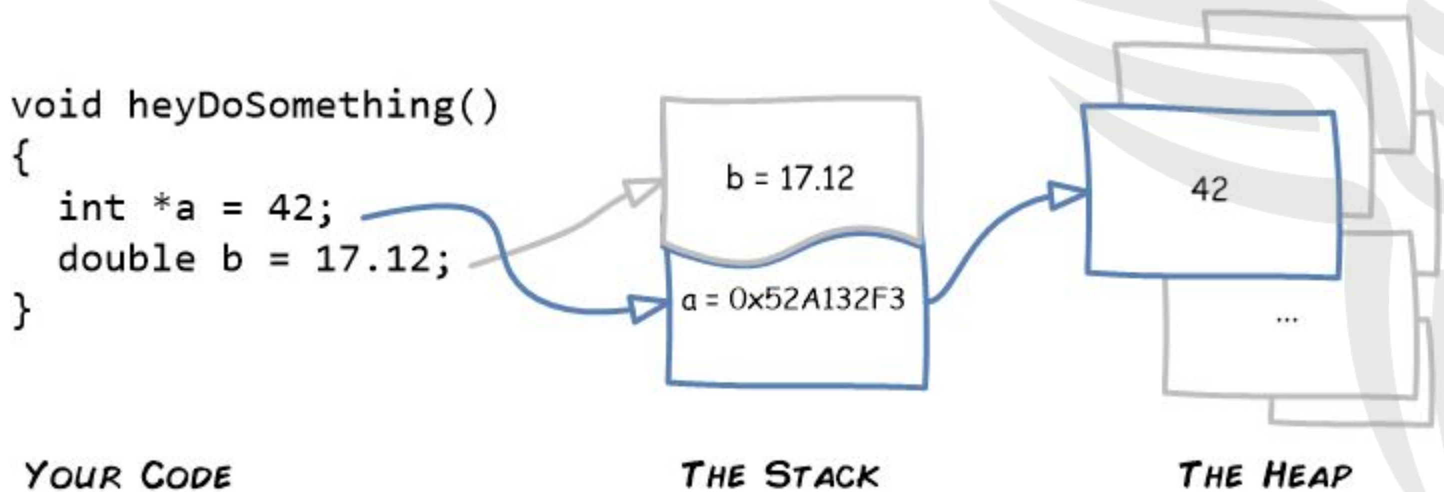
$$\frac{\{P \wedge Q\}\ C_1\ \{R\} \qquad \{P \wedge \neg Q\}\ C_2\ \{R\}}{\{P\}\ \texttt{if}\ Q\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{R\}}\ (\mathit{if})$$

$$\frac{\{P \wedge Q\}\ C\ \{P\}}{\{P\}\ \texttt{while}\ Q\ \texttt{do}\ C\ \{P \wedge \neg Q\}}\ (\mathit{while})$$

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}\ (\mathit{comp})$$

Why we need to extend the HL?

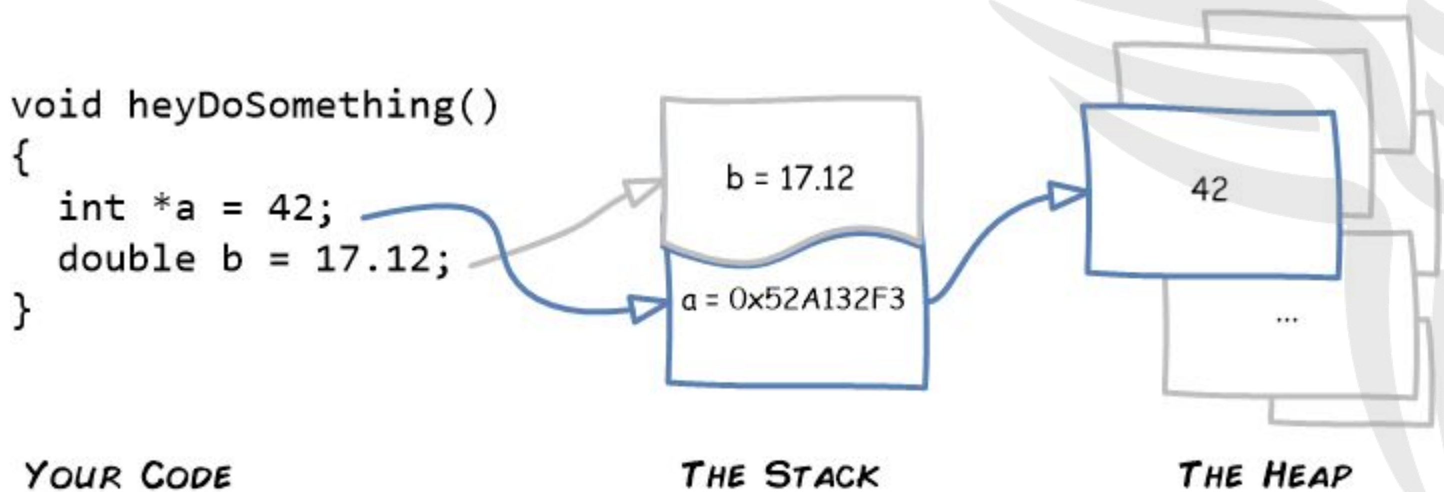
Programs are represented semantically as relations between initial and final states. To represent the pointer structures used to represent lists, trees etc. we need to add another component to states called the **heap**.



Why we need to extend the HL?

To be more **precise**:

Inside the Heap(RAM) at the address 0x52A132F3 we have the value **42**



Why we need to extend the HL?

In the previous example, the lookup operator was *, here
*__a__ is represented by **[a]**

$$
\begin{array}{lllr}
\langle comm \rangle & ::= & \ldots & \\
& | & \langle \mathsf{var} \rangle := \mathbf{cons}(\langle \mathsf{exp} \rangle, \ldots, \langle \mathsf{exp} \rangle) & \text{allocation} \\
& | & \langle \mathsf{var} \rangle := [\langle \mathsf{exp} \rangle] & \text{lookup} \\
& | & [\langle \mathsf{exp} \rangle] := \langle \mathsf{exp} \rangle & \text{mutation} \\
& | & \mathbf{dispose}\ \langle \mathsf{exp} \rangle & \text{deallocation}
\end{array}
$$

Why we need to extend the HL?

With HL we meet problem when the heap is introduced - e.g. assignment rule:

$$\{P[e/x]\}\ \mathsf{x} := \mathsf{e}\ \{P\}$$

$$\Downarrow$$

✓	$\{1 = 1\}$	$\mathbf{x := 1}$	$\{x = 1\}$
✗	$\{?\}$	$\mathbf{[x] := 1}$	$\{[x] = 1\}$

Why we need to extend the HL?

Preconditions have to explicitly specify that a set of addresses do not overlap with the addresses affected by a command.

$$\{x \mapsto - \wedge [y] = 2 \wedge x \neq y\}\ [x] := 1\ \{[x] = 1 \wedge [y] = 2 \wedge x \neq y\}$$

x point **to some** value into the heap

Separation Logic - Introduction

Simplifies the problem of specifying preconditions by introducing two new logical connectives:

$*$: **Separation conjunction**

[y] = 2

$-*$: **Separation implication**

$$\{x \mapsto - * y \mapsto 2\}\ [x] := 1\ \{x \mapsto 1 * y \mapsto 2\}\ (* \text{ has } x \neq y \text{ built into it})$$

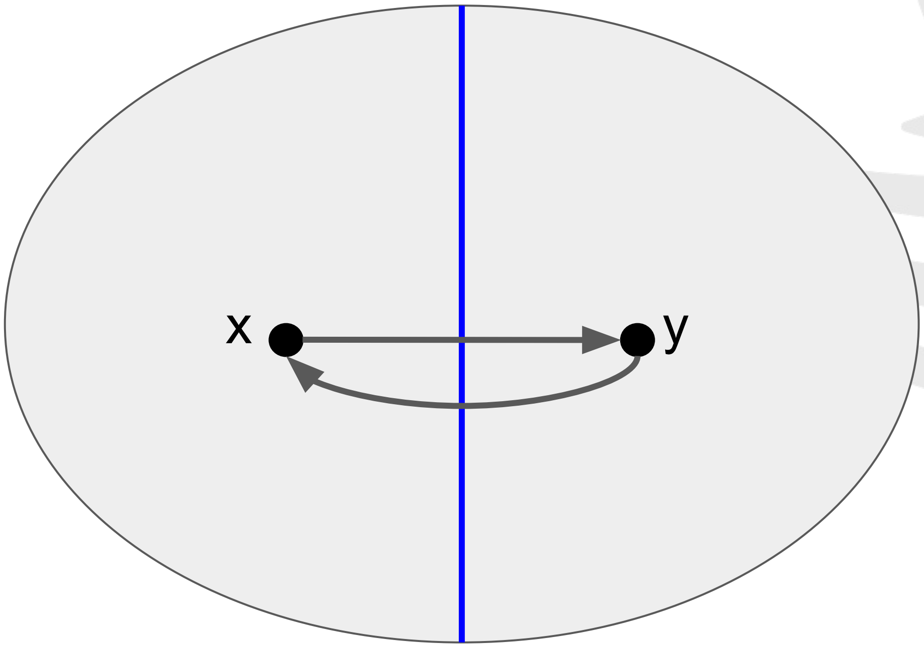
x points **to some** value into the heap. **[x] = _**

Separation conjunction - Intuition

$$x \mapsto y \ * \ y \mapsto x$$

x = 10
y= 42

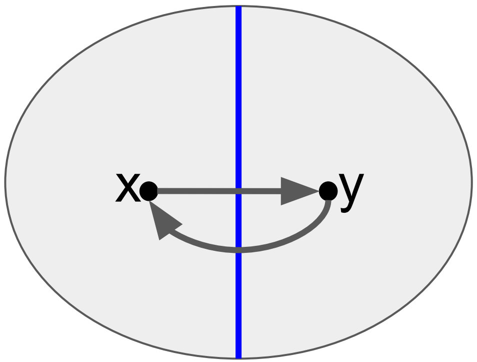
RAM(heap)	
Address	Value
10	42
42	10



Separation conjunction - Intuition

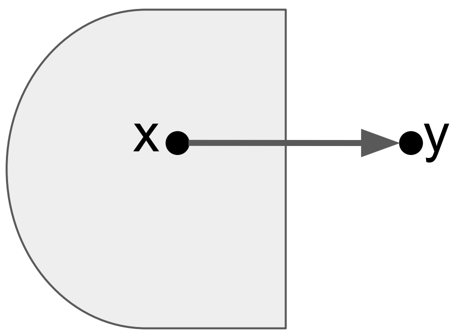
$$x \mapsto y \ * \ y \mapsto x$$

Roughly speaking: x points to y, and separately y points to x



Separation conjunction - Intuition

$x \mapsto y$

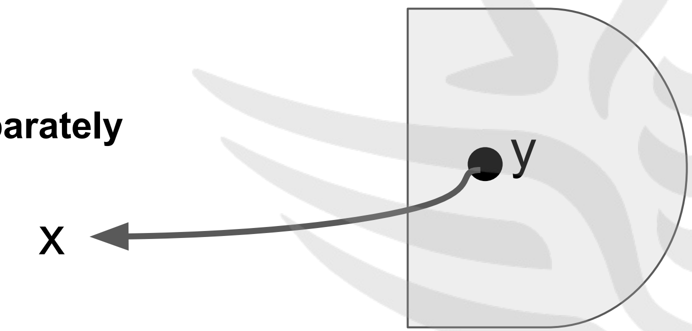


x = 10
y= 42

RAM splitted (heap)	
Address	Value
10	42

And Separately

$y \mapsto x$

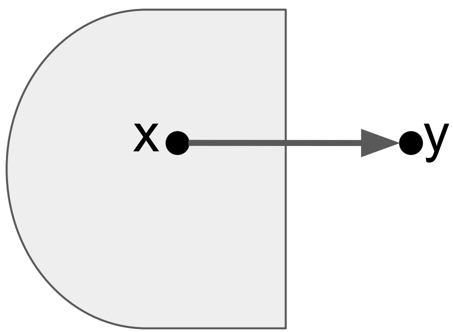


x = 10
y= 42

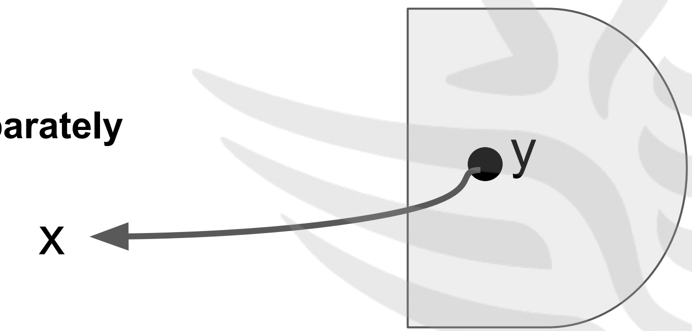
RAM splitted (heap)	
Address	Value
42	10

Separation conjunction - Intuition

Notice that: the separating conjunction splits the heap/RAM, but it does not split the association of variables to values: heap cells, but not variable associations



And Separately



x = 10
y= 42

RAM splitted (heap)	
Address	Value
10	42

x = 10
y= 42

RAM splitted (heap)	
Address	Value
42	10

Separation Logic - Formal Definition

The model has two components, the store and the heap:

$$State = (Store \times Heap)$$

- The store is a finite partial function mapping from variables to integers

$$Store = Var \Rightarrow Val$$

- The heap is a finite function from natural numbers to integers.

$$Heap = Address \Rightarrow Val$$

Separation Logic - Formal Definition

1. $dom(h)$ denotes the domain of definition of a heap $h \in \texttt{Heaps}$, and $dom(s)$ is the domain of $s \in \texttt{Stores}$;
2. $h \# h'$ says that $dom(h) \cap dom(h') = \emptyset$;
3. $h \bullet h'$ denotes the union of functions with disjoint domains, which is undefined if the domains overlap;
4. $(f \mid i \mapsto j)$ is the partial function like f except that i goes to j.

2. In other way states that the address space of h and h' doesn't overlap

Separation Logic - Semantic of Assn extended

Notice that, it requires a **single** element inside the heap's domain

$s, h \models B \quad \text{iff } [\![B]\!]s = true$

$s, h \models E \mapsto F \text{ iff } \{[\![E]\!]s\} = dom(h) \text{ and } h([\![E]\!]s) = [\![F]\!]s$

$s, h \models false \quad \text{never}$

$s, h \models P \Rightarrow Q \text{ iff if } s, h \models P \text{ then } s, h \models Q$

$s, h \models \forall x.P \quad \text{iff } \forall v \in \texttt{Ints}. [s \mid x \mapsto v], h \models P$

$s, h \models emp \quad \text{iff } h = [\,] \text{ is the empty heap}$

$s, h \models P * Q \quad \text{iff } \exists h_0, h_1.\ h_0 \# h_1,\ h_0 * h_1 = h,\ s, h_0 \models P \text{ and } s, h_1 \models Q$

$s, h \models P -\!\!* Q \quad \text{iff } \forall h'. \text{if } h' \# h \text{ and } s, h' \models P \text{ then } s, h \bullet h' \models Q$

Separation Logic - And Inference Rule

$$\text{Let } Mod(C) = \{x | \exists a \in \mathbf{Aexp}.x := a \in C\}$$

$$\frac{\{A\}C\{B\}}{\{A \wedge R\}C\{B \wedge R\}} \ \mathrm{FV(R)} \cap \mathrm{Mod(C)} = \emptyset$$

It works for the HL.
BUT In SL might not be enough.

Example…. →

Separation Logic - And Inference Rule

x points into some points on the heap

$$\frac{\{x \mapsto \_\}[x] := 4\{x \mapsto 4\}}{\{x \mapsto \_ \wedge y \mapsto 3\}[x] := 4\{x \mapsto 4 \wedge y \mapsto 3\}}$$

- $s, h \models x \mapsto \_ \wedge y \mapsto 3$ for some $n \in Addr$ $dom(h) = \{n\}$. and $s(x) = s(y) = n$, $h(n) = 3$
- $([x] := 4, s, h) \downarrow (\_, s, \{n \mapsto 4\})$ h'
- $s, h' \not\models x \mapsto \ 4 \wedge y \mapsto 3$

It requires that the heap is made up at by just one address

Separation Logic - Separation Conjunction

To overcome this issue, in SL we introduce another inference rule, which is called the Frame-Rule

$$\frac{\{A\}C\{B\}}{\{A * R\}C\{B * R\}} \ \mathrm{FV(R)} \cap \mathrm{Mod(C)} = \emptyset$$

The separation conjunction allows to split the heap in 2 parts. It avoids the heap's domain made up by just one element

Separation Logic - Tool Example

,

[Infer Static Analyzer]

References

- Hoare Logic: https://mitpress.mit.edu/books/formal-semantics-programming-languages
- Separation Logic: http://www0.cs.ucl.ac.uk/staff/p.ohearn/papers/Marktoberdorf11LectureNotes.pdf
- Separation Logic: , http://www0.cs.ucl.ac.uk/staff/J.Brotherston/slides/ANU_LSS_seplogic.pdf