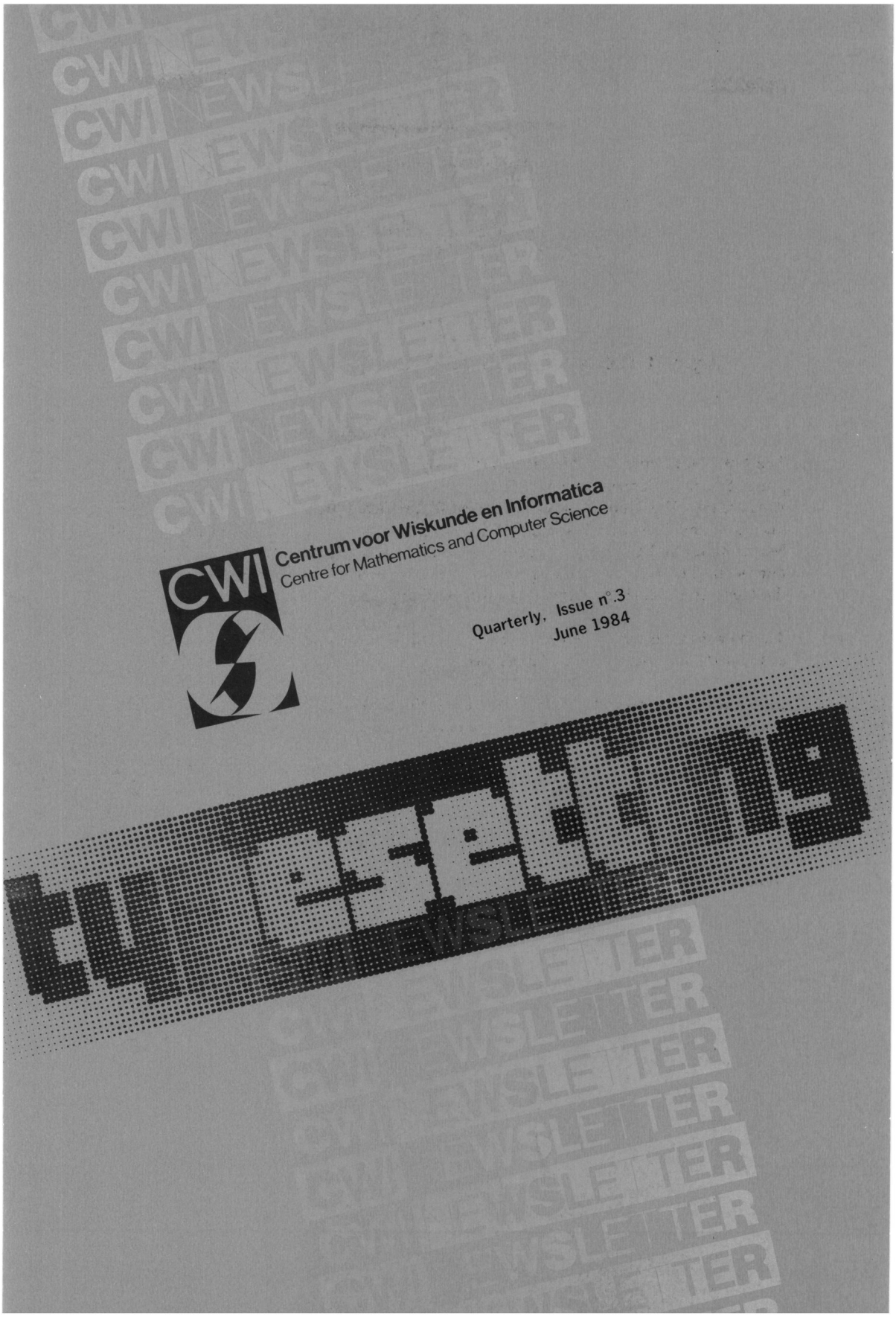

CWI
Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science
Quarterly, Issue n°.3
June 1984
NEWSLETTER

CWI NEWSLETTER

Number 3, June 1984

Editors

Arjeh M. Cohen Richard D. Gill Jan Heering

The CWI Newsletter is published quarterly by the Centre for Mathematics and Computer Science (Centrum voor Wiskunde en Informatica), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. The Newsletter will report on activities being conducted at the Centre and will also contain articles of general interest in the fields of Mathematics and Computer Science, including book reviews and mathematical entertainment. The editors encourage persons outside and in the Centre to contribute to the Newsletter. Normal referee procedures will apply to all articles submitted.

The Newsletter is available free of charge to all interested persons. The Newsletter is available to libraries on an exchange basis.

All correspondence should be addressed to: *The CWI Newsletter, Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.*

ISSN 0168-826x

Contents

Centre for Mathematics and Computer Science

Centrum voor Wiskunde en Informatica

The *B* Programming Language and Environment

A new programming language and environment for personal computing designed at the CWI

by Steven Pemberton

New computers, old languages

It is a common observation that the latest personal computers are very powerful. Certainly more powerful and more capacious than many of the previous generation of 'large' computers. Thus, it is quite feasible that many personal computers will spend most of their time idle, not from lack of use, but from under-filled capacity. And this is not because of delusions of grandeur on the part of the purchaser: the central processor, the part that is responsible for much of the measure of speed in a computer, is but a tiny part of the cost of a modern computer; there is no economic (or other) advantage in using a slower processor.

It is therefore rather surprising to realise that most programming on personal computers is done with programming languages that are for the most part 15 to 20 years old, languages designed for computers of a previous generation (or before). In particular, whatever personal computer you buy, you can be sure that the one language that the manufacturer has supplied for you is BASIC, a language designed in the mid-sixties, and which has been described as "an adaptation to early and very marginal computer technology" [1]. Thus you have the strange situation of people programming the computers of the eighties with a language of the sixties, a language unable to take advantage of the increased capabilities of the newer machines.

Two main advantages of BASIC are that it is interactive, and that it is simple. Interactiveness is the ability to type in and run a program immediately without going through any intermediate stages like translating the program into machine code, and to correct a program and re-run it immediately. Strictly speaking, this is a property of an *implementation* of a language, and not of the language itself, since it is in principle possible to make an interactive implementation of any language, or a non-interactive version of BASIC. But notwithstanding, a language usually has features that orient it more or less towards interactive implementation, and BASIC is usually implemented interactively, and most other languages are not.

Simplicity is a property often claimed for a programming language or system, without the term being properly clarified. For there are two, in some ways conflicting, senses to the word. You can have definitional simplicity, where there are only a few concepts, and you can have what might be called psychological simplicity, where the concepts are closest to your needs. A

couple of examples.

- The idea of a Turing machine is specifically to give the simplest model of a computational machine. No one however would consider it simple to program for.
- Boolean algebra can be expressed using a single operator 'not and'. However, no one would consider the expression

  (q nand (q nand q)) nand ((p nand p) nand (q nand q))

  as simpler than the equivalent

  not (p or q)

  despite the larger number of concepts in the second.

The simplicity of BASIC is actually a definitional one. It is easy to implement, and has few concepts to be learnt, but once learnt, it only remains easy to use for very small programs. Beyond that it is like cutting your lawn with a pair of scissors.

In schools

BASIC is also the principle programming language taught in schools. Apart from the perceived simplicity, it is usually the case that schools simply could not afford machines large enough to run anything other than BASIC. That situation will change quickly enough with the coming generation of cheap large computers, but the risk is that schools will continue using BASIC from sheer momentum, and perceived 'investment' in the language (a common barrier to change outside schools too). Risk, because BASIC has little to recommend it for educational use (and I say this as someone who has had to teach students coming from schools where they have learnt to program with BASIC). Just as with teaching yourself to type with one finger, where if you then want to learn to type properly you must first get rid of your old habits, BASIC's paucity of structuring facilities means that much time has to be dedicated to learning ways of getting round its expressive poverty, and the student ends up learning bad habits that must only be unlearnt in order to progress to other languages.

Well, either that, or they learn nothing at all, for quoting from [1] again: "For the vast majority of students who learn to program in BASIC, learning to program means learning a few set pieces of programming from a textbook and devoting the rest of their time at the terminal to playing computer games."

B

B is a programming language being designed and implemented at the CWI, together with an integrated programming environment (it should be noted that '*B*' is just a working title; the system will gain its definitive name when the language is frozen.) It was originally started in 1975 in an attempt to design a language for beginners as a suitable replacement for BASIC. In the intervening years the emphasis of the project has shifted from "beginners" to "personal computing", but the main design objectives have remained the same:

- simplicity;
- suitability for interactive use;
- availability of constructs for structured programming.

The design of the language has proceeded iteratively, and the language as it now stands is the third iteration of this process. The first two iterations were the work of Lambert Meertens and Leo Geurts of the CWI (then called the Mathematical Centre) in 1975-6 and 1977-9, and were more in the line of definitionally simple, being easy to learn and easy to implement.
In the third iteration of *B*, designed in 1979-81 with the addition of Robert Dewar of New York University, it became psychologically simple: it is still easy to learn, by having few constructs, but is now also easy to use, by having powerful constructs, without the sorts of restrictions that professional programmers are trained to put up with, but that a newcomer finds irritating, unreasonable or silly.

However, *B* is not just a language, but a complete programming environment. Traditional computer use for programming involves not only learning the programming language, but also a whole host of sub-systems and their commands, often completely separate and non-cooperating. *B* on the other hand shows one face at all times to the user, and it is not necessary to learn anything outside the *B* system.

Simplicity

B has just two basic data types: texts and numbers, and three ways of combining other values: compounds, lists and tables.

Numbers have two surprises for the seasoned computer user: firstly, on the basis of the maxim of no restrictions, numbers may be as big as wanted (within, of course, the physical limits of the computer's available memory); you may just as easily calculate 10^{200} as 10^2. In fact a Dutch newspaper recently dedicated a whole page to printing the value $2^{132049}-1$ (the current largest prime), which had been calculated with the *B* program

```
WRITE 2**132049-1
```

which, though it took a while to run, nevertheless produced the final answer consisting of more than 37000 digits.
Secondly, as long as it is possible, numbers are always kept exact, even

fractional numbers. Thus, as long as you use exactness-preserving operations, such as addition, subtraction, multiplication, and even division, a number is calculated exactly. Operations such as taking the square root cannot of course produce an exact result in general, and so in this case an approximate number results, rounded to some length.

Mathematicians and computer scientists alike are often surprised by the following little program that uses the properties of arithmetic in *B* to calculate the digits of π by evaluating the continued fraction

$$\cfrac{4}{1+\cfrac{1}{3+\cfrac{4}{5+\cfrac{9}{\cdots+\cfrac{k^2}{(2k+1)+\cdots}}}}}$$

```
HOW'TO PI:
    WRITE "3."
    PUT 3, 0, 40, 4, 24, 0, 1 IN k, a, b, c, d, e, f
    WHILE 1 = 1:
        PUT k**2, 2*k+1, k+1 IN p, q, k
        PUT b, p*a+q*b, d, p*c+q*d IN a, b, c, d
        PUT f, floor(b/d) IN e, f
        WHILE e = f:
            WRITE e<<1
            PUT 10*(a-e*c), 10*(b-f*d) IN a, b
            PUT floor(a/c), floor(b/d) IN e, f
```

Texts are strings of printable characters. Unlike many other languages *B* has a full range of operations on texts, such as joining texts together, replicating them, taking sub-strings and so on. Just as with all types in *B*, there is no maximum limit imposed on the size of a text, nor does the size have to be declared in advance.

Compounds are the way of making tuples, or records as they are called in some other languages, for instance for making complex numbers:

```
PUT 0, 1 IN z.
```

Lists are sorted collections of elements, again unrestricted in size. The elements of a given list must all be of the same type, but otherwise, and this is another surprise for the experienced programmer, may be of any type. Thus you may have lists of texts, numbers, compounds, lists of other lists, and so on. Elements may be duplicated; thus a list is a multiset or bag. You can insert elements, delete elements, find out if an element is present, find the size of a list, and so on. Here is a program that uses lists of numbers and the sieve method to calculate primes.

```
HOW'TO SIEVE'TO n:              \name is SIEVE'TO
    PUT {2..n} IN set           \set to be sieved
    WHILE set > {}:             \repeat indented part
        PUT min set IN p        \smallest member
        REMOVE'MULTIPLES        \refinement, see below
        WRITE p
REMOVE'MULTIPLES:
    PUT p IN multiple
    WHILE multiple <= n:
        IF multiple in set: \present in set?
            REMOVE multiple FROM set
        PUT multiple+p IN multiple
```

The last type is the table. Tables are mappings from values of any one type onto values of any one other type, and as such are a generalisation of arrays in other programming languages. Standard programming languages only allow you to map integers (and sometimes a few other similar types) onto other types. It is one of the biggest surprises, bordering on disbelief, for experienced programmers, that you may use any type for the indexes of *B* arrays. Thus if you want mappings from texts to lists, or tables to numbers, or tables to other tables, all are possible.

As an example, consider representing a directed acyclic graph as a mapping from nodes to lists of nodes:

```
PUT {[0]: {3}; [3]: {7; 8}; [7]: {8}; [8]: {}} IN graph
```

You can write a test to see if two elements are connected as follows:

```
TEST a connected'to b:
    SHARE graph
    REPORT b in graph[a] OR indirectly'connected
indirectly'connected:
    REPORT SOME e IN graph[a] HAS e connected'to b
```

and then write

```
IF 0 connected'to 8: ...
```

Other examples of surprises for the seasoned programmer that the newcomer will find unremarkable are in the READ command. If a running program is to input a value from the user, the READ command is used. In traditional languages, you can only read numbers and characters, and furthermore only constants of these types. However, in *B*, any type of value may be read, and further, any expression may be typed as input. This includes the use of variables, functions and so on.

It may be remarked from the above examples that although the data types of *B* are unusual, the kind of commands, or statements, are rather familiar.

There are the usual input and output commands, the assignment command, if and while commands, and so on. In fact the only unusual feature is the refinement, such as `REMOVE'MULTIPLES` and `indirectly'connected` in the above examples. These explicitly support the idea of 'step-wise refinement', so often practised in programming, but so rarely supported by programming languages.

As you can see, *B* has a small set of rather powerful data types. This is in comparison with most other languages that supply you with a number of *low-level* tools, that you must then use to build your own high-level tools.

B does it just the other way round. You get high-level tools which you can use for low-level purposes if you wish. For instance:

- If the numbers you use in a program are all less than a certain limit, you don't have to do anything special in *B*. In other languages, if your numbers go higher than a certain limit, you must write your own numerical package.
- In traditional languages, if you wish to use sparse arrays, you must write a package to implement them using the non-sparse arrays in the language. In *B*, sparse arrays (i.e. tables) are the default, but you can use them in a non-sparse way without extra effort.
- Traditional languages sometimes supply a pointer type, which you can then use to create data space dynamically. In *B*, data space is automatically dynamic. Furthermore, if you study the use of pointer types in other languages, you will see that they are almost always used for sorting and searching purposes. *B* supplies these sorting and searching facilities as primitives. If you still need to use pointers, you can represent them using *B* tables, but with additional advantages, for instance that you can print tables while you cannot print pointers.

Another feature of the simplicity of *B* lies in its environment. Global variables are permanent, in the sense that they remain not only while the programmer is working at the computer, but even after switching off, and returning later. Thus variables may be used instead of 'files' in the traditional sense, and so there is no need for extra file-handling facilities in the language. Since *B* variables are dynamic, and unrestricted in size, using them in place of files causes no difficulties. Quite the reverse in fact, since you now have the powerful data-types of *B* at your disposal, allowing random, and indeed associative, access to the contents.

Compare the following two programs in *B* and Pascal for counting the number of characters in a text file. In *B*:

```
PUT 0 IN size
FOR line IN document:
    PUT size+#line IN size
WRITE size
```

In Pascal:

```
program count(document, output);
var document: text;
    c: char;
    size: integer;
begin
     reset(document);
     size := 0;
     while not eof(document)
     do begin
             while not eoln(document)
             do begin
                     read(document, c);
                     size := size + 1
                end;
             readln(document)
        end;
     write(size)
end.
```

This, I contend, speaks for itself. In fact, these two programs illustrate clearly how compact *B* programs turn out to be. It is my experience that *B* programs are around a quarter or a fifth of the size of their equivalent Pascal programs (this comparison includes a 1000 line Pascal program which resulted in a 200 line *B* program). The ratio against BASIC would be even further in *B*'s favour. This clearly has consequences for programmer efficiency, especially as programmer effort is proportional not to program length, but a *power* of program length. Brookes [2] reports that empirical studies show this power to be around 1.5. This would imply that *B* is something like an order of magnitude easier to use than traditional languages.

The other side of this efficiency coin is that, because of its higher level, the language is no longer so straightforward to implement, and because it is interpreted, a given program in *B* will not run as fast as an equivalent program written in a non-interactive language. However, we have already noted that the personal computers of the new generation are so powerful that they will spend a large proportion of their time idle. This trade-off of computer time against programmer time is more than reasonable in view of this excess computational capacity. Furthermore, there are other trade-offs involved when comparing non-interactive languages with interactive ones, such as the absence of a translation phase in an interactive language.

Comparing *B* with BASIC on this score is another matter. BASIC implementations tend to be slow anyway, yet many people are willing to accept this slowness in return for interactive access. For instance, Bentley reports [3] that BASIC on an (apparently large) personal computer he bought ran at 100 instructions per second. This is even slower than the first commercially

produced computers of the 1950's which ran at 700 instructions per second!

Of course, higher-level commands like those of *B* take more time individually, but on the other hand fewer have to be executed to do the same job and more work is done at the (faster) system level than with a lower-level language. The combined effect depends on the application: simple programs — which take little time anyway — will generally run slower, but more complicated tasks may well run faster than if coded in a lower-level language.
But still, even if a given program in *B* runs slower than acceptable (for instance in the case of a commercial application which must run as fast as possible on a slow micro-computer), the programmer efficiency of *B* still makes it a good choice for the prototyping phase of a project.

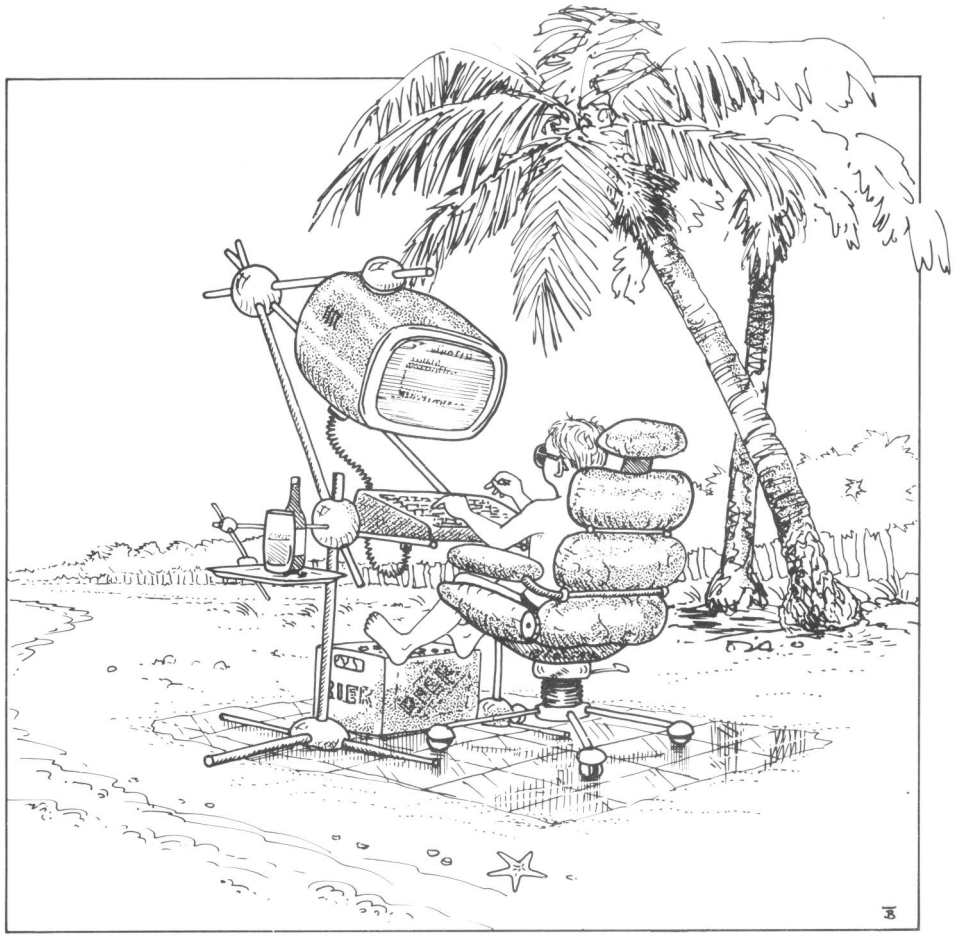
Interaction

Just as with BASIC, any command typed straight in at the terminal will be executed immediately. Thus you may use all the features of *B* as a sort of high-grade calculator:

```
WRITE root 2
1.41421356237
```

Furthermore, since user-written programs are called in exactly the same way as built-in *B* commands, much of the need for a separate command language so often found on computers disappears: as already pointed out, variables serve as files, and since programs are just the equivalent of subroutines in other languages, parameters can be passed to programs using the normal parameter passing mechanism of subroutines. In most systems, if parameters can be passed to a program at all, it is usually with a completely different mechanism.

One of the demands on an interactive language is that typing be minimised, since so much time is spent at the keyboard. One solution to this used by some interactive systems is to use abbreviated commands, but this generally results in very cryptic looking commands. *B* solves this by having a dedicated editor that knows much about the syntax and semantics of *B*. As an example, consider the above `WRITE` command. This is the second most used command in *B* (the first is `PUT`) and so when you type a `W` as first letter of a command, it is more than likely that you want a `WRITE` command. To this end, the moment you do type such a `W`, the system immediately suggests the rest of the command to you, by showing `WRITE` on the screen. If you do want a `WRITE` you can then press the 'accept' key and the system positions you so that you can type in the expression that you want to write. If you don't want a `WRITE`, but a `WHILE` say, then you just ignore the suggestion and type the next character, an `H`. The system then changes the suggestion to `WHILE`, and so on. This also works for the commands you write yourself (such as the `SIEVE'TO` unit defined earlier). The system also knows about things like matching brackets and supplies these for you. Thus certain typical sorts of typing error are just not possible in *B*.



The editor is also used in place of many functions that would normally be performed by a separate command language. For example, it is possible to edit the list of units (procedures and functions) that you have: if you delete an entry in the list using the editor, the corresponding unit disappears. Another feature of this is that you may edit the list of commands that you have typed in and executed: this then causes the changed commands to be re-executed as if you had typed the commands in in that way in the first place.

Another feature of the interactiveness of *B* is that declarations are not used. BASIC users usually perceive this as an advantage because it means less typing. Users of other languages, such as Pascal on the other hand, accept declarations on the grounds that they allow type inconsistencies and other similar errors to be detected before the program is run, therefore reducing the time taken to get a program correct.

B supplies the advantages of both, by inferring the types of variables from the way they are used (for instance, if you say `1+a`, `a` must be a number), and checking that all such uses are consistent. Furthermore, inconsistencies can be spotted as the command is typed in, increasing the interactive feel of the language.

Teaching

It is our feeling that *B* is well suited for teaching purposes. The availability of program and data structuring facilities, including support for step-wise refinement, means that students are less likely to adopt bad habits. More importantly, because of *B*'s high-level, a student can quickly get to a level of competence to produce useful working programs, rather than just trivial exercises.

B is currently being taught at a Dutch school in collaboration with the CWI to several classes of different school types. However, this has only recently started.

Implementation

Part of the effort at the CWI is to create an implementation of *B*. We have had a pilot implementation running for several years, and have now just finished a release version.

The original *B* implementation was written in 1981. It was explicitly designed as a pilot system, to explore the language rather than produce a production system, and so the priority was on speed of programming rather than speed of execution. As a result, it was produced by one person in a mere 2 months, and while it was slower than is desirable, it was still usable, and several people used it in preference to other languages.

The second version, just completed, is aimed at wider use, and therefore speed and portability have become an issue, though the system has also become more functional in the rewrite. Like the pilot system, it is written in the programming language C and was produced by first modularising the pilot system, and then systematically replacing modules, so that at all times we had a running *B* system. It was produced in a year by a group of four.
This implementation runs on larger machines that run Unix* (with at least 128 Kbytes of main store) and is freely available for non-commercial use at the cost of the media.

One of the features of the implementation is the way values are implemented, based on the scheme of Hibbard, Knueven and Leverett [4]. Here, each value includes a count of how many copies of it there exist. When a value has to be copied, instead of copying the whole object, only a pointer to it is copied, and the associated counts are updated. When a count reaches zero, the value is disposed of.
When a value has to be modified, such as by inserting a value in a list, if its

* A trademark of AT&T Bell Laboratories

count is greater than one then the value must first be 'uniquified' by (really) copying it to a fresh area of store (actually only part of it is usually copied because, for instance, if the list is a list of tables, the tables need only have their counts updated, since they are not changed themselves.) Already unique values are modified *in situ*.

This scheme has one outstanding feature, that the cost of copying is independent of the size of the value. Therefore there is a size of value above which this method becomes cheaper than ordinary copying. This critical size is rather small, and since *B* values easily become large, it is advantageous. Furthermore, PUT commands are typically the most executed sort of command in programs, and so it makes sense to choose a method that favours them.

The implementation uses B trees (no relation) [5] to represent texts, lists, and tables. These are a form of balanced trees, and the cost of modifying an element is only O(log n). Instead of having to copy a whole level of the value on modification, only a sub-section of the tree needs to be copied.

We have been lucky to receive, through the generosity of IBM Netherlands, an IBM Personal Computer, and we are now busy transporting the implementation to it.

The Future

There will be one final polishing of the language before it is finally frozen, to clear up a few odd corners. However, most work on the system is now focusing on the environment, for instance to try and do for graphics and data-entry what up to now we have done for programming.

Further information

For more details of the language, refer to reference [6]. There is a *B* newsletter published at the CWI, with further details of the *B* environment. An annotated list of *B* publications is given in an appendix.

Conclusion

The time has come that personal computers have so much power that a new programming language is called for to take advantage of that power. *B* has been designed with just such an aim, to satisfy the needs of people who, while not being professional programmers, nevertheless need to use personal computers. Although the language was designed with these non-professionals in mind, it turns out to be of interest to professionals too: several people in our institute now use it in preference to other languages.

References

[1] Seymour A. Papert, *Computers and learning*, in M.L. Dertouzos (ed.), *The Computer Age*, MIT Press, 1979, 73-86.

[2] F.P. Brookes, *The Mythical Man Month*, Addison Wesley, 1975.
[3] Jon Bentley, *Programming Pearls*, Comm. ACM, **27** (1984) 3, 181-184.
[4] P.G. Hibbard, P. Knueven, B.W. Leverett, *A Stackless Run-time Implementation Scheme*, in R.B.K. Dewar (ed.), *Proc. 4th Int. Conf. on Design and Implementation of Algorithmic Languages*, Courant Institute, New York, 1976, 176-192.
[5] T. Krijnen & L. Meertens, *Making B Trees Work for* B, Report IW 219/83, Mathematical Centre, Amsterdam, 1983.
[6] Leo Geurts, *An Overview of the* B *Programming Language*, SIGPLAN Notices, **17** (1982) 12, 49-58.

Appendix: Available publications about B

A number of publications about *B* are currently available. Unless otherwise stated, all are published by the CWI; an order form can be found at the end of this newsletter.

An Overview of the B *Programming Language, or* B *without Tears,*
Leo Geurts, CWI report IW 208/82, 11 pages.

This is the first place to go if you want to know more about *B*. Also published in *SIGPLAN Notices* **17** (1982) 12, 49-58.

Draft Proposal for the B *Programming Language,*
Lambert Meertens, CWI, ISBN 90 6196 238 2, 88 pages.

This book is a specification of the whole language, though rather technical for the casual reader. It also contains some thoughts on a *B* system. A part of the book, the *Quick Reference*, also appeared in the *Algol Bulletin* 48 (August 1982).

Description of B,
Lambert Meertens & Steven Pemberton, CWI note CS-N8405, 38 pages.

This is the informal definition of *B* promised in the Draft Proposal. It aims to provide a reference book for the users of *B* that is more accessible than the somewhat formal Draft Proposal. While it is not a text book, it should also be useful to people who already have ample programming experience and want to learn *B*.

Computer Programming for Beginners — Introducing the B *Language (Part I),*
Leo Geurts, CWI note CS-N8402, 85 pages.

This is a text-book on programming for people who know nothing about computers or programming. It is self-contained and may be used in courses or for self-study. The focus is on designing and writing programs, as opposed to entering them in the computer, and so on. It introduces the language, and how to write small programs. Part 2, which will appear later this year, will treat the language, and programming, in greater depth.

A User's Guide to the B *System,*
Steven Pemberton, CWI note CS-N8404, 10 pages.

A brief introduction to using the current *B* implementation.

B *Quick Reference Card.*

A single card containing all the features of the language, the editor, and the implementation, for quick reference when using *B*.

An Implementation of the B *Programming Language,*
Lambert Meertens & Steven Pemberton, 8 pages.

This gives an overview of the implementation and some of the techniques used in it. Not published by the CWI. To appear in USENIX Washington Conference Proceedings (January 1984).

Making B Trees Work for B,
Timo Krijnen & Lambert Meertens, CWI report IW 219/83, 13 pages.

This describes a method of implementing the values of *B*. It is rather technical.

Incremental Polymorphic Type-Checking in B,
Lambert Meertens, CWI report IW 214/82, 11 pages.

B allows you to use variables without having to declare them, and yet gives you all the safety that declarations would supply. This paper describes how this is achieved, but is *very* technical. Definitely not for the faint-hearted. Also published in *Conference Record of the 10th Annual ACM Symposium on Principles of Programming Languages*, ACM, 1983, 265-275.

On the Design of an Editor for the B *Programming Language,*
Aad Nienhuis, CWI report IW 248/83, 16 pages.

Gives an overview of the design of a pilot version of the *B* dedicated editor.

On the Implementation of an Editor for the B *Programming Language,*
Frank van Harmelen, CWI report 220/83, 18 pages.

Gives details of a pilot implementation of the *B* dedicated editor.

Towards a Specification of the B *Programming Environment,*
Jeroen van de Graaf, CWI report CS-R8408, 23 pages.

This report contains an informal description and a tentative specification of the environment.

The B *Newsletter,*

This is produced to keep interested parties in touch with developments in the language and its implementation, and to provide a forum for discussions. Issues 1 and 2 have already appeared.

Typesetting at the CWI – Part 1

by Jaap Akkerhuis

Introduction

In this article I will discuss how the typesetting of various publications, like the Newsletter you are reading, is done at the CWI.

Since the founding of the Institute in 1946, the Centre has published reports. The typing and printing of its publications has always been done "in house". This made it possible to get a quick turnaround time between the writing and printing of a report. In this way it is also possible to have a close interaction between the author and the typist, which allows the author to have a good control not only over the contents but also over the appearance of his publication.

It was recognised at an early stage that a computer could provide some help with the composition of a publication. This lead to a program, called TEKSTSCHAAF [1], on the Electrologica X8 computer written by D. Grune. This system allowed you to format a text and it would automatically provide page headings, page numbering, *justification**, etc. There was a nice mechanism to prevent *widows*†, but the most interesting aspect of the program was the way the justification was done. Most programs do justification on a line by line basis, but in this system a couple of lines (normally 3) were considered before the final adjustment took place. Where the extra spaces were added in the line was dependent on an "elastic band" between the words. The stretchability of this band depended on certain conditions that could be changed by the user, and some analysis of the (Dutch) text. A similar approach, using stretchable glue and taking into consideration a total paragraph before justification, is used in the TEX system [2] designed by Knuth.

A successful system that has been used for a long time was a formatting program made by H. Noot [3]. It had good error reporting and a mechanism for avoiding widows. All these above-mentioned programs did not give support for mathematics, and could only handle non-proportionally spaced characters (i.e. typewriter fonts).

The current system in use consists of the UNIX‡ [4] formatting tools. These were first used by the department of computer science. When the CWI obtained a phototypesetter, the rest of the institute slowly started to use these

---

* Justification is the the process of adding extra space between words to improve the appearance of a text, for instance to get right adjusted margins.

† A widow is a single line of text or just one word on the top of a page before a new paragraph starts. It actually belongs to the paragraph before.

‡ UNIX is a trademark of Bell Laboratories.

tools too. Although they are simple and straightforward, they turn out to be very flexible and capable of producing a lot of different things, from simple text up to line drawings, all of high quality.

Computers and typesetting

In the last decade a revolution has taken place in the typographical world. Nearly everywhere the hot-metal typesetters have been taken over by modern equipment like laser- and ink-jet printers, phototypesetters, daisy wheel and matrix printers. This revolution did not only take place at the output side; the process of keying in text has also been strongly influenced by computer technology.

This has lead to two different approaches: interactive typesetting and batch typesetting.

Interactive typesetting starts at the low end with a typewriter with some editing facilities and leads via *word processors* and their derivates to systems which allow the mark-up of complete and complicated material, including digitised pictures. These can be characterised as simple to very sophisticated picture editors, where the picture is built from simple characters to very complex entities. Some of these systems are even able to automatically make the necessary corrections for reproducing colour pictures on a printing press [5]. Although these systems are very flexible and can produce high quality material, they are very expensive and cost a lot in computer power. A cheaper approach is batch oriented typesetting, also known as formatting. This is much more popular in the scientific world, because there is no need for huge productivity, and it is yet another application which will run on the probably already available research computer. The quality of the output can nevertheless be high.

At this moment there are three different popular and generally available formatting programs. A well-known program is the SCRIBE formatter [6]. One of the interesting aspects of SCRIBE is that the input does not specify the form of the output. Instead it specifies the different segments of the text, like titles, headers, paragraphs with or without *hanging tags**, etc. The only way the output is controlled is by specifying the style of the document. The actual output produced is dependent on the output device and on the way the different styles are implemented. Support for the typesetting of mathematics is quite poor.

The TEX formatter, designed by D. Knuth, gives the user close control over the final appearance of the output. It is known for its good output quality, due to the fact that it formats a complete paragraph at once. The mathematics support is good, since the program has a lot of built in information about the

* A hanging tag is the part of the text hanging like a tag on the left side of a somewhat indented paragraph. This line of text in this footnote is a bit superfluous, but it will make the footnote big enough to demonstrate the use of the hanging tag.

shape of the character. A disadvantage is its enormous size and the crude input syntax. The system is especially written for raster printer devices, which makes interfacing to real typesetters a problem [7].

The standard UNIX formatter, TROFF [7], is specially designed for a typesetter and has extensive macro capabilities. In accordance with the general UNIX-philosophy, it is a central formatting tool. Special tasks like mathematics typesetting are performed by *preprocessors*. This system is the one used at the CWI.

The three mentioned formatters have had a great impact on "in house" typesetting in the (scientific) world and their influence is noticeable everywhere, e.g. in the proposed international standard for text processing [9].

History of TROFF.

Around 1964 a formatting program called RUNOFF made its appearance at MIT on the Compatible Time Sharing System. This program had a lot of influence and implementations were made for many different systems. One of them was the program ROFF at Bell Labs. Around 1973 ROFF was completely revised to NROFF (new roff). The output device for NROFF was basically the Teletype 37. It also got a cousin called TROFF (typesetter ROFF) which used a Graphics System C/A/T-4 typesetter as its output device. These three programs were written in assembly language, but in 1975 TROFF and NROFF were recoded in the higher level language C, and at the same time its capabilities were expanded. In 1979 Bell Labs acquired a new phototypesetter. This would have been a good opportunity to replace TROFF by something better, but no one could come up with a better design. Brian Kernighan started to modify TROFF, so it would run "henceforth" without any change on a variety of typesetters. This renewed version of TROFF is known as typesetter independent TROFF, and is usually called DITROFF [10] .

In 1981 the CWI obtained its own typesetter, a Harris 7450. This machine is connected by a serial line to a port selector so it can be shared between various computers. In order to get the machine running as soon as possible, I developed a filter which took the TROFF binary output code specific to the C/A/T and translated it into yet another binary code for the Harris. It actually consists of two filters, one for the interpretation of the C/A/T code and another one which maintains the protocol with the Harris, which is rather too arcane to describe here. Having two filters makes it possible to have other programs like TEX and a plotting package produce Harris code without their having to know about the ghastly protocol. When I was investigating the changes that had to be made to TROFF, I learned about Kernighan's work and decided to wait until it was available, not wanting to reinvent the wheel. When DITROFF finally arrived, I made the necessary changes to let it drive the Harris typesetter [11]. Also, Dutch hyphenation rules were added, which actually amounted to incorporating the BESTESPLITS [12] program inside TROFF. This addition makes it possible to switch from the American to the Dutch hyphenation algorithm in a single text. German hyphenation rules will be

added soon*.

Using the system

As explained before, TROFF is actually the central program among the typesetting tools. It is hardly used on its own but most of the time it is used in combination with a macro package and/or with one or more preprocessors. Note, that the preprocessors do not know anything about the output device, but they merely generate TROFF *requests*† and macro/string definitions, which are processed by TROFF, which does all the necessary calculations. By means of examples I will show you the capabilities of a macro package and some of the preprocessors. The input of the examples will be set in a line printer style font, `OCR-B`, and the corresponding output will be generated in the laurel font, which differs from the Times font used for the running text. The macro package explained is a standard one and is known as the *—ms* macro package [13].

The —ms macros

These macros give TROFF SCRIBE style capabilities. By classifying parts of the text, the output is produced according to a certain standard. It is not uncommon to do this: the LaTeX system [14] is especially designed to give TeX these capabilities as well. In this system some of the features of the TBL TROFF preprocessor are also implemented.

So one could start a paper with:

```
.TL
The Title
.AU
The Author
The Ghostwriter
.AI
The Author's institution
The Ghostwriter's address
.AB
The abstract of the paper begins, it will be printed as a
centered
block using 5/6 of the current line length.
Note also the rearrangement and justification of
these
input lines in the output.
.AE (abstract ends)
```

and the output will be:

---

* This is a widow

† A request is a line of text, in the text to be typeset, requesting TROFF to perform a certain function, for instance to generate a new page or to print the current page number.

The Title

The Author
The Ghostwriter

The Author's institution
The Ghostwriter's address

ABSTRACT

The abstract of the paper begins, it will be printed as a centered block using 5/6 of the current line length. Note also the rearrangement and justification of these input lines in the output.

As you can see, a line starting with a period means something special to TROFF.

Normally text is divided into paragraphs. To start an indented paragraph, a `.PP` command is used. For a left aligned paragraph, the `.LP` command is used.

Paragraph headings
Headings like the previous one are generated with

```
.SH
Paragraph headings
.LP
Headings like the ...
```

It is also possible to have the headers numbered automatically. The following example shows this.

```
.NH
Basic CPUs, Processor options, Memories
.NH 2
Central Processors
.NH 3
PDP-8 Kits
.NH 3
PDP-11 Kits
.NH 2
Memories
.NH
Mass Storage
.NH 2
Cartridge Disk Kits
.NH 0
Appendix
```

1. Basic CPUs, Processor options, Memories

1.1. Central Processors

1.1.1. PDP-8 Kits

1.1.2. PDP-11 Kits

1.2. Memories

2. Mass Storage

2.1. Cartridge Disk Kits

1. Appendix

The numeric argument to the `.NH` macro call specifies which part of the generated number needs to be incremented. The `.NH 0` command will reset the top level to one.

Indented paragraphs
It is possible to have hanging paragraphs with or without hanging tags. The following example illustrates this.

```
.IP [1]
Text for first paragraph, typed normally as long as you
wish. The first argument is the so called hanging tag.
You may omit it as in:
.IP
and the paragraph will be just indented. A second
argument changes the amount of indentation like:
.IP first 12
and the indentation will be 12 en's.
This value will be the new default value so the next .IP
.IP next
will give you this.
If you don't want a tag but do want to change the
indentation use
.IP "" 5
As done here, the "" denotes an empty argument.
```

[1]Text for first paragraph, typed normally as long as you wish. The first argument is the so called hanging tag. You may omit it as in:

and the paragraph will be just indented. A second argument changes the amount of indentation like:

first and the indentation will be 12 en's. This value will be the new default value so the next .IP

next will give you this. If you don't want a tag but do want to change the indentation use

As done here, the "" denotes an empty argument.

Footnote generation

Footnotes* are placed at the bottom of the page. They are generated with

```
Footnotes*
.FS
* Like this one
.FE
are placed ...
```

The character size† will be automatically reduced by two points.

---

* Like this one

† In the American typographic world the size of a character is usually given in Pica points, while in Europe Didot points are used. There are 12 points in a Pica and 6 Picas in an inch. A Pica point breaks down to 0.35146mm and a Didot point to 0.376065mm, so a Pica point is 0.934572 Didot point. Of course, TROFF uses Pica points.

Keeping blocks together
Sometimes the output should be kept together. This is achieved with the keep commands.

```
.KS
Text inserted here will be kept together until the
.KE
```

If the output of the material placed between the `.KS` and the `.KE` won't fit on the current page, a new page is generated before the actual output takes place. Using a `.KF` instead of a `.KS`, the new page will not be issued, but the output of the material placed between the `.KF` and the `.KE` will be delayed until there is enough space. This way material will *float* trough the actual output.

Displays
To get an exact replica of the input, so TROFF will not rearrange (apart from the typeface) the output with respect to the (current output) line length, the material must be surrounded by the *display* macros.

```
.DS
This is how the input to the examples is made
.DE
```

To get this display as a block of text, the keep macros are used, to prevent a split of the material over a page.

Note the caveat in displays. If the output device has proportionally spaced characters the appearance of the output will be different from the input. Consider the next input:

```
.DS
We want this X to line
up with this Y.
.DE
```

And this is how it looks:

We want this X to line
up with this Y.

The above example shows a basic problem for the ignorant user. There is a difference between what has been written down and the way it looks —how it is read is yet another problem.

Controlling the typeface
As you have seen the typeface changes when there is a different function of the text. Titles will be set in the **bold**-typeface somewhat bigger, while headers will just become bold and footnotes just smaller.

Apart from these automatic changes of the typeface, it can of course be specified by the user.

```
.I
After this command everything will be italic until the
.R
```

After this command everything will be italic ...

If the `.I` has an argument, only that argument will be in italic, and an optional second argument will be put right after it. So this is the way

```
.B bold -item.
```

to produce **bold**-item.

To get (two points) smaller output:

```
.SM
This will be smaller,
.NL
and back to normal.
```

This will be smaller, and back to normal.

The `.LG` command will make the size Larger.

There are of course a lot more things you want, and can do with the *-ms* macros, for instance creating running page headers, etc. However I will not treat this here, nor will I give any details of the bare TROFF commands since I merely want to give an overview of the system and do not want to write a user manual.

In the next issue of the CWI-Newsletter I will discuss the typesetting of tables, mathematical formulas and graphics.

Finally, I am grateful to Sape Mullender for allowing me to use his programs which made it possible to create the "artwork" on the cover.

References

[1] D. Grune, *Handleiding bij de TEKSTSCHAAF*, LR 2.5, Mathematisch Centrum, Amsterdam, 1972.

[2] Donald E. Knuth, *TEX and METAFONT,* American Mathematical Society and Digital Press, Bedford, MA, 1979.

[3] Han Noot, "Structured Text Formatting", *Software Practice and Experience*, **13**(1983), 79-94.

[4] K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual*, Sixth Edition, Bell Laboratories, Murray Hill, NJ, 1975.

[5] A. F. de Winter, "De grafische industrie — Elektronica tussen pers en post", *Natuur en Techniek*, **51**(1983), 432-447.

[6] Brian K. Reid, *Scribe: A document Specification Language and its Compiler*, Carnegie Mellon University, Pittsburgh, PA, 1980.

[7] A. H. Noot, "DVI-Code to the Harris 7500", *TUG-Boat*, March 1984.

[8] J. F. Ossanna, *NROFF/TROFF User's Manual*, Computing Science Technical Report 54, Bell Laboratories, Murray Hill, NJ, 1976.

[9] ISO standard, *Information Processing Systems - Programming Languages - Text Interchange and Processing* (Sixth Working Draft), ANSI, New York, 1 June 1983.

[10] Brian W. Kernighan, *A Typesetter-independent TROFF*, Computing Science Technical Report 97, Murray Hill, NJ, revised, March 1982.

[11] J. N. Akkerhuis, "Typesetting and Troff", *Proceedings EUUG conference 7-9th September*, 29-41, Dublin, Trinity College, 1983, Also available as IW 247/83, CWI, Amsterdam 1983.

[12] J. C. van Vliet, *Bestesplits 1*, NR 28/72, Mathematisch Centrum, Amsterdam, 1972.

[13] M. E. Lesk, *Typing Documents on UNIX and GCOS: The −ms Macros for troff*, Bell Laboratories, Murray Hill, NJ, 1977.

[14] Leslie Lamport, *The LaTeX Document Preparation System*, Second Preliminary Edition, SRI international, Menlo Park, CA, December 13, 1983.

Multigrid Algorithms Run on Supercomputers

by P.W. Hemker

The aim of research at the CWI on multigrid-methods in elliptic partial differential equations is the construction of algorithms that efficiently yield a numerical solution to these problems. This research is motivated by numerous applications, mainly in physics and in the engineering sciences. Except for a few very simple cases, it is impossible to find explicit mathematical expressions for the solutions, so that one has to rely on a numerical approximation to the solution.

By the very nature of partial differential equations, their solutions are continuous functions of several variables. In the numerical approach, these functions are approximated by only a finite set of numbers. Usually these numbers represent the function values of the solution at an evenly spaced set of *"grid-points"* in the domain of definition of the equation.

In practice, many problems appear in the form of an equation for the function $u(x,y)$, with (x,y) in a rectangle Ω. The form of this equation is

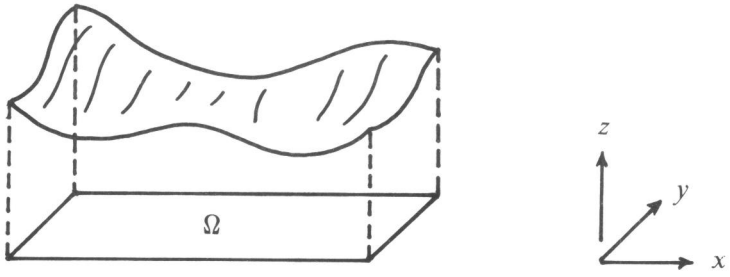
$$\left(\frac{\partial}{\partial x}\left(a_{11}\frac{\partial}{\partial x}+a_{12}\frac{\partial}{\partial y}\right)+\frac{\partial}{\partial y}\left(a_{21}\frac{\partial}{\partial x}+a_{22}\frac{\partial}{\partial y}\right)\right)u+b_1\frac{\partial u}{\partial x}+b_2\frac{\partial u}{\partial y}+cu = f, \qquad (\star)$$

with additional conditions for $u(x,y)$ on the boundary of Ω. The coefficients a_{ij}, b_i, c and f are given functions of x and y. Much of the research on multigrid methods is restricted to this equation. The computer programs that have been recently constructed at the CWI are almost all intended for equations of this type.

For those who are not familiar with elliptic partial differential equations a simple example is given by the Poisson equation:

$$\left(\frac{\partial}{\partial x}\right)^2 u+\left(\frac{\partial}{\partial y}\right)^2 u = 0 \quad \text{on } \Omega,$$

with $u(x,y)$ prescribed on the boudary of the 2-dimensional domain Ω. For this equation we can imagine the solution $z = u(x,y)$ as a surface in 3-dimensional space. On the boundary of Ω its position (x,y,z) is given and in the interior of Ω the surface behaves like a soap-film between the prescribed boundary values.



This example illustrates some essential properties of elliptic partial differential equations: boundary conditions are to be given all along the boundary and the solution in the interior is a smooth function.

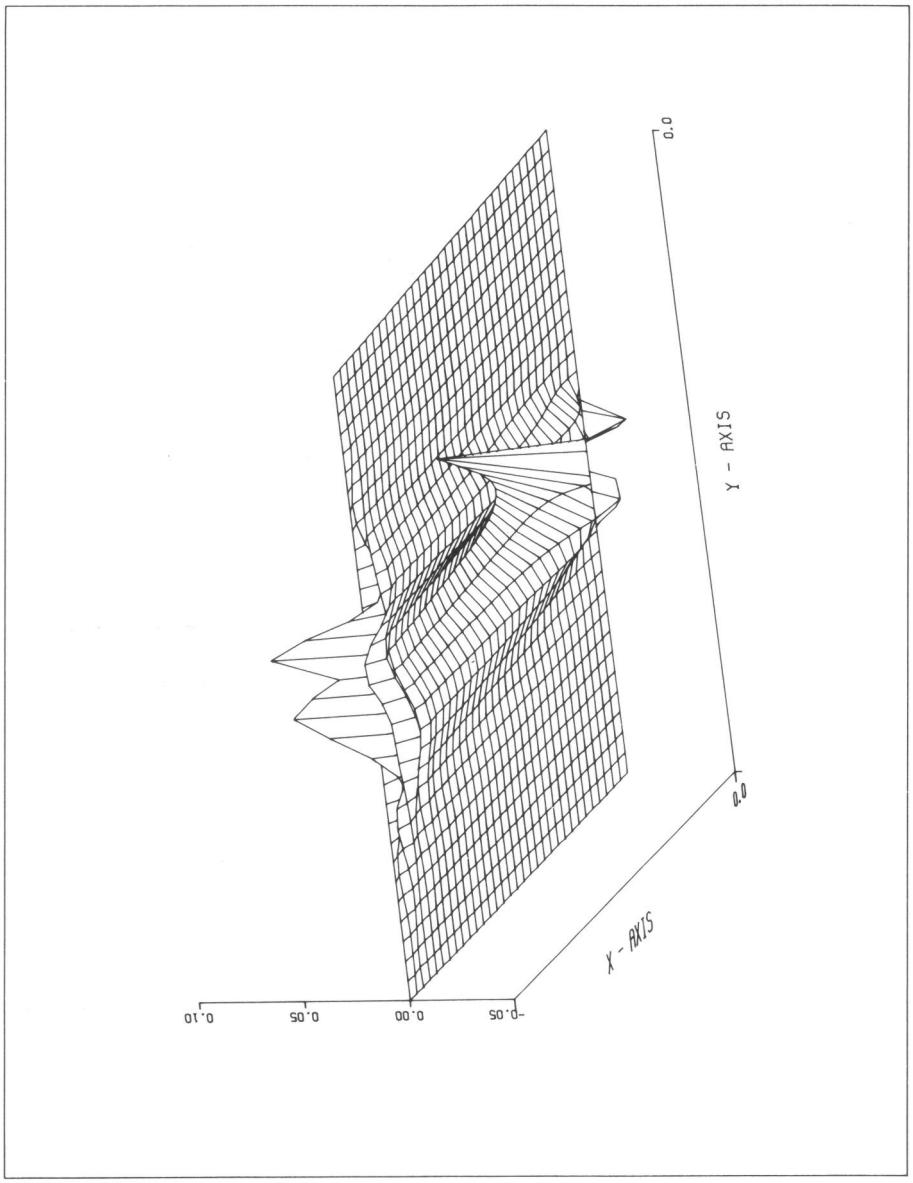
The usual technique for finding an approximation to $u(x,y)$ is to replace equation $(\star)$ by a set of N linear equations for N unknown values u_{ij} which are meant to represent the function values $u(x_i,y_j)$, where (x_i,y_j) is a gridpoint. All these gridpoints form a *"grid"* (or *"net"*) for the *"discretization"* of $(\star)$. The approximation u_{ij} for $u(x_i,y_j)$ becomes more accurate as the number of gridpoints N gets larger. Therefore, it is often necessary to solve linear systems with N very large. N may be so large that — with conventional solution methods such as Gaussian elimination — it can take many days to solve these systems on a computer.

Therefore the usual way to solve the large systems is by relaxation methods, i.e. iterative methods in which an initial guess of the solution is improved step by step. Well-known relaxation methods are Gauss Seidel-relaxation, SOR, zebra-relaxation, and Incomplete (Line) LU-decomposition-relaxation. Successful research in recent years has resulted in other, much faster converging, iteration methods such as ICCG (preconditioned conjugate gradient methods, cf.[6]). A disadvantage of all these methods is that the rate of convergence of the iterations decreases for larger N.

A significant improvement in solving these (very large) systems of equations is found in the multigrid method. This is a technique which accelerates the convergence of the relaxation methods so that the rate is independent of N. This is done by introducing coarser grid discretizations (linear systems with $N := N/4$, $N/16$ etc.) and by combining relaxation for the large system with the (less laborious and faster converging) relaxation on the coarser grids. A good account of the multigrid method is found in [2].

For those to whom the basic idea of multigrid is new, we give a very short explanation. The principle of multigrid is based on three facts:
1) The simple relaxation methods such as Gauss Seidel damp the rapidly varying components in the error much faster than the slowly varying components. In other words: they can be considered as efficient smoothers for the error rather than as reducers of the overall error.
2) The remaining (smooth) error components can be represented on coarser grids, where the number of gridpoints is much smaller. Consequently, the remaining error components can be reduced there much more efficiently.
3) On the coarse grid the solution is most efficiently reduced by a simple relaxation method and, again, coarser grid corrections. Thus a recursive procedure can be defined where on the very coarsest grid the linear system to be solved has a very small number of unknowns.

It will be clear that the above principle is rather general and that many variants are possible. The idea can also be applied to other equations in which the original problem has continuous solutions. The idea can for instance be used for integral equations [4]. Attempts are even made to use the multigrid


0.0
Y - AXIS
0.0
X - AXIS
-0.05
0.00
0.05
0.10

idea in cases where the linear systems do not originally stem from a continuous equation [5].

The convergence of multigrid for the equation (⋆) depends on the coefficients in the equation, on the operators that take care of the interaction between the various grids, and on the relaxation method used. On the one hand the efficiency of a multigrid algorithm depends on this convergence and on the other hand on the amount of arithmetic operations in each iteration. In recent years some research at the CWI was devoted to the selection of optimal efficient multigrid strategies for different equations (⋆), cf. [3].

It appears that for different classes of (⋆), different relaxation methods give optimal efficiency. For problems like the Poisson problem zebra- and ILU-relaxation are succesful (zebra is slightly more efficient, but ILU performs better for a larger class of equations), while ILLU-relaxation is particulary suited for problems where the coefficients b_i dominate the coefficients a_{ij}.

Several implementations of multigrid algorithms have been constructed at the CWI. Besides a comprehensive program for experimental purposes written in ALGOL 68, a number of programs was written in FORTRAN with efficient execution in practical applications in mind. Two particular programs, developed in cooperation with the Numerical Group of the University of Technology, Delft, are called MGDl (ILU relaxation) and MGD5 (ILLU).

In practice, the speed of these programs depends not only on the convergence rates or the number of arithmetic operations per iteration cycle, but also on the architecture of the computer used.

Here the programmer has to decide whether his aim is to develop his program for a particular machine or to pursue an efficient program for a general class of computers. We decided in favor of the latter and wrote two versions, one aimed at the usual sequential (=scalar) computer and one at vector computers (CRAY 1 or CYBER 205). In both cases we refrained from the use of features that are available only on one particular machine and we wrote the programs in a most elementary and portable FORTRAN. For the vector computers this means that we used the auto-vectorization capabilities of the FORTRAN compilers.

Thus for ILU- and ILLU-relaxation we constructed a scalar-version (MGDlS, MGD5S) and a vector-version (MGDlV, MGD5V). For the scalar architecture the computing time for an iteration cycle is proportional to the number of gridpoints in the finest grid. For different machines the execution times are given in table 1. From this we see that for an equation like Poisson's equation (for which 3 iterations and a preparational phase corresponding to 3 iteration cycles are necessary) a linear system with $N = 257 \times 257 \approx 66000$ equations can be solved in less than a second.

It is interesting to see to what extent the arithmetic operations in MGDl and MGD5 can be arranged so as to make effective use of the vector-architecture of the CRAY 1 or the CYBER 205 (i.e. to what extent the

algorithms are vectorizable). The acceleration factors of the vector-programs run (if possible) in vector-mode over the scalar-programs (run in scalar-mode) are given in Table 2.

	MGD1	MGD5
relaxation	ILU	ILLU
IBM 3081K	16.7	25.7
CYBER 170	15.4	24.9
CRAY 1S	9.1	12.7
CYBER 205	8.1	11.1

Table 1. CPU-times for the scalar versions on scalar architecture in μsec/(cycle $\times$ meshpoint).

	N	MGD1	MGD5
CYBER 170 (scalar mode)	65×65	0.86	0.95
CRAY 1S	65×65	3.2	2.7
(vector mode)	129×129	3.6	2.9
CYBER 205	65×65	3.2	2.2
(vector mode;	129×129	4.2	2.5
two pipes)	257×257	4.8	2.6

Table 2. Acceleration factor of the vector version over the scalar version for the algorithms MGD1 and MGD5.

We see that vectorization of the MGD5 algorithm has more effect on the CRAY than on the CYBER. The other algorithm, MGD1, is better vectorizable, especially on the CYBER 205. Now a Poisson type problem with $N = 257 \times 257$ is solved in 0.2 sec.

Other programs were made for zebra-relaxation. By its nature this relaxation method seems better suited for vectorization than ILU- or ILLU-relaxation, and on the CRAY 1 better acceleration factors were indeed found. However, to make it efficient on the CYBER the data structures in the program had to be changed drastically. In the MGD-programs the data (u_{ij}) are stored in a natural way in a rectangular array, corresponding to the location of the gridpoints (x_i, y_j) in the rectangle Ω. In order to prevent the frequent use of strides >1 in the zebra program (which is necessary for efficient vectorization on the CYBER), the data u_{ij} had to be re-ordered by even and odd lines. In this way the program could be accelerated by a factor 7.3 on the CYBER. The same program runs without problems on the CRAY.

We see that for efficient implementation of an algorithm we have to tune the structure of the program very much to the computer architecture. We are willing to do this as long as our programs remain portable.

A program can generally be made even faster if one tunes the programming really to one particular machine and even more if one restricts its use to only one particular case of equation (⋆). Such a program has been constructed by Barkai and Brandt [1]. It solves (only) the Poisson equation on a CYBER 205. In this program a checkerboard relaxation is used and the data structures have been specially adapted for this relaxation on this particular computer. The result is a non-portable program which is extremely fast. In [1] it is mentioned that the Poisson equation with $N = 129 \times 129$ can be solved in 0.006 seconds.

At the CWI we do not plan to proceed in the direction of non-portable programs. At the moment we are more interested in efficient algorithms for solving wider classes of equations. Besides our special interest in the solution of equations (⋆) of singular perturbation type, we are considering the implementation of an algorithm for (⋆) with (strongly) discontinuous coefficients.

Acknowledgements
I would like thank Paul de Zeeuw who did most of the programming and Walter Lioen who implemented the FORTRAN programs with zebra relaxation. Furthermore, I would like to thank Drs I.P. Jones and C.P. Thompson from AERE, Harwell (England), for their kind cooperation in running the programs on the CRAY 1 and the IBM 3081K.

References

1. D. Barkai & A. Brandt, (1983) Vectorized Multigrid Poisson Solver for the CDC CYBER 205. *J. Appl. Math. & Computat.* 13 *215-227.*
2. W. Hackbusch & U. Trottenberg (eds.), (1982) *Multigrid Methods.* LNM 960, Springer Verlag.
3. P.W. Hemker, R. Kettler, P. Wesseling & P.M. de Zeeuw, (1983) Multigrid Methods: Development of Fast Solvers. *J. Appl. Math. & Computat.* 13 *311-326.*
4. H. Schippers, (1983) *Multiple Grid Methods for Equations of the Second Kind with Applications in Fluid Mechanics.* Mathematical Centre Tract 163, CWI, Amsterdam.
5. K. Stüben, (1983) Algebraic Multigrid (AMG): Experiences and Comparisons. *J. Appl. Math. & Computat.* 13 *419-451.*
6. H.A. van der Vorst, (1982) *Preconditioning by Incomplete Decompositions.* Doctoral Thesis, University of Utrecht.

Winning ways for your mathematical plays

by Aart Blokhuis

Winning Ways for your Mathematical Plays by Elwyn R. Berlekamp, John H. Conway & Richard K. Guy.
Vol. 1 Games in General, 426 pp.
Vol. 2 Games in Particular, 424 pp.
Academic Press (London, New York, etc.), 1982.

One-heap min is certainly the most boring game ! Two persons sit at a table with a heap of beans. At his turn one of the players takes any number of beans, at least one, from the heap. The first player unable to move loses. Usually the first player wins by taking the whole heap. Things become more interesting, though, if more heaps are involved. At his turn a player chooses a heap and removes any number from that heap. This game is the *disjunctive sum* of several one-heap min games. In general, games are considered in which the players move alternately, and with the rule that the first player unable to move loses. In the disjunctive sum of games, a move consists of choosing a game and making a move in that game. In Volume 1 of Winning Ways, the authors develop a theory for addition of games, best illustrated using a partial variant of min called hackenbush.

A hackenbush game is drawn in figure 1.

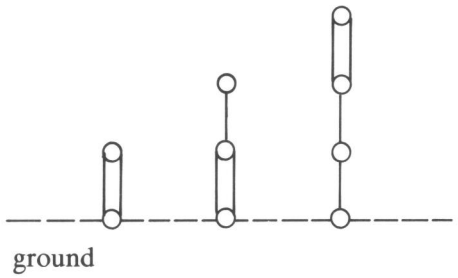


Fig.1

The two players, blue and red say, may hack the bush in any segment of their colour (single edges = blue, double = red). The part of the stalk that is disconnected from the ground disappears, together with the chopped segment. Clearly, this example is the disjunctive sum of three 'simple' hackenbush games. A little analysis also shows that in this example the player who starts always loses. This is called a *zero-game.* In hackenbush it is possible to assign to every position a number, corresponding to the number of 'free moves' for blue. If the number is positive blue always wins and if it is

negative red wins (irrespective of who starts); zero means that the starter loses. In figure 1 the three parts have values $-1, -\frac{1}{2}$ and $1\frac{1}{2}$, adding up to 0. How to compute these numbers is all in the book. As an example here are strings with value π and $1/\omega$, where ω is the first infinite ordinal:

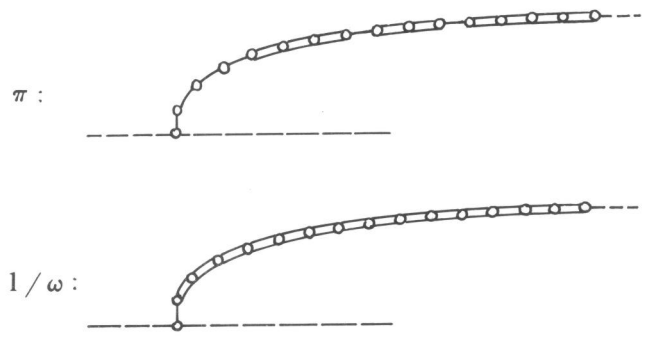


Fig.2

Also, other startling values like $\sqrt{\omega}$ or $\omega^{-\omega}$ can be constructed.

So far things were easy, but what number should we assign to a min-heap of one bean, or (which is the same) to a hackenbush stalk having just one segment which is red and blue at the same time (purple say) ? In this game the first player wins, but if you add to it any game with positive value, i.e. blue wins, the result is still positive. Hence the value of this game is something like zero, but it's not zero itself (which means the starter loses). In this way the 'nimbers' make their appearance. This one is called $\star$, and it is easy to prove that $\star + \star = 0$! Another interesting 'number' is $\uparrow$ which has the property $\uparrow > 0$, but $\uparrow < 2^{-n}$ for all n. It is here that the real problems begin, and that the reader should read the book instead of this review.

Volume 2 is called *games in particular,* and the only way to give an impression of its contents is to look at some games in particular. In chapter 18 we meet the following game: A and B choose a number in turn, with the restriction that no new number may be the sum of multiples of previously chosen ones. If you choose 1 you lose. For example, A :2, B :5, A :3, B :1 and loses.

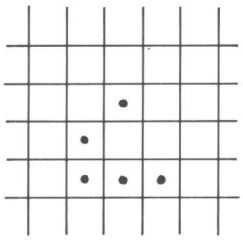
The game is called Sylvester coinage. Now B, of course, did not play very smartly, since choosing 3 instead of 5 would have won immediately. Hence 2 was not a clever choice for A either. Less obvious is that 4, 6, 8, 9, 12 are all losing opening moves. Can the reader prove that p is a winning opening choice for each prime $p \geqslant 5$? And what about 16, 18, 24, or any number of the form $2^a 3^b$?

The game Sylvester coinage can possibly be analyzed completely; however the following two-person game (which cannot) is really interesting. It is

called 'philosopher's football', or Phutball. The ball, a black stone, is put on the central point of a go-board. Each player, when he moves, *either* places a white stone somewhere on the board, *or* jumps the ball over a series of white stones in any of the eight directions any number of times, removing the jumped-over stones. One side of the board is Left's goal line, the opposite is Right's. Left wins if he succeeds in bringing the ball on (or behind) Right's goal line and conversely. For a little introduction, see p. 688.

Further games analyzed include tic-tac-toe, hare and hounds, fox and geese and many other less well-known games.

After 700 pages on two-person games come an additional 120 on 1-person-games or puzzles, including of course the celebrated Hungarian cube and the game of solitaire. The final 30 pages are about the most interesting no-person game: *life. Life* is a 'game' which is played on an infinite chess-board. At every stage some squares are 'alive' and others 'dead'. In the next stage squares become alive or die according to the following rules: A square is born if exactly 3 of its neighbours were alive in the last stage. A square dies if more than 3 or less than 2 neighbours were alive in the previous stage. As an exercise one should look at the development of the 'most spectacular small living object, the glider'. Black dots are living cells.



the glider

Fig.3

After its 'discovery' by J.H. Conway around 1970 *life* has become very popular, as a result of which many amazing starting configurations have been found, whose names suggest their properties: spaceships, flip-flops and finally Gosper's *glider-gun,* emitting a new glider every 30 generations. The final pages of Volume 2 are devoted to a construction of a computer using glider guns, eaters and other ingenious configurations, thus proving that *life is universal.*

Abstracts

of Recent CWI Publications

When ordering any of the publications listed below please use the order form at the back of this issue.

CS-R8401. J.A. Bergstra & J.V. Tucker. *Top-down design and the algebra of communicating processes.*
AMS 68B10; 38 pp.; **key words:** hierarchical and modular systems, composition tools, system architectures, concurrency, communicating processes, process algebra, fixed point equations, hand-shaking.
Abstract: We develop an algebraic theory for the top-down design of communicating systems in which levels of abstraction are represented by algebras, and their stepwise refinements are represented by homomorphisms. Particular attention is paid to the equational specification of these levels of abstraction. A number of examples are included for illustration, most notably a top-down design for a communication protocol.

CS-R8402. P.M.B. Vitányi. *Square time is optimal for simulation of one pushdown store by an oblivious one-head tape unit.*
AMS 68C40; 4 pp.; **key words:** multitape Turing machines, pushdown stores, queues, time complexity, square lower bounds, on-line simulation by oblivious single-head tape units, Kolmogorov complexity.
Abstract: To simulate a pushdown store or queue on-line by an oblivious one-head tape unit takes at least square time. Since each multitape Turing machine can be trivially simulated by an oblivious one-head tape unit in square time this result is optimal.

CS-R8403. J.A. Bergstra & J.W. Klop. *Algebra of communicating processes with abstraction.*
AMS 68B10; 43 pp.; **key words:** concurrency, communicating processes, internal actions, process algebra, bisimulation, process graphs, handshaking, terminating rewrite rules, recursive path ordering.
Abstract: We present an axiom system ACP_τ for communicating processes with silent actions ('τ-steps'). The system is an extension of ACP, Algebra of Communicating Processes, with Milner's τ-laws and an explicit abstraction operator. By means of a model of finite acyclic process graphs for ACP_τ syntactic properties such as consistency and conservativity over ACP are proved. Furthermore the Expansion Theorem for ACP is shown to carry over to ACP_τ. Finally, termination of rewriting terms according to the ACP_τ axioms is proved using the method of recursive path orderings.

CS-R8404. J.A. Bergstra & J.W. Klop. *Verification of an alternating bit protocol by means of process algebra.*
AMS 68B10; 17 pp.; **key words:** process algebra, alternating bit protocol, abstraction, fair abstraction.
Abstract: We verify a simple version of the alternating bit protocol in the system ACP_τ (Algebra of

Communicating Processes with silent actions) augmented with Koomen's fair abstraction rule.

CS-R8405. J.A. Bergstra & J.W. Klop. *Fair FIFO queues satisfy an algebraic criterion for protocol correctness.*
AMS 68B10; 17 pp.; **key words:** process algebra, queue, communication protocol, verification.
Abstract: An algebraic criterion for protocol correctness is presented, as well as a proof method for establishing the criterion. As an example we consider FIFO queues with unbounded capacity.

CS-R8406. P.M.B. Vitányi. *One queue for two pushdown stores take square time on a one-head tape unit.*
AMS 68C40; 5 pp.; **key words:** multitape Turing machines, pushdown stores, queues, time complexity, square lower bounds, on-line simulation by single-head tape units, Kolmogorov complexity.
Abstract: To simulate one virtual queue or two virtual pushdown stores by a one-head tape unit takes at least square time. Since each multitape Turing machine can be trivially simulated by a one-head tape unit in square time this result is optimal.

CS-R8407. J.C.M. Baeten, J.A. Bergstra & J.W. Klop. *Priority rewrite systems.*
AMS 03F65; 51 pp.; **key words:** term rewrite systems, priority rewrite set, signature, modularity, specification.
Abstract: Term rewrite systems with rules of different priority are introduced. The semantics are explained in detail and several examples are discussed, including a rewrite rule interpretation of Backus functional programming.

CS-N8401. R. van den Born. *Structure preserving data flow analysis for programs with GOTO-statements.*
AMS 68B10; 47 pp.; **key words:** high level data flow analysis, (micro)code-generation, optimalisation.
Abstract: The generation of efficient (micro)code requires information about the parallelism and the data flow in the source program as well as its high level structure. The graph described in this article provides a clear and flexible representation of this information. A new method for data flow analysis is introduced, which can be used to compute such a graph. The method can handle arbitrary GOTO statements.

OS-R8401. J.K. Lenstra & A.H.G. Rinnooy Kan. *New directions in scheduling theory.*
AMS 90B35; 9 pp.; **key words:** scheduling, jobs, machines, algorithm, computational complexity, probabilistic analysis.
Abstract: This is an assessment of new developments in the theory of sequencing and scheduling. After a review of recent results and open questions within the traditional class of scheduling problems, we focus on the probabilistic analysis of scheduling algorithms and next discuss some extensions of the traditional problem class that seem to be of particular interest.

OS-R8402. J.P.C. Blanc. *Asymptotic analysis of a queueing system with a two-dimensional state space.*
AMS 60K25; 22 pp.; **key words:** asymptotic analysis, queueing system, two-dimensional state space, conformal mapping, relaxation time.
Abstract: A technique is developed for the analysis of the asymptotic behaviour in the long run of queueing systems with two waiting lines. The generating function of the time-dependent joint

queue length distribution is obtained with the aid of the theory of boundary value problems of the Riemann-Hilbert type and by introducing a conformal mapping of the unit disk into a given domain. In the asymptotic analysis an extensive use is made of theorems on the boundary behaviour of such conformal mappings.

OS-R8403. J. Han & M. Yue. *A study of elimination conditions for the permutation flow-shop sequencing problem.*
AMS 90B35; 15 pp.; **key words:** flow-shop, elimination criterion, branch-and-bound.
Abstract: We give a few elimination criteria for the permutation flow-shop problem and prove that one of them is equivalent to Szwarc's elimination criterion. We also propose a lower bound to be used in a branch-and-bound method.

OS-R8404. J.H. van Schuppen. *Overload control for an SPC telephone exchange. An optimal stochastic control approach.*
AMS 93E20; 32 pp.; **key words:** stored program control exchange, overload control, queueing theory, stochastic control.
Abstract: The current stored program control (SPC) telephone exchanges are the operational units of the telephone networks. One of the problems with these exchanges is the performance degradation during time periods of peak demand. The problem of overload control is then to maximize the number of admitted and successfully completed calls under technical constraints of which the main one is the available processor capacity. In the paper the processor load of an SPC telephone exchange is modelled as a hierarchical queueing system, while the problem of overload control is formulated as an optimal stochastic control problem. The latter problem is solved. An implementation of the derived control law is suggested.

OS-R8405. M.W.P. Savelsbergh & P. van Emde Boas. *Bounded Tiling, an alternative to satisfiability ?*
AMS 68C25; 10 pp.; **key words:** computational complexity, NP-completeness, master problem, master reduction, Bounded Tiling.
Abstract: The Bounded Tiling problem is presented and the question is raised whether it provides a viable alternative to the foundation of the NP-completeness theory. To answer this question we take the standard results and investigate how they will look when they are based upon Bounded Tiling.

NM-R8401. P.W. Hemker & P.M. de Zeeuw. *Some implementations of multigrid linear system solvers.*
AMS 65F10; 34 pp.; **key words:** elliptic differential equations, solutions of linear systems, multigrid methods.
Abstract: In this paper portable and efficient FORTRAN implementations for the solution of linear systems by multigrid are described. They are based on ILU- or ILLU-relaxation. Scalar and vector versions are compared. Also a complete formal description of a more general multigrid algorithm is given in ALGOL 68.

NM-R8402. P.J. van der Houwen, B.P. Sommeijer & H.B. de Vries. *Generalized predictor-corrector methods of high order for the time integration of parabolic differential equations.*
AMS 65M10; 34 pp.; **key words:** parabolic differential equations, methods of lines, predictor-corrector methods, stability.
Abstract: A general class of predictor-corrector methods is presented and explicit expressions for

the local truncation error and the stability polynomial are derived. Examples of methods of orders up to 6 are given which are suitable for the integration of semi-discrete parabolic differential equations. By means of a large number of numerical experiments we show that the higher order methods are generally more efficient than the lower order methods. As a further illustration we compare the generalized predictor-corrector methods with the familiar ADI method confirming our general belief that for smooth parabolic problems high order time integrators are superior to lower order integrators.

NM-R8403. J.M. Sanz-Serna & J.G. Verwer. *A study of the recursion* $y_{n+1}=y_n+\tau y_n^m$.
AMS 65L05; 4 pp.; **key words:** numerical analysis, recursion formulas, energy method, Euler's method.
Abstract: We provide a detailed study of the recursion $y_0=1, y_{n+1}=y_n+\tau y_n^m$, $n=0,1,\ldots,\ m>1$, which arises either as a model discretization of a nonlinear ordinary differential equation or in the use of the energy method. Sharp bounds and asymptotic estimates are given for the size of the iterates y_n.

NM-N8401. G.T. Symm, B.A. Wichmann, J. Kok & D.T. Winter. *Guidelines for the design of large modular scientific libraries in ADA. Final report for the Commission of the European Communities.*
AMS 69D49; 146 pp.; **key words:** Ada programming language, scientific software.
Abstract: The new programming language Ada has been designed primarily for real-time, embedded computer applications development. However, it is envisaged that it will also be widely used in large-scale scientific computation. Several features of the language require careful consideration if large portable and modular scientific algorithms libraries are to be implemented successfully. Accordingly, in this report we attempt to identify the problems associated with the overall design and implementation of such libraries in Ada and make recommendations for their solution. The problem areas considered are precision, basic mathematical functions, composite data types, information passing, error handling, working-space organisation and real-time environment.

MS-8401. D.M. Chibisov & W.R. van Zwet. *On the Edgeworth expansion for the logarithm of the likelihood ratio, II.*
AMS 62E20; 11 pp.; **key words:** Edgeworth expansions, contiguity.
Abstract: In this paper we discuss conditions under which the distribution function of the logarithm of the likelihood ratio possesses an Edgeworth expansion. The underlying model is that of independent but not necessarily identically distributed random variables and the two sequences of product distributions are assumed to be contiguous. First we deal with the problem in full generality and obtain a result in the spirit of Oosterhoff and Van Zwet (1979), who characterized contiguity and asymptotic normality of the logarithm of the likelihood ratio. Next we show how this result may be simplified for differentiable likelihoods. In a companion paper, Chibisov and Van Zwet (1984), we discuss the special case of independent and identically distributed random variables and differentiable likelihoods in considerably more detail.

MS-R8402. C.A.J. Klaassen & W.R. van Zwet. *On estimating a parameter and its score function.*
AMS 62F11; 12 pp.; **key words:** adaptation, score function, Cramér-Rao inequality, semi-parametric models.
Abstract: We consider the problem of estimating a real-valued parameter θ in the presence of an abstract nuisance parameter η, such as an unknown distributional shape. Attention is restricted to the case in which the 'score functions' for θ and η are orthogonal, so that fully asymptotically

efficient estimation is not a priori impossible. For fixed sample size we provide a bound of Cramér-Rao type. The bound differs from the classical one for known η by a term involving the integrated mean square error of an estimator of a multiple of the score function for θ for the case where η is known. This implies that an estimator of θ can only perform well over a class of shapes η if it is possible to estimate the score function for θ accurately over this class.

MS-R8403. P. Groeneboom. *Estimating a monotone density.*
AMS 62E20; 14 pp.; **key words:** monotone densities, isotonic estimation, Brownian motion, jump processes, concave majorant, heat equations, Volterra integral equations, first exit densities.
Abstract: Some local and global results on estimating a monotone density are discussed. In particular, it is shown that a centered version of the L_1-distance between a smooth strictly decreasing density and its Maximum Likelihood Estimator is asymptotically normal and has an asymptotic variance which is independent of the density. The results are derived from the structure of a jump process generated by Brownian motion.

MS-R8404. R.J.M.M. Does, R. Helmers & C.A.J. Klaassen. *On the Edgeworth expansion for the sum of a function of uniform spacings.*
AMS 62E20; 15 pp.; **key words:** Edgeworth expansions, uniform spacings, Cramér's condition.
Abstract: An Edgeworth expansion for the sum of a fixed function g of normed uniform spacings is established under a natural moment assumption and a Cramér type condition. This condition is shown to hold under an easily verifiable and mild assumption about the function g.

MS-R8405. R.J.M.M. Does, R. Helmers & C.A.J. Klaassen. *Approximating the percentage points of Greenwood's statistic with Cornish-Fisher expansions.*
AMS 62E20; 6 pp.; **key words:** Greenwood's statistic, Cornish-Fisher expansions, uniform spacings, goodness-of-fit.
Abstract: It is suggested that approximating the exact percentage points of Greenwood's statistic with Cornish-Fisher expansions is useful for not too small sample sizes.

MS-R8406. P. Groeneboom & D.R. Truax. *A monotonicity property of the power function of multivariate tests.*
AMS 62H10; 11 pp.; **key words:** monotonicity of power functions, non-central Wishart matrix, characteristic roots, orthogonal groups, Euler angles, correlation inequalities, hypergeometric functions of matrix arguments, FKG inequality, pairwise total positive of order two.
Abstract: Let $S = \sum_{k=1}^{n} X_k X'_k$, where the X_k are independent observations from a 2-dimensional normal $N(\mu_k, \Sigma)$ distribution, and let $\Lambda = \sum_{k=1}^{n} \mu_k \mu'_k \Sigma^{-1}$ be a diagonal matrix of the form λI, where $\lambda \geqslant 0$ and I is the identity matrix. It is shown that the density ϕ of the vector $\tilde{\ell} = (\ell_1, \ell_2)$ of characteristic roots of S can be written as $G(\lambda, \ell_1, \ell_2)\phi_0(\tilde{\ell})$, where G satisfies the FKG condition on $\mathbb{R}^3_+$. This implies that the power function of tests with monotone acceptance region in ℓ_1 and ℓ_2, i.e. a region of the form $\{g(\ell_1, \ell_2) \leqslant c\}$, where g is nondecreasing in each argument, is nondecreasing in λ. It is also shown that the density ϕ of (ℓ_1, ℓ_2) does not allow a decomposition $\phi(\ell_1, \ell_2) = G(\lambda, \ell_1, \ell_2)\phi_0(\tilde{\ell})$, with G satisfying the FKG condition, if $\Lambda = diag(\lambda, 0)$ and $\lambda > 0$, implying that this approach to proving monotonicity of the power function fails in general.

MS-R8407. A.J. van Es. *On the weak limits of elementary symmetric*

polynomials.
AMS 60F05; 16 pp.; **key words:** symmetric polynomials, U statistics.
Abstract: In this paper we extend recent results of Székely and others on the weak limits of elementary symmetric polynomials $S_n^{(k_n)}(X_1, \ldots, X_n)$ in the case where the order k_n of the polynomials is proportional to the number of variables n.

MS-R8408. H.C.P. Berbee. *A limit theorem for the superposition of renewal processes.*
AMS 60G55; 5 pp.; **key words:** stationary, renewal process, Palm measure, superposition, asymptotics.
Abstract: The asymptotics of a superposition of renewal point processes is studied from the point of view of Palm theory.

AM-R8401. H.J.A.M. Heijmans. *On the stable size distribution of populations reproducing by fission into two unequal parts.*
AMS 92A15; 27 pp.; **key words:** size-structured populations, proliferating cells, exponential individual growth, continuous culture, strongly continuous semigroup, dynamical system.
Abstract: A nonlinear model describing the dynamics of a continuous culture of cells characterized by their size only, and reproducing by fission into unequal parts, is formulated. It is assumed that cells grow proportionally to their size. Using techniques from dynamical systems theory, we establish results concerning the existence of a globally stable equilibrium.

AM-R8402. H.A. Lauwerier. *Global bifurcation of a logistic delay map.*
AMS 58F14; 19 pp.; **key words:** logistic delay map, bifurcation, nonlinear difference equation, period-doubling.
Abstract: We consider the planar map $x' = y, y' = \phi(x,y)$ connected with the difference $x_{n+1} = ax_n(1-(1-b)x_n - bx_{n-1})$ with $a > 1, 0 < b < 1$. The unstable manifold of the fixed point (0,0) is determined explicitly as an analytic curve. The stable manifold consists of an infinity of algebraic curves and forms the boundary of the escape region, the set of starting-points of orbits going to infinity. The case $b = \frac{1}{2}$ is studied in more detail and most illustrations are for this case. For $a > 3$ there always exists an unstable 4-cycle. For this cycle the secondary unstable manifolds are also determined explicity. Even for the case $b = \frac{1}{2}$ a Feigenbaum scenario of repeated period-doubling has been observed. However, the convergence to Feigenbaum's universal constant appears to be rather slow.

AM-R8403. J. Grasman & J.V. Lankelma. *The exit problem for a stochastic dynamical system in a domain with almost everywhere characteristic boundaries.*
AMS 35A40; 13 pp.; **key words:** random perturbation, Fokker-Planck equation, exit problem, WKB approximation, hypercycle.
Abstract: For a dynamical system with a stable equilibrium point the influence of small random perturbations is analyzed with singular perturbation techniques. The WKB approach to the asymptotic solution of the exit problem for domains with characteristic boundaries containing a critical point is not valid, because of the turning-point behaviour of the Fokker-Planck equation near such a point. In this paper this difficulty is resolved by changing the domain for the characteristic exit problem slightly. Explicit computations are carried out for a problem originating from theoretical population biology: the 3-dimensional hypercycle.

AM-R8404. H.A. Lauwerier. *Entire functions for the logistic map II.*
AMS 30D05; 34 pp.; **key words:** entire functions, iterated maps, fractals.

Abstract: The asymptotic behaviour of the entire functions F introduced in part I of this article is studied on rays in the complex plane. The Julia set of the logistic map is brought into relation with the zeros of F(z).

AM-R8405. S.M. Verduyn Lunel. *Linear autonomous retarded functional differential equations; a sharp version of Henry's theorem.*
AMS; 44 pp.; **key words:** completeness, functional differential equation, small solution, Volterra convolution integral equation.
Abstract: Small solutions play a crucial role in the theory of completeness of the generalized eigenfunctions of the infinitesimal generator of the c_0-semigroup $\{T(t)\}$ associated with a linear autonomous retarded functional differential equation. In this paper we prove a sharp version of Henry's Theorem on small solutions and, as a corollary, that the ascent α of $\{T(t)\}$ is equal to the ascent δ of the adjoint c_0-semigroup $\{T(t)^*\}$.

AM-R8406. H.A. Lauwerier. *A case of a not so strange strange attractor.*
AMS 58F13; 13 pp.; **key words:** iterated maps, strange attractors, fractals.
Abstract: An example is given of an iterative two dimensional map of the horseshoe type in which everything can be expressed in terms of simple trigonometric functions. The strange attractor is an analytic curve with a fractal dimension.

AM-R8407. H.J.A.M. Heijmans. *Holling's 'hungry mantid' model for the invertebrate functional response considered as a Markov process. Part III: Mathematical elaborations.*
AMS 92A15; 43 pp.; **key words:** satiation, functional response, forward equation, backward equation, positive operator, semigroup, Trotter-Kato theorem, weak * solution, first order partial differential equation with transformed arguments.
Abstract: In this paper, we study an analytical model describing predatory behaviour. It is assumed that the parameter describing the predator's behaviour is its satiation. Using semigroup methods and compactness arguments we prove that a stable satiation distribution is reached as $t \to \infty$. Furthermore, using a Trotter-Kato theorem we justify the transition to the much simpler problem that is obtained if the prey biomass tends to zero.

AM-R8408. H.E. de Swart. *Spectral modelling of a potential vorticity equation for a barotropic flow on a beta-plane.*
AMS 34B25; 23 pp.; **key words:** scale analysis, hydrostatic balance, Rossby number, quasi-geostrophy, beta-plane, barotropic assumption, Ekman boundary layer, orography, spectral model.
Abstract: A quasi-geostrophic potential vorticity equation, including forcing and dissipation mechanisms, is derived for a barotropic flow over a large scale topography on a β-plane. The model is assumed to describe large scale motions in the atmosphere. Finally, from the vorticity equation a spectral model is constructed.

PM-R8401. A. Blokhuis & A.E. Brouwer. *Locally 4-by-4 grid graphs.*
AMS 05C25; 19 pp.; **key words:** Johnson scheme, locally grid graph.
Abstract: We investigate locally grid graphs. Main results are: (i) a characterization of the Johnson graphs (and certain quotients of these) as locally grid graphs such that two points at distance two have precisely four common neighbours, and (ii) a complete determination of all graphs that are locally a 4x4 grid (it turns out that there are four such graphs, with 35, 40, 40 and 70 vertices).

PM-R8402. A.M. Cohen & A.G. Helminck. *Trilinear alternating forms on a vector space of dimension 7.*
AMS 15A72; 20 pp.; **key words:** alternating forms, invariants.
Abstract: For vector spaces of dimension almost 7 over fields of cohomological dimension almost 1 (including algebraically closed and finite fields) all trilinear alternating forms and their isotropic groups are determined.

PM-R8403. J. de Vries & J.C.S.P. van der Woude. *Invariant measures and the equicontinuous structure relation I.*
AMS 54H20; 12 pp.; **key words:** equicontinuous structure relation, regionally proximal relation, invariant measures, weak mixing.
Abstract: In this introductory paper we introduce and illustrate some notions and problems from Topological Dynamics. This discipline originated from the qualitative theory of differential equations (work of Poincaré, Lyapunov, Birkhoff and others). This paper concerns 'abstract' Topological Dynamics: there is no direct relationship with differential equations. After the necessary definitions (Sections 1,2,3) we consider the problems of when the regionally proximal relation of a minimal flow is an equivalence relation and when a minimal flow which has no non-trivial equicontinuous factors is weakly mixing. In Sections 4 and 5 we state and prove a result of MacMahou's, namely that the answer to both problems is affirmative if the flow has an invariant measure. Although the results are not new, the proofs are simpler than the existing ones.

PM-R8404. J. de Vries & J.C.S.P. van der Woude. *Invariant measures and the equicontinuous structure relation II: The relative case.*
AMS 54H20; 15 pp.; **key words:** minimal flow, weakly mixing, relatively invariant measure, relative equicontinuous structure relation, relative regionally proximal relation.
Abstract: In this expository paper we discuss some notions from (abstract) Topological Dynamics. Moreover, we present self-contained simple proofs of the following results. Let $\phi: X \to Y$ be an open extension of minimal flows and suppose that ϕ admits a relatively invariant measure. Then $Q_\phi = E_\phi$, i.e. the relative regionally proximal relation is an equivalence relation. Also, if $E_\phi = R_\phi$ (that is, ϕ has no non-trivial almost periodic factor), then ϕ is weakly mixing.

PM-R8405. J. de Vries. *A note on the G-space version of Glicksberg's theorem.*
AMS 54H15; 2 pp.; **key words:** G-space, G-compactification, pseudocompactness.
Abstract: In an earlier paper, Glicksberg's theorem about the Stone-Čech compactification of products was generated by the author to the context of G-spaces and their maximal F-compactifications, where G is an arbitrary locally compact group acting on all spaces under consideration. However, in that paper only products of finitely many factors were considered. In the present note, infinite products are taken into account.

CWI Activities

Summer 1984

With each activity we mention its frequency and (between parentheses) a contact person at CWI. Sometimes some additional information is supplied, such as the location if the activity will not take place at CWI.

Study group on Analysis on Lie groups. Semisimple pseudo-Riemannian symmetric spaces. Joint with University of Leiden. Biweekly. (T.H. Koornwinder)

Lecture series 'The Spherical Fourier Transform for Semisimple Lie groups'. Biweekly. (T.H. Koornwinder)

Seminar on Algebra and Geometry. The Leech lattice. Biweekly. (A.E. Brouwer)

Study group on Cryptography. Biweekly. (A.E. Brouwer)

Colloquium 'STZ' on System Theory, Applied and Pure Mathematics. Twice a month. (J. de Vries)

Lecture series 'The Hamiltonian Formalism'. 6, 8, 12, 13, 15 June. Invited speaker: B. Kupershmidt (University of Tennessee, USA). (M. Hazewinkel)

Study group 'Biomathematics'. Lectures by visitors or members of the group. Joint with University of Leiden. (J. Grasman)

Study group 'Nonlinear Analysis'. Lectures by visitors or members of the group. Joint with University of Leiden. (O. Diekmann)

Progress Meetings of the Applied Mathematics Department. New results and open problems in biomathematics and analysis. Weekly. (N.M. Temme)

Study group 'Semiparametric Estimation Theory'. Lectures by members of the group on non-parametric maximum likelihood estimators, density estimation, etc. Biweekly. (R.D. Gill)

National study group on statistical mechanics. Joint with Technological University of Delft, Universities of Leiden and Groningen. Monthly. University of Amsterdam. (H. Berbee)

Progress meetings of the Mathematical Statistics Department. New results in research and consultation projects. Monthly. (R.D. Gill)

Colloquium on Queueing Theory. Triweekly. (E.A. van Doorn)

Progress meetings on Combinatorial Optimization. Biweekly. (J.K. Lenstra)

System Theory Days. 13 July. Main Speaker: P.R. Kumar (University of Maryland, Baltimore County, USA). (J.H. van Schuppen)

Study group on System Theory. Biweekly. (J.H. van Schuppen)

Study group on Differential and Integral Equations. Lectures by visitors or group members. Biweekly. (H.J.J. te Riele)

Study group Numerical Flow Dynamics. Lectures by group members. Every Wednesday. (J.G. Verwer)

Summer course 1984 for high-school teachers. Lectures on Operations Research, Matrix Algebra, Statistics and Probability Theory, 3-Dimensional Geometry, Mathematical Models, and Electronic Data Processing. Zwolle, 9, 10 August. Eindhoven, 16, 17 August. Amsterdam, 23, 24 August.

Visitors to CWI from abroad

J. Adams (University of California, Berkeley, USA) 30 May. **W. Alt** (University of Heidelberg, West Germany) 12 June. **D.G. Aronson** (University of Minnesota, USA) 24 May. **J. Beirlant** (University of Louvain, Belgium) 23 May. **L. Birgé** (University of Paris X, France) 3-6 April. **R.E. Bixby** (Rice University, Houston, USA) 11 May. **C. Blume** (University of Karlsruhe, West Germany) 18-19 May. **D. Chaum** (University of California, Santa Barbara, USA) 26 March - 15 August. **V. Chvátal** (McGill University, Montreal, Canada) 29 May. **W. Cook** (University of Bonn, West Germany) 28 May. **M. Csörgó** (Carlton University, Ottawa, Canada) 6 June. **N. Cutland** (University of Hull, UK) 3 April. **S.R. Dunbar** (University of Heidelberg, West Germany) 24 May. **H. Fujii** (Kyoto Sangyo University, Japan) 2-4 April. **G. Goos** (GMD, Bonn, West Germany) 9 February. **P. Hammerstein** (University of Bielefeld, West Germany) 13 April. **Hoang Huu Nhu** (University of Hanoi, Vietnam) 29 March. **A.F. Karr** (John Hopkins University, USA) 4-5 June. **B. Kupershmidt** (University of Tennessee, Tullahoma, USA) 6-26 June. **H.T. Lau** (Bell-Northern Ltd., Quebec, Canada) 17 May. **E.L. Lehmann** (University of California, Berkeley, USA) 14-18 May. **G. de Meij** (University of Gent, Belgium) 9 March. **C.L. Monma** (Bell Laboratories, Holmdel, USA) 24-27 June. **A. Neumaier** (University of Freiburg, West Germany) 26-27 March. **A. Odlyzko** (Bell Laboratories, Murray Hill, New Jersey, USA) 16-19 April. **T.J. Ott** (Bell Communications Research, Holmdel, USA) 4 June. **E. Pardoux** (University of Marseille, France) 19 April. **M. Pavon** (University of Padua, Italy) 7-18 May. **A. Pnueli** (Weizmann Institute, Rehovot, Israel) 16 March. **D.K. RayChaudhuri** (Ohio State University, USA) 1-2 April. **B. Schmitt** (University of Metz, France) 14 May. **A. Schönhage** (University of Tübingen, West Germany) 14-18 May. **J. Sekiguchi** (Tokyo Metropolitan University, Japan) 25 May. **T. Sellke** (Purdue University, USA) 18-20 June. **P. Tanner** (CRC, Ottawa, Canada) 15-18 May. **J.J. Tyson** (Virginia Polytechnic Institute, USA) 28-30 May. **D.A. Vogan** (M.I.T., Cambridge, Massachusetts, USA) 13 April. **R.A. Volz** (University of Michigan, USA) 18 May. **S. Walukiewicz** (Polish Academy of Sciences, Warsaw, Poland) 11 May. **R.A. Williamson** (University of Cambridge, UK) 5-6 April.

Order form for CWI publications

Centre for Mathematics and Computer Science
Kruislaan 413
1098 SJ AMSTERDAM
The Netherlands

☐ Please send the reports marked below on an exchange basis
☐ Please send the reports marked below with an invoice

	Publication code	Price per copy	Number of copies wanted
☐	CS-R8401	Dfl. 6.--	.....
☐	CS-R8402	3.70	.....
☐	CS-R8403	6.--	.....
☐	CS-R8404	3.70	.....
☐	CS-R8405	3.70	.....
☐	CS-R8406	3.70	.....
☐	CS-R8407	7.20	.....
☐	CS-N8401	7.20	.....
☐	OS-R8401	3.70	.....
☐	OS-R8402	3.70	.....
☐	OS-R8403	3.70	.....
☐	OS-R8404	3.70	.....
☐	OS-R8405	3.70	.....
☐	NM-R8401	4.80	.....
☐	NM-R8402	4.80	.....
☐	NM-R8403	3.70	.....
☐	NM-N8401	20.30	.....
☐	MS-R8401	3.70	.....
☐	MS-R8402	3.70	.....
☐	MS-R8403	3.70	.....
☐	MS-R8404	3.70	.....
☐	MS-R8405	3.70	.....
☐	MS-R8406	3.70	.....
☐	MS-R8407	3.70	.....
☐	MS-R8408	3.70	.....
☐	AM-R8401	4.80	.....
☐	AM-R8402	3.70	.....
☐	AM-R8403	3.70	.....
☐	AM-R8404	4.80	.....
☐	AM-R8405	7.20	.....
☐	AM-R8406	3.70	.....
☐	AM-R8407	6.--	.....

	Publication code	Price per copy	Number of copies wanted
☐	AM-R8408	3.70	.....
☐	PM-R8401	3.70	.....
☐	PM-R8402	3.70	.....
☐	PM-R8403	3.70	.....
☐	PM-R8404	3.70	.....
☐	PM-R8405	3.70	.....

Publications about *B* (see page 13)

☐	IW 208/82	Dfl. 3.70	.....
☐	IW 214/82	3.70	.....
☐	IW 219/83	3.70	.....
☐	IW 220/83	370	.....
☐	IW 248/83	3.70	.....
☐	CS-N8402	11.90	.....
☐	CS-N8404	3.70	.....
☐	CS-N8405	6.--	.....
☐	CS-R8408	3.70	.....
☐	ISBN 90 6191 238 2	12.10	.....

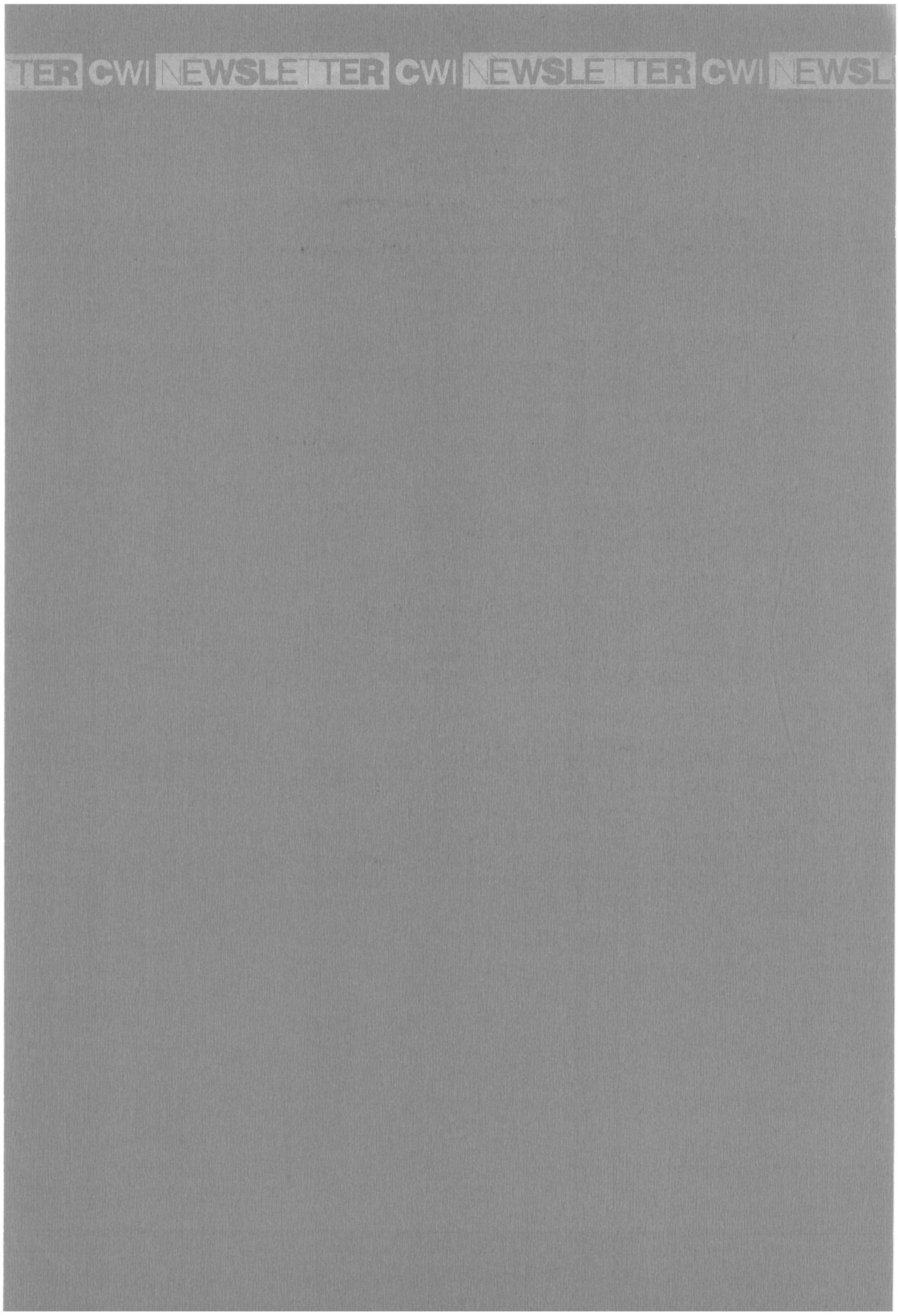
If you wish to order any of the above publications please tick the appropriate boxes and return the completed form to our Sales Department.
Don't forget to add your name and address !
Prices are given in Dutch guilders and are subject to change without notice. Foreign payments are subject to a surcharge per remittance to cover bank, postal and handling charges.

Name .....

Street .....

City .....

Country .....


TER CWI NEWSLE TER CWI NEWSLE TER CWI NEWSL

Contents