Research Proposal 1976-7

C.A.R. Hoare

In the year 1976-7, I wish to pursue the hypothesis that the concept of a communicating sequential process is a useful one for programming in many areas of application, including conventional scientific and commercial applications, real time, simulation, construction of compilers and operating systems, as well as for the design of algorithms for execution on networks of interconnected autonomous processors.

In addition, I wish to explore the possibility that this concept is useful in the large-scale structuring of large programs, and in their expression as abstract algorithms.

Finally, some thought must be given to the implementation of the concept in a high level programming language, where there may be a conflict between uniformity and efficiency.

The method of investigation is to select small to medium scale problems from the various application areas, and show how their solution can be expressed in terms of communicating sequential processes. The problems will be taken from the published literature; where they have been used to illustrate the merits of other program structures, such as monitors, processes, coroutines, classes, modules, clusters, etc. If their solution in terms of a single uniform structure is acceptably natural, convenient, and efficient, this will be strong evidence of the value of the uniform structure as a primitive programming concept.

If a programming concept is to be used in abstract programming, it is highly desirable that the abstract program should be correct before any attempt is made to code it into more concrete form. For abstract programs, even the usual ineffective option of program testing is not available. Thus it is necessary to give some attention to problems of establishing the correctness of programs, both by formal and by informal techniques. One of the best formal methods is by use of Dijkstra's "weakest precondition" predicate transformer, which can be used as an aid in program design as well as proof. Dijkstra shows how it can deal with both termination and nondeterminism; its extension to communicating parallel processes would be highly desirable, and would add to the evidence that it is a proper programming primitive.

Although it is far too early to contemplate a practical implementation of the concept, some preliminary design studies for object codes on various types of machine could provide some valuable insights, and validate the soundness, usefulness and generality of the concepts. For example, if the concepts can be adequately implemented on a conventional processor with homogeneous main store, on a multiprocessing machine with single store, on an array of interconnected processors with separate stores, and even perhaps by the hardware of large-scale integrated circuits - this result would be of practical as well as theoretic interest. The actual construction of a "compiler" is not necessary for the investigation of these questions.

In the following sections, particular sub-projects have been described in greater detail; there is no undertaking that the work will be completed by the end of the year.

1. Preliminaries.

The following documents will be required at an early stage in the project:

1.1 The design of a notation suitable for the design and description of abstract computer programs, and which includes a notation for communicating processes.

1.2 The rigorous description of the syntax and semantics of the language in the form of the ALGOL 60 report.

1.3 A graded series of example problems and solutions, taken from a variety of application areas, and suitable as a tutorial introduction to the concepts and notations, and illustrating a variety of methods of using them.

These documents may need to be revised as a result of desirable changes brought to light during the study.

2. Applications.

The language should then be applied to the design of programs in various application areas. In each case, only an abstract program is written, and concessions can be made on efficiency and realism. Thus, quite large projects can be tackled, for example:

2.1 A compiler (perhaps for the language itself), of about the complexity of our PASCAL S (a student interpretive subset of PASCAL).

2.2 An operating system of about the complexity of the T.H.E. system, or the IBM 704 FORTRAN Batch Monitor.

2.3 A simulation package, including simulated time, queues and random numbers, together with a substantial application, for example, simulation of an airport's passenger handling.

2.4 The analysis of a major administrative system, such as the student records of this University. The example should show the application of more rigorous methods to the task of systems analysis.

2.5 The design of the implementation of a communication protocol for communication between machines and devices of varying word length, character codes, message lengths, etc.

In addition to these major projects, the collection of typical small problems and elegant solutions should be increased.

3. Proof Methods.

On the more formal and theoretical side, work can proceed in three directions:

3.1 The selection of small problems (e.g., Dining philosophers, readers and writers) which are frequently used to test structuring and proving methods; and their solution and proof using communicating processes.

3.2 The extension of formal proof methods to parallel programming, and the construction of a complete axiomatic definition for the programming notation.

3.3 The construction of an operational definition of the concept, and the proof of its consistency with the axiomatic. This is required both to ensure the validity of the axioms, and to give a more intuitive description to the programmer of what his program does.

4. Implementation studies.

The implementation studies may include the following:

4.1 The design of an "efficient" implementation for a conventional machine of a highly restricted subset of the language, at about the level of PASCAL.

4.2 The design of an "abstract"implementation, using interpretive techniques, and disregarding economy in space and time.

4.3 The design of an implementation for multiple communicating processors, including design of suitable hardware interfaces for communicating microprocessors.

5. Conclusion.

The expected outcome of this work is to establish that the concept of a communicating sequential process is too general and too primitive for many purposes. In particular

(1) The use of the concept will reveal common special cases which require cumbersome coding, and for which an abbreviated notation would be helpful both to reader and to writer.

(2) The general proof rule may be too complicated, and much simpler rules could be found to cover certain common specific cases.

(3) The fully general implementation may involve inescapable overhead in space and time, and much more efficient methods are available to deal with common special cases.

If it turns out that the common cases referred to under the three headings above are the same, then there will be a strong argument for extending a programming notation in a manner carefully designed to solve all three problems simultaneously.

It is highly probable that this will involve reintroduction into the language of some of the structures (e.g., procedures, classes, monitors), which were intended to be replaced by communicating processes. But as a result of the exercise, the definition of these concepts will have been sharpened and simplified, their proof rules will have been discovered and validated, and their relationship to each other will have been clarified, and their design should completely avoid the arbitrary complexity which is characteristic of many proposed extensions to programming languages; and finally, even the extended language will be very small.