Assume processes $P_1, \ldots, P_k$.

**Global State** of the system consists of $loc[1], \ldots, loc[k]; x_1, \ldots, x_\ell$ where $x_1, \ldots, x_\ell$ denote the (values of) the shared variables.

System is represented by a *labelled graph*.

Each node denotes a global by the state and is labelled by it.

Each edge denotes a transition and is labelled by the index of the process in charge of it.

In our example, $k = 2$ (A and B), $\ell = 1$ (the *turn* variable).

We denote A’s transitions in *red* and B’s in *blue*. 
Generating Reachability Graph

1. Start with *initial state*; mark it *pending*

2. While there are *pending* states:
   (a) Choose a *pending* state;
   (b) compute all its outgoing transitions;
   (c) If *new* states are discovered, mark them *pending*;
   (d) Connect edges and label accordingly.
- States enclosed in red (blue) dashed lines are those for which turn is A (B)

- Note Self loops
Suppose we want to prove that

All states are $p$-states

where $p$ is some assertion

Then all we have to do is verify that

every reachable state is a $p$-state

In our example, the *mutual exclusion* property is

$p = \neg(A.loc = 4 \land B.loc = 4)$ which holds for all reachable states
Suppose we want to verify that

Every $p$-state is \textit{eventually} succeeded by a $q$-state
Response is violated with there is some infinite fair path \( \pi = s_0, \ldots \) in the graph that:

1. Starts with the initial node \( (s_0) \)
2. Reaches a \( p \land \neg q \)-state (say at \( s_i \))
3. After that, never reaches a \( q \)-state (for every \( j \geq i \), \( s_j \) is not a \( q \)-state)

An infinite path \( \pi \) defines a **Strongly connected component**, \( \text{inf}(\pi) \), by the set of nodes it visits infinite many times

Thus,

\[
\pi = \underbrace{s_0, \ldots, s_k}_{\text{initial segment}}, \underbrace{s_{k+1}, \ldots}_{\text{nodes appearing i.m. times}}
\]
An SCC is *fair* if it contains an $i$-edge for each process $i$.

**Theorem:** A path $\pi$ is fair iff $\inf(\pi)$ is a *fair SCC*

- To establish the response property we restrict to the subgraph that consists of all $\neg q$ nodes reachable from $p$-nodes, and show it contains no fair SCCs.
- A fair SCC in this graph implies the existence of a *fair computation* that violates the response property.
Algorithm

- Remove from graph all but nodes reachable from $p$-nodes, and violating $q$

In practice

1. Remove all $q$-nodes and edges leading to/from them
2. Remove every $\neg p$-node (and incoming/outgoing edges) for every node that is reachable from the initial state by $\neg p$-path (using $\text{dfs}$, e.g.)
When A wishes to, she can take dog out:

\[ p = (\text{loc}[A] = 1) \] and \[ q = (\text{loc}[A] = 4) \]

Brown: \( \neg p \)-states reachable from the initial state

Green: \( q \)-states
After Removal

Diagram showing a network of nodes and connections.
Decomposition into MSCCs

Assume the graph has only $\neg q$-nodes, leading from reachable from $p$-states. We need to show that graph has no fair SCCs.

- A subgraph is maximal strongly connected component (MSCS) is it is
  - strongly connected
  - does not properly contain any SCC
- There is an algorithm (of Tarjan) that decomposes a graph into its MSCCs $S_1, \ldots, S_m$ such that $S_i$ leads into $S_j$ only if $i < j$ (thus imposing a dag structure)
- Some $S_i$’s may be singular – single nodes w/o self loops (not fair SCCs)
Algorithm for Response

(Assumes subgraph as above)
separate graph into its MSCCs
while graph not empty do
  Find a terminal MSCC \( C \)
  if \( C \) is fair then output failure; halt
  else remove \( C \) from graph
end (while)
output success
Example: Step 1

Unfair MSCC – can be removed
Example: Step 2

Unfair MSCC – can be removed
Example: Step 3

Singleton MSCC – can be removed
Example: Step 4

Unfair MSCC – can be removed
Example: Step 5

Unfair MSCC – can be removed
Unfair MSCC – can be removed
Unfair MSCC – can be removed
Singleton MSCCs – can be removed
Singleton MSCCs – can be removed
Example: Step 10/10a

Unfair MSCCs – can be removed
Example: Step 11

Singleton MSCC – can be removed
Example: Step 12

Unfair MSCC – can be removed
Example: Step 13

Singleton MSCC – can be removed
Example: Step 14

Singleton MSCC – can be removed
Unfair MSCC – can be removed
Note on Ranking

The algorithm also implies a *ranking* on states:

- A state can be ranked by the step it was removed, thus, a rank measures, in a sense, the distance from the goal

- This guarantees that for every state ranked $r$:
  1. All successor states are ranked $\leq r$
  2. Eventually a state whose rank is $< r$ is reached (using a *helpful*, or fair, transition)

- The *well founded domain* is the natural numbers $\mathbb{N}$ with the usual $<$ ordering, which guarantees there are no infinitely decreasing sequences $a_0 < a_1 < \ldots$
Your Assignment (I)

Is to write a C++ or Java program that checks for safety and reponse of a finite-state program involving processes 1 and 2. The program should:

- Generate and print the program graph (nodes and labelled edges) of the program (states should be generated until exhausted or safety violation is obtained)
- Generate and print graph that includes only \( q \)-states that are reachable from \( p \)-states
- Perform MSCC removal until failure or empty graph
- Print every removed MSCC
- If failure, print offending MSCC
Your Assignment (II)

- Run your program on the algorithm we saw in class
- Try to flip the two assignment statements at locations 1 and 2 and then try to prove safety and response
- Try to keep your program modular so that you can use it for other algorithms (which we may do later on)

Extra Credit.
1. Design another solution to the problem (e.g., that does not use shared variables)
2. Verify your solution using the program
• Assignment is due February 24
• Alex will post information as to how to hand it in
• Alex will be available for help; if necessary, he will hold several office hours
• You can contact him at sasha1979@yahoo.com
General Rules

- You may work in groups of up to four students.
- If you do work in a group, your assignment has to include a statement as to who did what.
- The larger the group, the better the assignment should be.
- If you do use outside sources, you have to reference them.
- You are not to use someone else’s program.
- When in doubt, ask!