J M Bishop 20/3/90

South African Computer Journal
Number 1
January 1990

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 1
Januarie 1990

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

Assistant Editor: Information Systems
Professor Peter Lay
Department of Accounting
University of Cape Town
Rondebosch 7700

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Editorial

At last the first edition of SACJ is available. I trust that readers will find it worth the waiting. There have been a number of teething problems in getting things together, the many details of which need not be spelt out here. One significant challenge was to cope with the consequences of the resignation of Quintin Gee, QI's highly competent production editor. He assisted in the initial phases of getting this publication together but had to resign for personal reasons. It is fitting to acknowledge here not only his initial advice and assistance in getting this first issue of SACJ off the ground, but also the many hours of work that he spent in previously producing QI.

Quintin's resignation meant that a new *modus operandi* for typesetting and printing had to be established. The exercise was not only time-consuming, but also has significant cost implications. Fortunately, the Unit for Software Engineering (USE) at Pretoria University has generously agreed to sponsor this first edition. On behalf of the South African computing community, I should like to thank them for their generosity. Now that they have made a first issue of SACJ possible, it is hoped to solicit the sponsorship of one of the larger computer companies for future editions.

It might be of interest to take readers on a walk through the new journal to highlight various aspects. To begin with, the cover design follows that of several journals whose titles have the format: *The South African Journal of Subject / Die Suid-Afrikaanse tydskrif vir Vakgebied* (where *Subject* and *Vakgebied* are appropriately instantiated). While colours vary, these journals generally have *Subject* and *Vakgebied* restated on the darker portion of the cover. SACJ's title was chosen in preference to a more descriptive but also more cumbersome title such as *The South African Journal of Computer Science and Information Systems*. The appearance of the words *Computer Science and Information Systems / Rekenaarwetenskap en Inligtingstelsels* on the cover are thus out of step with the original inspiration, but seem appropriate under the circumstances.

The inside cover is of interest for several reasons. Firstly, note that Peter Lay has kindly agreed to lighten my task by acting as an assistant editor. He will deal with matters relating to Information Systems. *Contributions in this area should henceforth please be sent directly to him.* Also note that an editorial board of distinguished persons has been assembled. I should like to once again thank board members for adding status to SACJ by agreeing to serve in this capacity. They will be consulted on matters of editorial policy whenever appropriate. Finally, the subscription costs have been increased to keep pace with production costs. This increase does not affect SAICS members, who will continue to receive the journal as one of the benefits of membership.

The guest editorial by Pieter Kritzinger makes for interesting reading. Several points of concern about computer-related research in South Africa are raised. I trust that the article will focus attention on these problems and stimulate a debate which will lead to eventual solutions. It is hoped to make guest editorials a regular feature of future SACJ issues.

Of the eight research papers offered in the journal, four have been gone through the normal channel of refereeing and revisions. The remainder were submitted to the Vth SA Computer Symposium and are published here by invitation. Each paper submitted to the chairman of the symposium's program committee was sent to three referees. A ranking scheme, reflecting an aggregate measure of referee evaluation, was used as a basis for deciding on papers to be presented. After further editorial evaluation, the authors of four of the five highest ranking papers were invited to submit their papers to SACJ. While it was not possible to contact the fifth author in time for this edition, but it may be possible to publish that paper, together with a selection of others from the symposium, in future SACJ editions.

In the section marked *Communications* various items of news arriving at the editor's desk have been published. It was particularly gratifying to receive book review submissions in response to a prior general appeal. There has also been an enthusiastic response from book publishers, who have sent in a number of books for review. Titles are listed in the *Communications* section. Please contact me if you are willing to review one (or more) of these. Naturally, reviews of other books of interest in your possession will also be welcomed.

The final point to highlight in this walk through the journal is the increase in page charges indicated on the back inside cover. These reflect the increased cost of production. Since research papers in SACJ qualify for state subsidy at academic institutions, the charges should not, in general, present major problems for authors. However, it is worth pointing out that the final format of papers submitted significantly impacts on both the financial and editorial load. Submissions in camera-ready format (or nearly so) result in both a cost savings and a speed up of turn-around time by several orders of magnitude. Since many readers may not be familiar with the printing process, it may be helpful to say something about it in order to substantiate this claim.

The printing process basically involves typesetting, shooting (or photographing), and then reproduction and binding. Apart from limiting the amount of material, the printer's client has very little control over the cost of shooting, reproduction and binding. On the

other hand, anyone equipped with moderate text- or word processing facilities and a laser printer can go a long way (if not all the way) towards typesetting a paper. Even a partially typeset paper helps significantly, as I will explain below.

By typesetting I simply mean knocking the paper into the right shape and producing a laser printout. The printers regard this is a tedious, error-prone task, even if they start off with an ASCII file rather than a hardcopy of the paper. Consequently, they tend to handle large-scale typesetting by subcontracting the task. Moreover, while they may be willing to typeset uncomplicated text, they tend to balk at text containing specialized mathematical and other notation. However, they are quite skilful at cutting and pasting text, and at enlarging or reducing photographed or scanned diagrams. They are even willing to redraw sketches which are not too complicated.

As a result of the above, I have pressed several authors to do their own typesetting. In cases where it was problematic to produce double column format, a single column of appropriate width was requested. While this is a second-best option, it allows for cutting and pasting to be done by the printers. Some sketches have either been directly reduced from the author's original, while others have been redrawn by the printer. By way of exception, I have personally undertaken the typesetting of a few papers using WordPerfect. However, I would like to avoid this as far as possible in future, and consequently appeal to potential authors to make every effort to do their own typesetting.

From SACJ's point of view encouraging authors to do their own typesetting involves a compromise in that there will inevitably be slight variations in the print from one article to the next (as is in fact the case in this issue). If you are pedantically inclined, you might consider this to be a disaster. Personally, I regard it as a rather neat advertisement for the typesetting skills of SACJ contributors.

As an aside, since the handling of TEX files was initially a problem for me, I was pleased to discover that Peter Wood and his colleagues at UCT have mastered the art of producing TEX printout in the format now before you. Future authors who use TEX should consult them on details.

As to the future, it is not possible at the this stage to commit to a fixed number of SACJ issues per year. The number of issues is constrained by finance, submissions of the right quality, and time available to the editorial staff (including our anonymous and unsung heroes - the referees). The ideal is to produce four issues per year, but this may not always be attainable.

In conclusion, if readers have as much fun in reading this first issue of SACJ as I have had in editing it, the hours spent on it will have been well worthwhile. Hopefully SACJ is destined not only to be a permanent feature of the Southern African computing scene, but also to significantly contribute to research in the region.

Derrick Kourie
Editor

This SACJ issue is sponsored by
The Unit for Software Engineering
(USE)
Department of Computer Science
Pretoria University

Guest Editorial

Funding Computer Science Research in South Africa

P S Kritzinger
Department of Computer Science, University of Cape Town, Rondebosch 7700

The word *research* has many connotations and is often abused. In everyday language a person does not simply *search for information in a library*, for example, but rather does *research*, thus pretentiously conferring an aura of intellectual activity on an effort which requires very little original thought.

Here I will interpret the term to mean work which generates results that gain international recognition. This implies that the work is published in good international journals or presented at international conferences. I believe this is the only valid index of the quality of research.

With very few exceptions, the computer industry in South Africa is a consumer of computer technology, rather than a developer. In contrast with, say, the chemical industry, there is therefore no tradition of research in computer science in the South African computer industry and computer science researchers therefore have, as virtually their only source of funding, the Foundation for Research Development (FRD) which has its origins in the CSIR.

The FRD was formed in April 1984 with the development and use of research expertise in the natural and applied sciences and engineering as its mission. This mission is primarily directed at the universities, museums and technicons with the ultimate aim of improving the life of all South Africans.

Although the FRD has several programmes, the two which are of main concern to computer scientists are the Core Programmes and the Special Programmes.

FRD Core Programmes foster the optimum development of a scientific and technological knowledge base by supporting individual self-initiated research. These programmes, started only about 4 years ago, have met with considerable acclaim, particularly in regard to the way in which research funding for a particular individual is decided. To qualify for support within a Core Programme, researchers must obtain a certain evaluation status within the FRD and funding is then linked directly and exponentially to the merit of the individual concerned, rather than being linked to the specific project proposed.

In the evaluation process, peer review is strongly emphasised. The researcher himself is expected to nominate referees, whose status and reports play a decisive role in the evaluation. As a result of this evaluation, an applicant is assigned a specific evaluation status category. There are currently 9 categories in all, but the ones of main interest are:

- **A** researchers who are without any doubt accepted by the international community as being amongst the leaders in their field (52);
- **B** researchers not in category **A** but who nevertheless enjoy considerable international recognition as independent researchers of high quality (182);
- **C** proven researchers who have maintained a constant high level of research productivity and whose work is regularly made known internationally, or proven researchers whose current research output is less but who are actively engaged in scholastic activity (433);
- **P** researchers younger than 35 years of age who have already obtained a doctoral degree and who have shown exceptional potential as researchers (10); and
- **Y** young researchers usually under 35 years of age, who are highly likely to achieve C status by the end of their support period (108).

The number of researchers in the various categories as of August 1989 has been indicated in parentheses above. Of these, only 7 persons are computer scientists: 1 in category B; 3 in category C; and 3 in category Y. Only 4 departments of computer science are involved.

The other main programmes of concern to computer persons are the Special Programmes which aim at developing research manpower in priority areas. After identification of an area that merits particular research development, given local expertise, a Special Programme is launched to address the problem in the national interest.

Although a manager of a Special Programme has to be an FRD evaluated researcher, the same need not be true for the other team members. Regular peer evaluation of researchers as well as evaluation of the progress and results of Special Programmes are considered essential. Special Programme awards will be made for the first time towards the end of 1989. It is therefore not yet known whether proposals already submitted for programmes in computer science have been successful.

It is clear that, in the context explained above, there is virtually no computer science research being done in South Africa - a scary thought which has considerable implications for this country! Why is this so? There are several reasons, but I would like to single out two in particular.

Qualified faculty and students is an abiding problem at the heart of computer science departments. Acquisition of new faculty members is an issue intimately linked to the number of graduate students successfully completing PhD degrees. This problem is by no means unique to South Africa. For instance, data gathered in North America indicates that in 1983 there were over 200 vacancies in the 91 departments that have doctoral programmes in computer science. At the same time, only approximately 250 PhD's were granted in North America - a figure that has remained relatively unchanged for the past several years. A large number of those graduates were attracted to industry and industrial research laboratories. Although I do not have solid data at my disposal, I would think that South Africa produces at most one PhD graduate in computer science per year. There are currently 20 departments of computer science at universities is South Africa. It will therefore take us 20 years to locally produce one new faculty member with a PhD in computer science for every university.

Contributing to the above problem is our current academic image. The graduate student usually sees concerned computer science faculty members as rather harried individuals, having large undergraduate classes, much committee and professional work, and labouring under an ill-fitting model (applicable to more established disciplines) for decisions on tenure, salary and promotion. Further, as undergraduates, many prospective graduate students were not engaged in research projects involving computer science faculty, and for that reason were not exposed to graduate students doing research, and rarely developed a camaraderie with any computer science professionals. At last count there were only 5 individuals in South Africa who completed their computer science doctorate at a university outside South Africa where they had the good fortune to work in an environment in which sufficient faculty and funds were available to create an ethos of research. It is difficult to convince students that their interests and goals can be served by a PhD in computer science or by an academic career.

The second problem, which is of greater concern to me since there is no immediate solution to it, has to do with the fact that senior persons who decide the fate and fortune of academic computer science departments are, in general, individuals whose professional careers started well before computing machines came into every day use - that is to say, in the years B.C. (Before Computers). These persons of influence do not always understand what "computers" are, and what their potential influence upon the workplace in particular and society in general are. As far as research (as opposed to teaching) is concerned, most of them understand that a medical school needs special and expensive equipment (not to mention, expensive faculty) and that engineers must have a workshop and special machinery to teach their students and conduct research. They understand that if one needs to build up a defense industry, it will cost billions of rands; but they are not so sure about computer science, even though many other countries have recognised it as of national strategic importance.

I believe that only time and dedication will lead to a solution of these seemingly insurmountable problems and allow computer scientists to take their rightful place in the research community in South Africa.

Bibliography

P J Denning, [1981], Eating our seed corn, *Communications of the A.C.M.*, **24**(6), pp.341 - 343.

J Tartar (Ed.), [1985], The 1984 Snowbird Report: Future issues in computer science, *Communications of the ACM*, **28**(5), pp490 - 493.

D Gries, R Miller, R Ritchie and P Young, [1986], Imbalance between growth and funding in academic computing science: two trends colliding, *Communications of the A.C.M.*, **29**(9), pp.870 - 878.

J E Hopcroft and D B Krafft, [1987], Towards better computer science, *IEEE Spectrum*, pp.58 - 60.

Programming Using Induction

C.S.M. Mueller
Department of Computer Science, University of the Witwatersrand, Johannesburg, 2050 Wits

Abstract

Induction has been used as a major tool in mathematics in proving theorems. This paper explores whether it has an equally important role in formulating programs. There are those who advocate that induction should be used in the design of algorithms. An attempt is made here to take the argument further and propose that induction should be the way to express iteration in programs. An alternative way of stating iteration in the form of a generalised expression is suggested in the place of loops or recursive constructs.
Keywords: *induction, programming, programming language constructs, verification.*
Computing Review Categories: *D.3.3 language constructs, F.3.1 specifying and verifying and reasoning about programs*

Recieved March 1989, Accepted September 1989

1. Introduction

Obviously formal methods of programming should ideally be used in everyday programming but it is not practical. Up to now, these methods have been developed to fit in with existing languages. Could not the problem with formal methods be that they are handicapped by the unsuitability of languages? The paper considers whether languages could be made more appropriate by considering whether it is possible to design a language construct based on formalism rather than the other way around. The inductive process has been selected to see if a construct can be developed based on standard mathematical methods for expressing induction. Being able to develop this construct supports the view that languages are unsuitable for formal methods and need to change.

Iteration, be it recursion or looping, is an essential aspect of any non-trivial program. They are the most complex constructs as they are semantically complex. The complexity of such constructs is illustrated by their formal definitions and the difficulty to prove them correct [6]. An alternative programming language construct is proposed here based on well established concepts of induction used in mathematics to prove theorems correct [12]. This alternative is argued to be an improvement on current approaches in that it

- simplifies the semantics of iteration
- requires less skill to prove correct
- suggests a method of programming iteration.

Induction is a widely used tool in mathematics to prove theorems correct and in particular to prove programs with iterative algorithms correct [8]. For this reason, it would be desirable if a program reflected the inductive process required to prove it correct explicitly in the code. Conversely the inductive process on which the program is based should be mirrored in the program. Thus ideally induction should be used for specifying, coding and verifying the correctness of the iterative parts of a program.

Iterative constructs' semantics are related to the mechanisms provided by a computer to evaluate these constructs. The design of these constructs has been influenced by the mechanism available on current computers [1]. These constructs were introduced with the advent of computers and did not form part of the mathematical notation prior to this time. A loop construct imposes a structure on a program typified by structured programming [3]. A loop needs to be abstracted out and the abstraction may have little relationship with the problem. This leads to the question being explored in this paper as to whether loop constructs as we know them today are the most appropriate way of expressing iteration.

2. Induction

There are three major stages to an inductive proof [12]. The first stage is to prove that a premise $P(n)$ holds for $n = 0$. The second stage is to show that if $P(n)$ is true for $n \geq 0$ then $P(n+1)$ holds. The final stage is to draw the inference that $P(n)$ holds for all integers $n \geq 0$. This can be expressed more formally as follows:

$$\begin{array}{ll} 1) & P(0) \\ 2) & \forall n\,[P(n) \Rightarrow P(n+1)] \\ \hline 3) & \therefore \forall n\, P(n) \end{array}$$

These stages are the same as those required for iteration. The first stage is related to defining the initial conditions of the iteration, in this way ensuring $P(0)$ is correct. The heart of the iteration is to define the function which gets

repeated or how $P(n+1)$ is defined in terms of $P(n)$ which really is the second stage of the induction proof. The final stage of the induction is to specify the termination conditions which should be the inference of the proof. Thus if iteration is specified using the three above steps then the proof of the iteration follows immediately from the specification of the iteration using an inductive proof.

To illustrate how iteration can be specified and proved, three examples are studied. The first is a trivial example to show the principle. The next example is a sort which, although a fairly complex algorithm, is small enough to be dealt with in a paper. The last problem is one which has cropped up recently in the correspondence in CACM [11, 5]. The problem is that of finding the first row of zeros in a matrix. The reason it was chosen is because it raised so much debate as to how it should be expressed in conventional programming languages.

3. Example 1 — Summation

This example is to express the iteration required to sum a list of numbers terminated by the first zero in the list. The proposition $P(n)$ in this case is that sum_n is equal to the summation of all $number_i$s such that $i < n$ and there does not exist an i such that $i < n$ and $number_i = 0$. The first stage is to define sum_0 such that $P(0)$ is true.

$$sum_0 = 0$$

The second stage is to define the mapping from sum_n to sum_{n+1} ensuring that if $P(n)$ is true than $P(n+1)$ holds. Such a mapping is:

$$sum_{n+1} = sum_n + number_n \quad \text{if } number_n \neq 0$$

Now the proposition $P(n)$ is assumed to be true for $n \geq 0$ and now $P(n+1)$ is shown to hold.

$P(n)$:

$$sum_n = \sum_{i=0}^{n-1} number_i \quad \text{if } \forall i,\ i<n\ number_i \neq 0$$

$P(n+1)$:

$$sum_{n+1} = sum_n + number_n \text{ if } number_n \neq 0$$

$$= \sum_{i=0}^{n-1} number_i + number_n$$

$$\text{if } \forall i,\ i<n\ number_i \neq 0 \text{ and } number_n \neq 0$$

$$= \sum_{i=0}^{n} number_i \quad \forall i,\ i<n+1\ number_i \neq 0$$

Having shown that $P(n+1)$ holds assuming $P(n)$ is true, it can be inferred that $P(n)$ for all $n \geq 0$. Hence it is correct to specify that:

$$answer = sum_n \quad \text{if } number_n = 0$$

The complete specification is as follows:

$$sum_0 = 0$$
$$sum_{n+1} = sum_n + number_n \quad \text{if } number_n \neq 0$$
$$answer = sum_n \quad \text{if } number_n = 0$$

4. Example 2 — Insertion Sort

This example is to express the standard insertion sort algorithm [7]. To simplify the algorithm, the numbers being sorted are assumed to be terminated with a zero and are all assumed to be greater than *min* and less than *max*. Each number to be sorted is represented by a $number_i$. On each pass of the sort, the numbers which are permuted are represented by $sort_{i,j}$ where i indicates the pass and j indicates the position in the permutation. The proposition $P(n)$ in this case is that the elements $\{sort_{n,0}, sort_{n,1}, \ldots, sort_{n,n+1}\}$ are a permutation of the first n elements $\{number_0, number_1, \ldots\}$ none of which are zero and *min* and *max* such that $sort_{n,j} \leq sort_{n,k}$ for all $j<k$. The first step is to specify $sort_{0,0}$ and $sort_{0,1}$ such that the premise $P(n)$ holds for $n=0$. This can be specified as follows:

$$sort_{0,0} = min$$
$$sort_{0,1} = max$$

The second step is to show how the $sort_{n,j}$'s map onto the $sort_{n+1,j}$'s such that the premise $P(n+1)$ holds if $P(n)$ is true. Such a mapping is:

$$sort_{n+1,i} = sort_{n,i} \quad \text{if } number_n \neq 0 \ \&\ number_n \geq sort_{n,i}$$
$$= number_n \quad \text{if } number_n \neq 0\ \&\ sort_{n,i} > number_n \geq sort_{n,i-1}$$
$$= sort_{n,i-1} \quad \text{if } number_n \neq 0\ \&\ sort_{n,i-1} > number_n$$

Given the above mapping it is necessary to prove that if $P(n)$ is true then $P(n+1)$ holds. It is thus necessary to prove:

1) $sort_{n+1,i} \leq sort_{n+1,j}$ for all $i < j$
2) $\{sort_{n+1,0}, sort_{n+1,1}, \ldots, sort_{n+1,n+2}\}$ is a permutation of $\{number_0, number_1, \ldots, number_n, min, max\}$

Since we can assume $P(n)$ is true, the following can be assumed:

$$sort_{n,0} \leq sort_{n,1} \leq \ldots \leq sort_{n,k-1} \leq sort_{n,k} \leq \ldots \leq sort_{n,n+1}$$

If $number_n \neq 0$ there must exist a k for $0 < k < n+1$ such that:

$$sort_{n,k-1} \leq number_n \leq sort_{n,k}$$

Since $number_n$ must be greater than *min* and less than *max* which are assumed to be two elements of the sequence of $sort_{n,j}$'s. It follows that:

$$sort_{n,0} \leq sort_{n,1} \leq \ldots \leq sort_{n,k-1} \leq number_n \leq sort_{n,k} \leq \ldots \leq sort_{n,n+1}$$

Since $\{sort_{n,0}, sort_{n,1}, \ldots, sort_{n,n+1}\}$ is a permutation of the first n numbers plus *min* and *max* then $\{sort_{n,0}, sort_{n,1}, \ldots, sort_{n,k-1}, number_n, sort_{n,k}, \ldots, sort_{n,n+1}\}$ is equal to $\{sort_{n+1,0}, sort_{n+1,1}, \ldots, sort_{n+1,n+2}\}$.

The last step of the inductive process is to draw the inference which is:

$$sorted_j = sort_{i,j} \quad \text{if } number_i = 0$$

The specification of the insert sort is as follows:

$sort_{0,0}$ = min
$sort_{0,1}$ = max
$sort_{n+1,i} = sort_{n,i}$ if $number_n \neq 0$ & $number_n \geq sort_{n,i}$
$= number_n$ if $number_n \neq 0$ & $sort_{n,i} > number_n \geq sort_{n,i-1}$
$= sort_{n,i-1}$ if $number_n \neq 0$ & $sort_{n,i-1} > number_n$
$sorted_j = sort_{i,j}$ if $number_i = 0$

5. Example 3 — Find First Row of Zeros in a Matrix

The problem to find the first row of zeros in a matrix was used in a letter in the Communications of the ACM [11] to argue the value of the "goto". The problem is stated as follows: "Let X be an $N \times N$ matrix of integers. Write a program that will print the number of the first all-zero row of X, if any." (Here, $X_{i,j}$ will be used to represent the entry in the i^{th} row and j^{th} column.) Subsequently there has been much debate as to how this problem should best be expressed. The volume of correspondence on the subject suggests that the problem is not trivial and is a suitable problem with which to explore iteration.

The proposition $P(n)$ chosen for this problem is that Q_n is true if the first n rows of X all contain at least one non-zero entry and Q_n false otherwise. The first stage of the inductive approach is to make sure $P(n)$ holds for $n=0$. The following statement ensures that this is the case.

$$Q_0 = \text{true}$$

The next stage is to define the mapping from Q_n to Q_{n+1} such that if the proposition $P(n)$ is true then $P(n+1)$ holds, which can be specified as follows:

Q_{n+1} = true if Q_n **and** $\exists$ j such that $X_{n+1,j} \neq 0$
= false if **not** Q_n **or** $X_{n+1,j} = 0$ $\forall$j

In the case that the first n rows of X all contain at least one non-zero, we can assume that Q_n is true since we are assuming $P(n)$ to be true and the $n+1^{st}$ row will contain a non-zero entry. The first line of the mapping implies that Q_{n+1} is true and thus $P(n+1)$ holds. If any of the first $n+1$ rows is a row of zeros then either one of the first n rows is a row of zeros and Q_n will be false or the $n+1^{st}$ row will be all zeros. The second line of the mapping implies that Q_{n+1} is false and thus $P(n+1)$ holds.

Establishing whether a row of the matrix X consists of all zeros is itself an iterative process and needs to be expressed in an inductive way. For this inner induction, the proposition chosen is that $Z_{i,n}$ is true if the first $n-1$ columns of row n are zero given that Q_i holds. This is obviously true for $n=0$.

$$Z_{i,0} = Q_i$$

The next step is to define the mapping from $Z_{i,n}$ to $Z_{i,n+1}$ such that if the premise is true for n it will hold for $n+1$. This can be expressed in a straight forward way as follows:

$$Z_{i,n+1} = Z_{i,n} \textbf{ and } (X_{i,n} = 0)$$

Now the inference at the inner level can be drawn:

$$Q_{i+1} = Z_{i,n} \quad \textit{where } n = N$$

The outer inference can also now be specified:

$$\text{first\_zero\_row} = n \quad \text{If } \textbf{not } Q_i$$

The whole specification can be stated as follows:

Q_0 = true
$Z_{i,0} = Q_i$
$Z_{i,n+1} = Z_{i,n}$ **and** $(X_{i,n} = 0)$
Q_{n+1} = **not** $Z_{n,m}$ *where m=N*
first_zero_row = n If **not** Q_n

6. Translating of an Inductive Specification into a Program

In the above three examples iteration is expressed in the form of induction which explicitly defines how to map the n^{th} values into the $n+1^{st}$ values. The benefit of such a specification is that it is easy to verify using induction. It also helps with the design of such a specification [8].

The question is how should these specifications be expressed as part of a program?

The conventional approach is to express iteration either imperatively as a loop or functionally as recursion. Both alternatives result in extra notation with the associated semantics to express the loop or recursive function. The loop requires code to specify which section of code must be repeatedly executed as well as the termination condition. The program's correctness depends on the loop being executed the correct number of times as well as the order of the statements within the loop. The recursive function requires parameters to be defined and the termination of the recursion to be coded. The program's correctness in this case depends on the correct arguments being passed, in particular, the initial values and the final test as to whether the recursion has completed. The semantics are considerably complicated by the recursive process which saves the current environment, creates a new environment on each recursive call and restores the old environment on returning from a function call.

In both cases it is argued that the mapping from the n^{th} onto the $n+1^{st}$ values is not immediately obvious when coded. In the case of a loop, the $n+1^{st}$ values are those assigned during the loop. Recursion uses parameters to pass the n^{th} values and returns the $n+1^{st}$ values. The result is that the coding of the specification no longer explicitly defines how the n^{th} values map onto the $n+1^{st}$ values.

Before looking at an alternative way of expressing the iterative process, let us consider how the mapping of the inductive specification is defined. The iteration specifies how a sequence of values $\{x_0, x_1, \ldots\}$ gets mapped onto a corresponding sequence of values $\{y_0, y_1, \ldots\}$ where x_{i+1} is defined in terms of y_i. The mapping is specified in terms of a generalised function which defines how to map any x_i onto the corresponding y_i and the function is specified in terms of how an element y_i is defined in terms of x_i. The argument is that a program should be expressed based on the same concepts without having to introduce extra semantic complexities such as loop constructs or recursive functions.

Consider the example of summation, here the sequence *sum* can be defined as follows:

$$sum=\{sum_0=0, sum_1=number_0+sum_0, sum_2=number_1 + sum_1, \ldots, sum_{i+1}=number_i+sum_i, \ldots\}$$

A new sequence *sum'* can be defined as:

$$sum'=\{sum_0'=number_0+sum_0, sum_1'=number_1+sum_1, \ldots sum_i'=number_i+sum_i, \ldots\}$$

Using sequence *sum'* then sequence *sum* can be simplified to:

$$sum = \{sum_0=0, sum_1=sum_0', sum_2=sum_1', \ldots, sum_{i+1}= sum_i', \ldots\}$$

This allows the specification of the summation to be stated as follows:

number = sequence of $number_i$'s

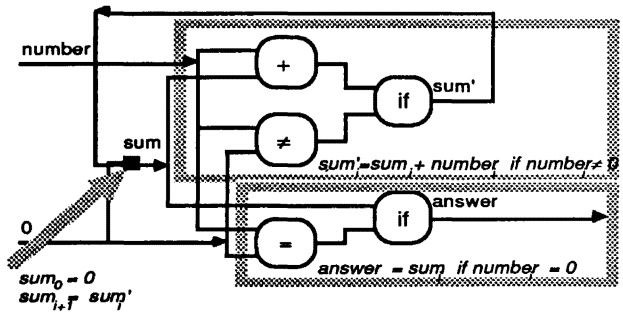
sum = sequence of sum_i's such that
$sum_0 = 0$
$sum_{i+1} = sum_i'$

sum' = sequence of sum_i''s such that
$sum_i' = sum_i + number_i$ if $number_i \neq 0$

answer = sum_i if $number_i = 0$

The above rewriting of the specification is a slight modification of the original. An important change is that the mapping is expressed in terms of how corresponding elements are related.

The specification in this form can now be expressed as a network as follows:



This network is similar to those used in data flow programs[13] except that the operations in this case apply to whole sequences of values. A language which allows programs to be expressed in a similar form to the above and an interpreter based on a data driven approach have been implemented [9, 10]. This language shows iteration can be expressed almost directly as an inductive proof.

7. Conclusions

In recent years there have been two calls: one for a more formal approach to programming and another for literate programs [2]. These two objectives can be achieved by moving towards thinking of programs as an expression of a proof which is meant to be read as a proof rather than a specification of what a computer is to execute. A proof is meant to be understood and in a similar way, a program should logically guide the reader in understanding the steps required to verify the correctness of a program. Surely this is what programming languages should be striving for.

This paper looks at an alternative way of expressing iteration based on the well established principle of induc-

tion. The paper shows that it is possible to express iteration as an outline of an inductive proof. The semantics required to understand the iteration are those required to prove the iteration. It does not rely on understanding the semantics of a loop construct artificially introduced to simplify the translation of a program into a suitable form for a computer. Hence the whole semantics of iteration is simplified by using standard mathematical concepts.

Since the objective of the method is to express iteration as a outline of an inductive proof, proving the iteration should be straight forward and verifying programs should become a practical possibility. If the three stages are clearly laid out for the reader of a program, the correctness should be easy to verify. Not only should it be valuable for verification but is should also help programmers with the design of a program [8].

Two schools of thought are developing in the programming community: those who feel that it is unprofessional not to use formal methods [4] and those who consider it infeasible to apply formal methods to any reasonable size of programming task. This paper questions whether the basic tools we are working with are not at fault and by devising better tools the gap between these two schools of thought can be bridged.

The paper has considered mainly iteration to see if it can be handled in a more suitable way. If a program is expressed in a way suitable for the reader such that it is easy to verify, will this not encourage formal verification methods to be applied to reasonable sized programs? It is argued that the examples show that it is.

References

[1] J. Backus, [1978], Can Programming be Liberated from the von Neumann Style? A functional Style and its Algebra of Programs, *Comm ACM*, **21**, 613-641.

[2] J. Bentley, [1986], Literate Programming, *Comm ACM*, **29**, 364-369.

[3] E.W. Dijkstra, [1968], GOTO Statement Considered Harmful, *Comm ACM*, **11**, 147-149

[4] E.W. Dijkstra, [1976], *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, N.J.

[5] E.W. Dijkstra, [1987], One More Time, *Comm ACM*, **30**, 661-662.

[6] C.A.R. Hoare, [1987], An Overview of Some Formal Methods for Program Design, *IEEE Computer*, **20**, 85-91.

[7] D.E. Knuth, [1973], *The Art of Computer Programming — Vol 3: Sorting and Searching*, Addison-Wesley, Reading Massachusetts.

[8] U. Manber, [1988], Using Induction to Design Algorithms, *Comm ACM*, **31**, 1300-1313.

[9] C.S.M. Mueller, [1987], Set-Oriented Functional Style of Programming, *Quæstiones Informaticæ*, **5**, 29-39.

[10] C.S.M. Mueller, [1988], *Towards Removing Sequential Ordering in Programs*, PhD Thesis, University of the Witwatersrand, Johannesburg.

[11] F. Rubin, [1987], "GOTO Considered Harmful" Considered Harmful, *Comm ACM*, **30**, 195-196.

[12] D.F. Stanat, [1977], *Discrete Mathematics in Computer Science*, Prentice-Hall, Englewood Cliffs, N.J.

[13] A.H. Veen, [1986], Dataflow Machine Architecture, *ACM Computing Surveys*, **18**, 365-396.

Computers and the Law

Submitted by Antony Cooper
CSIR

The SA Law Commission has established a commission on "The Legal Protection of Information".

The commission is still in its preliminary stages and the assigned researcher, Mr Herman Smuts, is still preparing the working paper. He does not know when it will be finished, but once the working paper has been prepared, they will invite comments for about two years, before preparing the final report. I have contacted Mr Smuts, and he would be most grateful to receive input at this stage, especially regarding the terms of reference of the commission. His address is:

C/o SA Law Commission
Private Bag X668
PRETORIA
0001

In addition, there is an ad-hoc committee at the Registrar of Copyright investigating numerous copyright issues, including those relating to software and data. Mr Smuts' commission will be liaising with the ad-hoc committee.

I feel that SAICS has an obligation to submit evidence to the commission, and I would appreciate it if you would circulate the members of the Council of SAICS, and perhaps the general membership as well, to solicit ideas concerning SAICS's input.

I shall prepare something for the commission, either in my personal capacity, or in my professional capacity here at CSIR. I would be willing to assist in the preparation of any evidence SAICS might submit.

4th National MSc/Phd Computer Science Conference

Report by Danie Behr
University of Pretoria

This conference was held from 7th to 10th September 1989 at the Cathedral Peak Hotel in the Drakensberg. The conference was attended by 61 postgraduate students from 11 South African universities. Most were engaged in MSc studies, although 5 Phd students also attended. These numbers are encouraging for the South African computer science community. This type of conference is rather unique in that it affords students the opportunity of sharing their research, and getting to know other researchers in the country. The number of Afrikaans and English speaking students attending the conference were roughly equal. Presentations were made in the language preferred by the student. Invitations were sent to all universities with computer science departments. The conference was organized by the students themselves.

Some of the more popular research topics that were presented included expert systems, data communications, computer security, graphics, software engineering, user interfaces and data bases. The main sponsor for this year's conference was the Division for Microelectronic Systems and Communication Technology of the CSIR. The conference was opened with an interesting talk on the myths and motivations of post graduate studies by Prof DG Kourie, acting head of the Computer Science Department at Pretoria University.

The next conference will be presented by the University of Port Elizabeth. People requiring further information about the next conference should contact Andre Calitz, Charmaine du Plessis or Jean Greyling of the Department of Computer Science at UPE.

A list of authors and papers presented at the symposium follows:

S Crosby, University of Stellenbosch
Performance Analysis of Wide Area Computer Communication Networks

A B Joubert, PU for CHE Vaal Triangle Campus
Image Processing Libraries

A Calitz, University of Port Elizabeth
An Expert System Toolbox to assist in the classification of objects

L von Backström, University of Pretoria
Integrated Network Management

R Foss, Rhodes University
The Rhodes Computer Music Network

A McGee, University of Natal
On Fixpoints and Nondeterminism in the Sigma-Lambda Calculus

P G Mulder, Randse Afrikaanse Universiteit
A Formal Language and Automota approach to Data Communications

A Tew, Randse Afrikaanse Universiteit
Drie dimensionele grafiek grammatikas

T C Parker-Nance, University of Port Elizabeth
Human-Computer Interaction: What Determines Computer Acceptance

E Coetzee, PU vir CHO Vaaldriehoekkampus
Opsporing van rande in syferbeelde dmv verskerping en drempelbepaling
D A Sewry, Rhodes University
Visual Programming
A Cooper, University of Pretoria
Improvements to the National Exchange Standard
E S Badier, University of Port Elizabeth
A Computer Assisted Diagnostic System (CADS)
C du Plessis, Universiteit van Port Elizabeth
Persoonsidentifikasie dmv naampassing in 'n genealogiese databasis
J Greeff, University of Stellenbosch
The Entity-Relationship Model and its Implementation
D A de Waal, PU vir CHO
Flat Concurrent Prolog (FCP) en Flat Guarded Horn Clauses (FGHC): 'n Vergelyking
E Naude, UNISA
Interne metodes in Liniêre Programmering
A Deacon, University of Stellenbosch
Global consistency in non-locking DDBMS
A Wilks, Rhodes University
The Synchronisation and Remote Configuration of the Resources in a Computer Music Network
J Greyling, University of Port Elizabeth
The design of a User Interface with special reference to an Interactive Molecular Modelling Program
L Drevin, PU vir CHO
Rekenaarsekuriteit: Verskillende vlakke van kontrole
Dieter C Barnard, University of Stellenbosch
The design and implementation of a modest, interactive proof checker
R A Schmidt, University of Cape Town
Knowledge Representation Systems and the Algebra of Relations
J Hartman, Randse Afrikaanse Universiteit
Die Gebruik van Objek-georiënteerde Programmering in die Moderne Sneltrein Omgewing
S Lawrie, Rhodes University
The Design and Implementation of a System for the Interactive Control of a MIDI-based Studio
E Mulder, Rand Afrikaans University
A Formalisation of Object-Oriented Principles
C J Tolmie, UOVS
Die Ontwikkeling van 'n Ekspertrekenaarstelsel vir die beoordeling van die resultate van die Technicon H1-Bloedselanaliseerder
R Breedt, University of Pretoria
Realism with Ray Tracing
J van Jaarsveld, University of Pretoria
Developing Medical Expert Systems: A knowledge acquisition perspective
W Appel, University of Pretoria
TCP/IP Implementation on Ethernet
E Goedeke, University of Natal
Eggspert's Control Structure
M Harmse, University of Stellenbosch
Modelling of I/O Subsystems
H L Viktor, University of Stellenbosch
A Quantitative Model for Comparing Recovery Techniques in a Distributed Database
M Olivier, Randse Afrikaanse Universiteit
Rekenaarvirusse in Suid-Afrika

Book Reviews

An Introduction to Functional Programming Through Lambda Calculus
by Greg Michaelson, Addison-Wesley, 1988.
Reviewer: Dr. E P Wentworth, Rhodes University

Recently we have seen a number of excellent *second generation* texts on Functional Programming. Michaelson's text assumes some previous programming experience with imperative languages, and presents the functional approach as an alternative paradigm. He begins with a very accessible exposition of the Lambda Calculus, and carefully develops this foundation to encompass the important aspects and paradigms of functional programming. The programming notation is language-independent, although the last chapters are devoted to a brief look at two specific languages, Standard ML and Lisp. The examples and exercises are mainly utility in nature, e.g. "insert a sublist after the first occurrence of another sublist in a list", and can generally be solved in a couple of lines. Answers to the exercises are provided in an appendix.

The approach is slanted towards developing a solid base for understanding functional languages and computing. In this respect the book achieves a good balance between the theoretical underpinnings and their practical application. On the practical side, however, I found the lack of more substantial examples and exercises disappointing. Most programming texts tackle a set of 'standard' problems which are well-understood in the academic community and provide an informal benchmark for comparisons. Since the book is targeted for those already versed in imperative languages and standard algorithms, one might expect the examples to clearly demonstrate the elegance and power of the *problem-oriented* functional approach in these areas. Having laid an excellent foundation I was left with the feeling that the book failed to capitalize and deliver the cherry on the top.

The book is highly recommended as one of the new breed of Computer Science books which gives substantial attention to the fundamentals of the subject without becoming bogged down in over-rigorous formality.

Artificial Intelligence and the Design of Expert Systems
by George F Luger & William A Stubblefield, The Benjamin/Cummings Publishing Co., 1989.
Artificial Intelligence: A Knowledge-based Approach
by Morris W Firebaugh, PWS-Kent Publishing Co., 1989.
Reviewer: Prof G D Oosthuizen, University of Pretoria

One of the primary goals of an Honours course is to introduce students to a field in such a way that they arrive at enough insight into relevant issues to enable them to conduct further research on their own. To this end a text book which is used ought to reflect the current view of the field. Because of the rapid expansion of the field of Artificial Intelligence (AI), we have now finally outgrown the era dominated by the books by Winston and Charniak and McDermott. In the past five to ten years much new work has been done, and new insights have been gained. Introducing AI, therefore, requires a marked shift from the previous emphasis on a few historical systems embodying a number of famous methods, to a more generic approach - an approach which highlights those fundamental representation and search models that span all the different application areas and strategies of problem solving. Of course, since AI still does not have a well developed theory, references to seminal systems continues to fulfil an important role.

Both of the above books are good text books, characterised by a balanced coverage of Prolog and Lisp. They also reflect and consolidate much of the work of the past few years done in areas such as knowledge representation, machine learning, the work done under the heading of Expert Systems and even the recent work on neural networks. But the most important feature that they share is the accurate and up to date overall picture of the subject provided; the broad framework for the understanding of AI that is created without neglecting work of historical importance. There are still references to these works, but they are placed in perspective in relation to new developments.

The book of Luger & Stubblefield (L&S) is more language oriented than Firebaugh's book. A characteristic of L&S is that AI approaches to representation are related to the Object Oriented approach. Whereas L&S includes chapters on advanced AI programming techniques in Prolog and Lisp, it does not address pattern recognition, computer vision and robotics. (Firebaugh has chapters on each of these themes.) These omissions are understandable, since AI has diversified so much recently that it is difficult to cover all applications in one book.

If I had to select one of the books, it would be L&S. Although L&S gives poor coverage of Machine Learning, the book's overall presentation is very good. In particular, the chapters are well-organised, and the overall approach to AI - starting with the core aspects of *representation* and *search*, followed by chapters on AI languages - is coherent. The authors also make very good use of graphical representations and illustrations to convey ideas.

Books Received

The following books have been sent to SACJ. Anyone willing to review a book should contact the editor. The book will be sent to him for review, and may be kept provided that a review is received.

- D Bustard, J Elder & J Welsh, [1988], *Concurrent Program Structures*, Prentice-Hall Inc., Englewood Cliffs.
- R Cafolla & A D Kauffman, [1988], *Turbo Prolog Step by Step*, Merrill Publishing Company, Columbus, Ohio.
- S Hekmatpour, [1988], *Introduction to LISP and Symbol Manipulation*, Prentice-Hall Inc., Englewood Cliffs.
- K L Clark & F G McCabe, [1984], *micro-PROLOG: Programming in Logic*, Prentice-Hall Inc., Englewood Cliffs.
- D Crookes, [1988], *Introduction to Programming in Prolog*, Prentice-Hall Inc., Englewood Cliffs.
- M J C Gordon, [1988], *Programming Language Theory and its Implementation*, Prentice-Hall Inc., Englewood Cliffs.
- J G Hughes, [1988], *Database Technology : A software engineering approach*, Prentice-Hall Inc., Englewood Cliffs.
- R Milner, [1989], *Communication and Concurrency*, Prentice-Hall Inc., Englewood Cliffs.
- T J Myers, [1988], *Equations, Models and Programs*, Prentice-Hall, Inc., Englewood Cliffs.
- N C Rowe, [1988], *Artificial Intelligence through Prolog*, Prentice-Hall Inc., Englewood Cliffs.
- D A Protopapas, [1988], *Microcomputer Hardware Design*, Prentice-Hall Inc., Englewood Cliffs.
- H Eisner, [1988], *Computer-aided Systems Engineering*, Prentice-Hall Inc., Englewood Cliffs.
- S H Unger, [1989], *The essence of logic circuits*, Prentice-Hall Inc., Englewood Cliffs.
- R J Young, [1989], *Practical Prolog*, Van Nostrand Reinhold, New York.

NOTES FOR CONTRIBUTORS

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between such cases as:
  - upper and lower case letters;
  - the letter O and zero;
  - the letter I and the number one; and
  - the letter K and kappa.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may provided in one of the following three formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies.

Charges

A charge per final page, scaled to reflect scanning, typesetting and reproduction costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format:	R80-00
ASCII format:	R60-00
Camera-ready format :	R20-00

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 1, January 1990
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 1, Januarie 1990
ISSN 1015-7999

Contents