# Lecture 23 - Hoare logic

**Vaishnavi Sundararajan**

COL703 - Logic for Computer Science

# So far...

- Want logics to be **sound**, so we can believe what is proved
- Want logics to be **complete**, so we can look for proofs instead of searching for truth!
- Propositional and first-order logic both sound and complete
- Want efficient proof search procedures
- Because logic gets used for **verification**

# Logic in CS: Formal verification

- "If I type my password in this box, nobody except me gets to know it"
- Testing maybe fine for small programs with restricted use-cases
- What about safety-critical programs that perform financial transactions? manage national data? fly planes?
- Need to **prove** that **no** bad things **ever** happen
- Even missing one possible test execution could be disastrous!
- Perform **formal verification**; use logic
- What does this involve?

## Logic in CS: Formal verification

- Could cast the system in some logic and make inference
- We saw some models for games etc earlier
- Need to abstract out useless details, but keep the important core
- Sometimes you might not be able to model everything
- Use an expressive-enough logic!
- But what of existing software programs? Abstraction is an extra chore
- Want a quicker way to verify that they do what they're supposed to!

# Verification of imperative programs

- Suppose I have a program written in some imperative language
- How do I figure out if it does exactly what it is supposed to do?
- Need **all possible executions** to satisfy the required guarantee
- Think of the program as **transforming machine state**
- Essentially a flowchart, where every command box is a transformation
- The overall transformation somehow implies the requirement? Great!

# Hoare logic

- Annotate a command by two assertions
- **Precondition**: holds before command is run
- **Postcondition**: holds after command is run
- Can annotate an entire program like this!
- If the states before and after executing the program satisfy some desired properties, the guarantee is met.

# Syntax

- Need to be able to talk about the annotation as well as about commands
- Arithmetic expressions: quantities one assigns to variables

$$e_1, e_2 \coloneqq n \mid X \mid e_1 + e_2 \mid e_1 \times e_2$$

- $n$ is a natural number, $X$ is a program variable
- Boolean expressions: quantities one can branch on

$$b_1, b_2 \coloneqq \textsc{True} \mid \textsc{False} \mid e_1 == e_2 \mid e_1 \leqslant e_2 \mid \neg b_1 \mid b_1 \wedge b_2$$

- $e_1, e_2$ are arithmetic expressions (as expected)
- Commands: refer to one or both of the above

$$c_1, c_2 \coloneqq \mathit{skip} \mid X = e \mid c_1; c_2 \mid \mathit{if}\ b\ \mathit{then\ do}\ c_1\ \mathit{else}\ c_2\ \mathit{end} \mid \mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}$$

# Semantics of arithmetic expressions

- Since programs transform machine state, semantics in terms of states
- What is a state? A finite partial function from program variables to $\mathbb{N}$
- We give semantics to the expressions first
- Denote by $[\![e]\!]s$ the meaning of the arithmetic expression $e$ in state $s$

$$[\![n]\!]s \coloneqq n$$

$$[\![X]\!]s \coloneqq s(X)$$

$$[\![e_1 + e_2]\!]s \coloneqq [\![e_1]\!]s + [\![e_2]\!]s$$

$$[\![e_1 \times e_2]\!]s \coloneqq [\![e_1]\!]s \times [\![e_2]\!]s$$

- Need to provide a semantics for the Boolean expressions next

# Semantics for Boolean expressions

- This we do in the "satisfaction" kind of style

$$
\begin{array}{rl}
s \models \textsc{True} & \text{always} \\
s \models \textsc{False} & \text{never} \\
s \models e_1 == e_2 & \text{iff} \quad [\![e_1]\!]s = [\![e_2]\!]s \\
s \models e_1 \leqslant e_2 & \text{iff} \quad [\![e_1]\!]s \leqslant [\![e_2]\!]s \\
s \models \neg b & \text{iff} \quad s \not\models b \\
s \models b_1 \wedge b_2 & \text{iff} \quad s \models b_1 \text{ and } s \models b_2
\end{array}
$$

- How do we now provide semantics to commands?
- A command $c$ transforms one state $s_1$ into another $s_2$
- **Big-step semantics** in terms of $c$, $s_1$, and $s_2$
- Captures the state change effected by the entire command in one go

# Semantics for commands

$$s \text{—}[\mathit{skip}]\text{→} s \qquad \frac{[\![e]\!]s = n}{s\text{—}[X = e]\text{→} s[X \mapsto n]} \qquad \frac{s\text{—}[c_1]\text{→} s_1 \quad s_1\text{—}[c_2]\text{→} s_2}{s\text{—}[c_1; c_2]\text{→} s_2}$$

$$\frac{s \vDash b \quad s\text{—}[c_1]\text{→} s'}{s\text{—}[\mathit{if}\ b\ \mathit{then\ do}\ c_1\ \mathit{else}\ c_2\ \mathit{end}]\text{→} s'} \qquad \frac{s \nvDash b \quad s\text{—}[c_2]\text{→} s'}{s\text{—}[\mathit{if}\ b\ \mathit{then\ do}\ c_1\ \mathit{else}\ c_2\ \mathit{end}]\text{→} s'}$$

$$\frac{s \nvDash b}{s\text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\text{→} s} \qquad \frac{s \vDash b \quad s\text{—}[c]\text{→} s_1 \quad s_1\text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\text{→} s_2}{s\text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\text{→} s_2}$$

where

$$(s[X \mapsto n])(Y) = \begin{cases} n & \text{if } Y = X \\ s(Y) & \text{otherwise} \end{cases}$$

**Theorem (Determinism of commands)**: For any command $c$ and state $s$, there is at most one $s'$ such that $s\text{—}[c]\text{→} s'$. **Exercise**: Prove this!

# Example program

```
x = 3; y = 1; z = 0;
while (x > z) do
    z = z + 1; y = y * z
end
```

- What does this piece of code do?

# Example program

```
x = 3; y = 1; z = 0;
while (x > z) do
    z = z + 1; y = y * z
end
```

- What does this piece of code do?
- Let $c_1 = x = 3; y = 1; z = 0$ and $w$ be the while loop after it.
- Denote by $(p, q, r)$ the state $[x \mapsto q, y \mapsto q, z \mapsto r]$
- Can we show that $(0, 0, 0) \xrightarrow{[c_1; w]} (3, 6, 3)$?
- Easy to analyze assignment statements
- How does one deal with a while loop?

# Example program: Analysis

- First prove that for all $m, n, p \in \mathbb{N}$ where $p \leqslant m$, it is the case that $(m, n, p) \text{—}[w]\!\rightarrow (m, f(m, n, p), m)$, where $f(m, n, p) = m * (m-1) * (m-2) * \cdots * (p+2) * (p+1) * n$.

# Example program: Analysis

- First prove that for all $m, n, p \in \mathbb{N}$ where $p \leqslant m$, it is the case that $(m, n, p) \text{—}[w]\!\rightarrow (m, f(m, n, p), m)$, where $f(m, n, p) = m * (m-1) * (m-2) * \cdots * (p+2) * (p+1) * n$.
- By induction on $m - p$.
- **Base case:** $m - p = 0$, i.e. $m = p$ and $f(m, n, p) = n$. $(m, n, p) \not\models x > z$, so we have $(m, n, p) \text{—}[w]\!\rightarrow (m, n, p)$.
- **Induction case:** $m - p > 0$, i.e. $p < m$ so $p + 1 \leqslant m$. By IH,

$$(m, n * (p+1), p+1) \text{—}[w]\!\rightarrow (m, f(m, n * (p+1), p+1), m)$$

# Example program: Analysis (contd.)

- What is $f(m, n * (p + 1), p + 1)$?

# Example program: Analysis (contd.)

- What is $f(m, n * (p+1), p+1)$? Nothing but $f(m, n, p)$
- So $(m, n * (p+1), p+1) \xrightarrow{[w]} (m, f(m, n, p), m)$
- Note that $(m, n, p) \xrightarrow{[z = z+1; y = y * z]} (m, n * (p+1), p+1)$
- So $(m, n, p) \xrightarrow{[w]} (m, f(m, n, p), m)$
- In particular, $(0, 0, 0) \xrightarrow{[x = m; y = 1; z = 0]} (m, 1, 0)$, and $(m, 1, 0) \xrightarrow{[w]} (m, f(m, 1, 0), m)$, where $f(m, 1, 0) = m!$

# Hoare triples

- Reasoning directly with the transitions of a program: complex
- Instead: reason about assertions that hold before and after a program
- **Hoare triples** $\{\alpha\}\, c\, \{\beta\}$
  - $c$ is a command
  - $\alpha, \beta$ first-order formulas involving expressions (arithmetic & Boolean)
  - $\alpha$ is the precondition, $\beta$ is the postcondition of the triple
- Informally, $\{\alpha\}\, c\, \{\beta\}$ means that whenever $c$ is run in a state satisfying $\alpha$, if it terminates, then the end state satisfies $\beta$.
- **Partial correctness assertions**: we do not require that $c$ terminates
- Hoare logic gives us rules to reason about these triples directly

# Hoare logic rules

$$\frac{}{\{\alpha\}\ \mathit{skip}\ \{\alpha\}}\ \textbf{Skip} \qquad \frac{}{\{\alpha(e)\}\ X = e\ \{\alpha(X)\}}\ \textbf{Assign}$$

$$\frac{\{\alpha\}\ c\ \{\beta\} \quad \{\beta\}\ c'\ \{\varphi\}}{\{\alpha\}\ c; c'\ \{\varphi\}}\ \textbf{Seq} \qquad \frac{\vDash \alpha' \supset \alpha \quad \{\alpha\}\ c\ \{\beta\} \quad \vDash \beta \supset \beta'}{\{\alpha'\}\ c\ \{\beta'\}}\ \textbf{Con}$$

$$\frac{\{\alpha \wedge b\}\ c\ \{\beta\} \quad \{\alpha \wedge \neg b\}\ c'\ \{\beta\}}{\{\alpha\}\ \mathit{if}\ b\ \mathit{then\ do}\ c\ \mathit{else}\ c'\ \mathit{end}\ \{\beta\}}\ \textbf{If} \qquad \frac{\{b \wedge \iota\}\ c\ \{\iota\}}{\{\iota\}\ \mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}\ \{\iota \wedge \neg b\}}\ \textbf{While}$$

We say that $\vdash \{\alpha\}\ c\ \{\beta\}$ if there is a proof of $\{\alpha\}\ c\ \{\beta\}$ using these rules.

# About the rules: Assign and While

- **Assign**: $\alpha(X)$ is an assertion in which program variable $X$ possibly occurs, and $\alpha(e)$ obtained by replacing all occurrences of $X$ in $\alpha$ by $e$.
- If $\alpha$ is to be satisfied by $X$ after $X = 3$, $\alpha$ should hold of $3$ **to begin with**.
- Suppose $\alpha(X)$ asserts that $X$ is odd. $\alpha$ true after $X = 3$; $3$ is **already** odd.
- Thus $\alpha(3)$ is an adequate precondition for $\alpha(X)$.
- Same logic works even when $e$ contains program identifiers (even $X$).
- **While**: $\iota$ is a **loop invariant**
- A loop invariant is a property that, if it is true at the beginning of a loop iteration, is re-established at the end of the iteration
- Loop invariants are critical to proving the correctness of programs

# Hoare logic

- A Hoare triple $\{\alpha\}\, c\, \{\beta\}$ is said to be **valid** (denoted $\vDash \{\alpha\}\, c\, \{\beta\}$) if for all states $s, s'$, if $s \vDash \alpha$ and $s \xrightarrow{[c]} s'$, then $s' \vDash \beta$.
- Having defined $\vdash \{\alpha\}\, c\, \{\beta\}$ and $\vDash \{\alpha\}\, c\, \{\beta\}$, what do we ask for next?

# Hoare logic

- A Hoare triple $\{\alpha\}\, c\, \{\beta\}$ is said to be **valid** (denoted $\models \{\alpha\}\, c\, \{\beta\}$) if for all states $s, s'$, if $s \models \alpha$ and $s \xrightarrow{[c]} s'$, then $s' \models \beta$.
- Having defined $\vdash \{\alpha\}\, c\, \{\beta\}$ and $\models \{\alpha\}\, c\, \{\beta\}$, what do we ask for next?
- **Theorem (Soundness)**: If $\vdash \{\alpha\}\, c\, \{\beta\}$, then $\models \{\alpha\}\, c\, \{\beta\}$
- **Proof sketch**: Proof is by induction on the structure of the proof.
- **Exercise**: Show that **Skip**, **Con**, and **Seq** preserve validity.
- **Assign**: Also easy, but a friendly old lemma is required!
- **If**: Needs two cases, both work out thanks to IH
- **While**: This is the only tough case that needs some analysis.

# Hoare logic: Soundness (While case)

- We show that for any $n \in \mathbb{N}$, and for any $s, s'$ s.t. $s \vDash \iota$ and there is a proof of $s \text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\!\rightarrow s'$ of size $\leqslant n$, we have $s' \vDash \iota \wedge \neg b$.
- Consider a proof $\pi$ of $s \text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\!\rightarrow s'$ of size $n_0$.
- Suppose the above statement holds for all $m < n_0$ (Call this $\mathrm{IH}_{\mathrm{prf}}$)
- Two cases arise now.
  - $s \nvDash b$ and $s = s'$: Easily done
  - $s \vDash b$, and there is some $s''$ s.t. $s \text{—}[c]\!\rightarrow s''$ and there is a proof of $s'' \text{—}[\mathit{while}\ b\ \mathit{do}\ c\ \mathit{end}]\!\rightarrow s'$ of size $< n_0$: Use IH and $\mathrm{IH}_{\mathrm{prf}}$
- **Exercise**: Fill in the details of this proof
- What next? Completeness
- Remember that we do not require termination

# Hoare logic: Completeness

**Theorem (Weakest liberal precondition)**: For every assertion $\psi$ and command $c$, there is an assertion $wlp(c, \psi)$ such that (1) for all states $s$, we have that $s \vDash wlp(c, \psi)$ iff for all states $s'$, if $s \text{—}[c]\!\rightarrow s'$, then $s' \vDash \psi$, and (2) $\vdash \{wlp(c, \psi)\}\, c\, \{\psi\}$.

Suppose we can prove the above theorem. Then we can prove completeness.

**Theorem (Completeness)**: If $\vDash \{\varphi\}\, c\, \{\psi\}$, then $\vdash \{\varphi\}\, c\, \{\psi\}$.

**Proof sketch**: Suppose $\vDash \{\varphi\}\, c\, \{\psi\}$. Use the (1) part of the above theorem to show that $\vDash \varphi \supset wlp(c, \psi)$. Then, use the (2) part of the above theorem to apply the **Con** rule to get $\vdash \{\varphi\}\, c\, \{\psi\}$. **Exercise**: Finish this proof

So now we need to prove the wlp theorem!