$t \in \text{Nat}$.

We were thinking of denoting the steps of the its system $J(t) : 3 \rightarrow 8^3$

$t : \text{Nat}$

If we were going to allow $J(0) = 3$, then the property $\forall t : J(t) \neq 3$ would not have been true!

If we fixed on $J(t) = 0$ then the property $J(t) \neq 3$ is true of the system.

Then $k$-induction achieves the proof w/o forcing you to specify the Reach state space.
Here \( \sin = \frac{310}{2} \), 46°

\( \sin = \frac{30}{2468} \) 

What's the \( k \) for which the system is \( k \)-inductive?

Has to relate to the diameter of the graph!

Diameter = longest shortest path

\( k \) = diameter works!

Shorter may work

The \( k = 3 \) works

Diameter is 4.
Floyd's logic rule. (also Hoare's)

We are working with wlp

\[ \{ \text{true} \} \text{ while}(I) \Rightarrow \{P \} \]

for any \( p \).

\[ \{ p \left[ E/x \right] \} \ x := E \Rightarrow \{ p \} \]

\[ \text{wlp} \ (x := E, p) = p \left[ E/x \right] \]

**Strongest post condition**

\[ \{ p \} \ x := E \Rightarrow \{ \text{fold: } P \left[ \text{old}/x \right] \land x = E \left[ \text{old}/x \right] \} \]

where \( \text{old} \in B \).

\[ \text{sp} \ (x := E, p) = \]

\[ \text{wlp} \ (x := x + 1, x = 27) = x + 1 = 27 \]

\[ \text{sp} \ (x := x + 1, x = 26) = \exists d : (x = 26) \left[ \frac{d}{x} \right] \land x = (x + 1) \left[ \frac{d}{x} \right] \]
Most general start

SP:

P \rightarrow C \rightarrow \text{Tightest end.}

Program March:

\[ \text{[Diagram showing transition from general start to tight end]} \]
WIN:

\[ \text{WIN}(\sigma, Q) = \bigwedge \text{mlp}(\lambda, Q) \quad \lambda \in \sigma^* \]

usually \( Q = \text{desired property} \).

\[ \text{sin}(\sigma, p) = \bigvee_{\lambda \in \sigma^*} \text{sp}(\lambda, p). \]

usually \( p = \text{init.} \).
Hoare logic with arrays:

\[ \{ P [ E_2 \leftarrow A(E_1)] \} \ A(E_1) := E_2 \ \{ P \} \]

Does not work because changing \( A[x] \) may affect \( A[y] \) etc.

Instead, treat

\[ A[E_1] := E_2 \text{ as} \]

\[ A := A \uparrow E_1 \leftarrow E_2 \]

Then we have

\[ \{ P[A[E_1 \leftarrow E_2]/A] \} \ A[E_1] := E_2 \ \{ P \} \]

One form

\[ \vdash A \uparrow E_1 \leftarrow E_2 \ \{ E_1 \} = E_2 \]

Satisfies these axioms.

\[ E_1 \neq E_2 \Rightarrow \vdash A \uparrow E_1 \leftarrow E_2 \ \{ E_2 \} = A \uparrow E_3 \]
\textbf{Example 3}
\begin{align*}
R & := x \cdot 3 \\
Q & := 0 \\
\text{while } Y \leq R \text{ do} & \\
R & := R - Y \\
Q & := Q + \lambda \\
\{ R < Y \land X = R + Y Q \} & \\
\{ (E_1 \leq V) \land V \leq E_2 \} & \lambda : P \vdash C \{ P[V+1/V] \} \\
\{ P[E_1/V] \} & \text{for } V := E_1 \text{ until } E_2 \text{ do } C \{ P[E_2+1/V] \} \\
\text{no var in } E_1, E_2 \text{ or } V \text{ are changed in } C. \\
\{ P[E_1/V] \land E_1 \leq E_2 \} & \\
V := E_1 & \\
\begin{cases}
T & \text{if } V \leq E_2 \\
F & \text{if } V > E_2 \\
\end{cases} & \\
V := V + 1 & \\
C & \\
\{ P[V+1/V] \} \\
\text{for axiom:} & \{ E_1 \leq E_2 \land E_2 < E_1 \} \\
\text{for } V := E_1 \text{ until } E_2 \text{ do } C \\
\text{See Wickerson's and Boulton's rules for } \text{MSCC} \text{.}
for $i := 2 \text{ to } N$ do
\[
X[i] := X[i] + X[i-1] \\
X[N] = \sum_{i=1}^{N} X[i]
\]

\[
\begin{align*}
\{2 \leq i \leq N\} \\
X[i] := X[i] + X[i-1] \\
X[i-1] = \sum_{j=1}^{i-1} x_0[j] \\
X[i] = x_0[i] & \quad j=1
\end{align*}
\]

\[
\left( x[i] = \sum_{j=1}^{i} x_0[j] \right) \left[ \frac{x[i] + x[i-1]}{x[i]} \right]
\]

\[
x[i] + x[i-1] = \sum_{j=1}^{i} x_0[j] \\
x[i] + x[i-1] = x_0[i] + \sum_{j=1}^{i-1} x_0[j]
\]
\[ 2 \leq i \leq N \wedge X[i-1] = \text{Sum}(x_0, 1, i-1) \wedge X[i] = x_0[i] \]

\[ X[i] := X[i] + X[i-1] \]

\[ \exists 2 \leq i \leq N \wedge X[i] = \text{Sum}(x_0, 1, i) \wedge i+1 \leq j \leq N : X[j] = x_0[j] \]

\[ \exists X[N] = \text{Sum}(x_0, 1, N) \wedge \text{True} \]

\[ i \text{ is } \]

\[ 2 \leq i \leq N \wedge X[i] + X[i-1] = \text{Sum}(x_0, 1, i) \wedge i+1 \leq j \leq N : X[j] = x_0[j] \]

\[ \text{Sum}(x_0, 1, i-1) + x_0[i] \]

follows!
The Repeat Rule:

\[ \text{repeat } C \text{ until } S \in QS \]

"Invariant" for C set up based on Q, the "output" of Repeat.
\[ n = \text{len}(A) \]

repeat

\[ s = F \]

for \( i = 1 \ldots n-1 \) do

\[ \text{if } A[i-1] > A[i] \]

\[ \text{Swap } (A[i-1], A[i]) \]

\[ s = T \]

\[ n = n - 1 \]

until \( s \)

---

first attempt:

\[ \{ T \} \xrightarrow{\text{sort}} \{ T \} \]

\[ \{ A[1 \ldots n] \} \xrightarrow{\text{array}} \{ \forall 2 \leq j \leq n: A[j-1] \leq A[j] \} \]

is this through?
Floyd proof of B.S.

\[ n_0 = \text{len}(A) \]

\[ \exists \{ \]

\[ n := \text{len}(A) \]

\[ s := F \]

\[ i := 1 \]

\[ i \leq m-1 \]

\[ A[i-1] \geq A[i] \]

\[ \text{if} \]

\[ A[i-1] > A[i] \]

\[ \text{swap}(A[i-1], A[i]) \]

\[ i := i + 1 \]

\[ s := T \]

\[ 2 \leq j \leq m : \]

\[ A[i-1,j] \leq A[j] \]

\[ A = \text{perm}(a) \]

\[ j \]

\[ s := T \]

\[ \forall n < j \leq n_0 : \]

\[ A[j] \text{ vs } A[1:n] \]
### Exercises

21.1. Write a Promela model for a two-process mutual exclusion algorithm that is based on Dekker's solution (many operating systems books contain a description of this algorithm). State one safety and one liveness property, express these using **never** automata, and verify these properties in turn.

21.2. Repeat Exercise 21.1, but following Peterson's mutual exclusion algorithm, again described in most books on operating systems.

21.3. Modify the dining philosophers protocol to eliminate the liveness bug described in Section 21.4.1. Also make sure that your solution is deadlock free.

21.4. Even sequential programs are easily gotten wrong. This exercise shows that even though transformational programs are most often required to be analyzed through Floyd-Hoare-Dijkstra's logic [40, 55, 37], they can sometimes be easily verified through **finite-state** methods as well.

In an old textbook on Algorithms (name withheld), the following Bubble sort program is offered, accompanied by the assertion, "It takes a moment’s reflection to convince oneself first that this works at all, second that the running time is quadratic."

```plaintext
procedure bubblesort;
    var j, t: integer;
begin
    repeat
        t:=a[1];
        for j:=2 to N do
            if a[j-1]>a[j] then
                begin t:=a[j-1]; a[j-1]:=a[j]; a[j]:=t end
        until t=a[1];
    end;
```

- Examine the above program and find a bug such that the program can exit with an **unsorted** array!\(^6\)
- Next, run the following Promela model encoding this Bubblesort program and find the bug in it.

```plaintext
#define Size 5
#define a MinIndx 1
#define a MaxIndx (Size-1)
```

\(^6\) It may be that the author meant to write a few side conditions, but going by exactly what he wrote, I assert that there is a bug in the program.
21.5. Modify the Bubblesort algorithm of Exercise 21.4, recode in
Promela, and prove that it is now correct.