Semantics and Verification of Software

Summer Semester 2019

Lecture 19: Separation Logic III (Soundness) & Wrap-Up

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

`https://moves.rwth-aachen.de/teaching/ss-19/sv-sw/`

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Semantics of Separation Logic

Definition (Semantics of SL assertions)

Let $A \in SLA$ and $(s, h) \in \Sigma$. The relation $(s, h) \models A$ is inductively defined by:

$$
\begin{array}{lll}
(s,h) \models \mathrm{emp} & \text{if } \mathrm{dom}(h) = \emptyset \\
(s,h) \models a \mapsto a' & \text{if } h = h_\emptyset[\mathfrak{A}[\![a]\!]s \mapsto \mathfrak{A}[\![a']\!]s] \\
(s,h) \models A_1 * A_2 & \text{if } \exists h_1, h_2 \in \mathit{Heap} : h = h_1 \uplus h_2, \\
 & \quad (s, h_1) \models A_1 \text{ and } (s, h_2) \models A_2 \\
(s,h) \models \forall x : A & \text{if } \forall z \in \mathbb{Z} : (s[x \mapsto z], h) \models A \\
(s,h) \models \mathrm{true} & \\
(s,h) \models a_1 = a_2 & \text{if } \mathfrak{A}[\![a_1]\!]s = \mathfrak{A}[\![a_2]\!]s \\
(s,h) \models a_1 > a_2 & \text{if } \mathfrak{A}[\![a_1]\!]s > \mathfrak{A}[\![a_2]\!]s \\
(s,h) \models \neg A & \text{if not } (s,h) \models A \\
(s,h) \models A_1 \wedge A_2 & \text{if } (s,h) \models A_1 \text{ and } (s,h) \models A_2 \\
(s,h) \models A_1 \vee A_2 & \text{if } (s,h) \models A_1 \text{ or } (s,h) \models A_2
\end{array}
$$

Furthermore we let $\Sigma(A) := \{\sigma \in \Sigma \mid \sigma \models A\}$, and A is called valid (notation: $\models A$) if $\Sigma(A) = \Sigma$.

Recursive Predicate Definitions

Definition (SL predicates)

- A (recursive) predicate definition is an equation of the form

$$P(x_1, \ldots, x_n) := A$$

  where P is a predicate symbol, $n \in \mathbb{N}$, $x_1, \ldots, x_n \in Var$ and $A \in SLA$ an SL assertion, additionally containing recursive predicate calls of the form $P(a_1, \ldots, a_n)$ with $a_1, \ldots, a_n \in AExp$. Syntactic restriction (to ensure monotonicity): each call must be in the scope of an even number of negations.
- It induces the functional $\Phi : (\mathbb{Z}^n \to 2^\Sigma) \to (\mathbb{Z}^n \to 2^\Sigma)$, given by

$$\Phi(p)(z_1, \ldots, z_n) := \Sigma(A[P \mapsto p, x_1 \mapsto z_1, \ldots, x_n \mapsto z_n]).$$

- A state $(s, h) \in \Sigma$ satisfies a predicate call $P(a_1, \ldots, a_n)$ (notation: $(s, h) \models P(a_1, \ldots, a_n)$) if $(s, h) \in \text{fix}(\Phi)(\mathfrak{A}[\![a_1]\!]s, \ldots, \mathfrak{A}[\![a_1]\!]s)$.

Recap: Separation Logic

Partial Correctness Properties in Separation Logic

Here we take a fault-avoiding interpretation of Hoare triples:

- programs must be memory-safe, i.e., never reach a fault
- all successfully terminating programs must satisfy the postcondition

Definition (Partial correctness properties)

- For $A, B \in SLA$ and $c \in Cmd$, $\{A\}\, c\, \{B\}$ is called a partial correctness property (PCP) with precondition A and postcondition B.
- A state $(s, h) \in \Sigma$ satisfies PCP $\{A\}\, c\, \{B\}$ (notation: $(s, h) \models \{A\}\, c\, \{B\}$) if, whenever $(s, h) \models A$,
  1. $\langle c, (s, h) \rangle \not\to \lightning$ and
  2. if $\langle c, (s, h) \rangle \to (s', h')$, then $(s', h') \models B$.
- PCP $\{A\}\, c\, \{B\}$ is called valid (notation: $\models \{A\}\, c\, \{B\}$) if $(s, h) \models \{A\}\, c\, \{B\}$ for every $(s, h) \in \Sigma$.

The Proof System

Definition (SL proof rules)

$$\text{(alloc)}\ \frac{x \notin FV(\vec{a})}{\{\mathrm{emp}\}\ x := \mathtt{alloc}(\vec{a})\ \{x \mapsto \vec{a}\}} \qquad \text{(lookup)}\ \frac{x \notin FV(a)}{\{a \mapsto v\}\ x := [a]\ \{a \mapsto v \wedge x = v\}}$$

$$\text{(free)}\ \frac{}{\{a \mapsto -\}\ \mathtt{free}(a)\ \{\mathrm{emp}\}} \qquad \text{(mutate)}\ \frac{}{\{a \mapsto -]\}\ [a] := a'\ \{a \mapsto a'\}}$$

$$\text{(asgn)}\ \frac{x \notin FV(a)}{\{\mathrm{emp}\}\ x{:=}a\ \{\mathrm{emp} \wedge x = a\}} \qquad \text{(frame)}\ \frac{\{A\}\ c\ \{B\} \quad Var(C) \cap Mod(c) = \emptyset}{\{A * C\}\ c\ \{B * C\}}$$

$$\text{(skip)}\ \frac{}{\{A\}\ \mathtt{skip}\ \{A\}} \qquad \text{(seq)}\ \frac{\{A\}\ c_1\ \{C\} \quad \{C\}\ c_2\ \{B\}}{\{A\}\ c_1 ; c_2\ \{B\}}$$

$$\text{(if)}\ \frac{\{A \wedge b\}\ c_1\ \{B\} \quad \{A \wedge \neg b\}\ c_2\ \{B\}}{\{A\}\ \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end}\ \{B\}} \qquad \text{(while)}\ \frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \mathtt{while}\ b\ \mathtt{do}\ c\ \mathtt{end}\ \{A \wedge \neg b\}}$$

$$\text{(cons)}\ \frac{\models (A \Rightarrow A') \quad \{A'\}\ c\ \{B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{B\}}$$

List Reversal Example I

Example 19.1 (List reversal: $\mathsf{sll}(x, y) := (x = y \wedge \mathsf{emp}) \vee \exists x' : x \mapsto x' * \mathsf{sll}(x', y)$)

$\{\texttt{head} \neq 0 \wedge \mathsf{sll}(\texttt{head}, 0)\}$ Precondition

$\Rightarrow \{\texttt{head} \neq 0 \wedge \exists x' : \texttt{head} \mapsto x' * \mathsf{sll}(x', 0)\}$ (unfold sll)

```
cur := [head];
```

$\{\texttt{head} \neq 0 \wedge \exists x' : \texttt{head} \mapsto x' * \mathsf{sll}(x', 0) \wedge \texttt{cur} = x'\}$ (lookup)

```
rev := head;
```

$\{\texttt{head} \neq 0 \wedge \exists x' : \texttt{head} \mapsto x' * \mathsf{sll}(x', 0) \wedge \texttt{cur} = x' \wedge \texttt{rev} = \texttt{head}\}$ (asgn)

$\Rightarrow \{\mathsf{sll}(\texttt{cur}, 0) * \mathsf{sll}(\texttt{rev}, \texttt{head})\}$ Invariant

```
while ¬(cur = 0) do
  next := [cur];
  [cur] := rev;
  rev := cur;
  cur := next
end
```

$\{\mathsf{sll}(\texttt{rev}, \texttt{head})\}$ Postcondition

List Reversal Example II

Example 19.1 (List reversal: $\mathsf{sll}(x, y) := (x = y \land \mathsf{emp}) \lor \exists x' : x \mapsto x' * \mathsf{sll}(x', y)$)

$$
\begin{array}{ll}
\{\mathsf{sll}(\mathtt{cur}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head})\} & \text{Invariant} \\
\mathtt{while}\ \neg(\mathtt{cur}\ \mathtt{=}\ \mathtt{0})\ \mathtt{do} & \\
\quad \{\mathsf{sll}(\mathtt{cur}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head}) \land \mathtt{cur} \neq 0\} & \text{(while)} \\
\quad \Rightarrow \{\exists x' : \mathtt{cur} \mapsto x' * \mathsf{sll}(x', 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head})\} & \text{(unfold sll)} \\
\quad \mathtt{next\ :=\ [cur];} & \\
\quad \{\exists x' : \mathtt{cur} \mapsto x' * \mathsf{sll}(x', 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head}) \land \mathtt{next} = x'\} & \text{(lookup)} \\
\quad \mathtt{[cur]\ :=\ rev;} & \\
\quad \{\exists x' : \mathtt{cur} \mapsto \mathtt{rev} * \mathsf{sll}(x', 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head}) \land \mathtt{next} = x'\} & \text{(mutate)} \\
\quad \Rightarrow \{\mathsf{sll}(\mathtt{next}, 0) * \mathsf{sll}(\mathtt{cur}, \mathtt{head})\} & \text{(fold sll)} \\
\quad \mathtt{rev\ :=\ cur;} & \\
\quad \{\mathsf{sll}(\mathtt{next}, 0) * \mathsf{sll}(\mathtt{cur}, \mathtt{head}) \land \mathtt{rev} = \mathtt{cur}\} & \text{(asgn)} \\
\quad \Rightarrow \{\mathsf{sll}(\mathtt{next}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head})\} & \\
\quad \mathtt{cur\ :=\ next} & \\
\quad \{\mathsf{sll}(\mathtt{next}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head}) \land \mathtt{cur} = \mathtt{next}\} & \text{(asgn)} \\
\quad \Rightarrow \{\mathsf{sll}(\mathtt{cur}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head})\} & \\
\mathtt{end} & \\
\{\mathsf{sll}(\mathtt{cur}, 0) * \mathsf{sll}(\mathtt{rev}, \mathtt{head}) \land \mathtt{cur} = 0\} & \text{(while)} \\
\Rightarrow \{\mathsf{sll}(\mathtt{rev}, \mathtt{head})\} & \text{Postcondition}
\end{array}
$$

Soundness of Separation Logic

Theorem 19.2 (Soundness of Separation Logic)

For every partial correctness property $\{A\}\, c\, \{B\}$ *with* $A, B \in SLA$ *and* $c \in Cmd$,

$$\vdash \{A\}\, c\, \{B\} \quad \Rightarrow \quad \models \{A\}\, c\, \{B\}.$$

Proof.

We only consider the frame rule and use the auxiliary lemmas on the following slide. $\square$

Soundness of Frame Rule

The following operational facts about programs imply their locality:

Lemma 19.3 (Monotonicity of memory safety)

If $\langle c, (s, h_1)\rangle \not\to \lightning$ *and* $h_1 \# h_2$, *then* $\langle c, (s, h_1 \uplus h_2)\rangle \not\to \lightning$.

Proof.

Straightforward (increasing the heap cannot introduce memory faults, which are only caused by accessing unallocated memory addresses) $\square$

Lemma 19.4 (Frame property)

Suppose $\langle c, (s, h_1)\rangle \not\to \lightning$ *and* $\langle c, (s, h_1 \uplus h_2)\rangle \to (s', h')$.
Then there exists $h'_1 \in Stack$ *such that* $\langle c, (s, h_1)\rangle \to (s', h'_1)$ *and* $h' = h'_1 \uplus h_2$.

Proof.

on the board $\square$

More Topics in Separation Logic

- Completeness of proof system
- Symbolic heaps, symbolic execution and abstract interpretation [P.W. O'Hearn: *A Primer on Separation Logic (and Automatic Program Verification and Analysis)*]
- Frame inference [A. Gotsman, J. Berdine, B. Cook: *Interprocedural Shape Analysis with Separated Heap Abstractions*]
  - Problem: given $A, B \in SLA$, find $C \in SLA$ such that $\models A \Rightarrow B * C$
  - Application: A assertion before procedure call, B precondition of procedure specification
  - Goal: find "leftover" heap C needed for frame rule applied to procedure call
- Concurrent Separation Logic and permissions [V. Vafeiadis: *Concurrent Separation Logic and Operational Semantics*]
- ...

Operational Semantics of Functional Programming Languages

- Program = list of function definitions
- Simplest setting: first-order function definitions of the form

$$f(x_1, \dots, x_n) = t$$

  - function name f
  - formal parameters $x_1, \dots, x_n$
  - term t over (base and defined) function calls and $x_1, \dots, x_n$
- Operational semantics (only function calls; for terms t_i, numbers z_j and variables x_k)
  - call-by-value case:

$$\frac{t_1 \to z_1 \quad \dots \quad t_n \to z_n \quad t[x_1 \mapsto z_1, \dots, x_n \mapsto z_n] \to z}{f(t_1, \dots, t_n) \to z}$$

  - call-by-name case:

$$\frac{t[x_1 \mapsto t_1, \dots, x_n \mapsto t_n] \to z}{f(t_1, \dots, t_n) \to z}$$

Outlook: Semantics of Functional Programming Languages

Denotational Semantics of Functional Programming Languages

- Denotational semantics
  - program = equation system (for functions)
  - induces call-by-value and call-by-name functional
  - monotonic and continuous w.r.t. graph inclusion
  - semantics := least fixpoint (Tarski/Knaster Theorem)
  - coincides with operational semantics
- Extensions: higher-order types, data types, ...
- see [Winskel 1996, Sct. 9] and *Functional Programming* course [Giesl]

Outlook: Semantics of Logic Programming Languages

Syntax of Logic Programming Languages

- Program = list of predicate definitions
- Predicate definition = sequence of clauses of the form $q_0 \colon\!- q_1, \ldots, q_n$ with atoms q_i
- Atom = predicate call $p(t_1, \ldots, t_k)$ with predicate p and terms t_j over variables, constants and function symbols

Example 19.1

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Outlook: Semantics of Logic Programming Languages

Operational Semantics of Logic Programming Languages

- Defined by (SLD) resolution
- Starts with single goal, called query
- Iteratively apply clauses with matching heads
- Involves backtracking if several clause heads match

Example 19.2

```
father(tom,sally).
father(tom,erica).
father(mike,tom).
mother(anna,sally).
sibling(X,Y) :- parent(Z,X), parent(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Refutation proof:

```
sibling(sally,erica).
⇐ parent(Z,sally), parent(Z,erica).
⇐ mother(Z,sally), parent(Z,erica).
⇐ parent(anna,erica).
⇐ mother(anna,erica). ↯
⇐ father(anna,erica). ↯
⇐ father(Z,sally), parent(Z,erica).
⇐ parent(tom,erica).
⇐ mother(tom,erica). ↯
⇐ father(tom,erica).
⇐ □
```

Denotational Semantics of Logic Programming Languages

- meaning of program = $\{$fully instantiated valid atoms$\}$
- fixpoint iteration:
  - start with empty set
  - 1st step: all instantiations of facts (i.e., clauses with empty RHS)
  - $i+1$st step: all instantiations of facts that can be derived from known facts
- monotonic and continuous w.r.t. set inclusion
- semantics := least fixpoint (Tarski/Knaster Theorem)
- coincides with operational semantics

Example 19.3

```
father(tom, sally).
father(tom, erica).
father(mike, tom).
mother(anna, sally).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
```

Fixpoint iteration:

$A_0 = \emptyset$

$A_1 = \{\mathtt{f}(\mathtt{t},\mathtt{s}), \mathtt{f}(\mathtt{t},\mathtt{e}), \mathtt{f}(\mathtt{m},\mathtt{t}), \mathtt{m}(\mathtt{a},\mathtt{s})\}$

$A_2 = A_1 \cup \{\mathtt{p}(\mathtt{t},\mathtt{s}), \mathtt{p}(\mathtt{t},\mathtt{e}), \mathtt{p}(\mathtt{m},\mathtt{t}), \mathtt{p}(\mathtt{a},\mathtt{s})\}$

$A_3 = A_2 \cup \{\mathtt{s}(\mathtt{s},\mathtt{e}), \mathtt{s}(\mathtt{e},\mathtt{s})\}$

$A_4 = A_3$

Summary I

Semantics of programming languages

- Models the computational meaning of each program
- Provides abstract entities that represent just the relevant features of all possible executions
  - relationship between input and output
  - whether execution terminates or not
- Ignores details that have no relevance to the correctness of implementations
- Equivalences identify programs with identical semantics

(Structural) operational semantics

- Uses syntax-guided rules to specify transition relations (for evaluation/execution)
- Represents scoping of identifiers by splitting of data state into environment and store part
- Big-step and small-step semantics (latter required for modelling interaction between concurrent activities)

Summary II

Denotational semantics

- Denotations are defined inductively on program structure
- Recursion (loops, procedures) handled as least fixpoints of continuous functions on CCPOs

Axiomatic semantics

- A Hoare Logic gives rules for the relation between assertions about values of variables before and after execution of each construct
- Employs logical variables for remembering (initial) values of program variables
- Partial vs. total correctness
- Recursion handled by invariants

Separation Logic

- Extension of Hoare Logic to deal with pointer programs
- Crucial: frame rule to support scalable modular reasoning

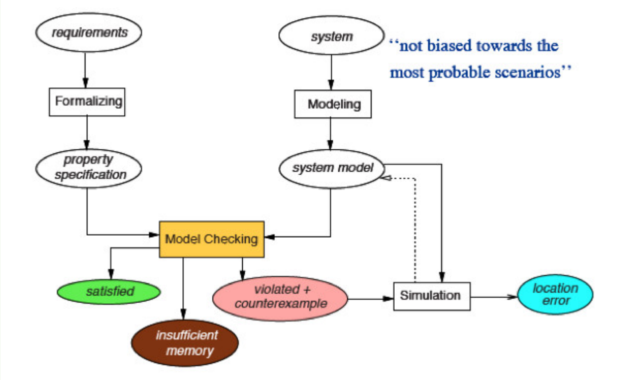
Miscellaneous

Oral Exam

- Appointment via Foodle poll at `https://terminplaner4.dfn.de/mW5s1bLwrApFk2Qs`
- Contents:
  + foundational concepts and connections between those
  + proof ideas
  – concrete (elaborated) examples
  – details of proofs
  – willing to omit correctness properties for execution time
- Typical (non-)questions:
  + operational/denotational/axiomatic semantics of `while`statement
  + alternative evaluation strategies for boolean expressions strict/sequential/parallel
  + algebraic foundations (partial orders, CCPOs, monotonicity, continuity, fixpoint theorem, ...)
  + idea of coincidence proof for operational/denotational semantics
  + (mild) extensions of programming language (side effects, ...)
  – Construct the derivation tree for $c := \ldots$ (many lines) and $\sigma := \ldots$
  – Prove the following claim: ...

Master-Level Teaching in Winter 2019/20

Course *Model Checking* [Katoen]



Course *Concurrency Theory* [Katoen/Noll]

1. Interleaving semantics (process algebras)
2. Property specifications (Hennessy-Milner Logic)
3. Process equivalence (traces, bisimulation)
4. True concurrency (Petri nets)

Seminar *[Advanced Topics in] Formal Semantics of Programming Languages* [Katoen, Noll, NN]

- Nondeterminism
- Concurrency
- Recursion
- Relaxed memory models
- Security and privacy

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY