One concurrent program: three attempts at its formal verification

Cliff Jones
Newcastle University

ETH *zürich* (by Zoom)

Funding:

LEVERHULME TRUST

Context: (coarse) timeline

- von Neuman, Post, Turing
- Bob Floyd [Flo67], Tony Hoare [Hoa69]
- Ed Ashcroft & Zohar Manna [AM71]
- Ashcroft [Ash75]
- Susan Owicki [Owi75]

sequential programs

concurrency

Today!

- original aim: history context + go through 3 proofs
- what's achievable on slides (in 25 minutes)
  - more on history (discuss "linear account?")
  - main conclusions from the (small type) proofs
  - reverse historical because …
- Chapter 5 of a forthcoming book (proofs are all written in detail)
  - possibly a paper?

“Floyd style” [Flo67]

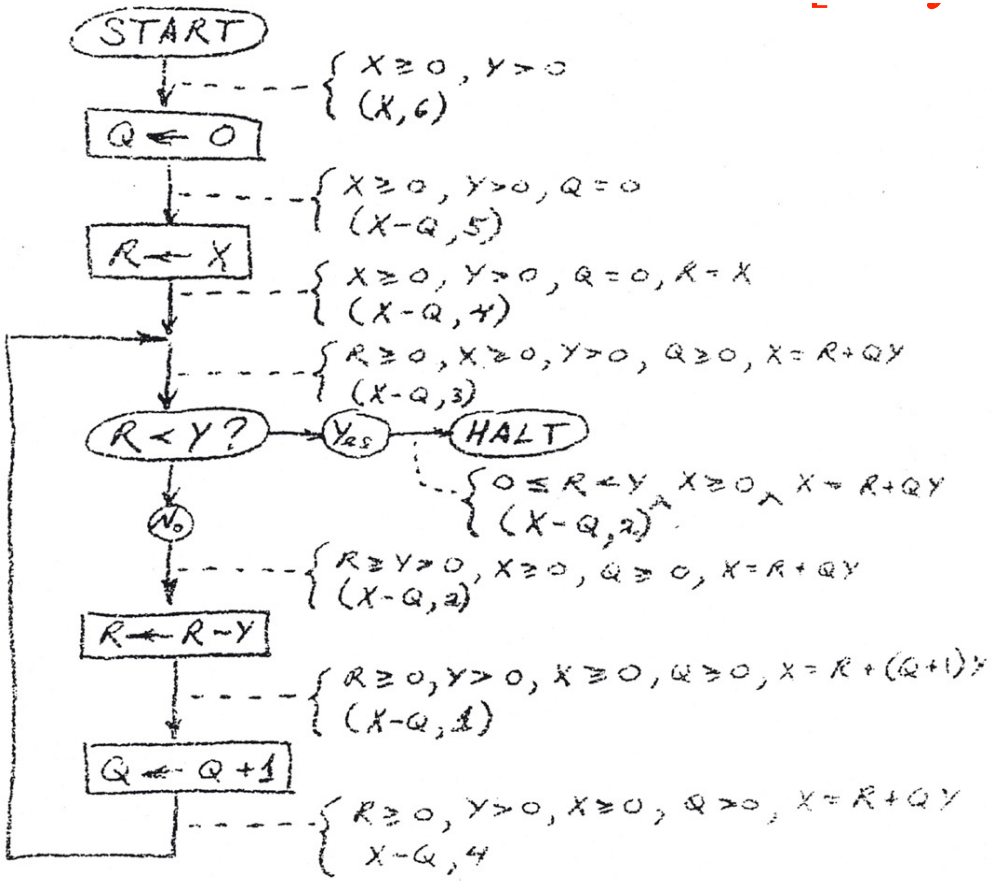


Figure 5.

Algorithm to compute quotient Q and remainder R of X ÷ Y, for integers X≥0, Y>0.

add assertions to flowchart
“state/memory assertions”

formal rules for consistency

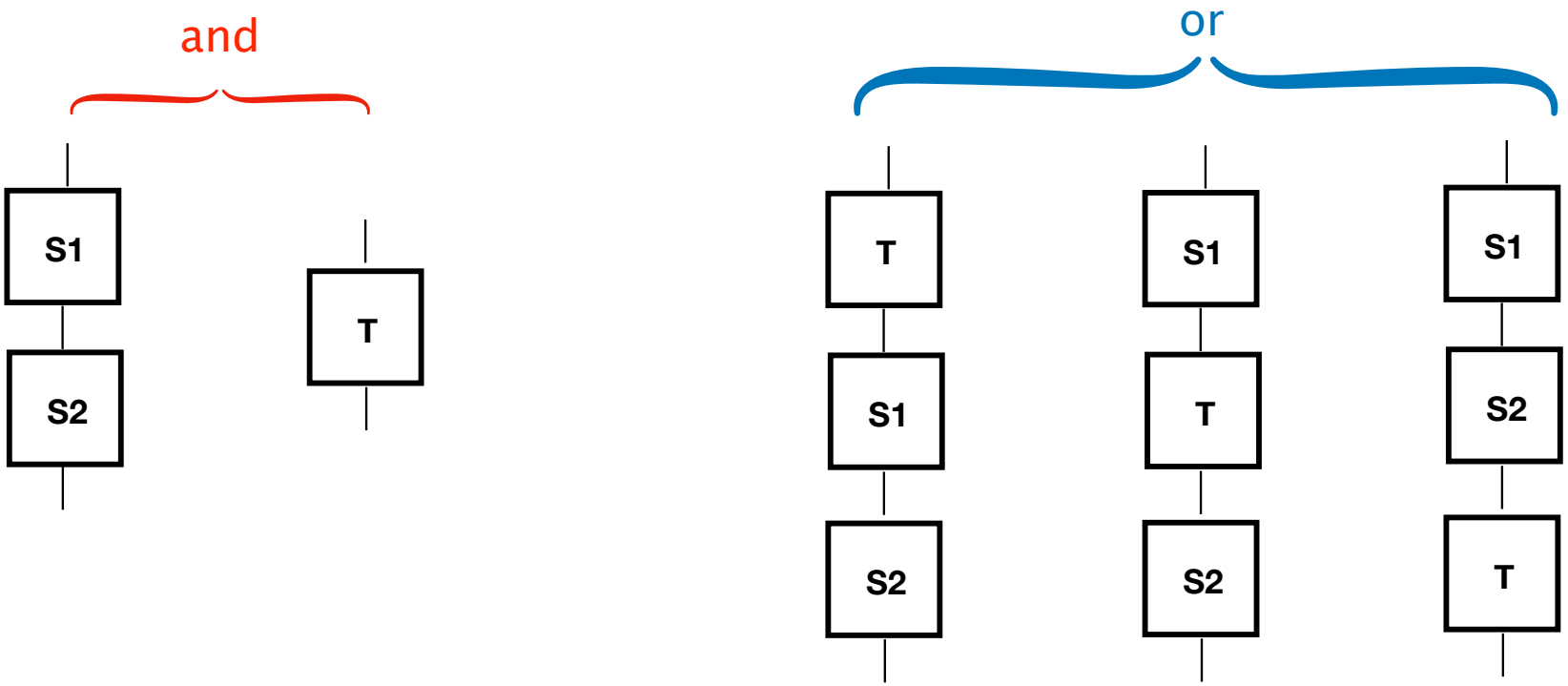
Why concurrency is difficult

- one key issue: “interference”
  - x := 1 || x := 2
  - Other || (x := 42; **if** x=42 **then** … **else** …)
  - moving money between bank accounts
- atomicity!
  - x := x + 1
  - **if** x = x **then** …

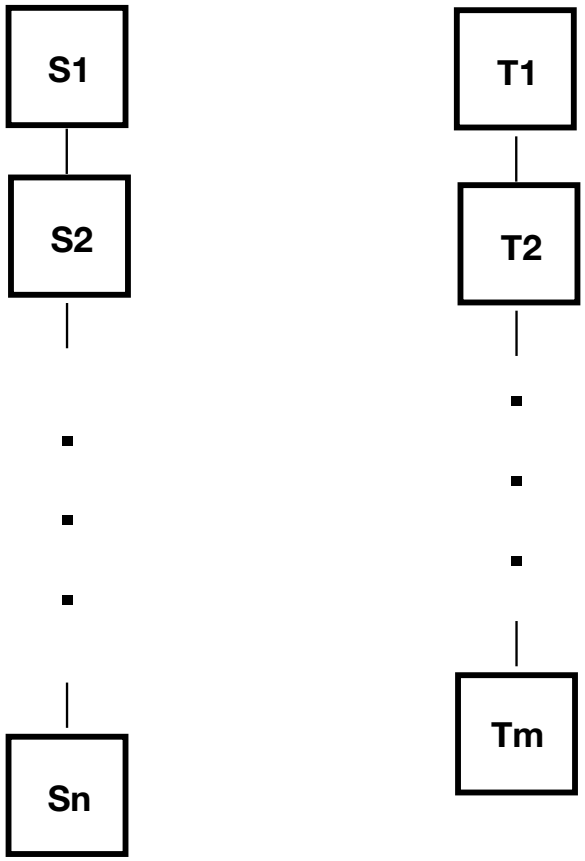
In Stanford

- Zohar Manna
  - CMU (previously “Carnegie Tech”) 1968 PhD with Bob Floyd + Al Perlis
    - (Jones (visited Floyd 1967 +) MTOC 1967-11)
  - paper on translating programs into predicate calculus: termination [Man69]
  - moved to Stanford 1969 (non-deterministic programs [Man70])
- Ed Ashcroft
  - London Uni PhD (awarded 1970, suspect moved to Stanford earlier)

concurrency as non-determinism



concurrency as non-determinism



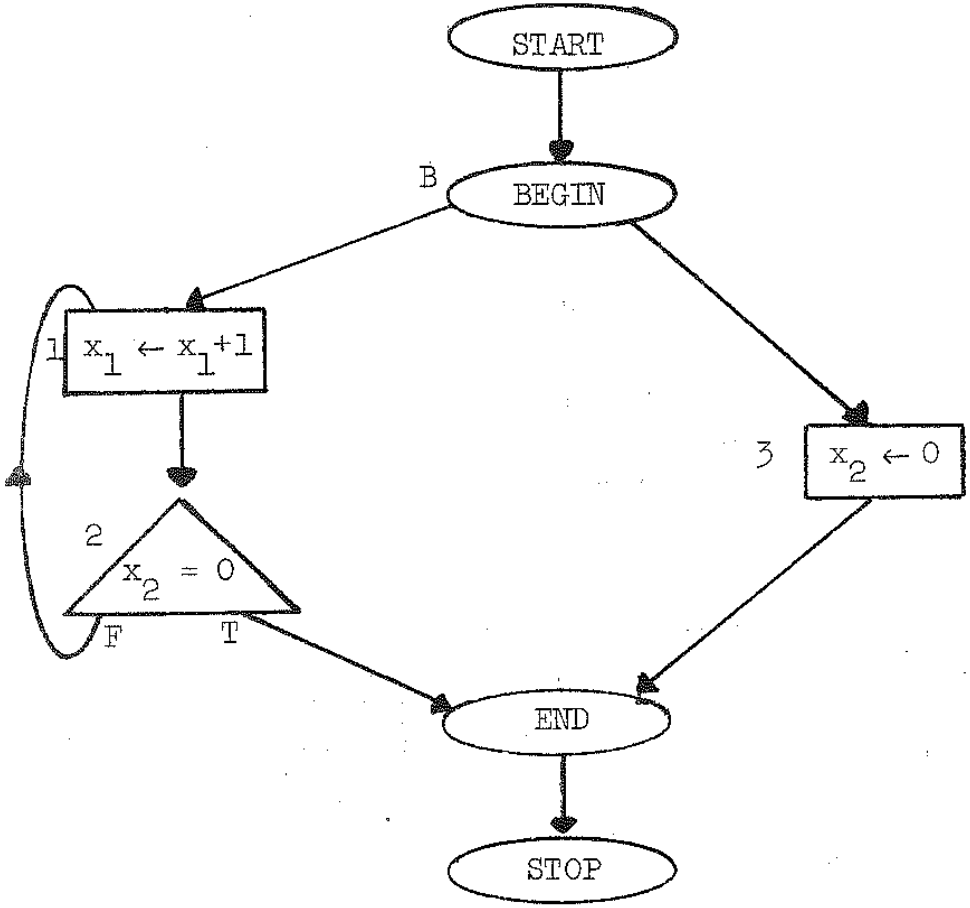
(even straight line)
exponential number of merges

(n+m)!/(n!*m!)

n=m=5: 252
n=m=10: >150 k
n=m=15: >155 million

then prove each alternative!

Ashcroft/Manna [AM71]



NB: right thread
has one statement
+
care!

Issues with Ashcroft/Manna

- very much Floyd based: annotated flow charts
  - (rather than structured combinators)
  - only a passing reference to [Hoare69]
  - all but one example are schemas (no specifications)
- *post-facto* verification (= “bottom-up” approach)
- assignments (and tests) assumed to execute atomically
- exponential expansion + non-trivial expansion of loops/conditionals
  - “blocks” as a way to reduce expansion

Avoiding expansion: Ashcroft (at Waterloo) [Ash75] *submitted 1973*

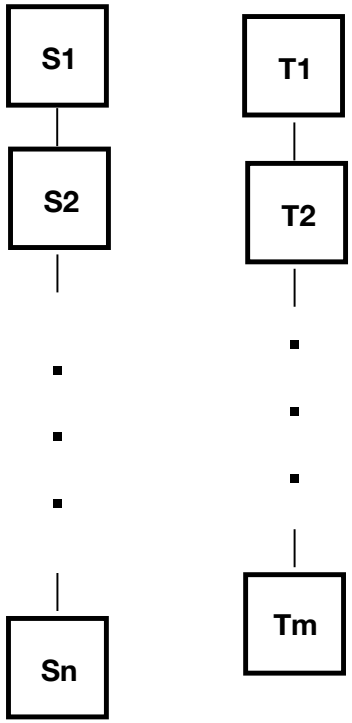
- (in addition to “memory states”) Ashcroft used “control states”
  - think of as fingers on what can execute next in each thread
    - shades of VDL “control trees” (Ashcroft’s PhD with John Florentin + IBM link)
- (potentially) still have exponential combinations
  - but can look for ways to combine cases - without (initial) expansion
    - hints here of how “Temporal Logic” separates proofs from programs
- a significant example: airline reservation (simplified)
- still effectively ignores [Hoa69]

Issues for Ashcroft

- observation: “concurrency as stimulus for formal verification”
- still post-facto verification
  - the starting point for verification is a finished program
- atomicity still unrealistic
- recently re-discovered his Jan 1972 Las Cruces paper
- also interesting: Matthew Hennessy did his (Waterloo) PhD with Ashcroft
  - Hennessy/Milner: process algebras
- later: Ashcroft/Wadge on “Lucid”

Enter: Susan Owicki

- (met again during recent “interview”)
- 1975 Cornell thesis [Owi75]
  - cites 1973 draft of Ashcroft’s paper
  - (order of pages in Cornell on-line copy is wrong)
- joint paper with supervisor David Gries [OG76]
  - normally referred to as “Owicki/Gries” approach
  - IFIP WG2.3 influences (and served to amplify)
- clearly Hoare-based: Hoare-like proofs of the independent threads
  - followed (crucially) by “interference freedom” proof obligation



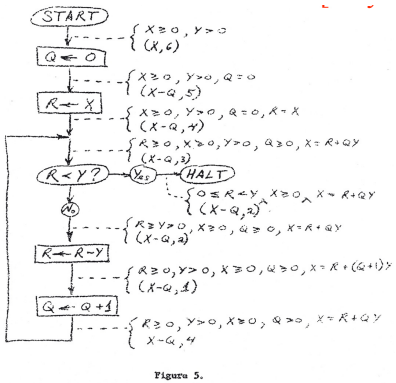


Figure 5.

Algorithm to compute quotient Q and remainder R of X ÷ Y, for integers X≥0, Y>0.

Hoare style [Hoa69]

Sequential composition:

$$\textbf{sequence}\ \frac{\{P\}\ S1\ \{Q\} \quad \{Q\}\ S2\ \{R\}}{\{P\}\ S1;\ S2\ \{R\}}$$

Iteration:

$$\textbf{while}\ \frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}\ \{P \wedge \neg\, b\}}$$

Conditional:

$$\textbf{if}\ \frac{\{P \wedge b\}\ S1\ \{Q\} \quad \{P \wedge \neg\, b\}\ S2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\ \textbf{fi}\ \{Q\}}$$

Assignment:

$$:=\ \frac{}{\{P^e_x\}\ x := e\ \{P\}}$$

where P^e_x is the result of systematically substituting the expression e for every free occurrence of the identifier x throughout P.

"triples" as judgements
inference rules

lent itself to "development" [Hoa71]
"top-down" = "posit and prove"

A useful example:
Findpos - sequential

$$
\begin{array}{l}
Findp: \\
\quad c := 1;\ t := M + 1; \\
\quad search: \\
\quad\quad \textbf{while}\ c < t\ \textbf{do} \\
\quad\quad\quad \textbf{if}\ x(c) > 0\ \textbf{then}\ t := c \\
\quad\quad\quad \textbf{else}\ c := c + 1
\end{array}
$$

Findpos - (only) parallel

$Findp$:
$ec := 2;\ oc := 1;\ et := ot := M + 1;$
cobegin
$Even$: **while** $ec < et$ **do**
if $x(ec) > 0$ **then** $et := ec$
else $ec := ec + 2$
||
Odd: **while** $oc < ot$ **do**
if $x(oc) > 0$ **then** $ot := oc$
else $oc := oc + 2$
coend
$t := min(ot, et)$
end

Findpos - concurrent

$Findp$:
$ec := 2;\ oc := 1;\ et := ot := M + 1;$
cobegin

$Even$: **while** $ec < min(ot, et)$ **do**	Odd: **while** $oc < min(ot, et)$ **do**
if $x(ec) > 0$ **then** $et := ec$	**if** $x(oc) > 0$ **then** $ot := oc$
else $ec := ec + 2$	**else** $oc := oc + 2$

coend
$t := min(ot, et)$
end

Add a “few” details:
(need assertions
between all statements)

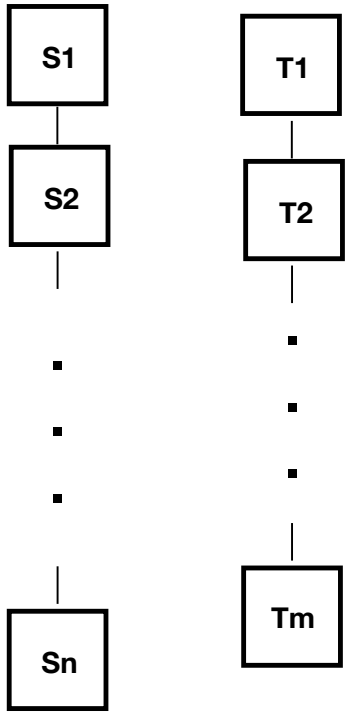
Findpos: **begin**
Initialize: $ec := 2;\ oc := 1;\ et := ot := M + 1;$
$\{ec = 2 \wedge oc = 1 \wedge et = ot = M + 1\}$
Search: **cobegin**
$\{ES\}$
Evensearch: **while** $ec < min(ot, et)$ **do**
$\{ES \wedge ec < et \wedge ec < M + 1\}$
Eventest: **if** $x(ec) > 0$
then $\{ES \wedge ec < et \wedge i < M + 1 \wedge x(ec) > 0\}$ *Evenyes*: $et := ec$ $\{ES\}$
else $\{ES \wedge ec < et \wedge x(ec) \leq 0\}$ *Evenno*: $ec := ec + 2$ $\{ES\}$
fi
$\{ES\}$
od
$\{ES \wedge ec \geq min(ot, et)\}$
||
$\{OS\}$
Oddsearch: **while** $oc < min(ot, et)$ **do**
$\{OS \wedge oc < ot \wedge oc < M + 1\}$
Oddtest: **if** $x(oc) > 0$
then $\{OS \wedge oc < ot \wedge oc < M + 1 \wedge x(oc) > 0\}$ *Oddyes*: $ot := oc$ $\{OS\}$
else $\{OS \wedge oc < ot \wedge x(oc) \leq 0\}$ *Oddno*: $oc := oc + 2$ $\{OS\}$
fi
$\{OS\}$
od
$\{OS \wedge oc \geq min(ot, et)\}$
coend
$\{OS \wedge ES \wedge ec \geq min(ot, et) \wedge oc \geq min(ot, et)\}$
$t := min(ot, et)$
$\{t \leq M + 1 \wedge (t \leq M \ \Rightarrow\ x(t) > 0) \wedge \forall i \cdot 0 < n < t \ \Rightarrow\ x(n) \leq 0\}$
end

Where:

$$ES = \left\{ \begin{array}{l} even(ec) \wedge \\ et \leq M + 1 \wedge \\ (et \leq M \ \Rightarrow\ x(et) > 0) \wedge \\ \forall n \cdot even(n) \wedge 0 < n < ec \ \Rightarrow\ x(n) \leq 0 \end{array} \right\}$$

$$OS = \left\{ \begin{array}{l} odd(oc) \wedge \\ ot \leq M + 1 \wedge \\ (ot \leq M \ \Rightarrow\ x(ot) > 0) \wedge \\ \forall n \cdot odd(n) \wedge 0 < n < oc \ \Rightarrow\ x(n) \leq 0 \end{array} \right\}$$

Breakout 5.2: Partial correctness proof of *Findpos* from [Owi75]



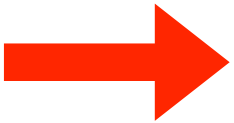
“Owicki-Gries” approach

- Hoare-style proofs of separate threads
  - could have been “development”
    - but do need all assertions for …
- but: the “interference freedom” PO is post-facto
  - if fails: start from beginning!
- atomicity

Findpos in Ashcroft's approach

- similar proof load
  - I actually re-used Owicki's predicate definitions
  - (full proof available in forthcoming book)

Add quite a “few” details!



Program:

Findpos: $ec := 2;\ oc := 1;\ et := ot := M + 1;$
Search: **fork go to**(*Evensearch*, *Oddsearch*);
Evensearch: **if** $ec < min(ot, et)$ **then go to** *Eventest* **else go to** J;
  Eventest: **if** $x(ec) > 0$ **then go to** *Evenyes* **else go to** *Evenno*;
    Evenyes: $et := ec;$ **go to** *Evensearch*;
    Evenno: $ec := ec + 2;$ **go to** *Evensearch*;
Oddsearch: **if** $oc < min(ot, et)$ **then go to** *Oddtest* **else go to** J;
  Oddtest: **if** $x(oc) > 0$ **then go to** *Oddyes* **else go to** *Oddno*;
    Oddyes: $ot := oc;$ **go to** *Oddsearch*;
    Oddno: $oc := oc + 2;$ **go to** *Oddsearch*;
J: **join**(*Evensearch*, *Oddsearch*);
$t := min(ot, et);$
F: **HALT**

The step to label *Search* is sequential so the control state c is a unit set and the state assertion is:

$c = \{Search\}$: $ec = 2 \wedge oc = 1 \wedge et = ot = M + 1$

Using the same definitions as in Breakout 5.2:

$$ES = \left\{ \begin{array}{l} even(i) \wedge \\ et \leq M + 1 \wedge \\ (et \leq M \ \Rightarrow\ x(et) > 0) \wedge \\ \forall n \cdot even(n) \wedge 0 < n < ec \ \Rightarrow\ x(n) \leq 0 \end{array} \right\}$$

$$OS = \left\{ \begin{array}{l} odd(oc) \wedge \\ ot \leq M + 1 \wedge \\ (ot \leq M \ \Rightarrow\ x(ot) > 0) \wedge \\ \forall n \cdot odd(n) \wedge 0 < n < oc \ \Rightarrow\ x(n) \leq 0 \end{array} \right\}$$

It is sufficient to consider groups of control states as follows (with their attached state assertions):

Evensearch $\in c$: ES
Eventest $\in c$: $ES \wedge ec < et \wedge ec < M + 1$
Evenyes $\in c$: $ES \wedge ec < et \wedge ec < M + 1 \wedge x(ec) > 0$
Evenno $\in c$: $ES \wedge ec < et \wedge x(ec) \leq 0$

Each of these steps follows by standard Floyd-like reasoning because there is no damaging interference. (Again the reasoning for the other thread is completely symmetric.)

The only difficult step is proving that both steps from *Evensearch* and *Oddsearch* to J are correct.

$c = \{J\}$: $ES \wedge OS \wedge ec \geq min(ot, et) \wedge oc \geq min(ot, et)$

since $c = \{Evensearch, Oddyes\}$ indicates that ot can be changed after the test. To conclude that the negation of that test is still true at J, it is necessary to note that:

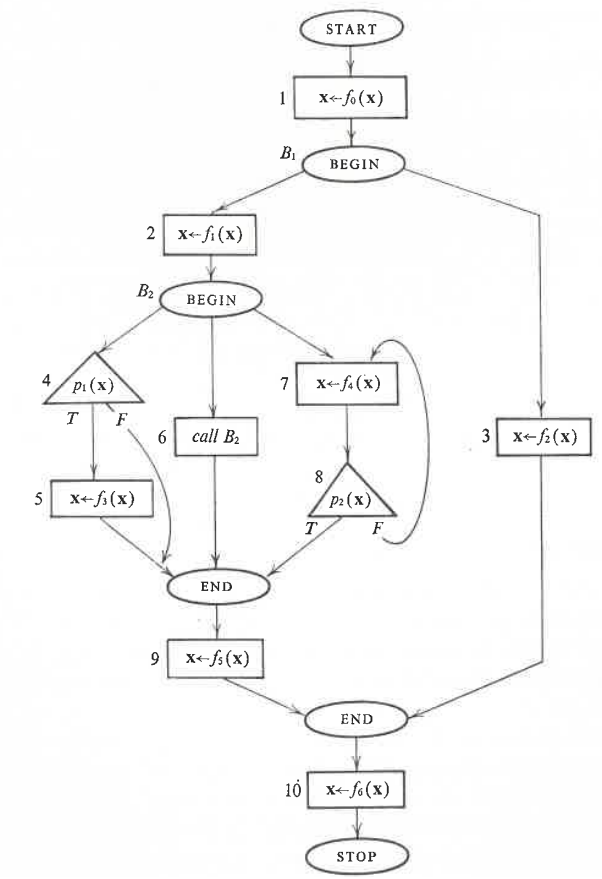
$ec \geq min(ot, et) \wedge ot' = oc \wedge oc < ot \ \Rightarrow\ ec \geq min(ot', et)$

The final step from J to F is again sequential reasoning (singleton control states):

$c = \{F\}$: $t \leq M + 1 \wedge (t \leq M \ \Rightarrow\ x(t) > 0) \wedge \forall i \cdot 0 < i < t \ \Rightarrow\ x(i) \leq 0$

Findpos in Ashcroft/Manna??

- they don't actually give a mapping *algorithm*
  - they give a fairly general example
    - (concurrent to non-deterministic programs)
- the merge is too large!
  - because loops+conditionals in *both* branches



Key insights

- concurrency can be replaced by non-determinacy [AshcroftManna]
  - but identification of “atomic steps” must be honest
- keeping track of all potential next steps can avoid expansion [Ashcroft]
  - points to separation of program and its justification
- “interference freedom” localises the pain [Owicki]
- “bottom-up” uses proof to replace testing
  - (IMHO) formal methods pay off *in design*

“Atomicity” = a problem for all

- x := x +1
- **if** x = x **then** … **else** …
- “Reynolds’ rule” doesn’t solve the problem;
  - t := x; t := t + 1; x := t
    - it just exposes it
    - (nor did John own it!)
- BTW there are neater versions of *Findpos*

Enhanced time line

Floyd	[Flo67]								
Manna		68	69						
Hoare			69	71			75		
AM				71					
Ashcroft					73		75		
Owicki							75		
Rely/Guar								81	
CSL									07

on **apparent** linearity:
I've focussed on 1 strand
+ fewer researchers in the 70s