CBJ

~~WALK~~ FILE

"OPERATIONS and Formal Development"

TN9004 (Unrestricted)
3 September 1972

C B Jones
Product Test Laboratory
IBM United Kingdom Limited
Hursley Park
Hursley
Nr Winchester
Hampshire

BACKGROUND

Refs 1 and 2 present examples of "Formal Development of Programs", which owe much, and are closely related, to the work in Refs 3, 4 and 5. One shortcoming of refs 1 and 2 is that they yield functions which have then to be translated into programs. The decision to use functions resulted from a desire to use rather more detailed arguments of correctness than given in refs 3 and 5 and the fact that certain problems had been encountered with the use of the axioms of ref 4. This note shows a way of overcoming these difficulties which appears to make more routine the construction , and add to the clarity, of "formal developments".

The first difficulty encountered with using Hoare's axioms was that termination is not treated: ref 4 in fact does not discuss termination until the complete algorithm has been developed. It is the view of the current author that termination should be proven at each level of refinement of the algorithm.

A second difficulty resulted from the fact that the domains of both the pre and post conditions is a single state. Thus to require that an operation does not change the value of a variable, requires the use of a free variable, thus:-

$$x=x_0\{OP\}x=x_0$$

the system given below has post conditions of state pairs thus reducing the use of free variables.

Ref 4 does not show how different levels of abstraction of an algorithm can use different data representations (although Professor Hoare has privately communicated his work in this area): the system below appears to handle this problem naturally.

Given a domain of states, say Σ, and members thereof σ, σ' etc, "operations" are considered as relations on states:-

$$OP \subseteq \Sigma \times \Sigma$$

Strictly this note considers $OP: \Sigma \to \Sigma$
see elsewhere for discussion of non-determ

written:-

$$\sigma[OP]\sigma'$$

Operations can be decomposed (combined) in a number of ways:-

$$\sigma[OP1;OP2]\sigma'' \equiv (\exists\sigma')(\sigma[OP1]\sigma' \wedge \sigma'[OP2]\sigma'')$$

presumed total
non side-eff

$$\sigma[\underline{if}\ p\ \underline{then}\ OP1\ \underline{else}\ OP2]\sigma' \equiv$$
$$(p(\sigma) \wedge \sigma[OP1]\sigma') \vee (\sim p(\sigma) \wedge \sigma[OP2]\sigma')$$

$$\sigma[\underline{while}\ p\ \underline{do}\ OP]\sigma'' \equiv$$
$$(\sim p(\sigma) \wedge \sigma''=\sigma) \vee$$
$$(p(\sigma) \wedge (\exists\sigma')(\sigma[OP]\sigma' \wedge \sigma'[\underline{while}\ p\ \underline{do}\ OP]\sigma''))$$

Notice that the while property does not give an immediate way of proving properties about while loops: this requires knowledge about (inductive) properties of the states.

By using, for example, restricted identity relations for the tests, it would be possible to present a more complete theory in terms of relations: this is not done since it is the system of relations between conditions which is of interset here.

CONDITIONS ON PURE OPERATIONS

It is obviously not possible to give properties required of operations by enumeration of the relations discussed above. The process of reasoning about the class of computations caused by an operation uses the following notation:-

$OP :: \Sigma$	an operation on states from Σ
$\alpha \;:\; \Sigma \rightarrow \{T,F\}$	a predicate on Σ
$\omega \;:\; \Sigma \times \Sigma \rightarrow \{T,F\}$	a predicate on pairs of elements of Σ

$\alpha\langle OP\rangle\omega$ is written only if:-

may not determine which deterministic operation

$\alpha(\sigma) \wedge \sigma[OP]\sigma' \supset \omega(\sigma,\sigma')$

$\alpha(\sigma) \supset (\exists\sigma')(\sigma[OP]\sigma')$

It is possible to decompose (combine) directly operations whose properties are given implicitly.

Sequencing, providing:-

extension to more terms

$\alpha_1(\sigma)\langle OP1\rangle\alpha_2(\sigma') \wedge \omega_1(\sigma,\sigma')$

$\alpha_2(\sigma')\langle OP2\rangle\alpha_3(\sigma'') \wedge \omega_2(\sigma',\sigma'')$

$\omega_1(\sigma,\sigma') \wedge \omega_2(\sigma',\sigma'') \supset \omega(\sigma,\sigma'')$

then:-

$\alpha_1(\sigma)\langle OP1;OP2\rangle\omega(\sigma,\sigma'') \wedge \alpha_3(\sigma'')$

Conditional, providing:-

$\alpha(\sigma) \wedge p(\sigma)\langle OP1\rangle\omega(\sigma,\sigma')$

$\alpha(\sigma) \wedge \sim p(\sigma)\langle OP2\rangle\omega(\sigma,\sigma')$

then:-

$\alpha(\sigma)\langle$<u>if</u> p <u>then</u> OP1 <u>else</u> OP2$\rangle\omega(\sigma,\sigma')$

Repetition (Hoare style), providing:-

$inv(\sigma) \wedge p(\sigma)\langle OP\rangle inv(\sigma')$

a function term can be found s.t.

$inv(\sigma) \supset term(\sigma) \geq 0$

$term(\sigma)=0 \equiv \sim p(\sigma)$

$\sigma[OP]\sigma' \supset term(\sigma') < term(\sigma)$

then:-

$inv(\sigma)\langle$<u>while</u> p <u>do</u> OP$\rangle inv(\sigma') \wedge \sim p(\sigma')$

Repetition (one of many alternatives), providing:-

$\alpha(\sigma) \wedge p(\sigma)\langle OP\rangle\alpha(\sigma') \wedge \omega(\sigma,\sigma')$

$c(\sigma,\sigma') \wedge \omega(\sigma',\sigma'') \supset c(\sigma,\sigma'')$

term ~~as above~~

$\alpha(\sigma) \supset term(\sigma) \geq 0$

$term(\sigma) = 0 \equiv \sim p(\sigma)$

$\omega(\sigma,\sigma') \supset term(\sigma') < term(\sigma)$

then:-

$\alpha(\sigma) \wedge c(\sigma,\sigma)\langle$<u>while</u> p <u>do</u> OP$\rangle\alpha(\sigma') \wedge c(\sigma,\sigma') \wedge \sim p(\sigma')$

notice that if:-

$\alpha\langle OP\rangle\omega$

providing:-

$strong\alpha(\sigma) \supset \alpha(\sigma)$

~~should be~~

$\cancel{\omega(\sigma,\sigma') \wedge c(\sigma',\sigma'') \supset c(\sigma,\sigma'')}$

then:-

$strong\alpha\langle OP\rangle\omega$

or providing:-

$\omega(\sigma,\sigma') \supset weak\omega(\sigma,\sigma')$

then:-

$\alpha\langle OP\rangle weak\omega$

(EXTENDED) OPERATIONS

It has in fact been found necessary to use operations which, as well as changing a state, accept arguments and produce results. One way of treating such operations, when they arise, is to consider a stack from which arguments are taken and to which results are returned. This could be written:-

$$OP :: \Sigma : \Delta \rightarrow P$$
$$\sigma, s^\frown\delta[OP]\sigma', s^\frown p$$
$$\alpha : \Sigma x \Delta \rightarrow \{T,F\}$$
$$\omega : \Sigma x \Delta x \Sigma x P \rightarrow \{T,F\}$$

$\alpha\langle OP\rangle\omega$ is written only if:-

$$\alpha(\sigma,\delta) \wedge \sigma, s^\frown\delta[op]\sigma', s^\frown p \supset \omega(\sigma,\delta,\sigma',p)$$
$$\alpha(\sigma,\delta) \supset (\exists\sigma',p)(\sigma, s^\frown\delta[op]\sigma', s^\frown p)$$

This would facilitate (if desired!) an extension of the conditional or repetitive constructs to permit state changes by the predicate.

$\rho = op(\delta)$?

MORE ON STATES

States can be structured and the notation used below is:

$$\begin{aligned}\Sigma = (&\langle n_1 : p_1\rangle,\\ &\langle n_2 : p_2\rangle,\\ &\quad\vdots\\ &\langle n_n : p_n\rangle)\end{aligned}$$

selection is written as

$\sigma(n_1)$ etc.

or, if no ambiguity is likely, parts of Σ, Σ' can be written:

n_1 or n_1' etc.

In spite of the liberties taken with the Vienna notation for objects, μ is used with its usual meaning.

"THE" EXAMPLE

Specification

$$\Sigma = (\langle n:I\rangle,$$
$$\langle fn:I\rangle)$$

where I is the set of non negative integers(this assumption about the state is used below to shorten the proofs.)

find:-

$$F :: \Sigma$$

such that:-

$$T < F > \omega$$

where $\omega(\sigma,\sigma') \equiv fn' = n!$

ie $\sigma'(fn) = (\sigma(n))!$

Stage 1

Assume we have two operations:-

$$OP1,\ OP2 :: \Sigma$$

such that:-

$T\langle OP1\rangle\ \omega_1$

where $\omega_1(\sigma,\sigma') \equiv \sigma' = \mu(\sigma;\langle fn:1\rangle)$

$T\langle OP2\rangle\ \omega_2$

where $\omega_2(\sigma',\sigma'') \equiv fn'' = fn' \,.\, (n'!)$

Assertion:-

F = OP1;OP2 satisfies the specification.

Justification required is $T\langle OP1;OP2\rangle\omega$
proof follows from combination of "conditions" since

$$\omega_1(\sigma,\sigma') \wedge \omega_2(\sigma',\sigma'') \supset \omega(\sigma,\sigma'')$$

Stage 2

Assume we have an operation

$$OP3 :: \Sigma$$

such that:-

$\alpha_3 \langle OP3 \rangle \omega_3$
where $\alpha_3(\sigma) \equiv n \geq 1$ required to ensure valid state, I

$\omega_3(\sigma,\sigma') \equiv fn' = fn \,.\, n \wedge$
$n' = n-1$

Assertion:-
OP2 = $\underline{\text{while}}$ n≥1 $\underline{\text{do}}$ OP3 satisfies the requirements

Justification required is $T\langle \underline{\text{while}}\ n \geq 1\ \underline{\text{do}}\ OP3 \rangle\ \omega_2$

proof for all σ by induction on $\sigma(n)$. Basis, suppose $\sigma(n)=0$:-

$\sigma[\underline{\text{while}}\ n\geq 1\ \underline{\text{do}}\ OP3]\sigma$
so $(\exists\sigma')(\sigma[\underline{\text{while}}\ n\geq 1\ \underline{\text{do}}\ OP3]\sigma')$
further since 0! = 1
$\omega_2\ (\sigma,\sigma)$
Thus $T\langle\underline{\text{while}}\ n\geq 1\ \underline{\text{do}}\ OP3\rangle\omega_2$

Suppose true for $0 \leq \sigma(n) < x$ prove for $\sigma(n)=x$

since $\alpha_3(\sigma)$
$(\exists\sigma')(\sigma[OP3]\sigma' \wedge \omega_3(\sigma,\sigma'))$
$n' < x$
thus by Induction Hypotheses
$(\exists\sigma'')(\sigma'[\underline{\text{while}}\ n\geq 1\ \underline{\text{do}}\ OP3]\sigma'' \wedge \omega_2(\sigma',\sigma''))$

since
$fn'' = fn' \,.\, (n'!)$
$= fn.n.((n-1)!)$
$= fn.(n!)$
$\omega_3(\sigma,\sigma') \wedge \omega_2(\sigma',\sigma'') \supset \omega_2(\sigma,\sigma'')$

$\omega_2(\sigma,\sigma'')$

thus
$T\langle\underline{\text{while}}\ n\geq 1\ \underline{\text{do}}\ OP3 \rangle\ \omega_2$

which concludes the proof.

<u>Program</u>

A "reasonable" language should allow:-

$T \langle\ fn := 1\ \rangle\ \omega_1$
$\alpha_3 \langle\ fn:=fn.n\ ;\ n:=n-1\ \rangle\ \omega_3$

<u>Comments</u>

Notice the effect of permitting operations to rely only on 302
properties of their initial state (<u>not</u> on the way it was formed), 303
and also that there is no requirement for a temporary variable to 305
avoid overwitting the original value of n. Termination follows 306
in the above from the way the induction was made.

The above proof can easily be made using the alternative 308
induction axiom with:-

$$c(\sigma,\sigma') = fn'.n'! = n! \cdot fn$$ 310

The proof using the Hoare axiom is left as an excercise to the 312
reader.

MAPPING

Suppose some stage of development uses:-

$$OPd :: D$$

such that:-

$$\alpha d < OPd > \omega d$$

that is:-

$$\alpha d(d) \wedge d[OPd]d' \supset \omega_1(d,d')$$
$$\alpha d(d) \supset (\exists d')(d[OPd]d')$$

Then the next stage could use:-

$$OPe :: E$$

such that:-

$$\alpha e < OPe > \omega e$$

provided a relation:-

$$\theta : D \times E \rightarrow \{T,F\}$$

is found such that:-

$$\theta(d^1,e) \wedge \theta(d^2,e) \supset d^1=d^2 \qquad ?$$
$$\alpha d(d) \supset (\exists e)(\theta(d,e))$$
$$\alpha d(d) \wedge \theta(d,e) \supset \alpha e(e)$$
$$\theta(d,e) \wedge \omega e(e,e') \wedge \theta(d',e') \supset \omega d(d,d')$$
$$\alpha e(e) \wedge \omega e(e,e') \supset (\exists d')(\theta(d',e'))$$

then:-

$$d[OPd]d' \equiv \theta(d,e) \wedge e[OPe]e' \wedge \theta(d',e')$$

satisfies the properties required for OPd

This general form, whose use will normally look far simpler than the above, is justified as follows:

$$\alpha d(d) \wedge \theta(d,e) \wedge e[OPe]e' \wedge \theta(d',e') \supset \omega d(d,d')$$

because:-

$\alpha d(d) \wedge \theta(d,e)$	give
$\alpha e(e)$	which with
$e[OPe]e'$	gives
$\omega e(e,e')$	which with above and
$\theta(d',e')$	gives
$\omega d(d,d')$	

and:-

$$\alpha d(d) \supset (\exists d')(\theta(d,e) \wedge e[OPe]e' \wedge \theta(d',e'))$$

because:-

$\alpha d(d)$	gives

$(\exists e)(\theta(d,e))$
let this be called e
$\alpha e(e)$ thus
$(\exists e')(e[OPe]e')$
let this be called e'
$(\exists d')(\theta(d',e'))$ thus
$(\exists d')(\theta(d,e) \wedge e[OPe]e' \wedge \theta(d',e'))$

A family of operations over some domain can be mapped to a new domain providing they are connected with "valid" sequencing constructs.

ACKNOWLEDGEMENTS

Apart from the influence of the referenced publications the author gratefully acknowledges the stimulus of private discussions on "Structured Programming" with Profs Dijkstra, Hoare and Wirth.

REFERENCES

1. C B Jones
"Formal Development of Correct Algorithms: an Example based on Earley's Recogniser."
December 1971

2. C D Allen, C B Jones
"The Formal Development of an Algorithm"
September 1972

3. E W Dijkstra
"A Short Introduction to the Art of Programming"
August 1971

4. C A R Hoare
"The Proof of a Program: FIND"
January 1971

5. N. Wirth
"Program development by Stepwise refinement"
April 1972