Hoare triples

Floyd-Hoare Logic, Separation Logic

1. Floyd-Hoare Logic *1969*

Reasoning about control

Hoare triples

- $\{A\}\, p\, \{B\}$ a 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

$$\frac{}{\{A[a/X]\}\, X := a\, \{A\}} \qquad \frac{}{\{A\}\, \texttt{skip}\, \{A\}} \qquad \frac{\{A_1\}\, p_1\, \{A_2\} \qquad \{A_2\}\, p_2\, \{A_3\}}{\{A_1\}\, p_1; p_2\, \{A_3\}}$$

$$\frac{\{A \wedge a \geq 0\}\, p_1\, \{B\} \qquad \{A \wedge \neg(a \geq 0)\}\, p_2\, \{B\}}{\{A\}\, \texttt{if}\ a \geq 0\ \texttt{then}\ p_1\ \texttt{else}\ p_2\, \{B\}}$$

$$\frac{\{A_I \wedge a \geq 0\}\, p\, \{A_I\}}{\{A_I\}\, \texttt{while}\ a \geq 0\ \texttt{do}\ p\ \{A_I \wedge \neg(a \geq 0)\}}$$

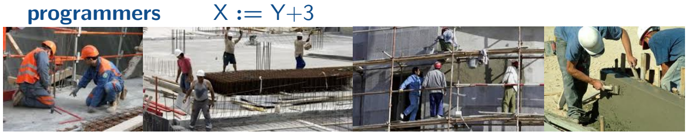
$$\frac{A_1 \Rightarrow A_2 \qquad \{A_2\}\, p\, \{B_2\} \qquad B_2 \Rightarrow B_1}{\{A_1\}\, p\, \{B_1\}}$$

- expressive properties

  functional correctness rather than absence of runtime errors

Hoare logic — main ingredients

programmers X := Y+3



(Hoare) logicians X $\geq$ Y+3

ingredients in Hoare logic:

1. a language for programs p IMP
2. a language for assertions A
3. inference rules

important aspects:

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

Hoare logic: metatheoretical properties

- **operational semantics and validity**
  - big step operational semantics for IMP: $\sigma, p \Downarrow \sigma'$
    - σ is an *environment*
    - $\sigma : \mathcal{V} \to \mathbb{Z}$ a map from variables to integers

      given some program p, σ is a partial mapping from a *finite set of variables* to $\mathbb{Z}$
  - the triple $\{A\}\, p\, \{B\}$ is **valid**:

    for all σ, if σ satisfies A and $\sigma, p \Downarrow \sigma'$, then σ' satisfies B
- **correctness** If the triple $\{A\}\, p\, \{B\}$ can be derived using the inference rules of Hoare logic, then it is valid.
  - NB: we could also rely on *denotational semantics*

    associate to each program p some function F_p from environments to environments
- **(relative) completeness** any valid triple can be constructed in Hoare logic, *provided* we can decide validity of the assertions *(i.e., decide whether A always holds)*
- logic rules capture the properties we want to express

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

$$\frac{\{A\}\, p\, \{B\}}{\{A \wedge C\}\, p\, \{B \wedge C\}} \quad \text{no variable in } C \text{ is modified by } p$$

- completeness: no new Hoare triple can be established if we add the rule of constancy
  - the 6 rules "tell everything"
  - using the rule of constancy makes proofs easier/more natural/more readable

  somehow, completeness is not only a theoretical question

The axiom for assignment

the axiom for assignment goes backwards

$$\frac{}{\{A[a/X]\}\, X := a\, \{A\}}$$

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

Floyd's **forward axiom**

$$\frac{}{\{A\}\, X := a\, \{\exists i.\, (X = a[i/X] \wedge A[i/X])\}}$$

see also $\frac{}{\{A \wedge X = i\}\, X := a\, \{A[i/X] \wedge X = a[i/X]\}}$

i: "ghost variable" *(should probably be written $\mathcal{I}$)*

Synonyms

assertion	formula	A
environment	store	σ
correctness	soundness	for a rule

2. Separation Logic $\sim$*2000*

Reasoning about memory

Programs manipulating pointers

- Hoare logic deals essentially with **control**

  `if` $a \geq 0$ `then` p_1 `else` p_2 $\quad p_1 ; p_2 \quad$ `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)
  - liberation/deallocation
- 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

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 the programming language

on the board

what does this program do?

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

on the board

Extending assertions: introducing *heap formulas*

- a memory state is (σ, h) where
  - σ is a store
  - h is a heap
- Hoare logic assertions state properties about the environment

  $X \geq Y * Z + Q \quad \wedge \quad T > 0$
- add formulas to reason about the heap
- NB: $X \mapsto 52$ usually makes more sense than $32 \mapsto 52$

  (both are assertions)

Hoare triples in Separation logic — interpretation

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

$\forall \sigma, h.$, if $(\sigma, h) \models A$, $\quad$ *((σ, h) satisfies A)*

then

- $(\sigma, h), p \not\Downarrow \underline{\texttt{error}}$, and
- if $(\sigma, h), p \Downarrow (\sigma', h')$, then $(\sigma', h') \models B$

like in traditional Hoare logic, but:

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

Small axioms

on the board

- axioms for heap-accessing operations are **tight**

  they only refer to the part of the heap they need to access

  (their **footprint**)
- along these lines,

  tight version of the axiom for (usual) assignment:

$$\overline{\{X = i \wedge emp\}\; X := a\; \{X = a[i/X] \wedge emp\}}$$

if X does not occur in a, the rule becomes simpler: $\overline{\{emp\}\; X := a\; \{X = a \wedge emp\}}$

- moreover, being tight tells us the following:
  - suppose we can prove $\{10 \mapsto 32\}\; p\; \{10 \mapsto 52\}$ $\quad$ whatever p is
  - then we know that

    if we run p in a state where cell 11 is allocated, then p will not change the value of 11

The frame rule

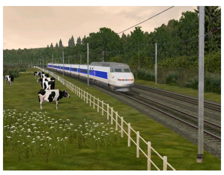
- the rules of Hoare logic remain sound
- the rule of consistency $\dfrac{\{A\}\, p\, \{B\}}{\{A \wedge C\}\, p\, \{B \wedge C\}}$ no variable in C is modified by p

  becomes unsound

$$\frac{\{x \mapsto \_\}\; [x] := 4\; \{x \mapsto 4\}}{\{x \mapsto \_ \wedge y \mapsto 3\}\; [x] := 4\; \{x \mapsto 4 \wedge y \mapsto 3\}} \qquad \text{what if } x = y?$$

- **the Frame Rule**

$$\frac{\{A\}\, p\, \{B\}}{\{A * C\}\, p\, \{B * C\}} \quad \text{no variable in } C \text{ is modified by } p$$



- separation logic is inherently **modular**

  as opposed to *whole program verification*

Separation logic: sum up

- inference rules
  - those of Hoare logic $\quad$ control
  - those for the new programming constructs $\quad$ memory
- important things:
  - invariants in while loops, backward rule for assignment, consequence rule
  - (tight) small axioms, footprint, frame rule
- metatheoretical properties
  - correctness
  - completeness

Beyond absence of runtime errors: recursive data structures

Reasoning about lists

- a linked list in memory is something like

$$(X_1 \mapsto k_1, X_2) * (X_2 \mapsto k_2, X_3) * \cdots * (X_n \mapsto k_n, \underline{nil})$$

$(X \mapsto a, b)$ stands for $X \mapsto a * (X+1) \mapsto b$

- describe the structure using assertions:

  add the possibility to write **(recursive) equations**

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

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

  we can rely on "mathematical lists" ($[\,], k::ks$) to provide a more informative definition

$$\begin{aligned} list([\,], i) &= emp \wedge i = \underline{nil} \\ list(k::ks, i) &= \exists j.\, (i \mapsto k, j) * list(ks, j) \end{aligned}$$

Recursive data structures

- we can specify similarly various kinds of data structures
- we can give a meaning to such recursive definitions using Tarski's theorem

- an **exercise**

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

  - write the code for a while loop that deallocates a linked list,
  - and prove $\{list(X)\}\, p\, \{emp\}$, where p is your program

On the weirdness of auxiliary variables

- in the lecture we saw the small axiom for lookup

$$\{a \mapsto i \ \wedge\ X = j\}\ X := [a]\ \{X = i \ \wedge\ a[j/X] \mapsto i\}$$

- in the TD you saw its simpler form

$$\{a \mapsto i\}\ X := [a]\ \{X = i \ \wedge\ a[j/X] \mapsto i\} \quad \text{if } X \text{ does not appear in } a$$

- how does one entail the other?

  rule of **auxiliary variable elimination** $\dfrac{\{A\}\, p\, \{B\}}{\{\exists u.A\}\, p\, \{\exists u.B\}}$

  if u does not appear in p

- if X does not appear in a, $a[j/X] = a$

moreover, $$\dfrac{\{a \mapsto i \ \wedge\ X = j\}\ X := [a]\ \{X = i \ \wedge\ a \mapsto i\}}{\{\underbrace{\exists j.\, (a \mapsto i \ \wedge\ X = j)}_{\Leftrightarrow\ a \mapsto i \,\wedge\, \exists j. X = j \ \Leftrightarrow\ a \mapsto i}\}\ X := [a]\ \{\underbrace{\exists j.\, (X = i \ \wedge\ a \mapsto i)}_{\Leftrightarrow\ X = i \,\wedge\, a \mapsto i}\}}$$

Going further

Reasoning about concurrent programs

concurrent separation logic

- shared memory, several threads
- permissions, locks, critical sections
- ownership

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_is are elementary commands.

- construction of the Hoare triple boils down to being able to prove *entailments* between assertions, $A \vdash B$

cf. the *Why3 tool* (Filliâtre et al.)

Towards automation of separation logic

- restrict the set of possible formulas: **symbolic heaps**

$$P \wedge H \qquad \begin{aligned} P &= P_1 \wedge \cdots \wedge P_k & &\text{pure formulas} \\ H &= H_1 \wedge \cdots \wedge H_n & &\text{simple heap formulas} \end{aligned}$$

(for instance, $\mapsto$, $list$, 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
    - $(list(i) \wedge i = \underline{nil}) \ \Rightarrow\ emp$
    - $(i \mapsto k, j \ \wedge\ list(j)) \ \Rightarrow\ list(i)$
- more automation: **discovering loop invariants**
  - back to *abstract interpretation*: abstract execution, generating a postcondition as we run through the loop
  - sometimes *abstracting* (narrowing) to insure termination
    - e.g., replacing $i \mapsto k, j \ \wedge\ j \mapsto (k', j') \ \wedge\ j' = \underline{nil}$ with $list(i)$

(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 $A * \texttt{?antiframe} \vdash B * \texttt{?frame}$
  - 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