CE 474 - LOGIC OF COMPUTER SCIENCE

Lecture Note 9 — Hoare Logic & Model Checking

Learning objectives

By the end of this lecture you should be able to:

- Understand the syntax and semantics of a Hoare triple $\{P\}\ C\ \{Q\}$.
- Apply core Hoare proof rules (assignment, sequence, if, while) to simple programs.
- Grasp model-checking basics: state-transition systems and temporal logics (LTL/CTL).
- Relate the complementary roles of Hoare logic (deductive proofs) and model checking (algorithmic verification).

Warm-up (Quick question)

What does the Hoare triple $\{x = 1\}\ x := x + 1\ \{x = 2\}$ assert? Explain in one sentence.

1. Hoare Logic — core ideas

Hoare triple

A Hoare triple $\{P\}\ C\ \{Q\}$ means: if precondition P holds before executing command C, and C terminates, then postcondition Q holds after execution.

P and Q are assertions (predicates) over program variables.

Core proof rules (summary)

- **Assignment rule:**

$$\frac{}{\{Q[x := E]\}\ x := E\ \{Q\}}$$

  Here $Q[x := E]$ denotes Q with each free occurrence of x replaced by E.

- **Sequence:** If $\{P\}\ C_1\ \{R\}$ and $\{R\}\ C_2\ \{Q\}$ then $\{P\}\ C_1; C_2\ \{Q\}$.
- **Conditional:** If $\{P \wedge B\}\ C_{then}\ \{Q\}$ and $\{P \wedge \neg B\}\ C_{else}\ \{Q\}$ then $\{P\}$ if B then C_{then} else C_{else} $\{Q$
- **While (invariant) rule:** If $\{I \wedge B\}\ C\ \{I\}$ then $\{I\}$ while B do C $\{I \wedge \neg B\}$. I is the loop invariant.

Checkpoint

Why does the assignment rule use $Q[x := E]$ on the precondition side? Give a short intuitive answer.

2. Worked example — factorial program (Hoare proof)

We prove partial correctness of:

Program

```
{ n >= 0 }
fact := 1;
i := 1;
while i <= n do
  fact := fact * i;
  i := i + 1
od
{ fact = n! }
```

Choose a loop invariant

A standard invariant is:

$$I(i, f): \quad f = (i-1)! \ \wedge \ 1 \le i \le n+1.$$

(Informally: before each loop iteration, `fact` equals $(i-1)!$.)

Proof sketch (steps)

1. Initialization (establish I before the loop): At the start, after `fact:=1; i:=1`, we have $fact = 1$ and $i = 1$; indeed $1 = (1-1)! = 0!$, and $1 \le 1 \le n+1$ holds since $n \ge 0$. So I holds.

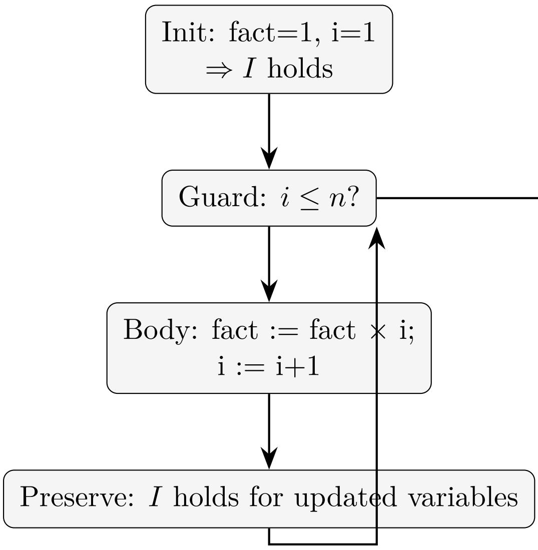
2. Preservation (body maintains I): Assume I and $i \le n$ (loop guard true). So $fact = (i-1)!$. After `fact := fact * i`, new $fact$ is $(i-1)! \cdot i = i!$. After `i := i+1`, new i is $i+1$. Then

$$\text{new } fact = i! = ((i+1)-1)!,$$

so I holds with the updated i and $fact$. Also bounds still satisfy $1 \le i+1 \le n+1$.

3. Termination/Exit: When the loop exits, $i > n$. From invariant $f = (i-1)!$ and $i = n+1$ (since exit after reaching $n+1$), we get $f = (n+1-1)! = n!$. Thus postcondition holds.

Compact invariant-flow diagram (inline)



Checkpoint

Explain briefly why the invariant $f = (i - 1)!$ is strong enough to imply the post-condition when the loop exits.

3. Model Checking — essentials

State-transition systems

A (Kripke) model is $M = (S, R, L)$:

- S: finite set of states;
- $R \subseteq S \times S$: transition relation;
- $L : S \to 2^{AP}$: labelling of each state with atomic propositions.

Temporal logics (very brief)

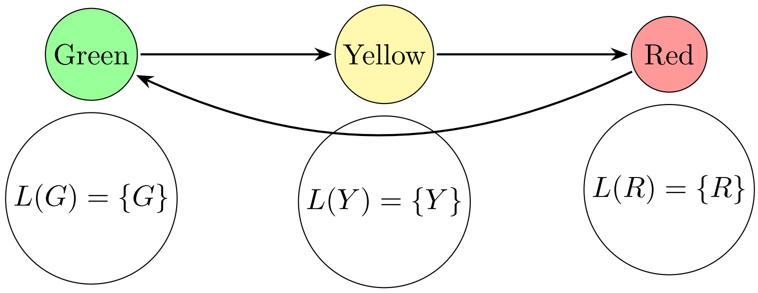
- **LTL** (linear time): properties over infinite paths. e.g. $X\varphi$ (next), $F\varphi$ (eventually), $G\varphi$ (always), $\varphi\, U\, \psi$ (until).
- **CTL** (branching time): state formulas quantify over paths, e.g. $AG\varphi$ (on all paths always φ), $EF\varphi$ (exists a path eventually φ).

Model checking problem

Given model M and temporal formula Φ, decide whether $M \models \Phi$ holds (usually at a designated initial state).

4. Worked example — traffic-light model

We model a simple controller with states $\{Green, Yellow, Red\}$ and atomic props G, Y, R.



CTL property check

Consider: $AG\,(G \to AF\,R)$ — "on all paths, whenever Green holds, then on all paths eventually Red holds."

Discussion: In this cyclic model, from Green the unique (and all) next paths go to Yellow then Red, so eventually R becomes true on all paths; thus the property *holds*.

Practice

Draw a variant model where from Green you can nondeterministically stay in Green forever (self-loop) or go to Yellow. Does $AG\,(G \to AF\,R)$ hold there? Explain.

5. In-class exercises

Exercise 1: Hoare proof

Prove partial correctness of the program given earlier (factorial) more formally:

$\{n \geq 0\}\ fact := 1;\ i := 1;$ **while** $i \leq n$ **do** $fact := fact \cdot i; i := i+1$ **od** $\{fact = n!\}$

Use invariant $I : fact = (i-1)! \wedge 1 \leq i \leq n+1$ and show initialization, preservation, and exit.

Exercise 2: Model checking

For the traffic-light model:

1. Express "it is always the case that eventually Red occurs" in LTL and CTL.
2. Verify whether $AG(G \to AF\,R)$ holds if we add a self-loop on Green.

6. Summary & takeaways

- **Hoare logic** gives a structured, deductive way to reason about program *partial correctness* via triples and invariants. Choosing the right invariant is the key step.
- **Model checking** explores finite-state models exhaustively against temporal logic specifications (LTL/CTL); it is automated and well-suited to reactive systems.
- Both approaches are complementary: Hoare logic for program-centered proofs and model checking for exhaustive, automated analysis of state-based systems.

References

- Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.
- Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model Checking*. MIT Press.
- Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 576–580.
- Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.