Separation Logic

Niccolò Piazzesi

January 3, 2022

Outline

Outline

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

Floyd-Hoare Logic in 1 slide

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

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

Partial correctness: **If the inital 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

The model

Ints $\triangleq \{\cdots, -1, 0, 1, \cdots\}$

Atoms, Locations $\subseteq Ints$

Stores $\triangleq$ Variables $\rightharpoonup_{fin}$ Ints

Variables $\triangleq \{x, y, \cdots\}$

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

Heaps $\triangleq$ Locations $\rightharpoonup_{fin}$ Ints

States $\triangleq$ Stores $\times$ Heaps

$[\![E]\!]_s \in Ints$, $[\![B]\!]_s \in \{\text{true}, \text{false}\}$

$h \in Heaps$, $h[E] \in Ints$

The language

Expressions:

$$
\begin{aligned}
E, F, G &:= x, y, \cdots \mid 0 \mid 1 \mid E + F \mid E \times F \mid E - F \\
B &:= false \mid B \Rightarrow B \mid E = F \mid E < F \mid isatom?(E) \mid isloc?(E)
\end{aligned}
$$

The language

Expressions:

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

Assertions:

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

The language

Expressions:

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

Assertions:

$$\begin{array}{llr} P, Q, R ::= B \mid E \mapsto F & & Atomic\ Formulae \\ & \mid false \mid P \Rightarrow Q \mid \forall x.P & Classical\ Logic \\ & \mid emp \mid P * Q \mid P -\!* \ Q & Spatial\ Connectives \end{array}$$

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

The language

Expressions:

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

Assertions:

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

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

$$P \wedge Q = \neg(\neg P \vee \neg Q)$$
$$\exists x.P = \neg\forall x.\neg P$$

Some notation

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

Semantics

For store s and heap h

Semantics

For store s and heap h

$$s, h \models B \; iff \; [\![B]\!]_s = true$$

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

$$s, h \models false \quad never$$

$$s, h \models P \Rightarrow Q \; iff \; if \; s, h \models P \; then \; s, h \models Q$$

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

Semantics

For store s and heap h

$$s, h \models B \; iff \; [\![B]\!]_s = true$$

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

$$s, h \models false \quad never$$

$$s, h \models P \Rightarrow Q \; iff \; if \; s, h \models P \; then \; s, h \models Q$$

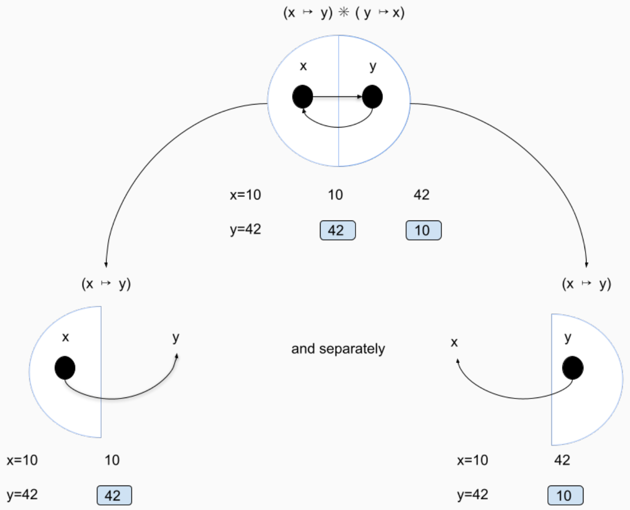
$$s, h \models \forall x.P \; iff \; \forall v \in Ints.[s \mid x \mapsto v], h \models P$$

$$s, h \models emp \; iff \; h = [] \; is \; the \; empty \; heap$$

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

$$s, h \models P -\!\!* \; Q \; iff \; \forall h'. \; if \; h' \# h \; and \; s, h' \models P \; then \; s, h * h' \models Q$$

Visual example



Outline

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

Outline

References I

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.