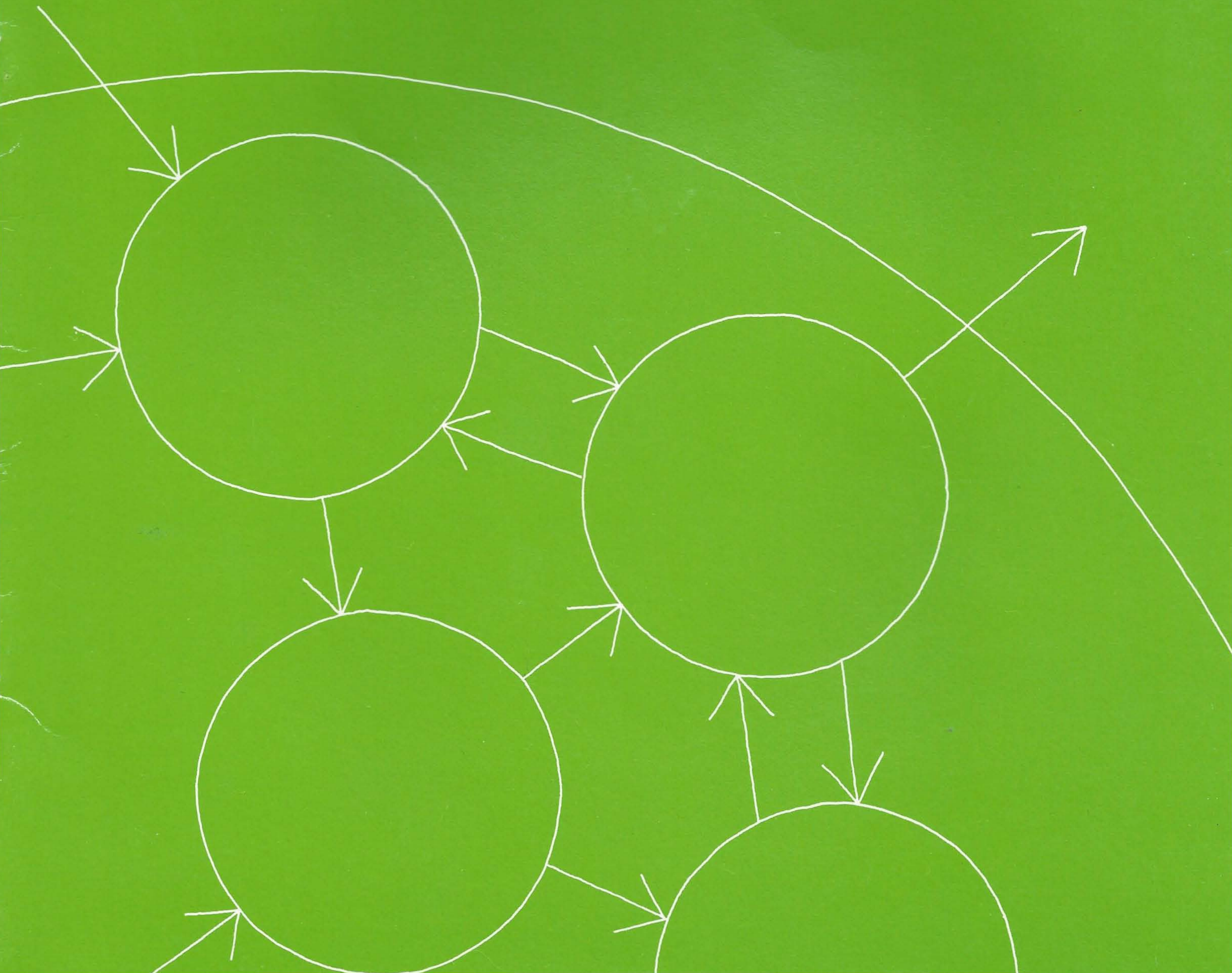
Language overview

occam™



Occam is the new programming language.

Occam is based on the concepts of concurrency and communication. These concepts enable today's applications of microprocessors and computers to be implemented more effectively. They are essential for tomorrow's systems built from multiple interconnected transputers.

Occam is designed for the professional programmer. The language is oriented to interactive use. It enables complex systems to be programmed in a concise and readable form. As a result, programmer productivity is enhanced.

Occam has a formal basis and uses the minimum of concepts. It is easy to understand and easy to compile for a wide variety of microprocessors and computers.

The choice of what is to be omitted from a new language is in practice much more critical than the choice of what is to be added
Niklaus Wirth

Niklaus Wirth is renowned as the designer of Pascal. Together with Dijkstra and Hoare, he has been influential in establishing the principles of good programming practice. Ada, the most ambitious language development ever attempted, is largely based on Pascal. Wirth's quote is from the original 'Green' submission to DoD for the Ada contest. Green won, but Wirth's comment was omitted from the final version.

Programmers are always surrounded by complexity; we cannot avoid it. If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution
CAR Hoare

Professor Hoare, Director of the Programming Research Group at Oxford University, is well known for his concern over the unnecessary elaboration of languages. He fears that systems programmed in complex languages may pose dangers in the real world. He received the Turing Award – the ACM's highest honour for technical contributions to the computing community – for his 'fundamental contributions to the definition and design of programming languages' with work 'characterised by an unusual combination of insight, originality, elegance and impact.'

Tony Hoare has been closely involved in the design of occam, the new language developed at INMOS Limited by David May to provide a better tool for programming microprocessors and future systems. The precursors of occam are well structured languages like Algol 60 and Pascal, system languages like BCPL and C and experimental languages like Hoare's CSP – which established the communication primitives subsequently elaborated in Ada.

Sequential systems will not be adequate for the future. There are an additional four orders of magnitude in computational capability available through concurrent systems
Carver Mead

Carver Mead is the foremost advocate of structured VLSI design. He holds the chair at Caltech endowed by Gordon Moore, Chairman of Intel. Carver Mead was joint winner (with Lynn Conway, manager of VLSI system design at Xerox PARC) of the Electronics 1981 Achievement Award. He is said to have significantly influenced the design of the Motorola 68000 and the Intel iAPX 432.

Concurrency is clearly the key to higher performance systems. Occam is designed to unlock the potential of VLSI by providing the concepts for describing and programming systems containing many interconnected processing elements – the fifth generation systems of the future.

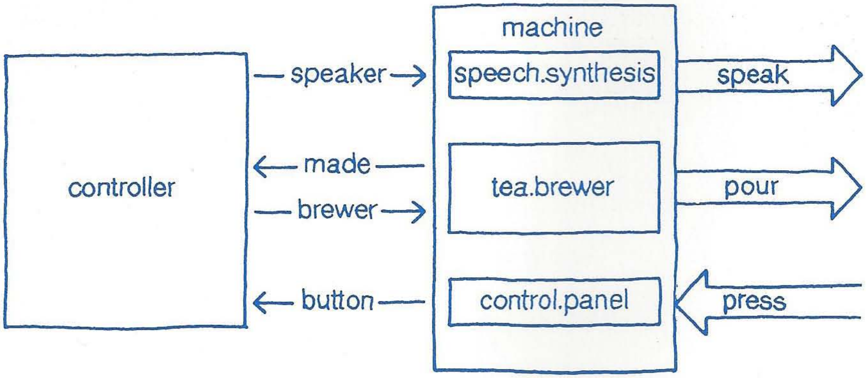
Entia non sunt multiplicanda praeter necessitatem
William of Occam

Occam's Razor – that entities are not to be multiplied beyond necessity – has been the guiding principle in designing this new language. William of Occam was a fourteenth century Oxford philosopher, whose teaching was condemned by the Pope. He is now recognised as having anticipated a basic principle of modern scientific method.

The use of occam is illustrated by the updating of an old fashioned tea maker. This wakes you up in the morning with a traditional message and offers a hot cup of tea. It is also a clock and will make tea at any other time, on request.

The tea maker has a number of units which interact with each other: the tea brewer which makes and pours the tea, a speech synthesiser for saying 'good morning' and telling the time, request buttons and an overall controller.

These units can be represented as a network.



In occam, each of the units is described by a process and each connection by a channel. The processes communicate by sending messages via the channels. A process can be constructed from smaller processes, as in the case of this machine which has a number of parts. Indeed the collection of processes is itself a process in occam, and could be part of some larger system.

This network is represented by defining the channels and the processes. CHAN introduces the channels through which the processes communicate, and the PAR construct causes the various processes to operate concurrently.

```
CHAN speaker, made, brewer, button:
PAR --tea.maker
  ...    --controller
  PAR  --machine
    ...      --speech.synthesis
    ...      --tea.brewer
    ...      --control.panel
```

The controller may do one of three things. Firstly, it may receive a message from the request buttons asking it to make tea, or tell the time.

```
button ? request
  IF
    (request = tea.please) AND NOT brewing
      PAR
        brewer ! make.tea
        brewing := TRUE
    request = time.please
      speaker ! say.time; NOW
```

This inputs a request from the button channel, and uses **IF** to determine whether it is a request for tea, or a request for the time. If it is a request for tea, a message is output to the brewer channel telling the tea brewer to make the tea, and the boolean variable **brewing** is set to prevent further attempts to initiate tea making. If the request is for the current time, a message is output to the speaker channel requesting the speech synthesiser to tell the time, which is the value **NOW**.

Secondly, the controller may receive a message from the tea brewer, telling it that the tea is made.

```
made ? ANY
  SEQ
    speaker ! say.message; tea.made
    brewer ! pour.tea
    brewing := FALSE
```

This uses **SEQ** to stop the tea being poured until the tea maker has said 'tea is made'. Finally, at daily intervals, the tea maker may say 'good morning' and make the tea.

```
WAIT NOW AFTER alarm.time
  SEQ
    alarm.time := alarm.time + one.day
    speaker ! say.message; good.morning
    IF
      NOT brewing
        PAR
          brewer ! make.tea
          brewing := TRUE
```

These individual program sections, each of which is a process, are combined into the complete controller process by declaring the local variables, and by using WHILE and ALT to enable the controller to perform whichever alternative is required.

```
VAR alarm.time, brewing :
SEQ
  alarm.time := 0
  brewing := FALSE
  WHILE TRUE
    ALT
      buttons ? request
        IF
          (request = tea.please) AND NOT brewing
            PAR
              brewer ! make.tea
              brewing := TRUE
          request = time.please
            speaker ! say.time; NOW
      made ? ANY
        SEQ
          speaker ! say.message; tea.made
          brewer ! pour.tea
          brewing := FALSE
      WAIT NOW AFTER alarm.time
        SEQ
          alarm.time := alarm.time + one.day
          speaker ! say.message; good.morning
          IF
            NOT brewing
              PAR
                brewer ! make.tea
                brewing := TRUE
```

Model

Programs are expressed in terms of concurrent processes, which communicate using channels. An obvious implementation of an occam program is a network of microcomputers, each executing one of the concurrent processes. However, the same occam program can also be executed by a single computer sharing its time between the concurrent processes.

Values

The basic data type is a word, which may be used to represent numbers, characters, truth values or bit patterns. Vectors and subscript operations, including record access, are provided. There is a wide range of logical and arithmetic operators for use in expressions.

Structure

Programs are constructed from a small number of primitive processes: assignment, input, output and wait. Processes are combined using the constructors sequence, parallel, conditional and alternative.

Assignment

An assignment may be used to set the value of a variable to the value of an expression.

Communication

A channel provides communication between two concurrent processes. The communication is synchronised, and takes place only when both the input process and the output process are ready; the values being copied from the output process to the input process.

Time

Execution of a process may be related to the passage of time. A wait process may be used to delay execution until a specified time is reached.

Sequence

The component processes are executed one after the other. A sequence construct terminates after the last of its components has terminated.

Parallel

The component processes are executed concurrently. Each component process operates on its own variables, communicating with other concurrent processes using channels. A parallel construct terminates only after all of its components have terminated.

Conditional

The component processes are tested in sequence. If one is ready, it is executed. At most, one of the component processes is executed.

Alternative

One of the component processes is chosen and executed. The alternative constructor chooses the first component process which is ready to be executed.

Repetition

A while construct causes its component process to be executed repeatedly until the result of evaluating a condition is false.

Abstraction

In the construction of a process, a name may be used in place of a component process which is to be used or defined elsewhere in the program. A process definition is used to associate such a name with a process.

Configuration

Configuration is used to meet speed and response requirements by distributing programs over separate, interconnected computers, and by placing and prioritising processes on single computers.

Syntax

Each primitive process and each constructor is represented by a single line of program. The component processes combined by a constructor follow it on successive lines. This makes interactive editors and compilers simple and efficient.

Semantics

The design of occam is based on a formal model which facilitates reasoning about the properties of the language constructs, and the behaviour of specific programs. Each process can be described by an assertion in the predicate calculus, and the composition of processes into networks can be described by the logical conjunction of the assertions describing each process.

Occam programming manual

This is a tutorial introduction and reference manual for the first release of the language.

Occam evaluation kit

This is a complete portable software kit to provide programmers with the opportunity to experiment with occam.

The kit is inexpensive and comprises a compiler and editor together with tutorial examples on a floppy disk. It includes manuals for occam and the compiler itself.

The kit is based on the UCSD p-System version IV and compiles occam into p-code. It can be used on a wide range of computers, from the Apple II to the VAX.

Occam development software

For development of applications in occam, a range of support products is provided. These include compilers, together with appropriate tools, optimised for occam program development, which are intended to run on a variety of widely available hosts, generating target code for a variety of processors.

and occam are trademarks of the INMOS Group of Companies
UCSD p-System is a trademark of the Regents of the University of California

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents, trademarks, or other rights of the INMOS Group of Companies.

Designed by HSAG
Printed in England by Syon Print Limited

January 1984

72-OCC-001 000

inmos

INMOS Limited
Whitefriars
Lewins Mead
Bristol BS1 2NP
UK
Telephone (0272) 290861
Telex 444723

INMOS Corporation
PO Box 16000
Colorado Springs
CO 80935
USA
Telephone (303) 630 4000
Telex 910 920 4904

INMOS SARL
Immeuble Monaco
7 rue Le Corbusier
SILIC 219
94518 Rungis Cedex
France
Telephone (1) 687 22 01
Telex 201222

INMOS GmbH
Danziger Strasse 2
8057 Eching
West Germany
Telephone (089) 319 10 28
Telex 522645

DATA12523