One concurrent program: three attempts at its formal verification

Cliff Jones
Newcastle University

ETH zürich (by Zoom)
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?
add assertions to flowchart
“state/memory assertions”
formal rules for consistency
Why concurrency is difficult

• one key issue: “interference”
  • \( x := 1 \parallel x := 2 \)
  • Other \( \parallel \) (\( x := 42; \) 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
Hoare style \([\text{Hoa}69]\)

Sequential composition:

\[
\{P\} \quad S_1 \quad \{Q\} \\
\{Q\} \quad S_2 \quad \{R\} \\
\text{sequence} \quad \{P\} \quad S_1; \quad S_2 \quad \{R\}
\]

Iteration:

\[
\{P \land b\} \quad S \quad \{P\} \\
\{P\} \quad \text{while } b \text{ do } S \text{ od } \{P \land \neg b\}
\]

Conditional:

\[
\{P \land b\} \quad S_1 \quad \{Q\} \\
\{P \land \neg b\} \quad S_2 \quad \{Q\} \\
\text{if } \{P\} \quad \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi } \{Q\}
\]

Assignment:

\[
:= \quad \{P^e_x\} \quad x := e \quad \{P\}
\]

where \(P^e_x\) is the result of systematically substituting the expression \(e\) for every free occurrence of the identifier \(x\) throughout \(P\).
A useful example:

*Findpos - sequential*

*Findp:*

\[ c := 1; \quad t := M + 1; \]

*search:*

\[ \text{while } c < t \text{ do} \]

\[ \quad \text{if } x(c) > 0 \text{ then } t := c \]

\[ \quad \text{else } c := c + 1 \]
Findpos - (only) parallel

Findp:

\[ ec := 2; \quad oc := 1; \quad et := ot := M + 1; \]

\text{cobegin}

\text{Even: while } ec < et \text{ do }
\quad \text{if } x(ec) > 0 \text{ then } et := ec \\
\quad \text{else } ec := ec + 2 \\
\text{coend}

\text{Odd: while } oc < ot \text{ do }
\quad \text{if } x(oc) > 0 \text{ then } ot := oc \\
\quad \text{else } oc := oc + 2 \\

\text{t := min(ot, et)}

end
Findpos - concurrent

\[ \text{Findp:} \]
\[ ec := 2; \quad oc := 1; \quad et := ot := M + 1; \]
\[ \text{cobegin} \]
\[ \text{Even: while } ec < \min(ot, et) \text{ do} \quad \text{Odd: while } oc < \min(ot, et) \text{ do} \]
\[ \quad \text{if } x(ec) > 0 \text{ then } et := ec \quad \text{if } x(oc) > 0 \text{ then } ot := oc \]
\[ \quad \text{else } ec := ec + 2 \quad \text{else } oc := oc + 2 \]
\[ \text{coend} \]
\[ t := \min(ot, et) \]
\[ \text{end} \]
Findpos: begin  
Initialize: \( ec := 2; \ oc := 1; \ et := ot := M + 1; \)  
\{ ec = 2 \land \ oc = 1 \land \ et = ot = M + 1 \}  
Search: cobegin  
\{ ES \}  
Evensearch: while \( ec < \min(ot, et) \) do  
\{ ES \land ec < et \land \ ec < M + 1 \}  
Eventest: if \( x(ec) > 0 \) then  
\{ ES \land ec < et \land \ i < M + 1 \land x(ec) > 0 \}  
Evenyes: \( et := ec \) \{ ES \}  
else  
\{ ES \land ec < et \land x(ec) \leq 0 \}  
Evenno: \( ec := ec + 2 \) \{ ES \}  
fi  
od  
\{ ES \land ec \geq \min(ot, et) \}  
||  
\{ OS \}  
Oddsearch: while \( oc < \min(ot, et) \) do  
\{ OS \land oc < at \land \ oc < M + 1 \}  
Oddtest: if \( x(oc) > 0 \) then  
\{ OS \land oc < at \land \ oc < M + 1 \land x(oc) > 0 \}  
Oddyes: \( ot := oc \) \{ OS \}  
else  
\{ OS \land oc < at \land x(oc) \leq 0 \}  
Oddno: \( oc := oc + 2 \) \{ OS \}  
fi  
od  
\{ OS \land oc \geq \min(ot, et) \}  
coend  
\{ OS \land ES \land ec \geq \min(ot, et) \land oc \geq \min(ot, et) \}  
t := \min(ot, et)  
\{ t \leq M + 1 \land (t \leq M \Rightarrow x(t) > 0) \land \forall i \cdot 0 < n < t \Rightarrow x(n) \leq 0 \}  
end  
Where:  
\[ ES = \left\{ \begin{array}{l} \text{even}(ec) \land \\
\text{et} \leq M + 1 \land \\
(\text{et} \leq M \Rightarrow x(\text{et}) > 0) \land \\
\forall n \cdot \text{even}(n) \land 0 < n < \text{ec} \Rightarrow x(n) \leq 0 \\
\text{odd}(ec) \land \\
\text{et} \leq M + 1 \land \\
(\text{et} \leq M \Rightarrow x(\text{et}) > 0) \land \\
\forall n \cdot \text{odd}(n) \land 0 < n < \text{oc} \Rightarrow x(n) \leq 0 \\
\end{array} \right. \]  
\[ OS = \left\{ \begin{array}{l} \text{odd}(oc) \land \\
\text{ot} \leq M + 1 \land \\
(\text{ot} \leq M \Rightarrow x(\text{ot}) > 0) \land \\
\forall n \cdot \text{odd}(n) \land 0 < n < \text{oc} \Rightarrow x(n) \leq 0 \\
\end{array} \right. \]  

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)
The final step from \( \text{Evensearch} \) concludes that the negation of that test is still true at
\( \text{Evenno} \); \( \text{Oddsearch} \); \( \text{Oddtest} \) and \( \text{Oddno} \)
are again sequential reasoning (singleton control states):
\[ c = \{ J \}: ES \land OS \land ec \geq \min(ot, et) \land oc \geq \min(ot, et) \]
since \( c = \{ \text{Evensearch}, \text{Oddyes} \} \) 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:
\[ ec \geq \min(ot, et) \land ot' = oc \land oc < ot \Rightarrow ec \geq \min(ot', et) \]
The final step from \( J \) to \( F \) is again sequential reasoning (singleton control states):
\[ c = \{ F \}: t \leq M + 1 \land (t \leq M \Rightarrow x(t) > 0) \land \forall i: 0 < i < t \Rightarrow x(i) \leq 0 \]
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

• \( x := x + 1 \)

• \( \text{if } x = x \text{ then } \ldots \text{ else } \ldots \)

• “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 \textit{Findpos}
Enhanced time line

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Floyd</td>
<td>[Flo67]</td>
<td>68</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Manna</td>
<td></td>
<td>68</td>
<td>69</td>
<td>71</td>
</tr>
<tr>
<td>Hoare</td>
<td></td>
<td>69</td>
<td>71</td>
<td>75</td>
</tr>
<tr>
<td>AM</td>
<td></td>
<td>68</td>
<td>69</td>
<td>71</td>
</tr>
<tr>
<td>Ashcroft</td>
<td></td>
<td>68</td>
<td>69</td>
<td>71</td>
</tr>
<tr>
<td>Owicki</td>
<td></td>
<td>68</td>
<td>69</td>
<td>71</td>
</tr>
</tbody>
</table>

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