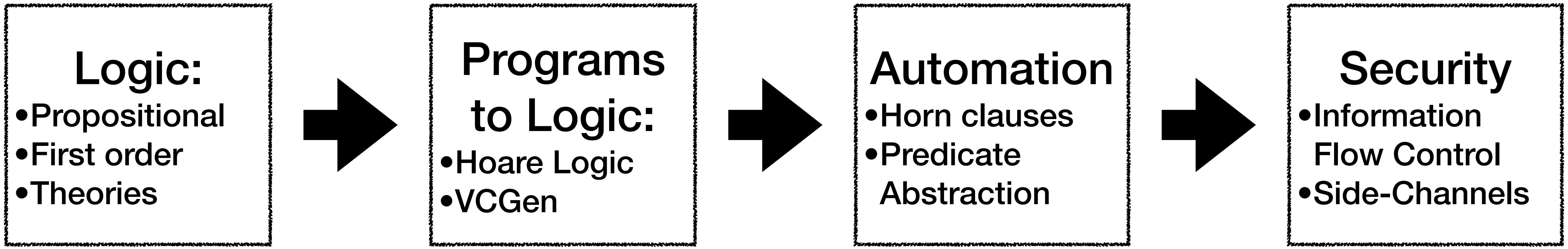
Nano & Hoare Logic

Klaus v. Gleissenthall


Logic:
•Propositional
•First order
•Theories
Programs to Logic:
•Hoare Logic
•VCGen
Automation
•Horn clauses
•Predicate Abstraction
Security
•Information Flow Control
•Side-Channels

Reminder

- Please interrupt at any time and ask questions!
- For example, these are perfectly good questions to ask:
  - I didn’t get this part, can you explain it again?
  - Can you explain this in a different way?
  - You’re talking too fast, can you slow down?
  - Wait, I still wanted to read that.
  - What does any of this have to do with anything?
  - Why are we doing this?

Questions

Quiz:

- Why do we need first order theories?
- What makes first order Theories different from standard FOL?
- Are satisfiability & validity easier problems in first order theories?
- What are some theories we can use in verification?

Recap: Congruence

<u>Example:</u> Consider universe U = {□,◍,★} and

- I(=) ≜ {⟨□,□⟩,⟨□,◍⟩,⟨◍,□⟩,⟨◍,◍⟩,⟨★,★⟩}

Quiz:

- Does I(=) satisfy the axioms of equality?

Which interpretations for a function *f* satisfies the axioms of congruence?

- I(f) ≜ {◍ → □, □ →★, ★→★ } ?
- I(f) ≜ {◍ → ◍, □ → ◍, ★→ ◍ } ?
- I(f) ≜ {◍ → □, □ →◍, ★→★ } ?

Where are we?

- Logic as the language of computation
- FOL is undecidable & hence not a good basis for verification
- Theories (SMT) are expressive & decidable which allows for reliable checking
- Next: step back from logic & look at <u>programs</u>
- How can we prove things about them?
- Plan for today:
    - Introduce a simple programming language (Nano)
    - Show how to prove things about it!
    - Next lecture: Checking proofs automatically via SMT solvers (VCGEN)

Plan

- To understand if programs are correct (= is it doing the right or wrong thing?), we first need to know what the program means!

- Today: meaning = operational semantics

- Then we'll talk about specifying and proving correctness

A Simple Imperative Language: Syntax

- We consider a small imperative programming language Nano
- Made up of expressions ranging over integers, Boolean expressions, and statements

Expressions: $e, e_1, e_2 \ni \text{Exp} ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2$ where $n \in \mathbb{Z}$, $x \in \text{Vars}$

Boolean Expressions: $b, b_1, b_2 \ni \text{BExp} ::= \top \mid \bot \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid e_1 = e_2 \mid e_1 \leq e_2$

Question: We are missing $<$, $>$, $\geq$, $\neq$, is this a problem?

A Simple Imperative Language: Syntax

- We consider a small imperative programming language Nano
- Made up of expressions, Boolean expressions, and statements

Expressions: $e, e_1, e_2 \ni \text{Exp} ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2$ where $n \in \mathbb{Z}$, $x \in \text{Vars}$

Boolean Expressions: $b, b_1, b_2 \ni \text{BExp} ::= \top \mid \bot \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid e_1 = e_2 \mid e_1 \leq e_2$

Statement:

$s, s_1, s_2 \ni \text{Stmt} ::= \text{skip}$	(no-op)
$\mid x := e$	(assignment)
$\mid s_1 \,;\, s_2$	(sequential composition)
$\mid \text{if } b \text{ then } s_1 \text{ else } s_2$	(if)
$\mid \text{while } b \text{ do } s$	(while)

A Simple Imperative Language: Semantics

- The meaning of Nano programs depends on the value of variables $x \in \mathit{Vars}$
- A <u>state</u> σ is a function from *Vars* to $\mathbb{Z}$; σ captures the <u>current</u> value of all variables

Expressions:

- We define a relation $\langle e, \sigma \rangle \Downarrow n$ saying that e evaluates to number n under σ.

$$\frac{}{\langle n, \sigma \rangle \Downarrow n} \qquad \frac{}{\langle x, \sigma \rangle \Downarrow \sigma(x)}$$

$$\frac{\langle e_1, \sigma \rangle \Downarrow n_1 \quad \langle e_2, \sigma \rangle \Downarrow n_2}{\langle e_1 + e_2, \sigma \rangle \Downarrow n_1 + n_2} \qquad \frac{\langle e_1, \sigma \rangle \Downarrow n_1 \quad \langle e_2, \sigma \rangle \Downarrow n_2}{\langle e_1 - e_2, \sigma \rangle \Downarrow n_1 - n_2} \qquad \frac{\langle e_1, \sigma \rangle \Downarrow n_1 \quad \langle e_2, \sigma \rangle \Downarrow n_2}{\langle e_1 * e_2, \sigma \rangle \Downarrow n_1 * n_2}$$

A Simple Imperative Language: Semantics

Boolean Expressions

- Boolean expressions evaluate to either $\top$ or $\bot$.
- We're skipping $\vee$ and $\neg$.

$$\frac{}{\langle \top, \sigma\rangle \Downarrow \top} \qquad \frac{}{\langle \bot, \sigma\rangle \Downarrow \bot}$$

$$\frac{\langle b_1, \sigma\rangle \Downarrow \bot}{\langle b_1 \wedge b_2, \sigma\rangle \Downarrow \bot} \qquad \frac{\langle b_2, \sigma\rangle \Downarrow \bot}{\langle b_1 \wedge b_2, \sigma\rangle \Downarrow \bot} \qquad \frac{\langle b_1, \sigma\rangle \Downarrow \top \quad \langle b_2, \sigma\rangle \Downarrow \top}{\langle b_1 \wedge b_2, \sigma\rangle \Downarrow \top}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 = e_2, \sigma\rangle \Downarrow n_1 = n_2} \qquad \frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \leq e_2, \sigma\rangle \Downarrow n_1 \leq n_2}$$

A Simple Imperative Language: Semantics

Quiz:

- What does the expression $x+1$ evaluate to in state $\sigma \triangleq \{x \rightarrow 2\}$?
- What about $x+1 \leq 3$?
- Does the order in which we evaluate expressions matter?

A Simple Imperative Language: Semantics

- Expressions evaluate to a number or Boolean value
- But what do statements evaluate to? They have no direct result!
- Instead, they yield a new program state: $\langle s, \sigma \rangle \Downarrow \sigma'$
- We call a pair $\langle s, \sigma \rangle$ a configuration
- This approach to semantics is called structural operational semantics (SOS)
- Introduced by Gordon Plotkin in 1981:

  https://web.eecs.umich.edu/~weimerw/590/reading/plotkin81structural.pdf

- There are other approaches, for example denotational semantics

Gordon Plotkin

A Simple Imperative Language: Semantics

Statements

(skip) $$\frac{}{\langle \text{skip}, \sigma\rangle \Downarrow \sigma}$$

(seq) $$\frac{\langle s_1, \sigma\rangle \Downarrow \sigma_1 \quad \langle s_2, \sigma_1\rangle \Downarrow \sigma_2}{\langle s_1 \,;\, s_2, \sigma\rangle \Downarrow \sigma_2}$$

(if-$\top$) $$\frac{\langle b, \sigma\rangle \Downarrow \top \quad \langle s_1, \sigma\rangle \Downarrow \sigma_1}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2, \sigma\rangle \Downarrow \sigma_1}$$

(if-$\bot$) $$\frac{\langle b, \sigma\rangle \Downarrow \bot \quad \langle s_2, \sigma\rangle \Downarrow \sigma_2}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2, \sigma\rangle \Downarrow \sigma_2}$$

(assn) $$\frac{\langle e, \sigma\rangle \Downarrow n}{\langle x := e, \sigma\rangle \Downarrow \sigma[x \mapsto n]}$$

A Simple Imperative Language: Semantics

Statements

- Last one that's missing: While

$$(\text{while-}\bot)\quad \frac{\langle b, \sigma\rangle \Downarrow \bot}{\langle \text{while } b \text{ do } s, \sigma\rangle \Downarrow \sigma} \qquad (\text{while-}\top)\quad \frac{\langle b, \sigma\rangle \Downarrow \top \quad \langle s \,;\, \text{while } b \text{ do } s, \sigma\rangle \Downarrow \sigma'}{\langle \text{while } b \text{ do } s, \sigma\rangle \Downarrow \sigma'}$$

- While-⊥ exits the loop
- While-⊤ unfolds the while loop once

A Simple Imperative Language: Semantics

Quiz:

- What does statement x :=x-1 evaluate to in state $\sigma \triangleq \{ x \to 2 \}$?
- What about: if (x+1 ≤ 3) then x:=x-1 else skip ?
- What about while (x+1 ≤ 3) then x:=x-1 ?
- Is $\langle e, \sigma \rangle \Downarrow n$ a total function?
- What about $\langle s, \sigma \rangle \Downarrow \sigma'$?

Hoare Logic

- Hoare logic forms the basis of all deductive verification techniques
- Named after Tony Hoare: inventor of quick sort, father of formal verification, 1980 Turing award winner

- Logic is also known as Floyd-Hoare logic: some ideas

  introduced by Robert Floyd in 1967 paper

  " Assigning Meaning to Programs"

- Also called "Axiomatic Semantics"

Dijkstra Quote

- “Program testing can be used to show the presence of bugs, but never to show their absence!”

Specifying Correctness: Hoare Triples

- In Hoare logic, we specify partial correctness of programs using **Hoare triples**:

$$\{P\}\ \mathrm{s}\ \{Q\}$$

- Here, s is a statement in Nano
- P and Q are SMT formulas
- P is called a <u>precondition</u> and Q is called <u>post-condition</u>

Meaning of Hoare Triples

- Meaning of a Hoare triple $\{P\}$ s $\{Q\}$
- Let us write $\sigma \models P$, if P holds on σ, (how could we define $\sigma \models P$?)
- If P holds in some state σ (i.e., $\sigma \models P$) and there exists σ' s.t. $\langle s, \sigma \rangle \Downarrow \sigma'$, then Q holds in state σ' (i.e., $\sigma' \models Q$).
- We write $\models \{P\}$ s $\{Q\}$ and say the Hoare triple is valid.
- What if s never terminates?
- Post-condition Q does not have to hold in that case!

Meaning of Hoare Triples

Quiz:

- Is $\{x=0\}$ x:=x+1 $\{x=1\}$ a valid Hoare triple?
- What about $\{x=0 \wedge y=1\}$ x:=x+1 $\{x=1 \wedge y=2\}$?
- What about $\{x=0 \wedge y=1\}$ x:=x+1 $\{x=1 \vee y=2\}$?
- And about $\{x=0\}$ while $\top$ do x:=x+1 $\{x=1\}$?

Parial vs. Total Correctness

- $\{P\}$ s $\{Q\}$ is called a partial correctness specification, as it doesn't require s to terminate
- There is a stronger requirement called total correctness
- Total correctness is written as $[P]$ s $[Q]$
- Meaning of $[P]$ s $[Q]$:
  - If P holds in some state σ (i.e., $\sigma \models P$), then there exists σ' s.t. $\langle s, \sigma \rangle \Downarrow \sigma'$,

    and Q holds in state σ' (i.e., $\sigma' \models Q$).

Quiz: Is $[x=0]$ while $\top$ do $x:=x+1$ $[x=1]$ valid?

Parial vs. Total Correctness

Quiz:

- What does $\{\top\}$ s $\{Q\}$ say?
- What about $\{P\}$ s $\{\top\}$?
- What about $[P]$ s $[\top]$?
- When does $\{\top\}$ s $\{\bot\}$ hold ?
- When does $\{\bot\}$ s $\{Q\}$ hold ?
- We'll only focus on only partial correctness (safety)
- Total correctness = partial correctness + termination

More Hoare Triples

Quiz: Are these Hoare triples valid or invalid?

- $\{i=0\}$ while $i<n$ do $i:=i+1\{i=n\}$?
- $\{i=0\}$ while $i<n$ do $i:=i+1\{i\geq n\}$?
- $\{i=0\}$ while $i<n$ do $i:=i+1$; $j:=j+i$ $\{2j=n(n+1)\}$?
- What if we strengthen the pre-condition?

Proving Partial Correctness

- Problem: How do we prove that a Hoare triple is valid?
- We want a proof system to show that $\models \{P\}$ s $\{Q\}$ holds
- We write $\vdash \{P\}$ s $\{Q\}$ to indicate that we can prove validity of the Hoare triple
- Hoare gave a sound and (relatively) complete proof system that allows a semi-mechanized correctness proof
- Soundness: If $\vdash \{P\}$ s $\{Q\}$ then $\models \{P\}$ s $\{Q\}$
- Completeness: If $\models \{P\}$ s $\{Q\}$ then $\vdash \{P\}$ s $\{Q\}$

Inference Rules

- Proof rules in Hoare logic are written as inference rules:

$$\frac{\vdash \{P_1\}\ s_1\ \{Q_1\} \qquad \ldots \qquad \vdash \{P_n\}\ s_n\ \{Q_n\}}{\vdash \{P\}\ s\ \{Q\}}$$

- Says: If Hoare triples $\{P_1\}\ s_1\ \{Q_1\}$ to $\{P_n\}\ s_n\ \{Q_n\}$ are provable in our proof system,

  then $\{P\}\ s\ \{Q\}$ is also provable.

- Rules without hypotheses are base cases

- One rule per statement in Nano, let's take a look

Quiz:

Proof Rules: Assignments

- Consider the assignment **x:=y** and and post-condition **x>2**
- What needs to hold before the assignment so that **x>2** holds afterwards?
- Consider **i:=i+1** and post-condition **i>10**
- What do we need to know before the assignment so that **i>10** holds afterwards?

Substitution

- We write *P*[e/x] to mean that we substitute e for variable x, in *P*
- Importantly, we substitute only <u>free occurrences</u>

Quiz:

- What is i>10[i+1/i] ?
- What is (∀i. i>10)[i+1/i] ?

Proof Rules: Assignments

- We write *P*[e/x] to mean that we substitute e for variable x, in *P*

$$\frac{}{\vdash \{Q[e/x]\}\ x := e\ \{Q\}}$$

- To prove Q holds <u>after</u> assignment x := e, it is sufficient to show that Q with e substituted for x holds <u>before</u> the assignment.

Quiz:

- Using this rule, which of these are provable?
  - $\{y = 4\}$ x := 4 $\{y = x\}$
  - $\{x+1=n\}$ x:=x+1 $\{x=n\}$
  - $\{y = x\}$ y:=2 $\{y = x\}$
  - $\{z = 3\}$ y:=x $\{z = 3\}$

Proof Rules: Assignments

- Your friend suggests the following proof rule for assignments:

$$\vdash \{(x=e) \rightarrow \quad Q\}\ x := e\ \{Q\}$$

- Is this proof rule correct?
- Let's try it out on $\{?\}\ x := 4\ \{y = x\}$

Precondition Strengthening

- Is the following Hoare triple valid?

$$\{z = 2\}\ y := x\ \{y = x\}$$

- Can we prove it with the assignment rule?

- Intuitively, we should be able to prove it <u>without</u> any assumptions; we should also be able to prove it <u>if we do</u> have assumptions!

Precondition Strengthening

- We write P $\Rightarrow$ P' to mean that formula P $\rightarrow$ P' is valid, i.e., $\models$ P $\rightarrow$ P'

$$\frac{\vdash \{P'\}\ s\ \{Q\} \qquad P \Rightarrow P'}{\vdash \{P\}\ s\ \{Q\}}$$

- To check P $\Rightarrow$ P', we need to call the SMT solver
- Let's now prove $\{z = 2\}$ y := x $\{y = x\}$

Postcondition Weakening

- We also need a dual rule for post-conditions called post-condition weakening:

$$\frac{\vdash \{P\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\vdash \{P\}\ s\ \{Q\}}$$

- If we prove some post-condition Q', we can always relax it to something weaker
- Again, we need to use an SMT solver when applying post-condition weakening

Postcondition Weakening

Quiz:

- Suppose we can prove $\{\top\}$ s $\{x = y \land z = 2\}$
- Using post-condition weakening, which of these can we prove?
  - $\{\top\}$ s $\{x = y\}$
  - $\{\top\}$ s $\{z = 2\}$
  - $\{\top\}$ s $\{z > 0\}$
  - $\{\top\}$ s $\{\forall y.\ x=y\}$
  - $\{\top\}$ s $\{\exists y.\ x=y\}$

Composition

$$\frac{\vdash \{P\}\ s_1\ \{Q\} \qquad \vdash \{Q\}\ s_2\ \{R\}}{\vdash \{P\}\ s_1;\ s_2\ \{R\}}$$

- Using this proof rule, let's prove validity of the Hoare triple:

$$\{\top\}\ x := 2;\ y := x\ \ \{y = 2 \wedge x = 2\}$$

If Statements

$$\frac{\vdash \{P \land b\}\ s_1\ \{Q\} \qquad \{P \land \neg b\}\ s_2\ \{Q\}}{\vdash \{P\}\ \text{if b then } s_1 \text{ else } s_2\ \{Q\}}$$

- Suppose P holds before the if-statement
- When executing the then-branch, what do we know?
- What about the else-branch?

If Statements

Example: Prove the correctness of this Hoare triple

$\{\top\}$ if $x > 0$ then $y := x$ else $y := -x$ $\{y \geq 0\}$

Where are we?

- Hoare Logic
- How can we prove things about programs?
- What we did:
  - Introduce a simple programming language (Nano)
  - Show how to prove things about it!
  - Next lecture: More Hoare logic!

    Automating proof checking via weakest pre-condition calculus
  - For more background on this part, I recommend the following book:
  - The formal semantics of programming languages by Glynn Winskel
  - https://www.cin.ufpe.br/~if721/intranet/TheFormalSemanticsofProgrammingLanguages
  - Also a good reference, this course: https://web.eecs.umich.edu/~weimerw/2015-6610/lectures.html