Hoare triples

- \{A\} p \{B\} \quad a \text{ Hoare triple}
- partial correctness:
  if the initial state satisfies assertion \(A\), and if the execution of
  program \(p\) terminates, then the final state satisfies assertion \(B\)
- inference rules:

  \[
  \{A\} p \{A\} \\
  \{A\} \text{skip} \{A\} \\
  \{A\} \text{if} p \{A\} \\
  \{A\} \text{while} \ a \geq 0 \ do\ p \{A\} \\
  \{A\} \text{do}\ \sigma \{A\} \\
  \{A\} \sigma \{A\} \\
  \{A\} \text{end}\sigma \{A\} \\
  \{A\} \{\text{end}\} \\
  \{A\} \{\text{return}\} \\
  \{A\} \{\text{goto}\} \\
  \{A\} \{\text{call}\} \\
  \{A\} \{\text{jump}\} \\
  \{A\} \{\text{error}\}
  \]

- expressive properties:
  functional correctness rather than absence of runtime errors

Hoare logic — main ingredients

1. Floyd-Hoare Logic
   - a language for programs \(\text{IMP}\)
2. a language for assertions \(A\)
3. inference rules

Correct rules and completeness

- the 6 rules of Hoare logic are not the only correct rules
- for instance, the rule of constancy is correct too

\[
\{A\} p \{B\} \\
\{A \land \sigma \} p \{B \land \sigma\}
\]

important aspects:
- invariants in loops
- logical deduction rule
- backward reasoning (in the rule for assignment)

Synonyms

assertion formula \(A\)

environment store \(\sigma\)
correctness soundness for a rule

\[
\{A\} X := a \{A\}
\]

(consider \(X := X + 3\) to convince yourself)

Floyd’s forward axiom

\[
\{A\} X := a \{\exists i. (X = a[i/X] \land A[i/X])\}
\]

see also

\[
\{A \land X = i\} X := a \{A[i/X] \land X = a[i/X]\}
\]

\(i\): “ghost variable” (should probably be written \(I\))
Extending Imp

- Structure of memory at runtime
  - In (traditional) Hoare-Floyd logic, programs manipulate variables
    - The environment just records the (integer) value of each variable that is all we know about the memory
  - Dynamically allocated memory: add a heap component

Extending assertions: introducing heap formulas

A memory state is $(\sigma, h)$ where
  - $\sigma$ is a store
  - $h$ is a heap

Hoare logic assertions state properties about the environment

$$X \geq Y + Z + Q \land T > 0$$

Add formulas to reason about the heap

- NB: $X \mapsto 52$ usually makes more sense than $32 \mapsto 52$ (both are assertions)

Separation logic: sum up

- Inference rules
  - Those of Hoare logic control
  - Those for the new programming constructs
- Important things:
  - Invariants in while loops, backward rule for assignment, consequence rule
  - (Tight) small axioms, footprint, frame rule
- Metatheoretical properties
  - Correctness
  - Completeness

Programs manipulating pointers

- Hoare logic deals essentially with control
  - If $a \geq 0$ then $p_1$ else $p_2$ $p_1 ; p_2$ while $a \geq 0$ do $p$
- Move to a richer language:
  - Add (some kind of) pointers and handling of memory
    - Allocation
    - Modification (move pointers around)
  - Different kinds of properties
    - Typical runtime errors we want to detect: memory leaks, invalid disposal, invalid accesses
    - Typically, other approaches either assume memory safety, or forbid dynamic memory allocation
  - Describe what programs manipulating pointers do
  - Adopt the same methodological framework

Separation Logic is an enrichment of Floyd-Hoare logic

Hoare triples in Separation logic — interpretation

$$\{A\} \ p \ {B}\ 	ext{holds iff}$$

$$\forall \sigma, h, \text{ if } (\sigma, h) \models A, \text{ (} (\sigma, h) \text{ satisfies } A)$$

- $(\sigma, h), p \not\models \text{error}$, and
- If $(\sigma, h), p \models (\sigma', h')$, then $(\sigma', h') \models B$

Like in traditional Hoare logic, but:

- The state has a heap component
- Absence of forbidden access to the memory

The frame rule

- The rules of Hoare logic remain sound
- The rule of consistency becomes unsound

$$\{x \mapsto x\} [x] := 4 \{x \mapsto 4\}$$

What if $x = y$?

Separation logic is inherently modular

As opposed to whole program verification
Reasoning about lists

- A linked list in memory is something like
  \[(X_1 \mapsto k_1, X_2) \ast (X_2 \mapsto k_2, X_3) \ast \cdots \ast (X_n \mapsto k_n, \text{nil})\]

  \[(X \mapsto a, \text{emp})\] stands for \[X \mapsto a \ast (X + 1) \mapsto \text{emp}\]

- We describe the structure using assertions:
  - Add the possibility to write \((\text{recursive})\) equations


\[\text{list}(i) = (i = \text{nil} \land \text{emp}) \lor (\exists j. (i \mapsto k, j) \ast \text{list}(j))\]

- The formula above just specifies that we have a list in memory

  - We can rely on "mathematical lists" \((\{1, k::ks\})\) to provide a more informative definition


\[\begin{align*}
\text{list}([1, i]) &= \text{emp} \land i = \text{nil} \\
\text{list}(k::ks, i) &= \exists j. (i \mapsto k, j) \ast \text{list}(ks, j)
\end{align*}\]

On the weirdness of auxiliary variables

- In the lecture we saw the small axiom for lookup

\[\{a \mapsto i \land X = j\} X \equiv [a] (X = i \land a[j/X] \mapsto i)\]

- In the TD you saw its simpler form

\[\{a \mapsto i\} X \equiv [a] (X = i \land a[j/X] \mapsto i) \quad \text{if } X \text{ does not appear in } a\]

- How does one entail the other?

  - Rule of auxiliary variable elimination

\[\{u. A\} p \{B\} \quad \text{if } u \text{ does not appear in } p\]

  - If \(X\) does not appear in \(a\), \(a[j/X] = a\)

\[\begin{align*}
\{a \mapsto i \land X = j\} X \equiv [a] (X = i \land a \mapsto i) \\
\{a \mapsto i \land X = j\} X \equiv [\exists j. (X = i \land a \mapsto i)] & \iff X = i \land a \mapsto i
\end{align*}\]

Reasoning about concurrent programs

- Shared memory, several threads
- Permissions, locks, critical sections
- Ownership

Going further

Towards automation

- Hoare logic and separation logic are used naturally in an interactive manner

- If loop invariants are provided (as well as the global pre and post conditions), we can automatically chop the verification task into the proof of slices of the form

\[\{A\} p_1; p_2; \ldots; p_k \{B\},\]

where the \(p_i\)s are elementary commands.

- Construction of the Hoare triple boils down to being able to prove \textit{entailments} between assertions, \(A \vdash B\)

  - Cf. the Why3 tool (Filliâtrel et al.)

Towards automation of separation logic

- Restrict the set of possible formulas:
  - \textbf{Symbolic heaps}
  - \[
P = P_1 \land \cdots \land P_k \quad \text{pure formulas}
  \]
  - \[
  H = H_1 \land \cdots \land H_k \quad \text{simple heap formulas}
  \]
  - \(\text{for instance, } \rightarrow, \text{list.emp})\)

- Small axioms are adapted to symbolic heaps.
  - Yielding a "specialised operational semantics"

- Deciding entailments
  - For the pure part of symbolic heaps, standard approaches (theorem provers/automatic decision procedures)

  - For the heap components, exploiting implications like

\[\{\text{list}(i) \land i = \text{nil}\} \Rightarrow \text{emp}\]

\[\{i \mapsto k, j \land \text{list}(j)\} \Rightarrow \text{list}(i)\]

- More automation:\textbf{ discovering loop invariants}

  - Back to \textit{abstract} interpretation: abstract execution, generating a postcondition as we run through the loop

  - Sometimes abstracting (narrowing) to insure termination

  - E.g., replacing \(i \mapsto k, j \land j \mapsto (k', j') \mapsto \text{nil}\) with \(\text{list}(i)\)

  (Loosing information about the size of the list)
Modular analysis

- use the automated framework to analyse
  functions manipulating pointers
- compute Hoare triples for functions,
  without information about the rest of the code
- solve $A + \text{antiframe} \vdash B + \text{frame}$
  - antiframe: missing portion of heap
    because of function calls, the outer function body should have
    some parts of the heap in its precondition
  - frame: leftover portion of heap
    the postcondition of the outer function body specifies what
    parts of the heap are left unchanged