Goal of the course

Study Separation Logic having automatic verification in mind

Learn how some notions of mathematical logic can be very helpful in reasoning about real world programs
A piece of a Windows device driver.

Is this correct?

Or at least: does it have basic properties like it won't crash or leak memory?
Today’s plan

- Motivation for Separation Logic
- Assertion language
- Mathematical model
- Data structures
Motivations...
Simple Imperative Language

Safe commands:

- \( S ::= \text{skip} | x := E | x := \text{new}(E_1, \ldots, E_n) \)

Heap accessing commands:

- \( A(E) ::= \text{dispose}(E) | x := [E] | [E] := F \)

where \( E \) is an expression \( x, y, \text{nil}, \text{etc.} \)

Command:

- \( C ::= S | A | C_1 ; C_2 | \text{if} \ B \{ \ C_1 \} \ \text{else} \{ \ C_2 \} | \text{while} \ B \ \text{do} \{ \ C \} \)

where \( B \) is a boolean guard \( E = E, E! = E, \text{etc.} \)
Example Program:
List Reversal

Some properties we would like to prove:

Does the program preserve acyclicity/cyclicity?
Does it core-dump?
Does it create garbage?

p:=nil;
while (c !=nil) do {
  t:=p;
p:=c;c:=[c];[p]:=t;
}

1 2 3 nil
3 2 1 nil
We are interested in pointer manipulating programs.
Why Separation Logic?

Consider this code:

Assume([x] = 3) && x!=y && x!=z) Add assertion?
Assume(y != z) Add assertion?
[y] = 4;
[z] = 5;
Guarantee([y] != [z])
Guarantee([x] = 3)

We need to know that things are different. How?
We need to know that things stay the same. How?
Framing

We want a general concept of things not being affected.

\[ \{ P \} \ C \ \{ Q \} \]
\[ \{ R \ \&\& \ P \} \ C \ \{ Q \ \&\& \ R \} \]

What are the conditions on C and R?
Hard to define if reasoning about a heap and aliasing

This is where separation logic comes in

\[ \{ P \} \ C \ \{ Q \} \]
\[ \{ R \ \ast \ P \} \ C \ \{ Q \ \ast \ R \} \]

Introduces new connective \( \ast \) used to separate state.
The Logic
### Storage Model

- **Vars** $\overset{\text{def}}{=} \{x, y, z, \ldots\}$
- **Locs** $\overset{\text{def}}{=} \{1, 2, 3, 4, \ldots\}$
- **Vals** $\supseteq$ Locs

- **Heaps** $\overset{\text{def}}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$
- **Stacks** $\overset{\text{def}}{=} \text{Vars} \rightarrow \text{Vals}$
- **States** $\overset{\text{def}}{=} \text{Stacks} \times \text{Heaps}$

#### Stack
- $x \rightarrow \text{7}$
- $y \rightarrow \text{42}$

#### Heap
- $7 \rightarrow \text{0}$
- $9 \rightarrow \text{11}$
- $42 \rightarrow \text{9}$
Mathematical Structure of Heap

\[
\text{Heaps} \overset{\text{def}}{=} \text{Locs} \to_{\text{fin}} \text{Vals}
\]

\[
h_1 \# h_2 \overset{\text{def}}{\iff} \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset
\]

\[
h_1 * h_2 \overset{\text{def}}{=} \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise} \end{cases}
\]

1) * has a unit
2) * is associative and commutative
3) \((\text{Heap},*,\{\{}\})\) is a partial commutative monoid
Assertions

Formulas in Lecture 1

Hongseok Yang
Queen Mary, University of London

1. Formulas

\[ \text{Vars} \quad \text{def} = \{ x, y, z, \ldots \} \]

\[ \text{Locs} \quad \text{def} = \{ 1, 2, 3, 4, \ldots \} \]

\[ \text{Vals} \supseteq \text{Locs} \]

\[ \text{Heaps} \quad \text{def} = \text{Locs} \rightarrow \text{fin Integers} \]

\[ \text{Stacks} \quad \text{def} = \text{Vars} \rightarrow \text{Vals} \]

\[ \text{States} \quad \text{def} = \text{Stacks} \times \text{Heaps} \]

\[ h_1 \cup h_2 \quad \text{def} = \begin{cases} h_1 \cup h_2 & \text{if } h_1 \cap h_2 = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases} \]

Heap-independent Exprs

\[ E, F ::= x | n | E + F | -E | \ldots \]

Atomic Predicates

\[ P, Q ::= E = F | E \geq F | E \leftrightarrow F | P \land Q | \neg P | \forall x. P \]

Heap-independent Exprs

Atomic Predicates

Separating Connectives

Classical Logic

Informal Meaning
Examples

Formula: \( \text{emp}^* x \rightarrow y \; * \; y \rightarrow z \; * \; z \rightarrow x \)
Expressions mean maps from stacks to integers.

$$\left[ E \right] : \text{Stacks} \rightarrow \text{Vals}$$

Semantics of assertions given by satisfaction relation between states and assertions.

$$(s, h) \models P$$
Semantics of Assertions

\[ (s, h) \models E \geq F \quad \text{iff} \quad [E]s, [F]s \in \text{Integers and } [E]s \geq [F]s \]

\[ (s, h) \models E \leftrightarrow F \quad \text{iff} \quad \text{dom}(h) = \{[E]s\} \text{ and } h([E]s) = [F]s \]

\[ (s, h) \models \text{emp} \quad \text{iff} \quad h = \emptyset \text{ (i.e., } \text{dom}(h) = \emptyset) \]

\[ (s, h) \models P \land Q \quad \text{iff} \quad \exists h_0 h_1. \ h_0 \ast h_1 = h, \ (s, h_0) \models P \text{ and } (s, h_1) \models Q \]

\[ (s, h) \models \text{true} \quad \text{always} \]

\[ (s, h) \models P \land Q \quad \text{iff} \quad (s, h) \models P \text{ and } (s, h) \models Q \]

\[ (s, h) \models \neg P \quad \text{iff} \quad \text{not } ((s, h) \models P) \]

\[ (s, h) \models \forall x. P \quad \text{iff} \quad \forall v \in \text{Vals. } (s[x \mapsto v], h) \models P) \]
The address $E$ is active:

$$ E \leftarrow - \triangleq \exists x', E \leftarrow x' $$

where $x'$ not free in $E$

$E$ points to $F$ somewhere in the heap:

$$ E \leftarrow F \triangleq E \leftarrow F * \text{true} $$

$E$ points to a record of several fields:

$$ E \leftarrow E_1, \ldots, E_n \triangleq E \leftarrow E_1 * \cdots * E + n - 1 \leftarrow E_n $$
Example

\[
\begin{align*}
x & \mapsto 3, y \\
y & \mapsto 3, x \\
x & \mapsto 3, y \times y \mapsto 3, x \\
x & \mapsto 3, y \land y \mapsto 3, x \\
x & \mapsto 3, y \land y \mapsto 3, x
\end{align*}
\]
An inconsistency

What’s wrong with the following formula?

\[ 10 \rightarrow 3 \times 10 \rightarrow 3 \]

Try to be in two places at the same time
Small details

- $E=F$ is completely heap independent.

$(E=F) \times P$ where is it true?

In a state where $E=F$ hold in the store and $P$ holds for the same store and a heap contained in the current one.

Example:

\[
\begin{align*}
  x &= y \quad z |-> 0 \\
  s(x) &= s(y) \\
  s(z) &= 10 \\
  \text{dom}(h1) &= \{10, 15\} \\
  h1(10) &= 0 \\
  h1(15) &= 37 \\
  \text{dom}(h2) &= \{10, 42, 73\} \\
  h2(10) &= 0 \\
  h2(42) &= 11 \\
  h2(73) &= 0
\end{align*}
\]
...but

- $E = F$ is completely heap independent.

$$(E = F) \land P \quad \text{where is it true?}$$

In a state where $E = F$ hold in the store and $P$ holds for the same store and exactly the current heap.

In other words: $P$ determines the heap

Example: $x = y \land z \mapsto 0$

holds in any state $(s, h)$ such that $s(x) = s(y)$

$\text{dom}(h) = \{ s(z) \}$ \quad $h(s(z)) = 0$

so many stores but the shape of the heap is fixed
Exercise

what is $h$ such that $s, h \models p$  

\[
\begin{align*}
  x \mapsto 1 & \quad h = h_1 \\
  y \mapsto 2 & \quad h = h_2 \\
  x \mapsto 1 \land y \mapsto 2 & \quad h = h_1 \ast h_2 \\
  x \mapsto 1 \land \text{true} & \quad h_1 \text{ contained in } h \\
  x \mapsto 1 \land y \mapsto 2 \land (x \mapsto 1 \lor y \mapsto 2) & \quad \text{Homework!}
\end{align*}
\]

$h_1 = \{(s(x), 1)\}$  

$h_2 = \{(s(y), 2)\}$  

with $s(x) \neq s(y)$
Validity

- P is valid if, for all s,h, s,h|=P

Examples:

- E|→3 ⇒ E>0  Valid!
- E|→– * E|→–  Invalid!
- E|→– * F|→–  ⇒ E != F  Valid!
- E|→3 ∧ F|→3  ⇒ E=F  Valid!
- E|→3 * F|→3  ⇒ E|→3 ∧ F|→3  Invalid!
Some Laws and inference rules

\[ p_1 * p_2 \iff p_2 * p_1 \]

\[ (p_1 * p_2) * p_3 \iff p_1 * (p_2 * p_3) \]

\[ p * \text{emp} \iff p \]

\[ (p_1 \lor p_2) * q \iff (p_1 * q) \lor (p_2 * q) \]

\[ (\exists x. p_1) * p_2 \iff \exists x. (p_1 * p_2) \quad \text{when } x \text{ not in } p_2 \]

\[ (\forall x. p_1) * p_2 \iff \forall x. (p_1 * p_2) \quad \text{when } x \text{ not in } p_2 \]

\[ p_1 \quad \implies \quad p_2 \quad \implies \quad q_1 \quad \implies \quad q_2 \quad \text{Monotonicity} \]

\[ p_1 * q_1 \implies p_2 * q_2 \]
Substructural logic

Separation logic is a substructural logic:

No Contraction \( A \not\vdash A \ast A \)

No Weakening \( A \ast B \not\vdash A \)

Examples:

\[ 10 \vdash 3 \not\vdash 10 \vdash 3 \ast 10 \vdash 3 \]

\[ 10 \vdash 3 \ast 42 \vdash 7 \not\vdash 42 \vdash 7 \]
Lists

A non circular list can be defined with the following inductive predicate:

\[
\begin{align*}
\text{list } \mathsf{} i & = \text{emp } \land \ i=\mathsf{nil} \\
\text{list } (s::S) \ i & = \exists j. \ i \Rightarrow s,j \times \text{list } S \ j
\end{align*}
\]
List segment

Possibly empty list segment

\[ \text{lseg}(x, y) = (\text{emp} \land x = y) \lor \exists j. x \mapsto j \ast \text{lseg}(j, y) \]

Non-empty non-circular list segment

\[ \text{lseg}(x, y) = x \neq y \land ((x \mapsto y) \lor \exists j. x \mapsto j \ast \text{lseg}(j, y)) \]
Trees

A tree can be defined with this inductive definition:

\[
\text{tree} \; [\;] \; i = \text{emp} \land i = \text{nil}
\]

\[
\text{tree} \; (t_1, a, t_2) \; i = \exists j, k. \; i \rightarrow j, a, k \cdot (\text{tree} \; t_1 \; j) \cdot (\text{tree} \; t_2 \; k)
\]
References