Goals:
- Introduce the concept of proving correctness of programs
- New concepts: Hoare triples, assignment axiom, stable operator

Reading:
- P. Sivilotti, *Introduction to Distributed Algorithms*, Section 3.1-3.3
Last Week: Models of Computation

UNITY model provides (seemingly) simple description of programs
- Program = variables + actions [assignments] (that’s it!)
- Guarded assignment ($g \rightarrow a$) allows modeling of finite state automata
- Distributed programs captured by nondeterministic execution model
- Termination = reaching a fixed point (variables remain constant)

Next: how to we prove that specifications are satisfied?
- A1: exhaustive testing [works for simple systems]
- A2: model checking [for specific instantiation]
- A3: formal proof [often generalizable]

Fri: how to prove things using predicate calculus and quantification (review + some new stuff)

This week: reasoning about program behavior and safety properties (invariants)
Examples to Consider

**FindAverage** (average consensus)
- Given a set of N sensors on a graph, would like to agree on the value of the average measurements
- Example: agree that it is too cold and warm up the room
- Q1: what protocol should we implement to solve this problem?
- Q2: is it *always* possible to agree?

**ChooseDefenders**
- Given a set of initial assignments in the “RoboFlag drill”, communicate with left and right neighbors and switch assignments such that we end up with no “crossed” assignments
- Q1: What are the properties we want to guarantee?
  - Termination: program terminates (variables remain constant)
  - Correctness: only fixed points are the desired ones
- Q2: What could go wrong?
  - Deadlock: get stuck in a state (= undesired fixed point)
  - Livelock: never terminate (eg, assignments “oscillate”)

Consensus is reached when:
\[ x_i(n) = \frac{1}{N} \sum_{j=1}^{N} x_j(0), \forall i \]

\( n \): Time index
N: number of nodes
Example: Average Consensus

Problem setup

• Variables: local estimate of average, initialized to local measurement
• Assignments: two agents communicate and share information

Program \textit{AverageConsensus}

constant \( N \) \{number of agents\}
\( G \) \{interconnection graph\}
\( \alpha : 0 < \alpha < 1 \)
var \( x \) : array of \( N \) numbers
assign
\[
(\forall i, j : j \in \mathcal{N}_i : x[i] := \alpha x[i] + (1 - \alpha)x[j] \quad |\mid x[j] := \alpha x[j] + (1 - \alpha)x[i]))
\]

Specification

• Show that we converge to a consensus (everyone agrees on average value)
• In practice, usually good enough to show that we get close within finite time

Why do we need a “proof”?

• Want to understand conditions under which this is not true (eg, directed graphs)
• Can extend to understand more interesting cases (eg, what happens if someone lies)
Properties of Programs

Notation: property(P) or property(P, Q) or P property Q
- A property operates on a set of states that satisfy a formula (predicate) P (and/or Q)
- The property is true if it holds for all possible executions

Reasoning about properties using graphs
- Formulas define subsets of the state space
- Can reason about whether a property holds by looking at how the transitions map to the (sets of states representing) properties

Reasoning about properties using formulas

\[
\text{stable}(r \leq M) \\
\equiv \quad \frac{\text{next}(r \leq M)}{(r < M)} \\
\equiv \quad \frac{(\forall a :: \{r \leq M\} \quad a \quad \{r \leq M\})}{(\forall x : 0 \leq x < N : \{r \leq M\} \quad r := \max(r, A[x]) \quad \{r \leq M\})} \quad \text{definition of program} \\
\equiv \quad \{ \text{assignment axiom} \} \\
(\forall x : 0 \leq x < N : \quad \text{stable}(Q) \\
\text{transient}(P) \\
\neg(P \text{ next } Q)
\]

Can also combine representations
\[
(P \subseteq Q) \land \text{stable}(Q) \Rightarrow \text{reachable}(P) \subseteq Q
\]
Reasoning About Actions

How are we going to prove things?
• A: show that sets of properties hold for all executions

Two main parts of a proof: safety and liveness (or progress)
• Safety: show that bad things don’t (ever) happen.
• Liveness: show that good things eventually do happen
• Roughly: can show that all specifications break down into safety and liveness

Notation: Hoare triple - \{P\} a \{Q\}
• P = precondition (predicate), a = program action, Q = postcondition (predicate)
• Interpretation: the triple evaluates to true if for any program state in which P holds, if we take the action a then Q will hold after the action is executed

Assignment axiom: \{P\} x := E \{Q\}
• In what state must we being execution in order for Q to hold after executing x := E?
• Written another way: find those states for which \([P \implies Q^x_E]\)
Reasoning About Guarded Actions

Hoare triple with a guarded action: \( \{P\} \, g \rightarrow x := E \, \{Q\} \)

- What we need to show depends on whether the guard is true or false
- \( g = \text{true} \): same as assignment
- \( g = \text{false} \): need \( Q \) to be satisfied

\[
[(P \land g \Rightarrow Q^x_E) \land (P \land \neg g \Rightarrow Q)]
\]

Example  \( \{x > y = 7\} \, x > y \rightarrow x, y := y, x \, \{x > 3\} \)

\[
(x > y = 7 \land x > y \Rightarrow y > 3) \land (x > y = 7 \land \neg(x > y) \Rightarrow x > 3)
\]

\[
\begin{aligned}
\iff & \quad \text{antecedent strengthening of } \Rightarrow : \quad [(X \Rightarrow Z) \Rightarrow (X \land Y \Rightarrow Z)] \\
& \quad \{ y = 7 \Rightarrow y > 3 \} \land (x > y = 7 \land \neg(x > y) \Rightarrow x > 3)
\end{aligned}
\]

\[
\begin{aligned}
& \equiv \quad \{ 7 > 3 \} \\
& \quad x > y = 7 \land \neg(x > y) \Rightarrow x > 3 \\
& \quad \equiv \quad \{ \text{definition of } \neg \} \\
& \quad x > y = 7 \land x \leq y \Rightarrow x > 3 \\
\iff & \quad x > y \land x \leq y \Rightarrow x > 3 \\
\equiv & \quad \text{false } \Rightarrow x > 3 \\
\equiv & \quad \{ \text{property of } \Rightarrow : [\text{false } \Rightarrow X \equiv \text{true}] \} \\
\end{aligned}
\]

Recall: \( x > y = 7 \equiv x > y \land y = 7 \)

\( x > y \land y = 7 \land x > y \Rightarrow y > 3 \)

\( (y = 7) \land (x > y) \Rightarrow (y > 3) \)

\( X \quad Y \quad Z \)
Use to reason about properties of a program $G$ as it executes

$$P \text{ next } Q \equiv (\forall a : a \in G : \{P\} \ a \ \{Q\})$$

- $P$ and $Q$ are predicates on states
- `next` is a binary relation between predicates

$P \text{ next } Q$ in terms of graphs means that

- (1) for all edges $(u, v)$ in a graph, if $u$ is in $P$ then $v$ is in $Q$,  
- (2) furthermore for all $u$ in $P$, $u$ is also in $Q$ (why: 

Some useful properties of next (prove in HW #2)

$$\begin{align*}
(P \text{ next } Q) \land (Q \subseteq Q') & \Rightarrow (P \text{ next } Q') \\
(P \text{ next } Q) \land (P' \subseteq P) & \Rightarrow (P' \text{ next } Q)
\end{align*}$$

RoboFlag Drill examples ($z = \text{defender pos'n}, y = \text{attacker height}$)

- Defenders never collide
  $$z_i < z_{i+1} \text{ next } z_i < z_{i+1}$$
- If attackers are far enough away, we won’t switch back and forth
  $$\forall i . \ y_i > 2\delta \land z_i + 2\delta < z_{i+1} \land \neg \text{switch}_{i,i+1} \text{ next } \neg \text{switch}_{i,i+1}$$
Stable

Definition: stable(P)
• Informal: once P becomes true, it remains true
• Formally: \( \text{stable}(P) \equiv P \text{ next } P \)
• Note: \( \text{stable}(P) \) does not mean that P is true for all (or even any) program executions

When do we use stable in a proof?
• Termination: \( \text{stable}([p]) \)
• Often combined with progress (Wed + W3)
  - Show that if we satisfy some conditions then we eventually get to a good set of states (and stay there)

Some useful results (will prove on the homework)
• \( \text{stable}(P) \land \text{stable}(Q) \implies \text{stable}(P \land Q) \)
  - Interpretation: if P is stable and Q is stable, then at the point that both of them are true, they will both remain true
• \( \text{stable}(P) \land \text{stable}(Q) \implies \text{stable}(P \lor Q) \)
  - Note: not true that \( \text{stable}(P) \lor \text{stable}(Q) \implies \text{stable}(P \lor Q) \)
Which of the following formulas are true?

- $\text{stable}(P) \land (P \subseteq Q) \implies \text{stable}(Q)$
- $\text{stable}(P) \land (Q \subseteq P) \implies \text{stable}(Q)$
- $\forall P : \text{stable}(\text{reachable}(P))$
- $(P \subseteq Q) \land \text{stable}(Q) \implies \text{reachable}(P) \subseteq Q$

Reachable(P) is the smallest stable set that includes P

- Reachable(P) = set of points that we can reach from states that satisfy predicate P
- Proof sketch (exercise: turn into a formal proof = sequence of implications/equivalents)
  - Let $Q = \text{reachable}(P)$. Clear that $P \subseteq Q$ and $\text{stable}(Q)$
  - Suppose $Q'$ is a smaller set ($Q' \subset Q$) with $P \subseteq Q'$ and $\text{stable}(Q')$
  - $Q' \subset Q \land \text{stable}(Q) \implies Q = \text{reachable}(P) \subset Q' \quad \therefore Q = Q'$
- Algorithm for finding reachable(P): start with P add neighbors until you stop growing
Examples: Properties for Average Consensus

Program \textit{AverageConsensus}
constant \(N\) \{number of agents\}
\(\mathcal{G}\) \{interconnection graph\}
var \(x\) : array of \(N\) numbers
assign
\[
\begin{align*}
\left( i, j : j \in \mathcal{N}_i : x[i] &:= \alpha x[i] + (1 - \alpha)x[j] \\
\| x[j] &:= \alpha x[j] + (1 - \alpha)x[i] \right)
\end{align*}
\]

What are some stable properties for this program? [assume \(\alpha = 1/2\)]
- stable\((x_i \leq x_i^0)\) ?
- __
- stable\((x_i + x_j \leq x_i^0 + x_j^0)\) ?
- __
- stable\((x_i \leq \max_i x_i^0)\) ?
- __
- stable\(((+i : 0 \leq i \leq N - 1 : x_i) \leq (+i : 0 \leq i \leq N - 1 : x_i^0))\) ?
- ___
Summary: Reasoning About Programs

Initial tools for reasoning about program properties

- UNITY approach: assume that any (enabled) command can be run at any time
- Hoare triple: show that all (enabled) actions satisfying a predicate $P$ will imply a predicate $Q$
- “Lift” Hoare triple to define $\text{next}$:
  \[ P \text{ next } Q \equiv (\forall a : a \in G : \{P\} a \{Q\}) \]
- Stability: $\text{stable}(P) \equiv P \text{ next } P$
- Wed: finish stability and introduce liveness properties