

Separation Logic

Niccolò Piazzesi

January 4, 2022

Outline

- 1 Introduction
- 2 Theoretical Foundations
- 3 Reasoning with separation logic
- 4 Tools

Outline

- 1 Introduction
- 2 Theoretical Foundations
- 3 Reasoning with separation logic
- 4 Tools

Brief recap: reasoning about code

- Program semantics described by logical conditions satisfied by language constructs
- Classical model, first put forward by Robert W. Floyd and Tony Hoare

$$\{P\}S\{Q\}$$

- P : pre-conditions
- S : statement
- Q : post conditions

Partial correctness: **If the initial state fullfils pre-conditions and the statement terminates**, the final state satisfies the post conditions.

Total correctness: **If the initial state fullfils the pre-conditions** then the statement terminates and the final state satisfies the post-conditions.

Limitations

Does not work for non terminating programs

Limitations

Does not work for non terminating programs

Becomes complex with modular constructs such as objects and unconditional jumps

Limitations

Does not work for non terminating programs

Becomes complex with modular constructs such as objects and unconditional jumps

Global view of state becomes a burden when introducing pointers(think of pointer aliasing..)

Motivating example

```
void deletetree(struct node *root){  
    if(root != NULL){  
        struct node *left = root->l;  
        struct node *right = root->r;  
        deletetree(left);  
        deletetree(right);  
        free(root);  
    }  
}
```

How can we prove
memory safety?

First attempt

Outline

- 1 Introduction
- 2 Theoretical Foundations
- 3 Reasoning with separation logic
- 4 Tools

The model

$$\text{Ints} \triangleq \{\dots, -1, 0, 1, \dots\}$$

$$\text{Atoms}, \text{Locations} \subseteq \text{Ints}$$

$$\text{Stores} \triangleq \text{Variables} \rightarrow_{fin} \text{Ints}$$

$$\text{Variables} \triangleq \{x, y, \dots\}$$

$$\text{Locations} \cap \text{Atoms} = \{\}, \text{nil} \in \text{Atoms}$$

$$\text{Heaps} \triangleq \text{Locations} \rightarrow_{fin} \text{Ints}$$

$$\text{States} \triangleq \text{Stores} \times \text{Heaps}$$

$$\llbracket E \rrbracket_s \in \text{Ints}, \llbracket B \rrbracket_s \in \{\text{true}, \text{false}\}$$

$$h \in \text{Heaps}, h[E] \in \text{Ints}$$

The language

Expressions:

$$E, F, G := x, y, \dots \mid 0 \mid 1 \mid E + F \mid E \times F \mid E - F$$
$$B := false \mid B \Rightarrow B \mid E = F \mid E < F \mid isatom?(E) \mid isloc?(E)$$

The language

Expressions:

$$E, F, G := x, y, \dots \mid 0 \mid 1 \mid E + F \mid E \times F \mid E - F$$
$$B := false \mid B \Rightarrow B \mid E = F \mid E < F \mid isatom?(E) \mid isloc?(E)$$

Assertions:

$$P, Q, R ::= B \mid E \mapsto F \qquad \textit{Atomic Formulae}$$
$$\mid false \mid P \Rightarrow Q \mid \forall x. P \qquad \textit{Classical Logic}$$
$$\mid emp \mid P * Q \mid P - * Q \qquad \textit{Spatial Connectives}$$

The language

Expressions:

$$E, F, G := x, y, \dots \mid 0 \mid 1 \mid E + F \mid E \times F \mid E - F$$
$$B := false \mid B \Rightarrow B \mid E = F \mid E < F \mid isatom?(E) \mid isloc?(E)$$

Assertions:

$$P, Q, R ::= B \mid E \mapsto F$$

Atomic Formulae

$$\mid false \mid P \Rightarrow Q \mid \forall x. P$$

Classical Logic

$$\mid emp \mid P * Q \mid P - * Q$$

Spatial Connectives

$$\neg P = P \Rightarrow False$$
$$true = \neg(false)$$
$$P \vee Q = \neg(P) \Rightarrow Q$$

The language

Expressions:

$$E, F, G := x, y, \dots \mid 0 \mid 1 \mid E + F \mid E \times F \mid E - F$$

$$B := false \mid B \Rightarrow B \mid E = F \mid E < F \mid isatom?(E) \mid isloc?(E)$$

Assertions:

$$P, Q, R ::= B \mid E \mapsto F \quad \text{Atomic Formulae}$$

$$\mid false \mid P \Rightarrow Q \mid \forall x.P \quad \text{Classical Logic}$$

$$\mid emp \mid P * Q \mid P - * Q \quad \text{Spatial Connectives}$$

$$\neg P = P \Rightarrow False$$

$$true = \neg(false)$$

$$P \vee Q = \neg(P) \Rightarrow Q$$

$$P \wedge Q = \neg(\neg P \vee \neg Q)$$

$$\exists x.P = \neg \forall x. \neg P$$

Some notation

- ❶ $dom(h)$ and $dom(s)$ denote the domain of definition for $h \in Heaps$ and $s \in Stores$, respectively
- ❷ $h \# h' \rightarrow dom(h) \cap dom(h') = \emptyset$
- ❸ $h * h'$ is the union of disjoint heaps
- ❹ $(f|i \mapsto j)$ represent the partial function that behaves like f except that i goes to j .

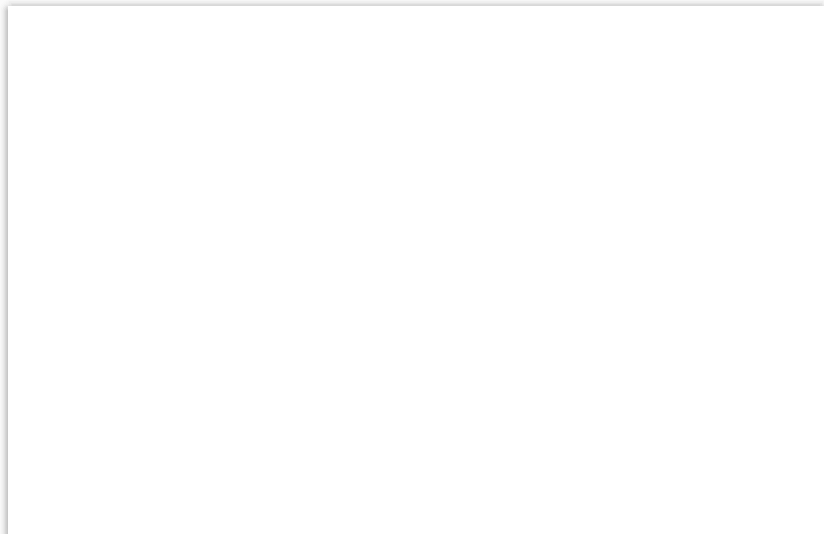
$$E \mapsto F_0, \dots, F_n \triangleq (E \mapsto F_0) * \dots * (E + n \mapsto F_n)$$

$$E \doteq F \triangleq (E = F) \wedge emp$$

$$E \mapsto - \triangleq \exists y. E \mapsto y$$

Semantics

For store s and heap h



For store s and heap h

$$s, h \models B \text{ iff } \llbracket B \rrbracket_s = \text{true}$$

$$s, h \models E \mapsto F \text{ iff } \{\llbracket E \rrbracket_s\} = \text{dom}(h) \text{ and } h(\llbracket E \rrbracket_s) = \llbracket F \rrbracket_s$$

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

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

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

For store s and heap h

$$s, h \models B \text{ iff } \llbracket B \rrbracket_s = \text{true}$$

$$s, h \models E \mapsto F \text{ iff } \{\llbracket E \rrbracket_s\} = \text{dom}(h) \text{ and } h(\llbracket E \rrbracket_s) = \llbracket F \rrbracket_s$$

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

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

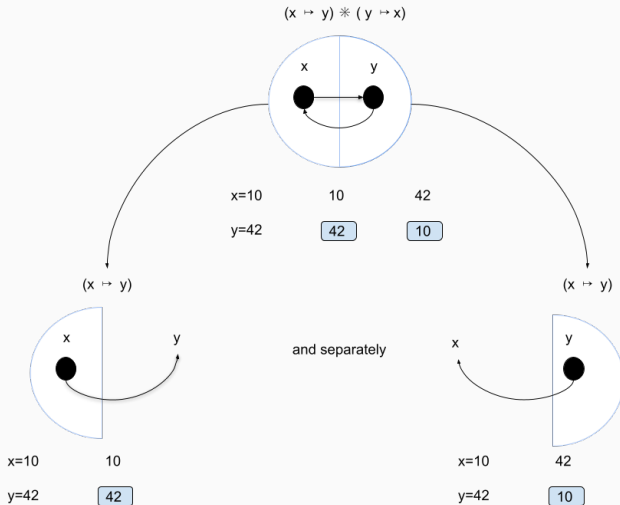
$$s, h \models \forall x. P \text{ iff } \forall v \in \text{Ints}. [s \mid x \mapsto v], h \models P$$

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

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

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

Visual example



Proof rules in seapartion logic are divide in:

- 1 Axioms for basic mutation commands \rightarrow *Small axioms*
- 2 Inference rules for modular reasoing \rightarrow *Structural rules*

Small axioms

Small axioms

$$\{E \mapsto -\}[\mathbf{E}] := \mathbf{F}\{E \mapsto F\} \text{ ("Store")}$$

Small axioms

$\{E \mapsto -\}[\mathbf{E}] := \mathbf{F}\{E \mapsto F\}$ ("Store")

$\{E \mapsto -\}\mathbf{free}(\mathbf{E})\{emp\}$ ("Reclaim memory")

Small axioms

$\{E \mapsto -\}[\mathbf{E}] := \mathbf{F}\{E \mapsto F\}$ ("Store")

$\{E \mapsto -\}\mathbf{free}(\mathbf{E})\{emp\}$ ("Reclaim memory")

$\{x \doteq m\}\mathbf{x} := \mathbf{cons}(\mathbf{E}_1, \dots, \mathbf{E}_k)\{x \mapsto E_1[m/x], \dots, E_k[m/x]\}$
("Allocate memory")

Small axioms

$\{E \mapsto -\}[\mathbf{E}] := \mathbf{F}\{E \mapsto F\}$ ("Store")

$\{E \mapsto -\}\mathbf{free}(\mathbf{E})\{emp\}$ ("Reclaim memory")

$\{x \doteq m\}\mathbf{x} := \mathbf{cons}(\mathbf{E}_1, \dots, \mathbf{E}_k)\{x \mapsto E_1[m/x], \dots, E_k[m/x]\}$
("Allocate memory")

$\{x \doteq n\}\mathbf{x} := \mathbf{E}\{x \doteq (E[n/x])\}$

Small axioms

$$\{E \mapsto -\}[\mathbf{E}] := \mathbf{F}\{E \mapsto F\} \text{ ("Store")}$$

$$\{E \mapsto -\}\mathbf{free}(\mathbf{E})\{emp\} \text{ ("Reclaim memory")}$$

$$\{x \doteq m\}\mathbf{x} := \mathbf{cons}(\mathbf{E}_1, \dots, \mathbf{E}_k)\{x \mapsto E_1[m/x], \dots, E_k[m/x]\} \\ \text{("Allocate memory")}$$

$$\{x \doteq n\}\mathbf{x} := \mathbf{E}\{x \doteq (E[n/x])\}$$

$$\{E \mapsto n \wedge x = m\}\mathbf{x} := [\mathbf{E}]\{x = n \wedge E[m/x] \mapsto n\} \text{ ("Load")}$$

Structural rules

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * \textit{frame}\}C\{Q * \textit{frame}\}} \quad \textit{Mod}(C) \cap \textit{Free}(\textit{frame}) = \emptyset$$

Structural rules

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * \text{frame}\}C\{Q * \text{frame}\}} \quad \text{Mod}(C) \cap \text{Free}(\text{frame}) = \emptyset$$

Auxiliary variable elimination

$$\frac{\{P\}C\{Q\}}{\{\exists x.P\}C\{\exists x.Q\}} \quad x \notin \text{Free}(C)$$

Structural rules

Variable substitution

$$\frac{\{P\}C\{Q\}}{(\{P\}C\{Q\})[E_1/x_1, \dots E_k/x_k]}$$

x_i free and if $x_i \in \text{Mod}(C)$ then E_i is not free in any E_j

Structural rules

Variable substitution

$$\frac{\{P\}C\{Q\}}{(\{P\}C\{Q\})[E_1/x_1, \dots E_k/x_k]}$$

x_i free and if $x_i \in \text{Mod}(C)$ then E_i is not free in any E_j

Rule of consequence

$$\frac{P \Rightarrow P' \quad \{P\}C\{Q\} \quad Q \Rightarrow Q'}{\{P\}C\{Q'\}}$$

Derived laws

The structural rules can be used to obtain more convenient derived laws.

As an example, we can simplify the rule for memory allocation by assuming $x \notin \text{Free}(E_1, \dots, E_k)$.

$$\{emp\}x := cons(E_1, \dots, E_k)\{x \mapsto E_1, \dots, E_k\}$$

Derived laws

The structural rules can be used to obtain more convenient derived laws.

As an example, we can simplify the rule for memory allocation by assuming $x \notin \text{Free}(E_1, \dots, E_k)$.

$$\{emp\}x := cons(E_1, \dots, E_k)\{x \mapsto E_1, \dots, E_k\}$$

The core system can also be extended with the usual Hoare rules

$$\frac{\{P \wedge B\}C\{Q\} \quad \{P \wedge \neg B\}C'\{Q\}}{\{P\}if \ B \ then \ C \ else \ C'\{Q\}}$$

Revisiting the tree example

```
void deletetree(struct node *root){  
    if(root != NULL){  
        struct node *left = root->l;  
        struct node *right = root->r;  
        deletetree(left);  
        deletetree(right);  
        free(root);  
    }  
}
```

Revisiting the tree example

```
void deletetree(struct node *root){  
    if(root != NULL){  
        struct node *left = root->l;  
        struct node *right = root->r;  
        deletetree(left);  
        deletetree(right);  
        free(root);  
    }  
}
```

Specification:

$\{tree(root)\} deletetree(root) \{emp\}$

$tree(root) = \text{if } root == 0 \text{ then } emp$
 $\text{else } \exists xy.root \mapsto [l : x, r : y] * tree(x) * tree(y)$

Proof:

$$\{ \textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red}) \}$$

Proof:

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red})\}$$

$\textcolor{black}{deletetree}(\textcolor{black}{left});$

Proof:

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red})\}$$

deletetree(left);

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{blue}{tree}(\textcolor{blue}{red})\}$$

Proof:

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red})\}$$

deletetree(left);

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{red}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{blue}{tree}(\textcolor{blue}{red})\}$$

deletetree(right);

Proof:

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red})\}$$

deletetree(left);

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{blue}{tree}(\textcolor{blue}{red})\}$$

deletetree(right);

$$\{\textcolor{blue}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{red}{emp}\}$$

Proof:

$$\{\text{root} \mapsto [l : \text{left}, r : \text{right}] * \text{tree}(\text{left}) * \text{tree}(\text{red})\}$$

deletetree(left);

$$\{\text{root} \mapsto [l : \text{left}, r : \text{right}] * \text{emp} * \text{tree}(\text{red})\}$$

deletetree(right);

$$\{\text{root} \mapsto [l : \text{left}, r : \text{right}] * \text{emp} * \text{emp}\}$$

free(root);

Proof:

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{blue}{tree}(\textcolor{blue}{left}) * \textcolor{red}{tree}(\textcolor{red}{red})\}$$

$\textcolor{blue}{deletetree}(\textcolor{blue}{left});$

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{blue}{tree}(\textcolor{blue}{red})\}$$

$\textcolor{blue}{deletetree}(\textcolor{blue}{right});$

$$\{\textcolor{red}{root} \mapsto [l : \textcolor{blue}{left}, r : \textcolor{red}{right}] * \textcolor{red}{emp} * \textcolor{red}{emp}\}$$

$\textcolor{blue}{free}(\textcolor{red}{root});$




$$\{\textcolor{blue}{emp} * \textcolor{blue}{emp} * \textcolor{blue}{emp}\}$$
$$\{\textcolor{blue}{emp}\}$$

Outline

- 1 Introduction
- 2 Theoretical Foundations
- 3 Reasoning with separation logic
- 4 Tools

Outline

- 1 Introduction
- 2 Theoretical Foundations
- 3 Reasoning with separation logic
- 4 Tools

-  Peter O'Hearn, John Reynolds, and Hongseok Yang.
Local reasoning about programs that alter data structures.
In International Workshop on Computer Science Logic, pages 1–19. Springer, 2001.
-  Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez.
Moving fast with software verification.
In NASA Formal Methods Symposium, pages 3–11. Springer, 2015.
-  Dino Distefano, Peter W. O'Hearn, and Hongseok Yang.
A local shape analysis based on separation logic.
In Holger Hermanns and Jens Palsberg, editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 287–302, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.



Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn.
Smallfoot: Modular automatic assertion checking with
separation logic.

In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf,
and Willem-Paul de Roever, editors, *Formal Methods for
Components and Objects*, pages 115–137, Berlin, Heidelberg,
2006. Springer Berlin Heidelberg.



James Brotherston, Nikos Gorogiannis, Max Kanovich, and
Reuben Rowe.

Model checking for symbolic-heap separation logic with
inductive predicates.

ACM SIGPLAN Notices, 51(1):84–96, 2016.



Josh Berdine, Byron Cook, and Samin Ishtiaq.

Slayer: Memory safety for systems-level code.

In *International Conference on Computer Aided Verification*,
pages 178–183. Springer, 2011.