distribute

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304
(415) 494-4000

XEROX

October 21, 1976

Professor C.A.R. Hoare
Computer Science Department
The Queen's University
Belfast BT7 1NN
Northern Ireland

Dear Tony,

Many thanks for your letter and the revised paper on *cooperating sequential processes.* I confess to be overwhelmed by your boundless modesty of not even hinting at your prestigious new appointment at Oxford University. Perhaps, misguided by a true Englishman's considerateness, you wanted to spare me from the "trouble" of writing a letter of congratulations? Let me assure you then, that this is no trouble, but rather a definite pleasure! When will you be moving? Will there be a party? How are your plans with the research grant affected?

Indeed, your paper exhibits a new framework for thinking about algorithms with concurrent activities. It is both fascinating and aggravating. The former when one recognises the appealing simplicity of many concepts that hitherto seemed complicated, the latter when one is forced to give up ideas that one had just been getting familiar and comfortable with (ideas such as monitors and conditions postulated very recently by CARH!). The paper has, in my view, gained significantly by the deletion of the "sets of traces". Without altering its essential contents, it could gain even further by more careful choice of notation and syntax, and in particular by careful typesetting, which is a prerequisite as soon as heavy mathematical symbols with super- and subscripting are used. A more substantial comment, however, is that precisely because there are so many new ideas, utmost care should be given to present them in a disentangled fashion, and to help the reader understand by relating the new constructs to analogous concepts in languages of "conventional" style. The reader should not have to build such bridges between his and your proposed new world by himself.

As far as I could make out, there are the following essential and quite independent ideas that you wish to propagate. Please correct me, should I have misinterpreted any of your statements. May I suggest, then, that such a summary be included in the introduction, so that the reader knows what he is in for:

1. Adoption of the *guarded command* and control structures as used by Dijkstra. (May I

suggest that if you adopt his semantics, you would do the reader a service by also adopting his syntax unchanged).

2. A program consists of a fixed number of concurrently active *processes.* They communicate via *channels,* sending data by output commands and receiving data by input commands. Synchronization is achieved exclusively by exchanging data through channels. There are no shared variables.

3. *Procedures* are considered as circular processes, receiving their arguments (input parameters by value) via input commands, and they yield their results (output parameters) by output commands.
   They have no access to nonlocal variables.

4. *Input commands* may be included in guards, thus combining the test for availability with the actual receiving of data. Such a guard is sait to the *ready,* if input is available on the specified channel.

Moreover, it would be most helpful if you stated quite clearly what led you to this view of programming fundamentals. May I suggest that it is above all the speculation that future computers will consist of large numbers of chips (processes) connected by a few wires (channels) and commanding over relatively small private stores (local variables)? This concept differs quite radically from the one which is currently predominant. "Our" computer has few processors, usually one, and a large monolithic store. Hence the concept of shared variables is most natural in its context. It is difficult to understand why you wish to eliminate the shared variable concept, unless you confess that you assume a different view of computers. Perhaps you should select a title like *A conceptual basis for distributed computing.*

Now let me add the comments that I wrote down while reading the paper. *Page* 4: BNF in honour, but why not use charts? - <skip command>: skip what? *Page* 5: Is the statement () := () intended to be legal? P() looks unusual and unmotivated. Instead of *cons* use *pair,* so that also nonlispers may understand. Is the introduction of lists essential to you? It looks rather like a secondary feature for embellishing your notation. I dislike the assignment

(x1, x2, ... , xn) := (y1, y2, ... , yn)

It looks quite nice and innocent in your special example: (x,y) := (y,x). However, if the list is long, and the ys are complicated expressions, the notation is unsuitable, because it is not easy to see the corresponding pairs of x and y. Hence I personally would prefer

(x1 := y1, x2 := y2, ... , xn := yn)

where the comma might indicate concurrency.
Page 6: I believe it was CARH who postulated the most sensible rule of good language design that notation should be suggestive and that there should always be a way to pronounce each symbol when reading a formula. But how do you pronounce the weird s?x and d!y ? Why suddenly insist on infix notation? May I suggest three possibilities that look more appealing to me?

from s **get** x	**get** x **from** s	get(s,x)
to d **send** y	**send** y **to** d	send(d,y)

Are there any process variables? Or else are the source or destination always fixed for

each command? That would seem rather restrictive (unless you are willing to look at channels as hardware wires).
Page 7: When does a value *match* a target variable? When their types are identical? When is a source *terminated* ? When no data are available? But perhaps more will arrive at a later time. The last sentence is a good way of effectively embalming the fact that indices run from 0 to n-1. Either leave the lower bound open or else set it to 1. Why is this syntax so complicated (baroque)? *Page* 8: When saying that each process is disjoint, explain what is meant. Line -7: the "bound variable i" is not a variable in the programmer's sense, but rather a constant for each process. You are a victim of conflict of terminology between mathematics and computing!
Page 9: It is of doubtful advantage to intersperse declarations throughout the program, as they easily submerge. Guarded commands appear inherently as components of alternative and repetitive commands. They can be defined syntactically in isolated fashion, but not semantically. You cannot say what happens, if a guard is false, unless you look at the context. Perhaps chapters 2.4 and 2.5 should therefore be merged.
Page 10: The idea of allowing an input command in a guard is quite central and elegant. But alas also controversial! It smacks awfully much like legalising side-effects. Remember the old controversy about statements like

```
        a[i] := a[i := i+1]
```

I surely remember on which side of the controversy you were! It is courageous to throw old points of view overboard. Is it also honourable, if you do it just because it is momentarily convenient? Of course you see the consequences: a lengthy and confusing elaboration on what a guard now means, and when and where what is allowed or disallowed. If a Boolean expression is true, "it has no effect, and otherwise it fails". Forgive me for failing to understand! If a guard contains several inputs, is it ready if *all* inputs are ready, or if *some* input is ready? Are then all ready inputs executed? The matter would be greatly simplified, if a guard were either a Boolean expression or a single input command.
Page 11: On line 4, the [] should be a | . Line 5: Why use [] instead of Π, which would be consistent with process arrays. Line 8 is badly conceived syntax, because it does not reflect the semantic structure.

```
    <alternative command> ::= [ <guarded command set> ]      if fi
    <repetitive command> ::= { <guarded command set> }      do od
```

Page 14: It was CARH who once critizised me for using the example

```
    var card: array [0 .. 79] of char
```

Quite sensibly he changed the index bounds to 1 and 80 (because otherwise every commercial programmer would doubt your wits.)
Page 18: You said earlier that a guard must not contain two inputs from the same source. So what about X?(x,y:integer) ? " ... which inputs from its user X ... ". Who is X ? A process variable? Such things don't seem to exist, or do they? Bottom page: Could I also write Y!mxn ? What would it mean?
Page 27: " ... gives some surprising results." Which ones? I think allowing X!more within guards would make things worse. It wouldn't be *neat*, perhaps clever.
Page 28: I fully agree with the last paragraph, and sincerely hope that my numerous critical remarks will not discourage you from working on this line. I have given copies of the paper to Butler Lampson, Jim Morris, Jim Mitchell, Chuck Geschke, and Ed

Satterthwaite. You may hear from them directly.

Many thanks also for your comments on my paper. I am grateful that you pointed out that what I say about signals and semaphores looks like advocating the introduction of real time dependencies. This wasn't my intention. I shall revise the paper accordingly and then take the liberty of submitting it again to your scrutiny. I also enclose a note on buffering written just recently, primarily for my own record. Perhaps you care to comment on it too.

I surely hope you don't mind my critique and the many question marks. Your standards of publication are exceedingly high, and I simply try to apply them uncompromisingly. You must be aware that people follow you very faithfully on whatever you say, even if you emphasise that these ideas are speculative and not ripe for standardization. At the 2nd Software Conference in San Francisco last week, there was a whole session with 3 papers on nothing else but Monitors and Conditions, and it is already too late to point out that these ideas may not be the lasting word and that actually the chosen terminology is quite unfortunate.
Well, notice the many question marks in this letter. I am confident you will understand them correctly as input commands! Hence I am

looking forward to hear from you.

Yours sincerely,

Niklaus

Prof. N. Wirth