Goals:
  • Define safety properties, program invariants
  • New properties: next, stable, invariant

Reading:
  • P. Sivilotti, *Introduction to Distributed Algorithms*, Section 3.3
The ‘Next’ Relation

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 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)**

\[
(P \text{ next } Q) \land (Q \subseteq Q') \Rightarrow (P \text{ next } Q')
\]

\[
(P \text{ next } Q) \land (P' \subseteq P) \Rightarrow (P' \text{ next } Q)
\]

**RoboFlag Drill examples (z = defender pos’n, y = 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
  \[ \forall i . \ y_i > 2\delta \land z_i + 2\delta < z_{i+1} \land \neg\text{switch}_{i,i+1} \text{ next } \neg\text{switch}_{i,i+1} \]
The ‘Stable’ Property

Definition: $\text{stable}(P)$

- 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?

\[ \text{stable}(P) \land (P \subseteq Q) \Rightarrow \text{stable}(Q) \]

\[ \text{stable}(P) \land (Q \subseteq P) \Rightarrow \text{stable}(Q) \]

\[ \forall P : \text{stable} (\text{reachable}(P)) \]

\[ (P \subseteq Q) \land \text{stable}(Q) \Rightarrow \text{reachable}(P) \subseteq Q \]

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/equivals)
  - 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

There can be an edge from a vertex which is in Q and not in P to a vertex outside Q
Examples: Properties for Average Consensus

Program \textit{AverageConsensus}

constant \( N \) \{number of agents\}

\( G \) \{interconnection graph\}

var \( x \) : array of \( N \) numbers

assign \( \left[ i, j : j \in \mathcal{N}_i : x[i] := \alpha x[i] + (1 - \alpha) x[j] \right| \| x[j] := \alpha x[j] + (1 - \alpha) x[i]) \)

What are some stable properties for this program? [assume \( \alpha = 1/2 \)]

- \( \text{stable}(x_i \leq x_i^0) \)?
  - __

- \( \text{stable}(x_i + x_j \leq x_i^0 + x_j^0) \)?
  - __

- \( \text{stable}(x_i \leq \max_i x_i^0) \)?
  - __

- \( \text{stable}(\left[ +i : 0 \leq j \leq N - 1 : x_i \right] \leq \left[ +i : 0 \leq i \leq N - 1 : x_i^0 \right]) \)?
  - __

If time, add proof of the last property here?
The ‘Invariant’ Property

A predicate $P$ is *invariant* if it is always true

$$\text{invariant}(P) \equiv \text{initially}(P) \land \text{stable}(P)$$

- Invariants are a critical part of proofs; establish the key properties that a problem *always* satisfies
- Invariants are not unique; a program can have many invariants

Some examples of useful invariants

- Amount of memory required is less than $M$
- Values of a variable (e.g., address register) is in a given range

Proving properties about invariants comes down to evaluating Hoare triples

$$\text{initially}(P) \land (\forall a : a \in G : \{P\} \ a \ {P})$$

Example:

- For average consensus,

  $$\text{invariant}((+i : 0 \leq j \leq N - 1 : x_i) = (+i : 0 \leq i \leq N - 1 : x_i^0))$$

Reachability and invariants

- Recall that reachable($P$) is the smallest stable set of vertices that includes $P$. Hence:

  $$\text{invariant}([\text{reachable(init)}]) \quad \text{invariant}(I) \implies \text{reachable(init)} \subseteq I$$
Which of the following formulas are true?

\[ \text{invariant}(P) \land (P \subseteq Q) \Rightarrow \text{invariant}(Q) \]

\[ \text{invariant}(P) \land \text{invariant}(Q) \Rightarrow \text{invariant}(P \cap Q) \]

\[ \text{invariant}(P) \lor \text{invariant}(Q) \Rightarrow \text{invariant}(P \cup Q) \]
Example: FindMax

Let \( M = ( \max x : 0 \leq x < N : A[x] ) \). Prove that \( r \leq M \) is an invariant

1. initially \( (r \leq M) \)
   \[
   r = A[0] \\
   \Rightarrow \{ A[0] \leq M \} \\
   r \leq M
   \]

2. stable \( (r \leq M) \)
   
<table>
<thead>
<tr>
<th>Program</th>
<th>( \text{FindMax} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>( A : \text{array} \ 0..N-1 \text{ of int} ),</td>
</tr>
<tr>
<td>( r : \text{int} )</td>
<td></td>
</tr>
<tr>
<td>initially</td>
<td>( r = A[0] )</td>
</tr>
<tr>
<td>assign</td>
<td>(</td>
</tr>
</tbody>
</table>

   \[
   \begin{align*}
   \text{stable.}(r \leq M) & \equiv \\
   & \frac{(r \leq M) \text{ next } (r \leq M)}{}
   \\
   & \equiv \\
   & \frac{\{ (\forall a :: \{ r \leq M \} \ a \{ r \leq M \} ) \}}{}
   \\
   & \equiv \\
   & \frac{\{ \text{definition of program} \}}{}
   \\
   & \equiv \\
   & \frac{(\forall x : 0 \leq x < N : \{ r \leq M \} \ r := \max(r, A[x]) \{ r \leq M \} )}{\{ \text{assignment axiom} \}}
   \\
   & \equiv \\
   & \frac{(\forall x : 0 < x < N : \ r \leq M \Rightarrow \max(r, A[x]) \leq M)}{\{ x \leq \max(x, y) \}}
   \\
   & \equiv \\
   & \frac{(\forall x : 0 \leq x < N : \ r \leq M \Rightarrow r \leq M )}{\{ \text{predicate calculus} \}}
   \\
   & \text{true}
   \end{align*}
   \]
Example: RoboFlag Drill

| Red$(i)$    | Initial                           | $x_i \in [a, b] \land y_i > c$ |
|            | Commands                          |                                  |
|            | $y_i > \delta$ : $y'_i = y_i - \delta$ |                                  |
|            | $y_i \leq \delta$ : $x'_i \in [a, b] \land y_i > c$ |                                  |

$P_{\text{Red}}(n) = +\sum_{i=1}^{n} \text{Red}(i)$

| Blue$(i)$   | Initial                           | $z_i \in [a, b] \land z_i < z_{i+1}$ |
|            | Commands                          |                                  |
|            | $z_i < x_{\alpha(i)} \land z_i < z_{i+1} - \delta$ : $z'_i = z_i + \delta$ |                                  |
|            | $z_i > x_{\alpha(i)} \land z_i > z_{i-1} + \delta$ : $z'_i = z_i - \delta$ |                                  |

$P_{\text{Blue}}(n) = +\sum_{i=1}^{n} \text{Blue}(i)$
RoboFlag Control Protocol

\[ r(i, j) = \begin{cases} 
1 & \text{if } y_{\alpha(j)} < |z_i - x_{\alpha(j)}| \\
0 & \text{otherwise} 
\end{cases} \]

\[ \text{switch}(i, j) = r(i, j) + r(j, i) < r(i, i) + r(j, j) \]
\[ \lor \quad (r(i, j) + r(j, i) = r(i, i) + r(j, j) \land x_{\alpha(i)} > x_{\alpha(j)}) \]

| Proto(i) \n| Initial | \n| \n| \n| Commands | \n| \n| \n| \n| \n| switch(i, i + 1) : \alpha(i)' = \alpha(i + 1) \n\alpha(i + 1)' = \alpha(i) | \n
\[ P_{\text{Proto}}(n) = + \sum_{i=1}^{n-1} \text{Proto}(i) \]
Properties for RoboFlag program

Safety (Defenders do not collide)
\[ z_i < z_{i+1} \quad \text{next } z_i < z_{i+1} \]

Stability (switch predicate stays false)
\[ \forall i \cdot y_i > 2\delta \land z_i + 2\delta < z_{i+1} \land \neg \text{switch}_{i,i+1} \quad \text{next } \neg \text{switch}_{i,i+1} \]

Robots are "far enough" apart.

Progress (we eventually reach a fixed point)
- Let \( \rho \) be the number of blue robots that are too far away to reach their red robots
- Let \( \beta \) be the total number of conflicts in the current assignment
- Define the metric that captures “energy” of current state (\( V = 0 \) is desired)

\[
V = \left( \binom{n}{2} + 1 \right) \rho + \beta = \sum_{i=1}^{n} r(i,i) \quad \beta = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma(i,j) \quad \text{where } \gamma(i,j) = \begin{cases} 1 & \text{if } x_{\alpha(i)} > x_{\alpha(j)} \\ 0 & \text{otherwise} \end{cases}
\]

- Can show that \( V \) always decreases whenever a switch occurs
\[
\forall i \cdot z_i + 2\delta \delta m < z_{i+1} \land \exists j \cdot \text{switch}_{j,j+1} \land V = m \quad \text{next } V < m
\]
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 $\text{next}$:
  \[(\forall a : a \in G : \{P\} a \{Q\})\]
- Stability: $\text{stable}(P) \equiv P \text{ next } P$
- Invariants: $\text{invariant}(P) \equiv \text{initially}(P) \land \text{stable}(P)$

Hoare triple: $\{P\} a \{Q\}$