Lab 9
Hoare Triples and all that

Objective

This is an exercise session. The goals are:
- Manipulate separation logic rules for proving heap properties.
- Proving programs with lists.

9.1 Separation Logic: heap formulas

In this part, we study properties of the formulas of Separation Logic, by focusing on the specific operators: emp, $\mapsto$ and $*$.

Heap formula, semantics Here are the inferences rules for this logic (s stack, h heap):

- $(\sigma, h) \models a \geq 0$ iff $[\![a]\!]_\sigma = k \geq 0$.
- $(\sigma, h) \models \neg A$ iff not $(\sigma, h) \models A$.
- $(\sigma, h) \models A \wedge B$ iff $(\sigma, h) \models A$ and $(\sigma, h) \models B$.
- $(\sigma, h) \models \exists x, A(x)$ iff there exists $x \in \mathbb{N}$ such that $(\sigma, h) \models A(x)$
- $(\sigma, h) \models emp$ iff $dom(h) = \emptyset$.
- $(\sigma, h) \models a_1 \mapsto a_2$ iff $dom(h) = \{i\}$, $h(i) = k$ where $[\![a_1]\!]_\sigma = i$ and $[\![a_2]\!]_\sigma = k$.
- $(\sigma, h) \models A_1 * A_2$ iff $h = h_1 \uplus h_2$ and $(\sigma, h_i) \models A_i$.

EXERCISE #1 ▶ **Cells in heaps**
With the help of the recalled semantics:
1. Define an assertion, called 1, that does not use existential quantification ($\exists$), and that is satisfied by all and only memory states in which the domain of the heap is of size 1.
2. $E \hookrightarrow F$ is defined as $E \mapsto F * \texttt{true}$. Define, conversely, $\mapsto$ in terms of $\hookrightarrow$.

EXERCISE #2 ▶ **Two forms of conjunction**
Separating conjunction really is not like conjunction: give an assertion A and a memory state (σ, h) such that $(\sigma, h) \models A * \neg A$.

EXERCISE #3 ▶ **Relating assertions**
We write $\{\{A\}\}$ for the *extension* of assertion A, defined as follows:

$$\{\{A\}\} \;=\; \{(\sigma, h) \mid (\sigma, h) \models A\}$$

We write $A \vdash B$ if $\{\{A\}\} \subseteq \{\{B\}\}$, and $A \dashv\vdash B$ whenever $A \vdash B$ and $B \vdash A$.

Discuss the implications or equivalences between formulas given below. In each case, you can just provide an informal argument to justify an implication between formulas. If the implication does not hold, draw a memory state that proves it.
- $a \hookrightarrow a' \vdash a \mapsto a'$ (we recall that $A \hookrightarrow B = A \mapsto B * \texttt{true}$)
- $A \wedge B \dashv\vdash A * B$
- $A * A \dashv\vdash A$
- $X \mapsto 12 * X \mapsto 12 \vdash ??$ (try to guess what should replace the "??")
- $(A \wedge B) * C \dashv\vdash A \wedge (B * C)$ when A is *pure*, which means that A does not contain $emp, \mapsto$ or $*$.

EXERCISE #4 ▶ **An implication between heap formulas**
What could it mean for a memory state to satisfy assertion $(\exists i.\, X \hookrightarrow i) \Rightarrow (\exists j.\, Y \hookrightarrow j)$?

9.2 Proving a program

9.2.1 Rules

Small-step semantics Here is the small step semantics for new constructs:

- Lookup: $(\sigma, h), x := [a] \Downarrow \begin{cases} (\sigma[k/x], h) \text{ if } [\![a]\!]_\sigma = i, i \in dom(h), h(i) = k \\ \underline{error} \text{ if } [\![a]\!]_\sigma = i, i \notin dom(h). \end{cases}$
- Mutation: $(\sigma, h), [a_1] := a_2 \Downarrow \begin{cases} (\sigma, h[k/i]) \text{ if } [\![a_1]\!]_\sigma = i, i \in dom(h), [\![a_2]\!]_\sigma = k \\ \underline{error} \text{ if } [\![a]\!]_\sigma = i, i \notin dom(h). \end{cases}$
- Allocation: $(\sigma, h), x := cons(a_1, \dots a_n) \Downarrow (\sigma[i/x], h[k_1/i, \dots k_n/(i+n-1)])$, with $i, \dots i+n-1$ new addresses and $[\![a_j]\!]_\sigma = k_j$.
- Desallocation: $(\sigma, h), free(a) \Downarrow \begin{cases} (\sigma, h \setminus i) \text{ if } [\![a]\!]_\sigma = i, i \in dom(h) \\ \underline{error} \text{ if } [\![a]\!]_\sigma = i, i \notin dom(h). \end{cases}$

Hoare-like rules now mean:

$$\{A\}p\{B\} \text{ iff for all } (\sigma, h), \text{ if } (\sigma, h) \models A, \text{ then } (s, h) \not\Downarrow \underline{error} \text{ and } (\sigma, h), p \Downarrow (\sigma', h') \Rightarrow (\sigma', h') \models B$$

Hoare triples We recall the Hoare-like axioms for separation logic:

- Lookup: $\{a \mapsto i \wedge X = j\}X := [a]\{X = i \wedge a[j/X] \mapsto i\}$.
  If X is not in $vars(a)$, this rule becomes $\{a \mapsto i\}X := [a]\{X = i \wedge a \mapsto i\}$.
- Mutation: $\{\exists i, a_1 \mapsto i\}[a_1] := a_2\{a_1 \mapsto a_2\}$.
- Allocation: $\{X = i \wedge emp\}X := cons(a_1, \dots a_n)\{X \mapsto a_1[i/X] * X+1 \mapsto a_2[i/X] * \dots * X+n-1 \mapsto a_n[i/X]\}$
- Desallocation: $\{a \mapsto -\}free(a)\{emp\}$

and we recall the frame rule:

$$\frac{\{A\}p\{B\}}{\{A * C\}p\{B * C\}},$$

if p doesn't modify the variables of C.

9.2.2 Proving programs with recursive data structures

EXERCISE #5 ▶ **The exercise seen at the lecture - List destruction**
We revisit the code you saw (very quickly) at the end of last lecture. Let p be the program:

```
while X != nil do
    Y := [X+1];
    free(X);
    free(X+1);
    X := Y
```

We recall the definition of list $(\cdot)$:

$$\mathsf{list}(i) \quad = \quad (i = \underline{\mathsf{nil}} \wedge emp) \vee (\exists j, k.\, (i \mapsto k, j) * \mathsf{list}(j))$$

1. Suppose that the current memory state satisfies $\mathsf{list}(i)$, i.e., $(\sigma, h) \models \mathsf{list}(i)$. Can the heap contain a *cyclic* list starting at i? Explain why.
2. Prove the Hoare triple $\{\mathsf{list}(X)\}\, p\, \{emp\}$.

   You can use the following (simpler) small axiom for lookup in the case where the arithmetical expression does not depend on the modified variable:

   $$\{a \mapsto i\}\, X := [a]\, \{X = i \wedge a \mapsto i\} \qquad \text{if } X \text{ does not appear in } a$$

EXERCISE #6 ▶ **Trees and DAGs (Directed Acyclic Graphs)**
Draw inspiration from the equations you have seen for lists at the lecture to:

1. Write equations for binary trees (leafs are just leafs, nodes contain an integer value and two subtrees).
2. Write equations for DAGs (like the trees you just specified, but allowing sharing of subtrees).

EXERCISE #7 ▶ **List reversal**
Prove that the following code performs list reversal: state the Hoare triple it satisfies, and describe the proof in Separation Logic.

```
J := nil;
while I != nil do
   K := [I + 1];
   [I + 1] := J;
   J := I;
   I := K
```

You will have to use the following predicate:

$$\begin{array}{rcl} \mathsf{lista}(\epsilon, i) & = & (emp \wedge i = \underline{\mathsf{nil}}) \\ \mathsf{lista}(a\alpha, i) & = & (i \mapsto a, j * \mathsf{lista}(\alpha, j)) \end{array}$$

where α denotes a sequence of values. Use α^R to denote its reversal.