'COLD' stands for 'Common Object-oriented Language for Design', but do not be misled into thinking it is object-oriented. It combines 'a wide variety of notations for property-oriented and model-oriented specification, ... in equational style, pre- and post-condition style, inductive definitions, algorithmic definitions in functional as well as imperative style'. It is, in a word, the PL/I of formal development languages.

'The book is self-contained and instead of going into the formal semantics of the language, it will convey a working knowledge of the meaning of the language constructs via the examples, the explanations, and the pictures'. Your reviewer has to report that this working knowledge was not conveyed to him. Surely those that use notations should be able to give their translations into English, and not merely hope that we will latch on to what they mean.

A *sort*, we are told, is a value set. Then (p. 89) we are shown an array update definition with input `a, n, i`: `a` is of the sort Array, `n` an index, and `i` an item. 'a' is an 'object name'—not (p. 114) the name of an object, but a 'logical variable' (an *unvarying* pronoun) 'of a static nature'. This is, `a` is an (unchangeable) array *value*. The post-condition of the array update function is `val(a,n) = i`, i.e. `i` is item `n` of `a`. Surely this implies that `a` has changed? But the specification of the update says that only the `val` function, `val(a,n)`, can change. So perhaps `a` is unchanged, but `val` is changed? Or perhaps not: the sort Array is defined as *variable*, i.e. not a sort at all.

The book appeals, of course, not to the missing informal interpretation of its notation, but to the deliberately omitted formal semantics, which 'guarantee that the design can be rigorously analysed'. The motivating case study describes a vending machine: no surprise there, except that it is the weirdest such machine ever analysed; for instance, it is informed individually about each valid coin. What is more, insertion of a valid coin with a value less than the price of any selection stalls the machine; and, while the initial statement of requirements says that each product has a price, the formal specification allows multiple selections, at different prices, for the same product.

Rigorously analysed, not: see your reviewer's comments on other formal development language texts *passim* and *ad nauseam*.

ADRIAN LARNER
De Montfort University

MICHAEL G. HINCHEY AND STEPHEN A. JARVIS
Concurrent Systems: Formal Development in CSP. McGraw-Hill. 1995. ISBN 0-07-707649-4. £19.95. 181pp. softbound.

Parallel systems are notoriously complicated, and the myriad possibilities for interaction between components makes such systems difficult to understand and reason about. Communicating Sequential Processes (CSP) is an abstract language for describing concurrent systems, together with an underlying theory crafted to enable specification, design, development, and verification within a formal framework grounded in mathematics. It controls complexity by abstracting away details of components' internal structure, focusing on the communication patterns between them. The language was first proposed in 1978, and has proved extremely successful as a formal method. The infrastructure required to support its uptake in industry is developing: there is now tool support for CSP analysis and verification, and there are a range of courses available. However, in the 10 years since Tony Hoare's classic book *Communicating Sequential Processes* there has been a scarcity of books on this subject. This is a gap that Hinchey and Jarvis aim to fill.

This slim book, described on the jacket as 'a tutorial introduction with comprehensive reference material' is aimed at both the student and the research community. The preface indicates that the tutorial material consists of Chapters 1, 2 and 3 (introduction, language, semantic models), and that the rest of the book is intended as a reference. However, Chapter 3 consists essentially of a large number of laws, and its material is not illustrated or motivated, so it cannot really be considered as tutorial material. The structure of the book is as follows: it covers the language of CSP, its semantic models, variants, laws, elements of CSP style, a case study, the `occam` programming language, and finally refinement. This would appear to cover the most important aspects of CSP, though there are some startling omissions, the most serious being the lack of any treatment of the semantics of recursion, a topic that is absolutely central to CSP.

It is axiomatic that any academic text must be technically sound, particularly one intended as a reference. This is especially important in the area of formal methods where emphasis is placed on its mathematical basis. In addition tutorial material should be written extremely carefully, since it must withstand intense scrutiny by students wishing to gain an understanding of the subject: precise and careful explanations are essential, and proper and appropriate use of terminology is expected. It is also common for tutorial material to contain exercises. Regrettably, this book has severe deficiencies in all these areas, which renders it unsuitable as a tutorial or student text, and especially unsuitable as a CSP reference.

The first severe problem with the book concerns the extremely high number of technical errors. A single technical flaw in an otherwise excellent book may be forgiven, but Chapter 5 alone (Laws of CSP) contains *over 40* laws which are incorrect, incomplete, or simply nonsensical. An example is the meaningless assertion that the parallel operator is transitive. The errata slip currently appearing with the book addresses only one of these laws, replacing one incorrect law ($P|||P = P$) with another ($P|||P \neq P$). Using the laws of this chapter it is possible to prove, in more than one way, that $P = Q$ for

any processes P and Q. It is also possible to prove that $P \neq Q$ for any two processes, including the special case $P \neq P$ for any P! The rest of the book also contains its share of errors, including some elementary mistakes in the tutorial examples and in the chapter on semantic models.

The other main problem with this book is that many explanations throughout the text are confused or incorrect. It would be pointless to list all those I found, and instead I will confine myself to three typical examples from the tutorial part of the book. Firstly, Section 2.2.2 on 'Process definition' begins with the extraordinary assertion that 'a process is defined recursively in the format $(e \rightarrow P)$'. The rest of that section is also nonsense, and does not even address its topic on how processes are defined. A second example of poor explanation is in Section 2.2.4 (Standard processes) where 'chaos' or 'bottom' is described as denoting 'livelock or deadlock; it is a process that can do or fail to do anything, and at all costs must be avoided in specifications'. This will not make much sense to readers unfamiliar with the theory. A final example of confused explanation may be found in Section 3.2 (Traces) where it is claimed that a specification is strengthened by introducing hiding, but weakened by introducing interleaving. This does not make sense even with respect to the authors' own explanation, and cannot but confuse the reader. As well as poor and incorrect explanations, standard terminology such as 'deadlock', 'livelock', and 'angelic non-determinism', and even terms such as 'total-ordering' are used incorrectly, or confusingly.

The treatment of refinement at the end of the book is superficial and unsatisfactory. The definition of refinement in CSP is given, but there is no guidance presented on how or where it is to be used, except the comment that it 'is not a straightforward process'. Given the subtitle of the book and the central role that refinement plays in formal development, I would have expected refinement to be treated in greater depth than it is here. Furthermore, no notion of a refinement relation between CSP and `occam2` or Ada is presented. Instead we find an ad-hoc translation of the case study of Chapter 7 from CSP to `occam2`. The section on refinement to Ada boils down to informal rules for translating input and choice constructs into Ada. This contrasts with the jacket description, which claims that 'practical implementation in both Occam and Ada 9X is discussed in depth'.

There are some good aspects of the book. It has been very well typeset. The case study in Chapter 7 illustrates some aspects of verification in CSP, though I was disappointed with the number of typographical errors and use of undefined notation, and could not help feeling that this chapter would be more suited to a journal. The CSP and `occam` bibliography is very extensive, though it will require some work to keep it up-to-date in subsequent editions.

In conclusion, this book contains an extraordinarily high number of serious technical mistakes throughout which render it unusable as a reference. Furthermore, many of the explanations are confused or incorrect, which also makes it unsuitable for teaching. Sadly, the computer science community must wait a little longer for another good book in this area.

S. A. SCHNEIDER
Royal Holloway, University of London