CS 839 Systems Verification
Lecture 5: Hoare logic

this lecture will be on the board, this is just the plan / my notes

Learning outcomes

1. Explain what pre- and post-conditions mean
2. Formally analyze a "whiteboard" programming language

The Big Idea

$\{P\}\, e\, \{Q\}$

"*if* we run e in a state satisfying P and it terminates, *then* Q will be true of the final state"

Some more details

- semantics: what happens when we *run* e?
- logic: set of *rules* for proving $\{P\}\, e\, \{Q\}$
- soundness: are those rules *correct*?

The goal:

$$\begin{aligned} \mathrm{euclid}(a, b) &:= \textsf{if}\, b == 0\, \textsf{then}\, a\, \textsf{else}\, \mathrm{euclid}(b, \mathrm{mod}(a, b)) \\ \mathrm{mod}(a, b) &:= a - (a\, \textsf{div}\, b) * b \end{aligned}$$

$$\{a \geq 0 \wedge b > 0\}\, \mathrm{mod}(a, b)\, \{c.\, \exists k \geq 0.\, a = b \cdot k + c \wedge 0 \leq c < b\}$$

$$\{a \geq 0 \wedge b \geq 0\}\, \mathrm{euclid}(a, b)\, \{c.\, \mathrm{gcd}(a, b, c)\}$$

We want to reason about functions like these. `euclid` involves recursion, so that's one tricky thing to handle. The other interesting aspect is that we use `mod` inside `euclid`. The goal will be that the proof of `euclid` uses the proof of `mod`, saving us work.

Key principle of Hoare logic

Proof structure mirrors code structure

euclid is recursive -> proof by induction

euclid calls mod -> proof of euclid uses proof of mod as a lemma

Syntax

$$
\begin{array}{lrl}
\text{Expressions} & e ::= & x \mid v \mid \lambda x.\, e \mid e_1\, e_2 \\
& & \mid \textbf{if}\, e\, \textbf{then}\, e_1\, \textbf{else}\, e_2 \\
& & \mid e_1 + e_2 \mid e_1 == e_2 \mid e_1 < e_2 \\
& & \mid (e_1, e_2) \mid \pi_1\, e \mid \pi_2\, e \\
\text{Values} & v ::= & \lambda x.\, e \mid \overline{n} \mid \mathsf{true} \mid \mathsf{false} \mid (v_1, v_2)
\end{array}
$$

r:

$$
\begin{aligned}
\lambda x, y.\, e &::= \lambda x.\, \lambda y.\, e \\
\textbf{let}\ x := e_1\ \textbf{in}\ e_2 &::= (\lambda x.\, e_2)\, e_1 \\
e_1\, e_2\, e_3 &::= (e_1\, e_2)\, e_3
\end{aligned}
$$

Semantics

Rules defining $e_1 \to e_2$ (step relation):

$$
\begin{aligned}
(\lambda x.\, e)v &\to e[v/x] \quad \text{(beta reduction)} \\
\textbf{if}\ \text{true}\ \textbf{then}\ e_1\ \textbf{else}\ e_2 &\to e_1 \quad \text{(if-true)} \\
\textbf{if}\ \text{false}\ \textbf{then}\ e_1\ \textbf{else}\ e_2 &\to e_2 \quad \text{(if-false)} \\
\pi_1\,(v_1, v_2) &\to v_1 \quad \text{(proj-fst)} \\
\pi_2\,(v_1, v_2) &\to v_2 \quad \text{(proj-snd)} \\
\overline{n_1} + \overline{n_2} &\to \overline{n_1 + n_2}
\end{aligned}
$$

$$
\frac{n_1 = n_2}{\overline{n_1} == \overline{n_2} \to \text{true}} \qquad \frac{n_1 \neq n_2}{\overline{n_1} == \overline{n_2} \to \text{false}}
$$

$$
\frac{n_1 < n_2}{\overline{n_1} < \overline{n_2} \to \text{true}} \qquad \frac{n_1 \geq n_2}{\overline{n_1} < \overline{n_2} \to \text{false}}
$$

Activity: explain how to read these

Activity: add sums to language

5-min break

Program proofs without any techniques

example: `mod` above

```
mod(a, b) := a - (a / b) * b
```

maybe easy, but what about recursion?

```
euclid(a, b) := if b == 0 then a else
euclid(b, mod(a, b))
```

Scaling up

Now imagine verifying some assembly code using the official semantics

```
_start:
    mov r6, #0          // Initialize
accumulator r6 to 0
    mov r0, #0          // Initialize counter
r0 to 0
    mov r1, #10         // Set upper limit to
10

loop:
    add r6, r6, r0      // Add current counter
value to accumulator
    add r0, r0, #1      // Increment counter
    cmp r0, r1          // Compare counter
with limit (10)
    ble loop            // Branch if counter
<= 10

    // At this point, r6 contains the sum:
0+1+2+3+4+5+6+7+8+9+10 = 55
```

https://developer.arm.com/documentation/dui0231/b/arm-instruction-reference/arm-general-data-processing-instructions/add--sub--rsb--adc--sbc--and-rsc?lang=en

https://developer.arm.com/documentation/dui0231/b/arm-instruction-reference/conditional-execution?lang=en

The point is that we want to *abstract* away behavior and create *modular* reasoning principles that divide up the effort.

Hoare logic

$$\{P\}\, e \,\{\lambda v.\, Q(v)\}$$

Soundness: what does a Hoare triple mean?

$$\{P\}\, e\, \{\lambda v.\, Q(v)\}$$

$$\forall v', P \wedge (e \rightsquigarrow v') \implies Q(v')$$

Proof system

$$\frac{\{P\}\ e_1\ \{\lambda v.\, Q(v)\} \quad \forall v.\ \{Q(v)\}\ e_2[v/x]\ \{R\}}{\{P\}\ \mathbf{let}\ x := e_1\ \mathbf{in}\ e_2\ \{R\}}\ \text{hoare-let}$$

Logic rules

$$\frac{P' \vdash P \quad (\forall v.\, Q(v) \vdash Q'(v)) \quad \{P\}\ e\ \{Q(v)\}}{\{P'\}\ e\ \{\lambda v.\, Q'(v)\}}\ \text{consequence}$$

Exercise: prove pure step

Run into a problem: need determinism