Separation Logic - Bi-Abduction

Seminar Formal Methods

Franz-Xaver Reichl

February 1, 2018

Overview

- Separation Logic
- Bi-Abduction

Separation Logic

- The semantic of assertions
- Proof rules for programs
- Symbolic Executions

Separation Logic

The semantic of assertions - Prerequisites

Store - Heap

- Store: $Vars \rightharpoonup Vals$
- Heap: $Locations \rightharpoonup Vals$

Locations countably infinite and $Locations \subseteq Vals$

Valuation functions

- $[\![E]\!]s \in Vals$ — Valuation function of an expression in store s
- $[\![B]\!]s \in \{true, false\}$ — Valuation function of a boolean expression in store e

The semantic of assertions - Notations

- s denotes a store h, h_1, h_2 denote heaps
- Let f be some partial function then dom(f) denotes the set of all values for which f is defined.
- if $dom(h_1) \cap dom(h_2) = \emptyset$ we say $h_1 \# h_2$
- Let f be some function then $f[i \mapsto j]$ denotes a function which is everywhere equal to f except in i, which is mapped to j.

Separation Logic

The semantic of assertions - Satisfaction judgement

$s, h \models P$ The assertion P holds for store s and heap h.

Separation Logic

The semantic of assertions - Satisfaction judgement

$s, h \models P$ The assertion P holds for store s and heap h.

- $s, h \models B$ iff $[\![B]\!]s = \mathit{true}$
- $s, h \models E \mapsto F$ iff $\{[\![E]\!]s\} = \mathit{dom}(h)$ and $h([\![E]\!]s) = [\![F]\!]s$
- $s, h \models \mathrm{false}$ never
- $s, h \models P \Rightarrow Q$ iff $s, h \models P$ then $s, h \models Q$
- $s, h \models \forall x.P$ iff $\forall v \in \mathit{Vals}\ s[x \mapsto v], h \models P$
- $s, h \models \mathit{emp}$ iff h is the empty heap
- $s, h \models P * Q$ iff $\exists h_1, h_2\ h_1 \# h_2,\ h_1 \cup h_2 = h$, $s, h_1 \models P$ and $s, h_2 \models Q$
- $s, h \models P \mathrel{-\!\!*} Q$ iff $\forall h_1$ if $h_1 \# h$ and $s, h_1 \models P$ then $s, h \cup h_1 \models Q$

The semantic of assertions - Examples

- With these judgements we can determine rules for the remaining classical logical connectives.
  - negation: Since $\neg P \equiv P \Rightarrow \mathit{false}$ we get $s, h \models \neg P$ iff $s, h \not\models P$
  - conjunction: Since $P \wedge Q \equiv \neg(P \Rightarrow \neg Q)$ we get $s, h \models P \wedge Q$ iff $s, h \models P$ and $s, h \models Q$
- We can define a connective $\hookrightarrow$ s.t.
  $s, h \models E \hookrightarrow F$ iff $s, h \models (E \mapsto F) * \mathit{true}$
  This then means that there is some memory address such that $h([\![E]\!]s) = [\![F]\!]s$ but that the heap function can also be defined for other addresses.

Proof rules for programs - Prerequisites

Hoare calculus

- A Hoare triple $\{P\}\ c\ \{Q\}$ means that if a command c is executed in a state in which the condition P holds than after the execution Q holds.
  Example: $\{Q[e/x]\}x := e\{Q\}$ where $Q[e/x]$ means that x is replaced by e
- There are rules for several constructs of programming languages. e.g.
  - The rule for assignment which I stated above
  - The rule for command sequences:
    $$\frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$
  - Besides this there are rules for conditionals, loops,...

Proof rules for programs - Axiom for new Rule System

We now add to the rules for loops and conditionals from Hoare calculus the following rules.

- $\{E \mapsto -\}\ [E] := F\ \{E \mapsto F\}$
- $\{E \mapsto -\}\ \mathit{free}(E)\ \{\mathit{emp}\}$
- $\{(x = m) \wedge \mathit{emp}\}\ x := \mathit{cons}(E_1, \ldots, E_k)$
  $\{x \mapsto E_1[m/x], \ldots, E_k[m/x]\}$
- $\{(x = n) \wedge \mathit{emp}\}\ x := E\ \{x = (E[n/x]) \wedge \mathit{emp}\}$
- $\{E \mapsto n \wedge x = m\}\ x := [E]\ \{x = n \wedge E[m/x] \mapsto n\}$

Where $[E]$ denotes the heap at position $[\![E]\!]s$
Where $E \mapsto -$ means that the heap is defined for $[\![E]\!]s$

Proof rules for programs - New Rules

- Frame Rule

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

  Where no variable which is free in R is modified by c.

- Additionally rules for auxiliary variable elimination, variable substitution and the rule of consequence (i.e. weakening / strengthening) are used.

Symbolic Executions - Symbolic Heaps

- A pure formula Π is a formula in the form of $(P_1 \wedge \ldots \wedge P_n)$
  - Where P_i describes properties of the store - heap independent.
- A quantifier free symbolic heap Δ is a formula in the form of $\Pi \wedge (S_1 * \ldots * S_m)$
  - Where S_i describes properties of the heap.
- A symbolic heap H is a formula in the form of $\exists \vec{X}.\Delta$
  - Where $\vec{X}$ is a vector of logical variables (i.e. variables which we do not use in programs)

Remark: Since there is no unique way for defining P_i and S_i (for example this depends on the chosen set for the values) this informal description has to suffice.

Symbolic Executions - Rules I

- Empty command

$$\frac{H \vdash H'}{\{H\}\ \mathit{empty}\ \{H'\}}$$

- Assignment

$$\frac{\{\exists X' : x = E[X'/x] \wedge H[X'/x]\}\ C\ \{H'\}}{\{H\}\ x = E; C\ \{H'\}}$$

  Where X' is a fresh logical variable. This shall also hold for the remaining rules.

- Heap lookup

$$\frac{\{\exists X' : x = F[X'/x] \wedge H[X'/x] * E[X'/x] \mapsto F[X'/x]\}\ C\ \{H'\}}{\{H * E \mapsto F\}\ x = [E]; C\ \{H'\}}$$

Symbolic Executions - Rules II

- Change in heap

$$\frac{\{H * E \mapsto G\}\ C\ \{H'\}}{\{H * E \mapsto F\}\ [E] = G; C\ \{H'\}}$$

- New element on heap

$$\frac{\{\exists X' : H[X'/x] * x \mapsto E[X'/x]\}\ C\ \{H'\}}{\{H\}\ x = \mathit{cons}(E); C\ \{H'\}}$$

- Delete heap element

$$\frac{\{H\}\ C\ \{H'\}}{\{H * E \mapsto F\}\ \mathit{free}(E); C\ \{H'\}}$$

Symbolic Executions - Remarks

- Besides these rules there is also a rule for conditionals and for rearranging such formulae.
- If we assume that we have a program with given Pre- and Postcondition and with given loop invariants we can use Symbolic Execution to verify the program with respect to the Pre- and Postcondition.

Symbolic Executions - Example

We want to show:

$\{x \mapsto 1 \wedge y = 2\}\ y := [x]; y := y + 1; \mathit{free}(x)\ \{x = 3 \wedge \mathit{emp}\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y = 2 \wedge y' = 1 \wedge \mathit{emp} \vdash y = 2 \wedge \mathit{emp}}}{\exists y' : y = y' + 1 \wedge y' = 1 \wedge \mathit{emp} \vdash y = 2 \wedge \mathit{emp}}}{\{\exists y' : y = y' + 1 \wedge y' = 1 \wedge \mathit{emp}\}\ \mathit{empty}\ \{y = 2 \wedge \mathit{emp}\}}}{\{\exists y' : y = y' + 1 \wedge y' = 1 \wedge x \mapsto 1\}\ \mathit{free}(x)\ \{y = 2 \wedge \mathit{emp}\}}}{\{y = 1 \wedge x \mapsto 1\}\ y := y + 1; \mathit{free}(x)\ \{y = 2 \wedge \mathit{emp}\}}}{\{\mathit{true} \wedge x \mapsto 1\}\ y := [x]; y := y + 1; \mathit{free}(x)\ \{y = 2 \wedge \mathit{emp}\}}$$

Bi-Abduction

Problem Description

Bi-Abduction is the problem of finding two symbolic heaps called antiframe and frame such that:

$$A * \mathit{antiframe} \vdash B * \mathit{frame}$$

where A and B are given symbolic heaps.
Interpretation: Assume we have a procedure with precondition B which is called in the state A. Then we can use Bi-Abduction to infer what is missing in the calling state to fulfil the precondition and we can infer the parts of the calling state which are not needed in the preconditions.

Bi-Abduction

Solving Bi-Abduction

To solve the problem $A * \mathit{antiframe} \vdash B * \mathit{frame}$:

- Firstly we determine the antiframe as

$$\mathit{antiframe} = \mathit{Abduce}(A, B * \mathit{true})$$

- Secondly determine the frame as

$$\mathit{frame} = \mathit{Frame}(A * \mathit{antiframe}, B)$$

Where $\mathit{Frame}(X, Y) = L \quad \text{s.t.} \quad X \vdash Y * L$
Where $\mathit{Abduce}(X, Y) = M \quad \text{s.t.} \quad X * M \vdash Y$

Bi-Abduction

Abduction

Abduction is the problem of finding a symbolic heap Q such that:

$$A * Q \vdash B$$

Abduce(A,B) describes an algorithm for solving this problem:
$Abduce(A, B) =$

1. Find a symbolic heap M s.t. $A * [M] \rhd B$
2. If $A * M$ is inconsistent return fail, else M.

Bi-Abduction

Solving $A * [M] \rhd B$

Idea: recursively apply proof rules for abductive inference. We denote the alogorithm for solving this problem with AbduceAux $AbduceAux(A, B) =$

- If an axiom of our proof rules applies, return M indicated by the axiom.
- Else if some rule applies, return AbduceAux(A',B') where A' and B' are given by the condition of the rule
- If no rule applies, return fail.

Bi-Abduction

Selected Proof rules for abductive inference

- Axiom base-true

$$\frac{}{\Delta * [\exists \vec{X}.\Pi \wedge emp] \rhd \exists \vec{X}.\Pi \wedge true}$$

- missing

$$\frac{\Delta * [M] \rhd \Delta' \qquad \Delta * \exists \vec{X}.B(E, E') \not\vdash false}{\Delta * [M * \exists \vec{X}.B(E, E')] \rhd \Delta' * \exists \vec{X}.B(E, E')}$$

  Where $B(E, E')$ is $E \mapsto E'$ or $ls(E, E')$

- match

$$\frac{(E_0 = E_1 \wedge \Delta) * [M] \rhd \exists \vec{Y}.\Delta'}{\Delta * E \mapsto E_0 * [\exists \vec{X}.E_0 = E_1 \wedge M] \rhd \exists \vec{X}\vec{Y}.\Delta' * E \mapsto E_1}$$

  Where $\vec{Y} \cap FreeLogVar(E_1) = \emptyset$

Bi-Abduction

Proof rules for abductive inference - Example

$$\dfrac{\dfrac{\dfrac{}{(y = X \wedge emp) * [emp] \rhd true}\ \text{base-true}}{(y = X \wedge emp) * [ls(X, 0)] \rhd ls(X, 0) * true}\ \text{missing}}{x \mapsto y * [y = X \wedge ls(X, 0)] \rhd x \mapsto X * ls(X, 0) * true}\ \text{match}$$

Bi-Abduction

We now use Bi-Abduction to extend our rules for symbolic execution. If symbolic execution fails we use:

$$\frac{\{A\}C\{B\}}{\{P * M\}C\{B * L\}}$$

Where M and L are a solution of the Bi-Abduction Problem.
$P * M \vdash A * L$
In this case we have to add P to the preconditions of the start of symbolic execution and can continue with symbolic execution.

References

- Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. "Symbolic Execution with Separation Logic". In: *Programming Languages and Systems*. Ed. by Kwangkeun Yi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 52–68. ISBN: 978-3-540-32247-4.
- Cristiano Calcagno et al. "Compositional Shape Analysis by Means of Bi-Abduction". In: *J. ACM* 58.6 (2011), 26:1–26:66. DOI: 10.1145/2049697.2049700. URL: http://doi.acm.org/10.1145/2049697.2049700.

- Peter W. O'Hearn. "A Primer on Separation Logic (and Automatic Program Verification and Analysis).". In: *Software Safety and Security*. Ed. by Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann. Vol. 33. NATO Science for Peace and Security Series - D: Information and Communication Security. IOS Press, 2012, pp. 286–318. ISBN: 978-1-61499-028-4. URL: http://dblp.uni-trier.de/db/series/natosec/natosec33.html#OHearn12.
- Wolfgang Schreiner. *Lecture notes Formal Methods in Software Development*. 2016. URL: https://moodle.risc.jku.at/course/view.php?id=131.

Thank you for your attention!