Caltech

CS/IDS 142: Lecture 2.1
Reasoning About Programs

Richard M. Murray

7 October 2019

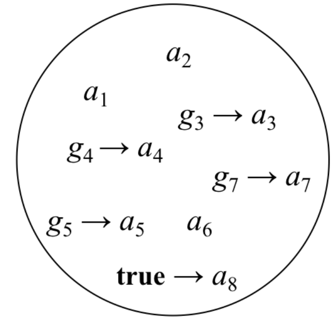
Goals:

- 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

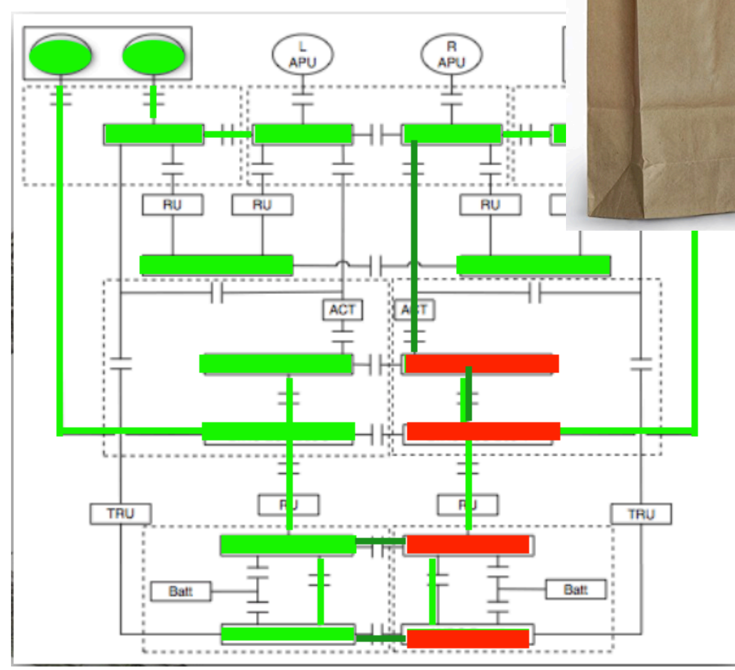


UNITY model provides (seemingly) simple description of programs

- Program = variables + actions [assignments] (that's it!)
- Guarded assignment (g → a) allows modeling of finite state automata
- Distributed programs captured by nondeterministic execution model
- Termination = reaching a *fixed point* (variables remain constant)

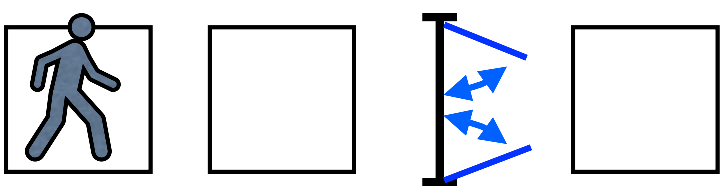
Next: how to we *prove* that specifications are satisfied?

- 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)

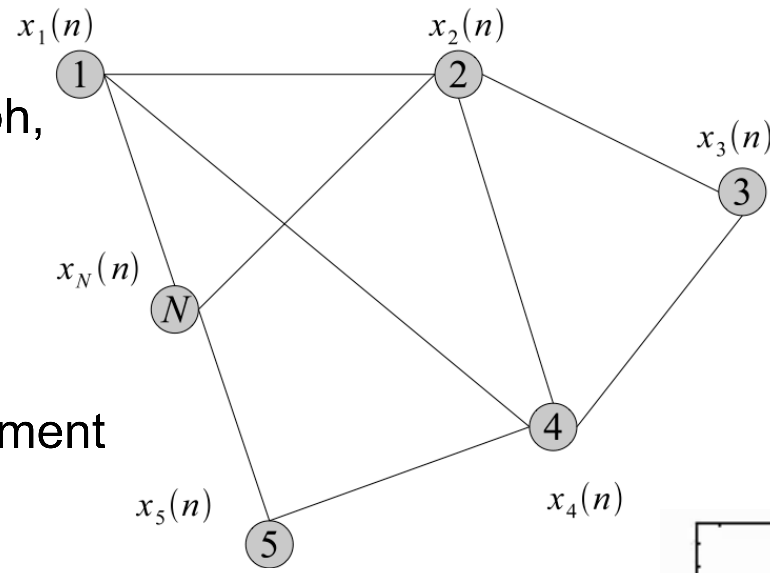
This week: reasoning about program behavior and safety properties (invariants)



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?



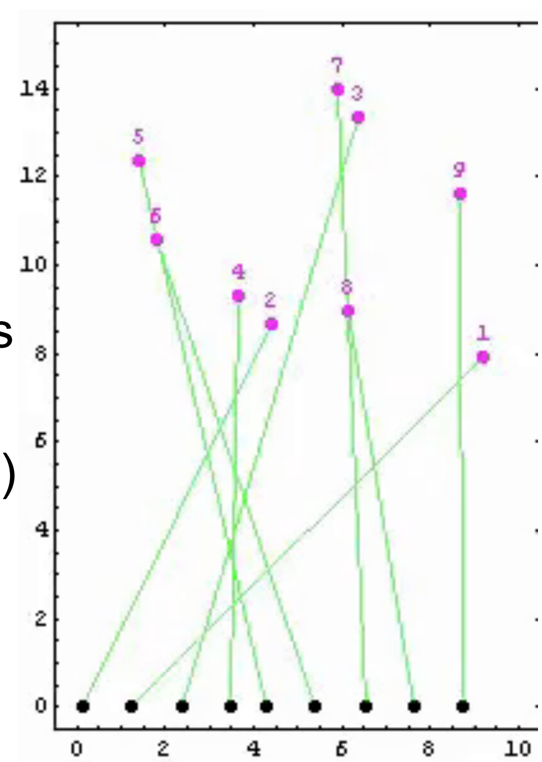
Consensus is reached when:

$$x_i(n) = \frac{1}{N}\sum_{j=1}^{N} x_j(0),\ \forall i$$

n: Time index
N: number of nodes

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")



Example: Average Consensus

Problem setup

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

Program $\quad AverageConsensus$

constant $\quad N \quad \{number\ of\ agents\}$

$\qquad \mathcal{G} \quad \{interconnection\ graph\}$

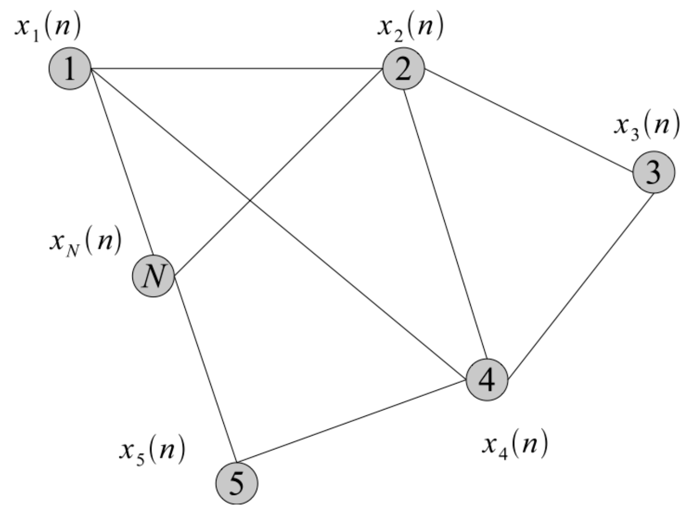
$\qquad \alpha : 0 < \alpha < 1$

var $\quad x$: array of N numbers

assign

neighbors of *i*

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



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

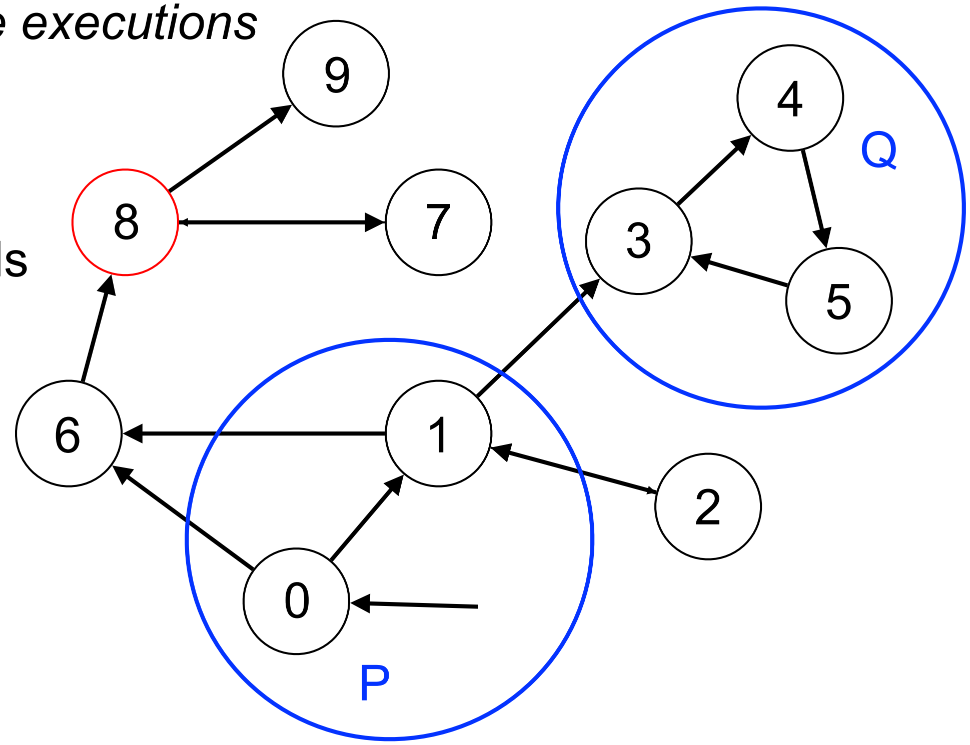
Why do we need a "proof"?

- 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 properties holds by looking at how the transitions map to the (sets of states representing) properties

Reasoning about properties using formulas

$$
\begin{array}{ll}
 & \textbf{stable}.(r \le M) \\
\equiv & \underline{\qquad\qquad\qquad\qquad\qquad} \\
 & (r < M)\ \ \textbf{next}\ \ (r < M) \\
\equiv & \underline{\qquad\qquad\qquad\qquad\qquad} \\
 & (\forall a :: \ \{r \le M\} \quad a \quad \{r \le M\}\ ) \\
\equiv & \quad \{\ \text{definition of program}\ \ \} \\
 & (\forall x\ :\ 0 \le x < N\ :\ \ \{r \le M\} \quad r := max(r, A[x]) \quad \{r \le M\}\ ) \\
\equiv & \quad \{\ \text{assignment axiom}\ \ \} \\
 & (\forall x\ :\ 0 \le x < N\ :\ \ \underline{\qquad\qquad\qquad\qquad\qquad}\ )
\end{array}
$$

Example properties

transient(P)

stable(Q)

¬(P **next** Q)

Can also combine representations

$$(P \subseteq Q) \wedge \text{stable}(Q) \quad \Rightarrow \quad \text{reachable}(P) \subseteq Q$$

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

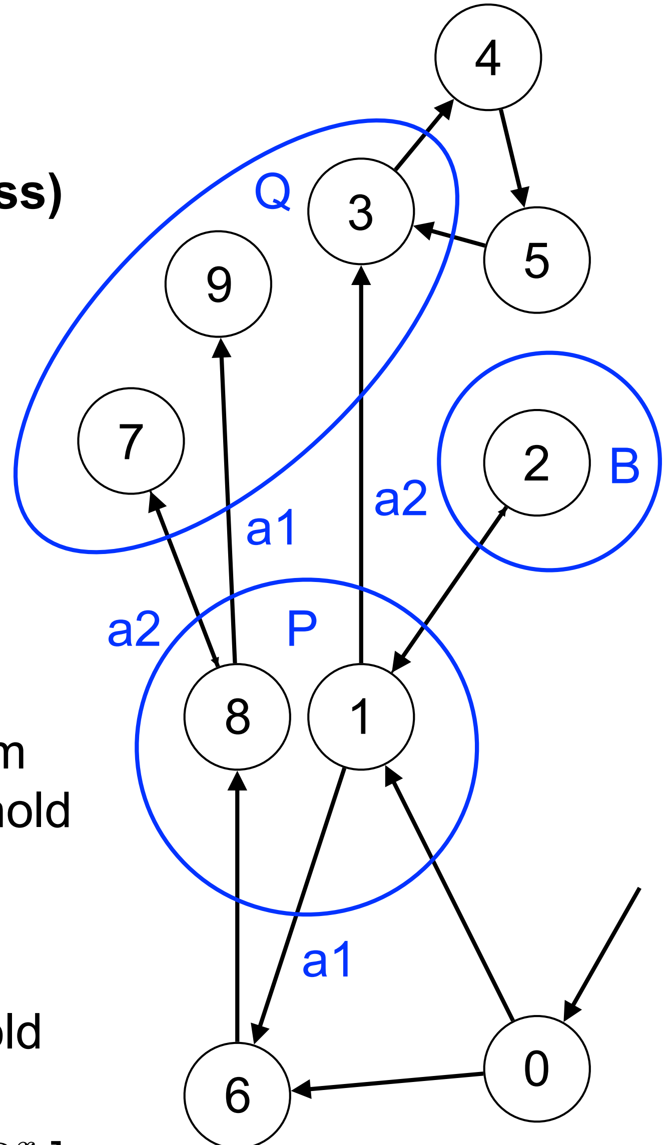
Notation: Hoare triple - {P} *a* {Q}

- 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 $[P \implies Q_E^x]$

predicate Q with x replaced by E



Reasoning About Guarded Actions

Hoare triple with a guarded action: {P} g → x := E {Q}

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

predicate Q with x replaced by E

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

Example $\{x > y = 7\} \quad x > y \longrightarrow x, y := y, x \quad \{x > 3\}$

$$(x > y = 7 \wedge x > y \Rightarrow y > 3) \wedge (x > y = 7 \wedge \neg(x > y) \Rightarrow x > 3)$$

$\Leftarrow$ { antecedent strengthening of $\Rightarrow$: $[(X \Rightarrow Z) \Rightarrow (X \wedge Y \Rightarrow Z)]$ }

$$(y = 7 \Rightarrow y > 3) \wedge (x > y = 7 \wedge \neg(x > y) \Rightarrow x > 3)$$

$\equiv$ { $7 > 3$ }

$$x > y = 7 \wedge \neg(x > y) \Rightarrow x > 3$$

$\equiv$ { definition of $\neg$ }

$$x > y = 7 \wedge x \leq y \Rightarrow x > 3$$

$\Leftarrow$

$$x > y \wedge x \leq y \Rightarrow x > 3$$

$\equiv$

$$\textbf{false} \Rightarrow x > 3$$

$\equiv$ { property of $\Rightarrow$: $[\textbf{false} \Rightarrow X \equiv \textbf{true}]$ }

true

Recall: $x > y = 7 \quad \equiv \quad x > y \wedge y = 7$

$$x > y \wedge y = 7 \wedge x > y \Longrightarrow y > 3$$

$$(y = 7) \wedge (x > y) \Longrightarrow (y > 3)$$

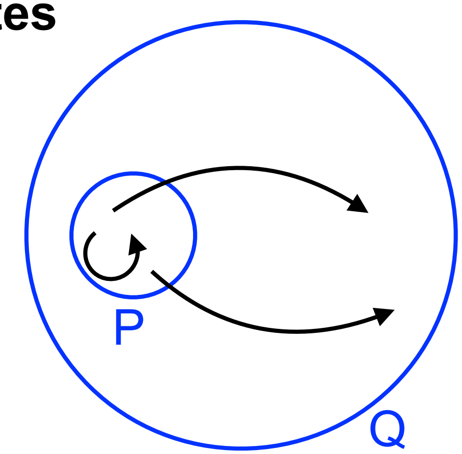
$$X \qquad Y \qquad Z$$

The Next Relation: P next Q

Use to reason about properties of a program G as it executes

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

- 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)

$$(P \text{ next } Q) \wedge (Q \subseteq Q') \Rightarrow (P \text{ next } Q')$$

$$(P \text{ next } Q) \wedge (P' \subseteq P) \Rightarrow (P' \text{ next } Q)$$

Note: $[P \implies Q]$ and $P \subset Q$ capture same concept

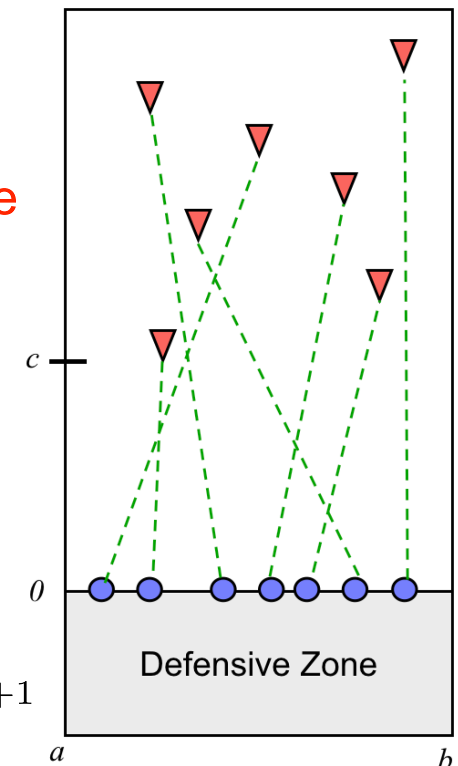
RoboFlag Drill examples (z = defender pos'n, y = attacker height)

- Defenders never collide

$$z_i < z_{i+1} \textbf{ next } z_i < z_{i+1}$$

- If attackers are far enough away, we won't switch back and forth

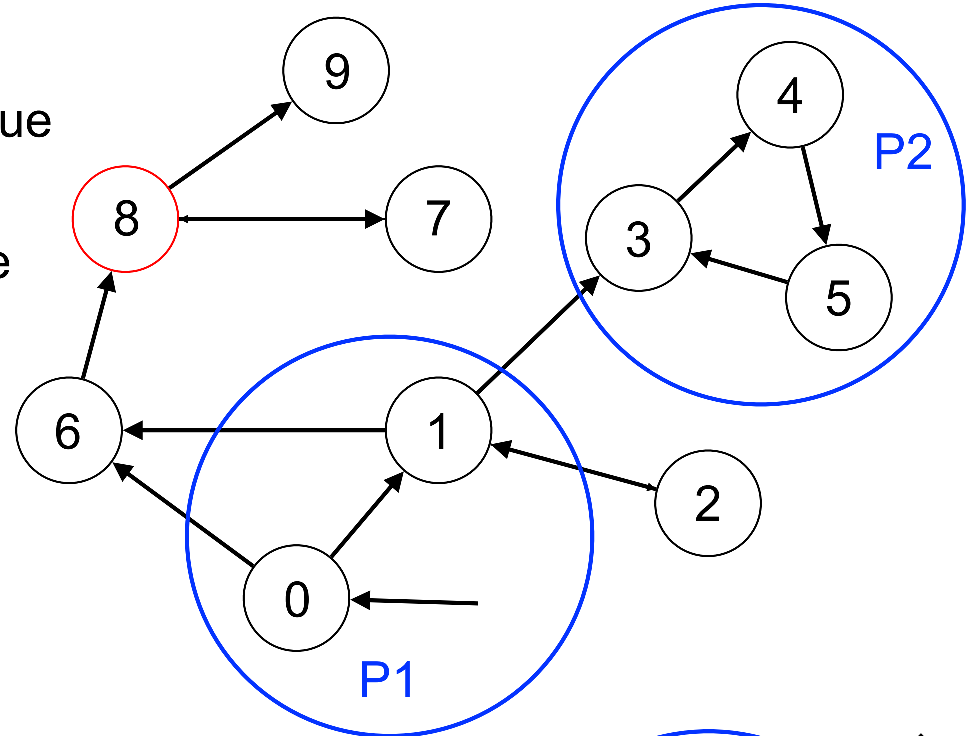
$$\forall i \,.\, y_i > 2\delta \wedge z_i + 2\delta < z_{i+1} \wedge \neg switch_{i,i+1} \textbf{ next } \neg switch_{i,i+1}$$



Stable

Definition: stable(P)

- Informal: once P becomes true, it remains true
- Formally: **stable**(P) ≡ P **next** P
- Note: **stable**(P) does not mean that P is true for all (or even any) program executions



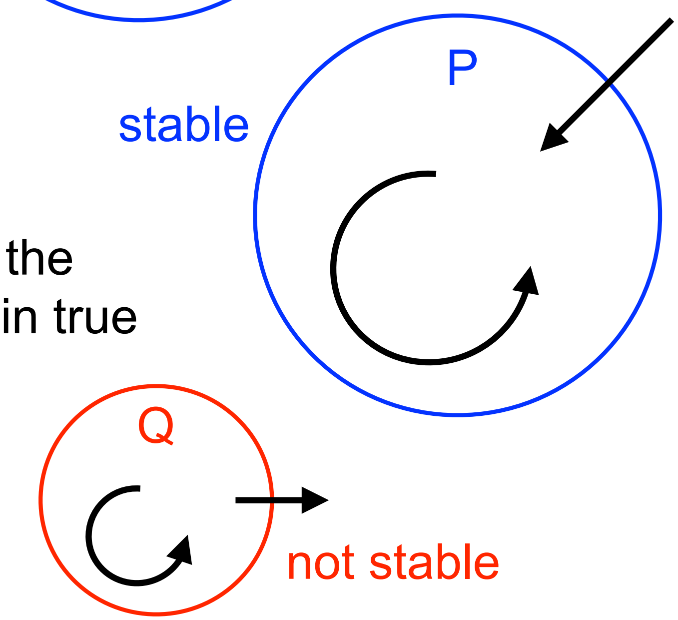
When do we use stable in a proof?

- Termination: **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)

- $\mathbf{stable}(P) \wedge \mathbf{stable}(Q) \implies \mathbf{stable}(P \wedge Q)$
  - Interpretation: if P is stable and Q is stable, then at the point that both of them are true, they will both remain true
- $\mathbf{stable}(P) \wedge \mathbf{stable}(Q) \implies \mathbf{stable}(P \vee Q)$
  - Note: *not* true that

$$\mathbf{stable}(P) \vee \mathbf{stable}(Q) \implies \mathbf{stable}(P \vee Q)$$



Which of the following formulas are true?

$\textbf{stable}(P) \wedge (P \subseteq Q) \implies \textbf{stable}(Q)$ _____

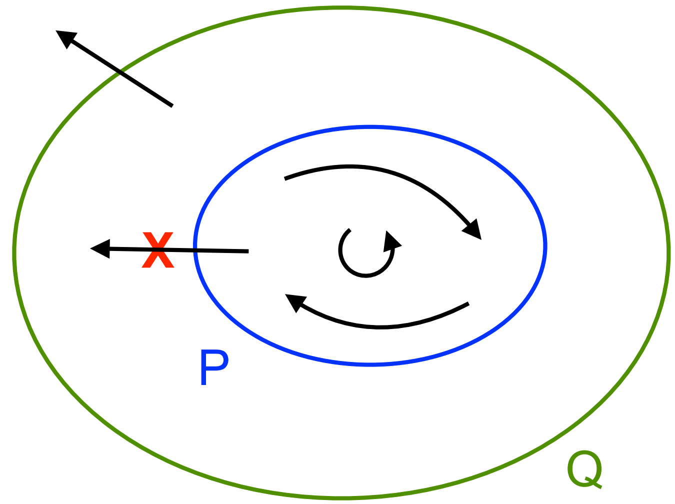
There can be an edge from a vertex which is in Q and not in P to a vertex outside Q

$\textbf{stable}(P) \wedge (Q \subseteq P) \implies \textbf{stable}(Q)$ _____

$\forall P : \textbf{stable}(\text{reachable}(P))$ _____

$(P \subseteq Q) \wedge \textbf{stable}(Q) \implies \text{reachable}(P) \subset Q$ _____

_____



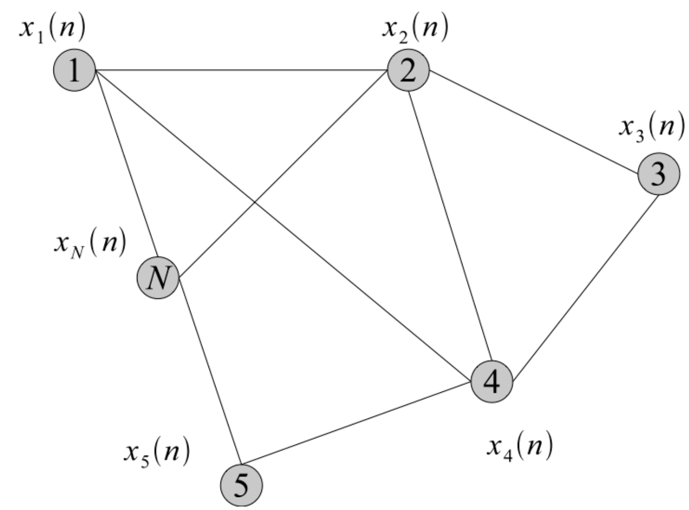
Reachable(P) is the smallest stable set that includes P

- 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 = reachable(P). Clear that $P \subseteq Q$ and **stable**(Q)
  - Suppose Q' is a smaller set ($Q' \subset Q$) with $P \subseteq Q'$ and stable(Q')
  - $Q' \subset Q \wedge \textbf{stable}(Q) \implies Q = \text{reachable}(P) \subset Q' \qquad \therefore Q = Q'$
- Algorithm for finding reachable(P): start with P add neighbors until you stop growing

Examples: Properties for Average Consensus

$$
\begin{array}{ll}
\textbf{Program} & \mathit{AverageConsensus} \\
\textbf{constant} & N \quad \{\mathit{number\ of\ agents}\} \\
 & \mathcal{G} \quad \{\mathit{interconnection\ graph}\} \\
\textbf{var} & x : \text{array of } N \text{ numbers} \\
\textbf{assign} &
\end{array}
$$

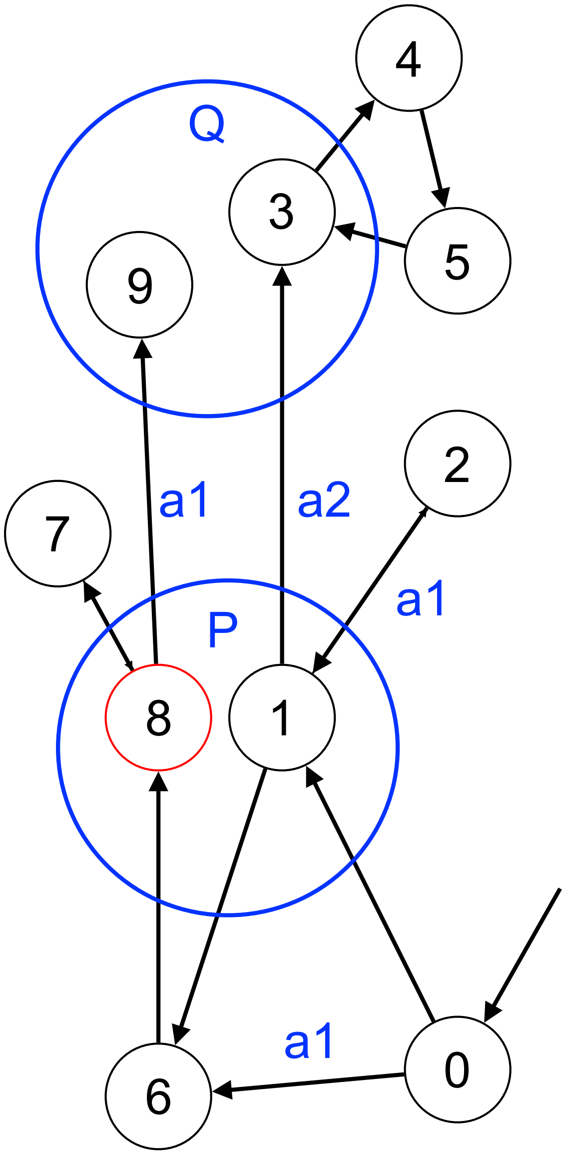
$$\big([\!] i, j : j \in \mathcal{N}_i : x[i] := \alpha x[i] + (1-\alpha) x[j] \quad || \; x[j] := \alpha x[j] + (1-\alpha) x[i]] \big)$$



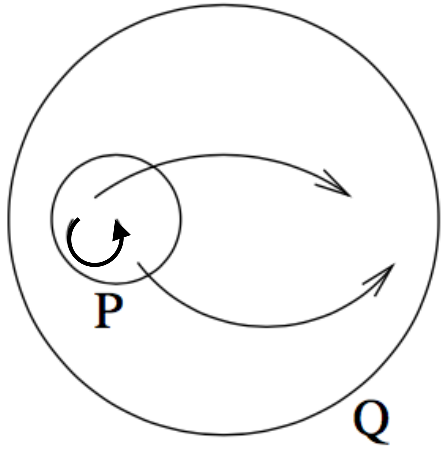
What are some stable properties for this program? [assume α = 1/2]

- $\textbf{stable}(x_i \leq x_i^0)$?
  - ___
- $\textbf{stable}(x_i + x_j \leq x_i^0 + x_j^0)$?
  - ___
- $\textbf{stable}(x_i \leq \max_i x_i^0)$?
  - ___
- $\textbf{stable}\big((+i : 0 \leq i \leq N-1 : x_i) \leq (+i : 0 \leq i \leq N-1 : x_i^0)\big)$?
  - ___

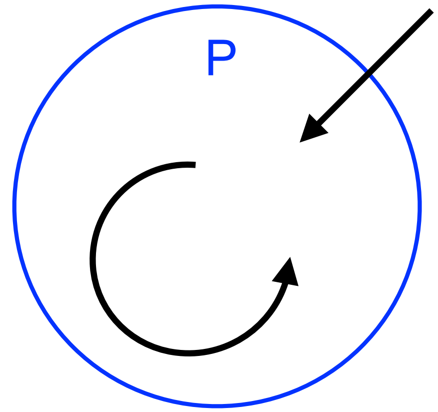
Summary: Reasoning About Programs



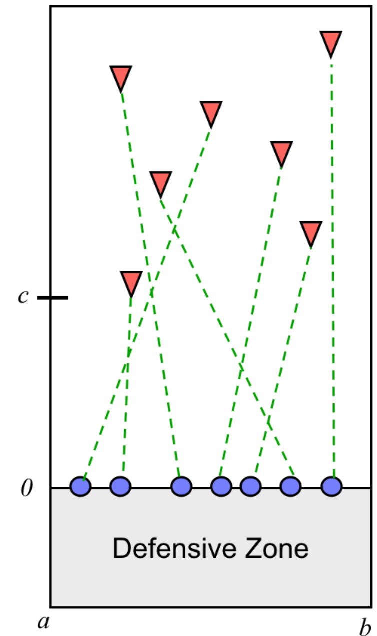
Hoare triple: {P} a {Q}



P **next** Q



stable(P)



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 **next**:

$$P \textbf{ next } Q \quad \equiv \quad (\forall a : a \in G : \{P\}\, a\, \{Q\})$$

- Stability: **stable**(P) ≡ P **next** P
- Wed: finish stability and introduce liveness properties