# II

# ALGOL 68 Session

**Chair:** *Jacques Cohen*

## A HISTORY OF ALGOL 68

*C. H. Lindsey*

5, Clerewood Avenue, Heald Green,
Cheadle, Cheshire SK8 3JU, U.K.

### ABSTRACT

ALGOL 68 is a language with a lot of "history." The reader will hear of discord, resignations, unreadable documents, a minority report, and all manner of politicking. But although ALGOL 68 was produced by a committee (and an unruly one at that), the language itself is no camel. Indeed, the rigorous application of the principle of "orthogonality" makes it one of the cleanest languages around, as I hope to show. Moreover, when the language came to be revised, the atmosphere was quite different, enabling a much more robust and readable defining document to be produced in a spirit of true cooperation. There are some lessons here for future language design efforts, but I am not optimistic that they have been learned.

CONTENTS



### 2.1 INTRODUCTION

The world seems to have a rather negative perception of ALGOL 68. The language has been said to be "too big," to be defined by an "unreadable Report" produced by a committee which "broke up in disarray," to have no implementations, and to have no users. The only phrase that *everybody* can quote

from the Report is the one that says, "It is recognized, however, that this method may be difficult for the uninitiated reader" [R 0.1.1]. While these stories do have some basis in fact, which it will be the purpose of this paper to explore, they are at the least a considerable oversimplification of what really took place. In fact most of the people who pass on these things have never read the Report, or tried to use the language, and the observed fact is that those who have used the language have become totally addicted to it.

I should point out that my own involvement with the project came after the basic design of the language, and of its original Report, were complete. I was an onlooker at the fracas, as confused as any outsider as to what was going on. It is only now, in the course of studying the minutes and other documents from that time, that I have come to see what the real fuss was about, and I hope that all this has enabled me to take a dispassionate view of the events. The reader of this paper will certainly see discord, but I believe he will see also how good design can win through in the end.

ALGOL 68 clearly grew out of ALGOL 60, but many other language developments were taking place at that time—Lisp, COBOL, PL/1, Simula—all of them aimed at wider markets than the numerical computations for which ALGOL 60 was so well suited. The territory into which ALGOL 68 was now pushing was largely unmapped. Individual research languages had probed far ahead; ALGOL 68 was to advance into this landscape on a broad front. We indeed discovered what was there, but it seems to have been left to other, later languages to complete the colonization.

### 2.1.1 The Political Background

ALGOL 60 [Naur *et al.* 1960] was produced by a group of persons, half nominated by ACM and half by various European institutions. Apart from that initial sponsorship, the group had no official standing or authority. Meanwhile, the International Federation for Information Processing (IFIP, founded in 1960), through the auspices of its Technical Committee 2 on Programming Languages (TC2), had founded a Working Group on ALGOL (WG 2.1). At their meeting in Rome in April of 1962, upon completion of the Revised ALGOL 60 Report, the authors

> accepted that any collective responsibility which they might have with respect to the development, specification and refinement of the ALGOL language will from now on be transferred [to WG 2.1].

Thus, it will be seen that IFIP is a hierarchical organization with a General Assembly and a Council, a layer of Technical Committees, and a layer of Working Groups below that. There is a procedure for promulgating an "IFIP Document," the product of some Working Group, with some variant of the following wording:

> This Report has been accepted by Working Group 2.1, reviewed by Technical Committee 2 on Programming Languages and approved for publication by the General Assembly of the International Federation for Information Processing. Reproduction of the Report, for any purpose, but only of the whole text, is explicitly permitted without formality.

Over the last thirty years, some eight IFIP Documents have been produced by WG 2.1 under this procedure, as shown in Table 2.1.

IFIP does not have any significant funds at its disposal. The costs of bringing members together at a Working Group meeting falls on those members' employers or funding agencies, and on the organization hosting the meeting. If it were possible to add up the total cost of bringing ALGOL 68 before the world, it would come to a tidy sum. Whether the world has now the benefit of this investment is an interesting question.

**TABLE 2.1**
IFIP Documents produced by WG 2.1.

| | |
|---|---|
| Revised report on the algorithmic language ALGOL 60 | Naur *et al.* 1962] |
| Report on SUBSET ALGOL 60 (IFIP) | [WG 2.1 1964a] |
| Report on input-output procedures for ALGOL 60 | [WG 2.1 1964b] |
| Report on the algorithmic language ALGOL 68 | [Van Wijngaarden 1969] |
| Revised report on the algorithmic language ALGOL 68 | [Van Wijngaarden 1975] |
| A supplement to the ALGOL 60 revised report | [De Morgan 1976] |
| A sublanguage of ALGOL 68 | [Hibbard 1977] |
| The report on the standard hardware representation for ALGOL 68 | [Hansen 1977] |

### 2.1.2 Historical Outline

As will be seen from Table 2.1, The early years of WG 2.1 were spent in various activities arising out of ALGOL 60. But even as early as March 1964 there was mention of a language, ALGOL X, to be a "short-term solution to existing difficulties," and a "radically reconstructed" ALGOL Y [Duncan 1964].

Table 2.2 shows the milestones in the life of ALGOL 68, which can be divided into three phases.

- 1965 through 1969, culminating, in spite of much dissention, in the original Report [Van Wijngaarden 1969].
- 1970 through 1974, leading to the Revised Report [Van Wijngaarden 1975].
- 1975 onwards—the maintenance phase, resulting in various supplementary documents.

The milestones themselves are most conveniently remembered by the names of the meeting places at which the events took place.

### 2.1.3 Some Conventions

ALGOL 68 and the documentation that surrounds it adopt by convention various usages that I shall follow in this paper. In particular, I shall refer to [Van Wijngaarden 1969] as [R], often followed by a section number, and to [Van Wijngaarden 1975] as [RR]. Also, the long series of draft Reports leading up to [R] will be known by the numbers given them by the Mathematisch Centrum, of the form [MR*nn*]. References of the form (*n.m*) are to sections within this paper.

[R] is noted for its number of arcane technical terms used with a very precise meaning. Some were introduced purely to facilitate the syntax and semantics, but others denote concepts of definite interest to the user, and of these many are now in common parlance (for example, 'dereference', 'coercion', 'elaborate', 'defining/applied occurrence', 'environment enquiry'). Often, however, the authors chose terms for existing (or only marginally altered) concepts that have failed to come into common usage, such as the following:

| | | |
|---|---|---|
| mode | instead of | type |
| multiple value | " " | array |
| name | " " | variable |
| scope | " " | extent (a dynamic concept) |

| | | |
|---|---|---|
| reach | " " | scope (a static concept) |
| routine | " " | procedure |
| **range** | " " | **block** |
| **identity-declaration** | " | **constant-declaration** |

To avoid confusion, I shall be using the terms in the right hand column in the rest of this paper.

**TABLE 2.2**

Summary of the main events in the history of ALGOL 68.

The ORIGINAL REPORT

| Date | Meeting Place | Document | Events |
|---|---|---|---|
| May 1965 | Princeton | | Start of ALGOL X. Draft proposals for a full language solicited. |
| Oct 1965 | St. Pierre de Chartreuse | [MR 76] | Drafts presented by Wirth/Hoare, Seegmüller and Van Wijngaarden. Commission given to these four to produce an agreed draft. |
| Apr 1966 | Kootwijk | | Meeting of the commissioned authors. |
| Oct 1966 | Warsaw | [W-2] | Van Wijngaarden commissioned to produce and circulate a draft ALGOL X Report. |
| May 1967 | Zandvoort | [MR 88] | Progress meeting. |
| Feb 1968 | | [MR 93] | Draft Report [MR 93] circulated as a supplement to the ALGOL Bulletin. |
| Jun 1968 | Tirrenia | | Intense discussion of [MR 93]. |
| Aug 1968 | North Berwick | [MR 95] | A week of political wrangling. |
| Dec 1968 | Munich | [MR 100] | ALGOL 68 Report accepted by WG 2.1.3 |
| Sep 1969 | Banff | [MR 101] | Final cleanup before publication. |

The REVISED REPORT

| Date | Meeting Place | Events |
|---|---|---|
| Jul 1970 | Habay-la-Neuve | Discussion of "Improvements." Subcommittee on "Data processing." |
| Apr 1971 | Manchester | Subcommittee on "Maintenance and Improvements to ALGOL 68". |
| Aug 1971 | Novosibirsk | Decision to produce a Revised Report by the end of 1973. |
| Apr 1972 | Fontainebleau | Editors for the Revised Report appointed. |
| Sep 1972 | Vienna | Progress meeting. Hibbard's Sublanguage proposal. |
| Apr 1973 | Dresden | Progress meeting. |
| Sep 1973 | Los Angeles | Revised Report accepted, subject to "polishing". Subcommittee on ALGOL 68 Support. |
| Apr 1974 | Cambridge | Meeting of Support Subcommittee to discuss transput. |

See Table 2.5 (Section 2.6) for events after 1974 (the Maintenance Phase).

[R] is writing about language, and this always causes problems when one needs to distinguish clearly between a word and the thing it denotes. Distinguish, for example, between the following: "*John* has four daughters", "*John* has four letters", "*John* is in italic." In these the word "*John*" stands for, respectively, the object denoted by the word, the syntactic structure of the word, and the appearance of the word.

To ease this sort of problem, it has long been traditional to print fragments of program text in italic, but [R] goes further by introducing a special font for constructs produced by the grammar (*foo* is an **identifier**). Moreover, when talking about the grammar symbols themselves (or fragments thereof) single quotes are used as well (the mode of the **identifier** *foo* is '**integral**'). To express the text of an ALGOL 68 program, two alphabets are necessary, and in the days when computers did not support two cases of characters a variety of 'stropping' conventions (this term comes from [Baecker 1968]) were used (for example, *'REAL' X* or *.REAL X*, or *REAL x* once two cases did become available). However, it has always been traditional to use a bold font for the stropped words in printed documents (***real*** *x*). When preparing [RR] we took special care in choosing suitable fonts. The stropping font, although still slanted like italic, is also sans serif, and the font for constructs and grammar symbols is bold roman (as opposed to the insufficiently distinctive sans serif used in [R]).

I follow the same conventions in this paper, but I also adopt a practice not used in the Report, which is to use a type (usually written as a stropped word) to indicate some value of that type (*foo* is an ***int***). Thus I will talk about ***string***s and ***struct***s and ***proc***s, and by analogy, about ***array***s, even though the word "***array***" is not an official part of the language. And I take the liberty of applying these conventions to other languages where appropriate. Finally, to avoid confusion with an ever changing language, the examples in this paper are generally written using the syntactic sugar of the final language, even when discussing features of earlier versions.

### 2.1.4 Layout of the Rest of This Paper

2.2 The history of the original Report, up to 1969. This is the period during which all the controversies arose.

2.3 The concepts of ALGOL 68. This traces the sources of the various ideas, including some that fell by the wayside, and a few that only appeared in the revision.

2.4 The history of the Revised Report, from 1970 to 1975.

2.5 The method of description. This changed substantially during the revision, which is why it is considered here.

2.6 The maintenance phase: enhancements and problems.

2.7 Implementations.

2.8 Conclusions, including "Whatever happened to ALGOL 68?" and "Whatever have we learned from ALGOL 68?"

I have included in this paper those incidents in the history of ALGOL 68 that I considered to be significant. Space did not permit me to tell all the stories that could have been told; for a different subset and an alternative perspective, especially of the early days, the reader is referred to [Van der Poel 1986].

## 2.2 HISTORY OF THE ORIGINAL ALGOL 68 REPORT

### 2.2.1 Dramatis Personae

Table 2.3 lists those members of WG 2.1 who were active in the development, in the order in which they became full members of the Group.

Most members of the Group were there on account of their involvement with, or experience of, ALGOL 60. Those belonging to academic institutions would have teaching duties and research interests not necessarily connected directly with ALGOL 68. Only the team at Amsterdam (Van Wijngaarden, Mailloux, Peck, and Koster) together with Goos, who was preparing an implementation, worked full-time on the project. McCarthy had been responsible for LISP, regarded as a tool for his work on artificial intelligence; Wirth, after initially working on ALGOL X, devoted his energy to his own ALGOL W; Ross was working on his own languages, AED-0 and AED-1; and Yoneda was working on the Japanese ALGOL N.

The nonacademics were subject to the bidding of their employers, although those in large research establishments would have much opportunity to do their own thing. Thus Woodger and Duncan at NPL and Sintzoff at MBLE could devote their time to programming language research. Randell had been responsible for a major ALGOL 60 implementation at English Electric, but at IBM he was into system architecture; and Hoare had implemented ALGOL 60 for Elliot and was now trying to provide operating systems to match; Bekic was part of the group that developed the Vienna Definition Language.

### 2.2.2 Princeton (May 1965)

The first public mention of ALGOL X (which eventually became ALGOL 68) and of the mythical ALGOL Y (originally conceived as a language that could manipulate its own programs, but in fact degenerating into a collection of features rejected for ALGOL X) was in a paper entitled "Cleaning up ALGOL 60" [Duncan 1964]. Although it had been discussed at Tutzing (March 1964) and Baden (September 1964), work started in earnest at the Princeton meeting in May 1965.

The chairman's Activity Report to TC2 for that meeting [Van der Poel 1965] shows a wide ranging discussion on basic aspects of the language. There was much interest in the language EULER [Wirth 1966a] and particularly in its 'trees' or 'lists'. There was a strong feeling towards what now we should call 'strong typing', and even a firm decision to have coercion from integer to real to complex. Draft proposals for a full language were solicited for the next meeting.

### 2.2.3 St. Pierre de Chartreuse (October 1965)

There were three drafts on the table: [Wirth 1965], [Seegmüller 1965b], and [MR 76], which was Aad van Wijngaarden's famous "Orthogonal design and description of a formal language."

[Wirth 1965] had been written while the author was on sabbatical at the Mathematisch Centrum over the summer, and was available sufficiently in advance of the meeting that a substantial set of comments on it was also available. It was a straightforward compilation of features as discussed at Princeton, including the trees from EULER. In the meantime, however, Record Handling [Hoare 1965b] had been proposed and Wirth much preferred this to his trees. Henceforth, the Wirth/Hoare proposal must be regarded as one.

[Seegmüller 1965b] was really an extension of [Wirth 1965] and never a serious contender. Its main feature was a system of References, also described in [Seegmüller 1965a] (see also Section 2.3.4).

**TABLE 2.3**

WG 2.1 members active in the original design of ALGOL 68.

| | |
|---|---|
| Fritz Bauer | Techn. Hochschule, Munich |
| Edsger Dijkstra† | Tech. University, Eindhoven |
| Fraser Duncan† | National Physical Lab., UK |
| Tony Hoare† | Elliot Automation, UK |
| P. Z. Ingerman | RCA, Camden, NJ |
| John McCarthy | Stanford University, Palo Alto, CA |
| J. N. Merner | General Electric, Phoenix, AZ |
| Peter Naur‡ | A/S Regnecentralen, Copenhagen |
| Manfred Paul | Techn. Hochschule, Munich |
| Willem van der Poel | Techn. Hogeschool Delft, Chairman of WG 2.1 |
| Klaus Samelson | Techn. Hochschule, Munich |
| Gerhard Seegmüller† | Techn. Hochschule, Munich |
| Aad van Wijngaarden | Mathematisch Centrum, Amsterdam |
| Mike Woodger† | National Physical Lab., UK |
| Jan Garwick† | Oak Ridge National Lab., Oak Ridge, TN |
| Brian Randell† | IBM, Yorktown Heights, NY |
| Niklaus Wirth‡ | Stanford University, Palo Alto, CA |
| Peter Landin | Univac, New York, NY |
| Hans Bekic | IBM Lab., Vienna |
| Doug Ross | M.I.T., Cambridge, MA |
| W. M. Turski† | Academy of Sciences, Warsaw, Secretary of WG 2.1 |
| Barry Mailloux | Mathematisch Centrum, Amsterdam |
| John Peck | University of Calgary |
| Nobuo Yoneda | University of Tokyo |
| Gerhard Goos | Techn. Hochschule, Munich |
| Kees Koster | Mathematisch Centrum, Amsterdam |
| Charles Lindsey | University of Manchester, UK |
| Michel Sintzoff | MBLE, Brussels |

† Signatories to the Minority Report.

‡ Resigned after [MR 93].

[MR 76], distributed only the week before the meeting, introduced three basic ideas:

1. The two-level grammar notation we now know as a W-Grammar (2.5.1.1).
2. The combination of a minimal number of language concepts in an orthogonal way (2.3.4.3), thus providing the power of the language. (This process is much facilitated by using a W-Grammar.)
3. An expression-oriented language, that is, no distinction between statements and expressions.

Apart from these, the language described by [MR 76] was nothing special, and the document itself was exceedingly hard to read (no examples, no pragmatics, and only 26 one-letter metanotions of not much mnemonic significance). Nevertheless, WG 2.1 was sufficiently impressed to resolve formally (but with misgivings from Hoare) that "whatever language is finally decided upon by WG 2.1, it will be described by Van Wijngaarden's metalinguistic techniques as specified in MR 76." Whether they had bought just the notation, or whether it included (2) and (3) as well, is not clear.

The meeting spent most of its time discussing language issues, mainly in terms of the Wirth/ Hoare proposal (which was not expression-oriented, be it noted), and by the end of the week the participants felt that they had a fair idea what ALGOL X was going to look like. Meanwhile, it had established a subcommittee consisting of Hoare, Seegmüller, Van Wijngaarden, and Wirth with instructions to prepare a Draft Report along the lines agreed upon. A second subcommittee, set up to consider I/O, consisted of Ingerman, Merner, and (in their absence) Garwick and Paul; it was charged with producing facilities somewhat between [Knuth *et al.* 1964] and [Garwick 1965].

Nevertheless, all was not plain sailing. A bad attack of cold feet occurred on the Thursday morning (a traditional time for cold feet in WG meetings), and serious suggestions were made by Naur and Randell that ALGOL X should be dropped and ALGOL Y proceeded with.

At the end of the meeting, the Subcommittee announced that Van Wijngaarden would write a draft report for the other three to consider, and that it would be circulated well in advance of the next meeting (with the proviso that members wishing to change anything would have to rewrite those parts of the report affected). The next meeting was fixed for six months hence, and it was well understood that the time for radically new concepts was already passed.

Throughout the whole project, the WG in general, and Van Wijngaarden in particular, consistently underestimated the time it would take by substantial factors. Recall that ALGOL 60 was put together in six days, albeit after various preliminary meetings over the previous six months. Naur's draft for [Naur *et al.*, 1960] was written in two months, and the final version was circulated within two months after the meeting [Naur 1981]. But the art of language design and definition had advanced since 1960 and things could not be done at that pace anymore. In the event, Van Wijngaarden's draft was nowhere near complete, and the next meeting had to be postponed for a further six months. However, the Subcommittee members themselves did get together on the date originally proposed at Kootwijk.

### 2.2.4 Kootwijk (April 1966)

In addition to the four members of the Subcommittee, this meeting was attended by Barry Mailloux, Van Wijngaarden's doctoral student from Canada, and W. L. van der Poel, chairman of WG 2.1. Contrary to the arrangement made at St. Pierre, there were two drafts available: Van Wijngaarden's incomplete one, written in the agreed formalism, and a Wirth/Hoare "Contribution to the development of ALGOL" [Wirth 1966b], which had been prepared from [Wirth 1965] and [Hoare 1965b] for publication in *Communications of the ACM* after the adoption of [MR 76] at St. Pierre. It was generally agreed that the Contribution described more or less the right language, but in the wrong formalism.

Due to the incompleteness of Van Wijngaarden's document, the meeting worked initially by amending the Contribution (although still with the intent that Van Wijngaarden's version should be brought into line and become the official definition, as agreed by the WG). However, there arose a complete disagreement between Seegmüller and Van Wijngaarden on the one hand, and Hoare and Wirth on the other, concerning the parameter mechanism (2.3.4.3), and as Van Wijngaarden's parameter mechanism could not easily be incorporated into the Contribution, it was clear that the two documents must now diverge. Certainly, there was little prospect that the Van Wijngaarden approach

would produce a document within 1966, whereas Hoare and Wirth were especially concerned about achieving a short timescale.

The Subcommittee made a long list of points that the Van Wijngaarden version should attend to, and he continued to work on it, assisted by Mailloux, a further incomplete draft being sent to WG 2.1 members during July, and a more-or-less complete draft just ten days before the next meeting.

The language defined in the Contribution was subsequently implemented by Wirth as ALGOL W (see also [Wirth 1966c]).

### 2.2.5 Warsaw (October 1966)

The draft submitted to this meeting was [W-2] (which apparently never did get an MR number). There was fierce debate as to whether this document by one member could be accepted as the report of the Subcommittee: Hoare was willing to accept it as such if time were allowed for missing parts to be completed; Wirth, who had had grave misgivings about the formalism at Kootwijk, had resigned from the Subcommittee and declined to attend this meeting, complaining of insufficient time to study the document, and insisting that it could no longer be regarded as a Subcommittee product.

There was also a report [Merner 1966] from the I/O subcommittee, which had held meetings at Oak Ridge and at the University of Illinois. However, this was not yet incorporated into [W-2].

At the start of the week there was some expectation that the two documents could be put together and submitted to TC2. However, there was strong debate on the subject of references. Orthogonality dictated that references could exist to any object, whether a record or not and whether that object were declared locally or on the heap, and this also considerably simplified the parameter-passing mechanism. Hoare and some others wanted to restrict references to records only (as in [Hoare 1965b] and [Wirth 1966b])—the so-called "diagonal" approach (2.3.4.3). Moreover, McCarthy had introduced fresh proposals for operator overloading (2.3.6), and had also been demanding serial elaboration of ***and*** and ***or*** (2.3.3.6), and Samelson had asked for anonymous routines (2.3.5). So by Thursday afternoon the customary depression had set in, to the extent that they even considered abandoning it altogether, and certainly the most to be expected was publication as a Working Document, while further editing and implementation studies continued.

Finally, the following were formally agreed:

1. [W-2] to be amended, incorporating at least [Merner 1966], together with proper pragmatics. (A 'pragmatic remark' is to the Report as a **comment** is to a **program**.)
2. The amended document to be published in the ALGOL Bulletin, and possibly other informal, non-refereed journals.
3. The results of implementation studies to be incorporated.
4. Van Wijngaarden to be charged with the editing, alone without any subcommittee (provided he was not pressed for any specific time schedule, although he could foresee mailing it the following February).
5. The next meeting in May would be too soon to consider the revised draft, and would be devoted to ALGOL Y. The following meeting in September would take the final decision.
6. McCarthy's overloading could be incorporated if the editor found it feasible.

It will be seen that the content of the language was growing steadily (and more was to come). It should be stressed that Hoare made several attempts to limit the content to something like [Wirth 1966b], or at least to "within the intersection of the agreement of WG 2.1," but to no avail. It is also

noteworthy that Van Wijngaarden was beginning to realize just how long the revision might take, although he was still out by a factor of two.

### 2.2.6 Zandvoort (May 1967)

This meeting was supposed to discuss ALGOL Y: it spent nearly all its time on ALGOL X. The document available (in spite of there being no obligation to produce one at all) was [MR 88], which was now beginning to look substantially like the eventual [R]. John Peck, on sabbatical from the University of Calgary, had now joined the editorial team. It had been mailed four weeks in advance of the meeting. McCarthy's overloading was there "in all its glory", as were Samelson's anonymous routines.

There was pressure to get the process finalized before a TC2 meeting in October (which could lead to final acceptance as an IFIP document at the forthcoming General Assembly meeting in Mexico), but the timing was very tight. So the motions passed were

1. The report published in the ALGOL Bulletin (as agreed in Warsaw) should also be submitted to TC2, who would accept it subject to subsequent adoption by WG 2.1 "without substantive changes."
2. The next meeting was to be held not less than three and a half months after the circulation of the report.

On this basis, the earliest date for the next meeting would be October 22 (but, by application of the formula, it did not actually happen until the following June 3).

The meeting spent much time in constructive discussion of language issues (and also of the description method) without (so far as I can tell from the minutes) threats of resignation or rebellion, apart from a considerable disgust expressed by Wirth. They even spent an afternoon on ALGOL Y, from which it emerged that no one had very concrete ideas of what it was about, and that the only role model was LISP, on account of the fact that that language could construct its own programs and then *eval* them.

### 2.2.7 MR 93

The draft Report commissioned at Warsaw was eventually circulated to the readership of the ALGOL Bulletin as [MR 93] in February 1968, and was the cause of much shock, horror, and dissent, even (perhaps especially) among the membership of WG 2.1. It was said that the new notation for the grammar and the excessive size of the document made it unreadable (but they had said the same about ALGOL 60 and, although it may have been the longest defining document at that time, it is rather shorter than most of its successors). Moreover, it was not all gloom—there was also a substantial number of positive reactions and constructive comments. Another of Van Wijngaarden's students, Kees Koster, had now joined the team, with special responsibility for "transput," as I/O was now to be known.

This is where I entered the picture myself. Everything described up to now has been gleaned from the minutes and other documents. However, a copy of [MR 88] happened to come into my possession, and I set out to discover what language was hidden inside it. Halfway through this process, I switched to [MR 93]. The task occupied me for fully six man-weeks (the elapsed time was much more, of course), and as it progressed I incorporated my knowledge into an ALGOL 68 program illustrating all the features, embedded within comments so as to read like a paper. At that time, the ALGOL

Specialist Group of the British Computer Society announced a meeting to discuss the new language, so I circulated my document in advance under the title "ALGOL 68 with fewer tears" (non-native English speakers need to be aware of an English school textbook entitled "French without Tears"). On arrival at the meeting, at which three members of WG 2.1 (Duncan, Hoare, and Woodger) were present, I found that I was the only person there who knew the language well enough to present it, and I was hauled out in front to do just that. "Fewer tears" went through several revisions, appearing in the *ALGOL Bulletin* [Lindsey 1968], and finally in the *Computer Journal* [Lindsey 1972].

Of course my report contained errors, but the only important feature of the language I failed to spot was the significance of what could be done with records (or ***struct***s as they had now become), and the necessity for garbage collection arising therefrom. Nevertheless, it served as an existence proof of the language, and when Duncan took a copy to the next WG meeting it helped to redeem some of the damage. In addition to myself, several people (notably G. S. Tseytin of Leningrad) succeeded in interpreting the Report "from cold."

In May 1968, the tenth anniversary of ALGOL 58, a colloquium was held in Zürich, where the recently distributed [MR 93] came in for much discussion, being "attacked for its alleged obscurity, complexity, inadequacy, and length, and defended by its authors for its alleged clarity, simplicity, generality, and conciseness" [AB 28.1.1]. Papers given at the colloquium included "Implementing ALGOL 68" by Gerhard Goos, "Successes and failures of the ALGOL effort" [Naur 1968], and a "Closing word" [Wirth 1968]. Naur's paper contained criticism of [MR 93], as providing "the ultimate in formality of description, a point where ALGOL 60 was strong enough," and because "nothing seems to have been learned from the outstanding failure of the ALGOL 60 report, its lack of informal introduction and justification." IFIP seemed to be calling for immediate acceptance or rejection, and thus IFIP was the "true villain of this unreasonable situation," being "totally authoritarian" with a committee structure and communication channels entirely without feedback and with important decisions being taken in closed meetings. "By seizing the name of ALGOL, IFIP has used the effort . . . for its own glorification." Strong words indeed! Wirth's contribution was also critical of [MR 93], and of the method whereby the WG, after a week of "disarray and dispute," and then "discouragement and despair" would accept the offer of any "saviour" to work on the problems until next time. Eventually the saviours "worked themselves so deeply into subject matter, that the rest couldn't understand their thoughts and terminology any longer." Both Naur and Wirth resigned from the Group at this time.

However, Naur's point about "lack of informal introduction and justification" was correct. Starting from [W-2], which contained no informal material whatsoever, the amount of pragmatics in the successive drafts had been steadily increasing, particularly under pressure from Woodger, but it still had far to go. And Wirth's caricature of the meetings was not far off the truth.

### 2.2.8 Tirrenia (June 1968)

Three and a half months after the distribution of [MR 93] (four days less, to be exact), WG 2.1 met to consider it. Van Wijngaarden had had a considerable response from readers of the ALGOL Bulletin, and came armed with pages of errata, including some modest language changes. There was concern that the use of the name "ALGOL 68" might turn out to be premature. Van Wijngaarden had been invited to talk on "ALGOL 68" at the forthcoming IFIP Congress, but there might be no ALGOL 68 by then. Randell questioned the need for the "mystical IFIP stamp," and others wanted publication delayed until implementations existed. But without the stamp, manufacturers would not implement it, so maybe it should be stamped, but with a proviso that it would be revised in the light of experience.

Seegmüller produced a list of demands:

1. An expanded syntax in an Appendix.
2. A formal semantics (perhaps using VDL [Lucas 1969]). But the WG had rejected that as far back as Princeton in order to avoid "unreadability." However, competitive descriptions ought to be invited.
3. There should exist implementations.
4. There should be an informal introduction to the language (and to its method of description), preferably within the final Report.
5. The final vote on the report should be delayed until all these requirements were met.

In a series of straw votes, the WG showed itself in broad agreement with these requirements.

But the meeting was not all like this. The bulk of the time was spent considering technical points of the language. Members would ask about some particular situation, and Van Wijngaarden would explain how the Report covered it. There seems to have been a consensus for approval of the language itself. It was the Report that was the sticking point.

Came the last day, and what were they to do? First, two decisions were made:

1. Changes already made to [MR 93] were too "substantive" for the Zandvoort mechanism to be applied (see 2.2.6 (1)).
2. The Warsaw resolutions (see 2.2.5 (2)) had been fulfilled.

This put the whole project into limbo, with nobody charged to do anything anymore.

At this point Van Wijngaarden presented a Statement (some saw it rather as a threat) in which the authors offered to make just one more edition of the document, which would be submitted at some agreed moment to WG 2.1 which would then either accept it or reject it. If it was rejected, then the authors would have the right to publish it as their own. And he pointed out that the WG had worn out its first editor (Peter Naur), and then two authors (Wirth and Hoare), and now it might have worn out four more.

After the tea break, this statement was turned into a formal resolution, and the authors were asked to produce their final draft by October 1 for consideration by a full WG meeting on December 16.

### 2.2.9 North Berwick (August 1968)

This meeting, held just before the IFIP Congress, had been fixed before the decision quoted above. Hence it was not clear what this meeting should do, except talk—which is just what it did. It was the first meeting that I attended myself, and it was five days of continuous politics. For example, a poll was taken concerning the future work of the group, and the best part of a whole day, both before and after the poll, was taken up with its form, and how its results were to be interpreted. Mutual trust among the parties had been entirely lost, and jockeying for position was the order of the day, and of every day. Much time was devoted to discussion of whether and how minority reports might be attached to the final Report. Barely half a day was spent in technical discussion.

The document before the meeting was [MR 95], a half way stage to the document requested in Tirrenia, and containing much improved pragmatics and also extensive cross referencing of the syntax. But at the end of the meeting it was clear that the future of ALGOL 68 was still very finely balanced.

On a personal note, it was here that I met Sietse van der Meulen who had been asked by Van Wijngaarden to write the Informal Introduction requested by Seegmüller at Tirrenia. He asked me to

join him in this task, and together we worked out the plan of our book during the IFIP Congress the following week.

### 2.2.10 Munich (December 1968)

By October, Van Wijngaarden had circulated his report [MR 99] to the Working Group as promised. In the meantime, Ross had been trying to organize a Minority Report that would be constructive and well thought out [Ross 1969]. Also, Dijkstra, Hoare, and Randell each circulated drafts of brief minority reports, which they invited others to join in signing.

At Munich, another draft was available [MR 100] whereas, some said, the meeting should have been considering only [MR 99]. To resolve this, three of us had to be dispatched to a side room to list all the differences between the two (not that I think anybody actually read our list). A draft of the first chapters of the Informal Introduction was also presented to the meeting by Van der Meulen and myself.

Although this meeting had much political content, there was more willingness than at North Berwick to reach a compromise, and there was also more technical discussion. Nevertheless, there were clearly some who would wish to move on from ALGOL 68 and who were already considering (following discussion at a NATO software engineering conference two months earlier) the possibility of a group devoted to Programming Methodology. It was therefore proposed that TC2 should be asked to form a new Working Group in this area (WG 2.3, as it later became).

To try to preserve the unanimity of the Group, a covering letter to be issued with the Report was drafted. This "did not imply that every member of the Group, including the authors, agreed with every aspect of the undertaking" but affirmed that "the design had reached the stage to be submitted to the test of implementation and use by the computing community." Then a motion to transmit the report with its Covering Letter to TC2 was proposed by Hoare and Woodger. There was heated discussion of the wording of the Covering Letter, but both it and the motion were accepted by substantial majorities, and we thought we were done.

However, in the last hour of the meeting some who were unhappy with the final form of the Covering Letter, and with interpretations which were being put upon it, produced a minority report. It had been constructed at the last moment, based upon the draft circulated by Dijkstra, and could clearly have been improved if its authors had had more time. Following the agreements at North Berwick the meeting had to accept it, and the point was clearly made that TC2 should be asked to publish it alongside the main Report.

In January of 1969, TC2 duly forwarded the Report and a mauled version of the Covering Letter to the IFIP General Assembly, which in a postal vote in March authorized publication. But TC2 refused to forward the Minority Report. The texts of the original Covering Letter and of the Minority Report are reproduced in Appendix A and Appendix B.

### 2.2.11 Banff (September 1969)

This was a very peaceful meeting by comparison with its predecessors. The agreed Report had been printed [MR 101], plus copious sheets of errata. Arrangements were in hand to have it published in *Numerische Mathematik* and *Kybernetika*. The decision of TC2 not to publish the Minority Report was regretted (but it had meanwhile been published in the *ALGOL Bulletin* [AB 31.1.1]).

During the meeting some small additional errata to [MR 101] were discussed (including a few extra representations to facilitate writing programs in ASCII). Thus was the final published text of [Van Wijngaarden 1969] established.

A much more substantial draft of the Informal Introduction was available, and discussions proceeded as to how it should be published. The decision was to ask TC2 to arrange for publication under IFIP copyright (which was subsequently done, and it appeared as [Lindsey 1971]).

In order to promote implementation of the language and to provide feedback from implementors, the WG empowered Mailloux to set up an Informal Implementers' Interchange as a means of getting a rapid dissemination of implementation techniques. Later, at Manchester, this responsibility was taken over by Branquart. It never did work fully as intended, but it enabled us to know who was implementing and to provide a channel for communication to them.

### 2.2.12 So What Had Gone Wrong?

It would be wrong to describe the seven who signed the Minority Report (or the two who resigned) as wreckers. They were all honourable men who cared about programming languages (even about ALGOL 68) and most of them had contributed substantially towards it. But their objections were diverse.

First, it should be said that Dijkstra had attended none of the crucial meetings from St. Pierre to Tirrenia, and Garwick had been entirely absent from St. Pierre to the end (although he had been active on the I/O subcommittee [Merner 1966]).

Some of them were opposed to the language. In particular, Hoare had grave misgivings about the language all along, as he has described in his Turing lecture [Hoare 1981], and he published his specific criticisms in [Hoare 1968]. He much preferred the way things had been done in SIMULA 67 [Dahl 1968]. Basically, he had hoped that more of the language definition could have made use of its own overloading facility (2.3.6), and he thought that the whole reference concept had been taken much too far (2.3.4.3). He had consistently expressed these opinions at the meetings, always in the most polite terms, always acknowledging the immense amount of work that Van Wijngaarden had put into it. It would seem that Wirth shared Hoare's views on most of these issues.

For some, notably Dijkstra, the whole agenda had changed. The true problem, as he now saw it, was the reliable *creation* of programs to perform specified tasks, rather than their *expression* in some language. I doubt if any programming language, as the term was (and still is) understood, would have satisfied him.

I think the rest were more concerned with the method of description than with the language, and with the verbosity and possible unsoundness of the semantics as much as with the W-Grammar (see [Turski 1968]). As Randell said at Tirrenia, "From our discussions . . ., it seems to follow that there is reasonable agreement on the language, but that there is the need for the investigation of alternative methods of description."

But how had the WG managed to support the description so far, only to become so upset at the last minute? After all, [MR 93] was not so different in style from [MR 88], and the discussion of [MR 88] at Zandvoort had been all about the language, not the document. It may be that having three months to read it had changed their perception. Duncan in particular did not like the W-Grammar, but he did try to do something about it by proposing an alternative syntax notation and even a complete rewrite of Chapter 1. And [MR 95] was much more readable than [MR 93], as Woodger admitted at North Berwick (but Ross, when trying to produce his own minority report [Ross 1969], still could not get to grips with [MR 99] although he dissociated himself from the Minority Report actually produced). Contrariwise, in spite of all the complaints from within the working group, I must report that I have never heard an actual implementor complain about the method of description.

The text of the Munich Covering Letter seems to have been the last straw, at least for Turski, who wanted a clearer expression of the preliminary nature of the Report and of the deep divisions within

the WG. He also shared the view, often expressed by Hoare, on the necessity for simplicity in programming languages, and he has written on this [Turski 1981].

However, all the dissenters shared an exasperation with Van Wijngaarden's style, and with his obsessional behaviour in trying to get his way. He would resist change, both to the language and the document, until pressed into it. The standard reply to a colleague who wanted a new feature was first to say that it was too late to introduce such an upheaval to the document, then that the colleague should himself prepare the new syntax and semantics to go with it. But then, at the next meeting, he would show with great glee how he had restructured the entire Report to accommodate the new feature and how beautifully it now fitted in. Moreover, there were many occasions when he propagated vast changes of his own making throughout the document.

### 2.2.13 The Editorial Team

At Tirrenia, Van Wijngaarden described the editorial team in Amsterdam as follows:

| | | |
|---|---|---|
| Koster | Transputer | (from Oct. 1967) |
| Peck | Syntaxer | (on sabbatical from Univ. of Calgary Sep. 1966 to Jul. 1967) |
| Mailloux | Implementer | (doctoral student Apr. 1966 to Aug. 1968) |
| Van Wijngaarden | Party Ideologist | |

Peck and Mailloux worked on the project full time, and all the pages of the Report were laboriously typed and retyped as changes were made. Van Wijngaarden, whose full time job was to direct the Mathematisch Centrum, would receive daily reports of progress. Mailloux was also responsible for ensuring that what was proposed could be implemented (as reported in his doctoral thesis [Mailloux, 1968]; see also 2.7.1).

The use of a distinctive font to distinguish the syntax (in addition to italic for program fragments) commenced with [MR 93], using an IBM golf-ball typewriter with much interchanging of golf balls. Each time Van Wijngaarden acquired a new golf ball, he would find some place in the Report where it could be used (spot the APL golf ball in the representations chapter). In fact, he did much of this typing himself (including the whole of [MR 101]).

From late 1967 on, intermediate drafts of parts of the document were circulated among an "inner circle" that included Yoneda, Goos' implementation team in Munich, Landin and, most importantly, the quadrumvirate of Michel Sintzoff, P. Branquart, J. Lewi, and P. Wodon from MBLE in Brussels. This latter group was particularly prolific in the number of comments that they submitted, to the extent that there were effectively two teams involved—one in Amsterdam creating the text and another in Brussels taking it apart again. Peck continued to submit comments and suggestions after returning to Calgary, and he returned to Amsterdam to resume the old routine during the hectic summer of 1968.

## 2.3 THE CONCEPTS OF ALGOL 68

Historically, ALGOL 68 arose out of ALGOL 60 and, although it soon became apparent that strict upwards compatibility with ALGOL 60 was not a realistic goal, there was always reluctance to adopt new syntactic sugar for familiar constructs, and there were many rearguard actions to maintain, at least the appearance of, certain cherished features (such as ***switch***es, call-by-name, and Jensen's device). All this led to some strange quirks in the final product.

From our present vantage, it is amazing to see how ill-understood in the early sixties were concepts that we now take quite for granted. It was at those early meetings of WG 2.1 that these concepts were painfully hammered out. In spite of the differences that surfaced, the membership of WG 2.1

comprised as good a collection of international programming language experts as could have been gathered together at that time. Everybody had something to contribute, especially, perhaps, those who were eventual signatories of the minority report.

### 2.3.1 The Easy Bits

In spite of the stormy passage of some items, many new features were proposed and went into the language with hardly any discussion.

Complex arithmetic (type ***compl***) was accepted without qualm, as were bit patterns (type ***bits***) and ***long*** versions of ***int***, ***real***, etc. (also ***short*** versions in [RR]). Naur [1964] proposed various standard **operators** (***mod***, ***abs***, ***sign***, ***odd***, and so forth) and also 'environment enquiries' such as *max int*, *max real*, *small real*, and *long* versions thereof.

It was agreed that there should be a ***string*** type, but some wanted them to be of indefinite length with a concatenation operator [Duncan 1964], and some assumed that each ***string*** variable would be declared with some maximum length [Seegmüller 1965a]. In both [MR 76] and [Wirth 1965] ***string*** was a primitive type. You could declare ***string*** variables, assign literal ***string***s to them, and concatenate ***string***s, but that was all; a delight to use, but requiring a heap for implementation. An inconclusive discussion at St. Pierre seemed to support the maximum length view, but in the event [MR 88] they turned out to be ***flex***ible ***array***s of the primitive type ***char***. This meant that sub-strings could easily be extracted and assigned to, but the ***flex*** facility itself was a mistake (2.3.2). ALGOL 68 is still the only major language not to require a maximum length with each ***string*** variable.

It was agreed that the ALGOL 60 **for-statement** was too complicated (although hankerings after **for-lists** continued to crop up). Hoare [1965a] proposed what eventually became the **loop-clause**:

***for*** *identifier* ***from*** *first* ***by*** *increment* ***to*** *last* ***do*** *statements* ***od***

where *identifier* is an implicitly declared ***int*** constant (unassignable) for the duration of the loop only, and *first*, *increment*, and *last* are ***int*** expressions to be evaluated once only at the start. It was agreed that there should be a separate ***while*** statement, although eventually it became just another option of the **loop-clause** (defaulting to ***true***).

That ALGOL 68 is an expression-oriented language is due to [MR 76], which made no distinction between **expressions** and **statements** (this idea having been first proposed in [Wirth 1966a]). Thus, all forms of **conditional-**, **case-** and **closed-clauses** (that is, **blocks**) return values, as do (after [W-2]) **assignations**. For example,

*x* := (***real*** *a* = *p***q*; ***real*** *b* = *p*/*q*; ***if*** *a*>*b* ***then*** *a* ***else*** *b* ***fi***) + (*y* := 2**z*);

In addition, it is also possible to declare an **identifier** as an (unassignable) constant (as *a* and *b* above). Neither of these features was in [Wirth 1965], although **constant-declarations** were in [Seegmüller 1965b]. They are, of course, now the basis of functional languages such as SML [Milner 1990], but contemporary imperative languages seem to have fought shy of being expression-oriented, although **constant-declarations** are now well accepted (as in Ada).

The example shows another important feature that appeared without comment in [W-2], namely the matching of ***if*** by ***fi*** (likewise ***case*** by ***esac*** and, in [RR], ***do*** by ***od***). This removes the dangling else problem and, at the same time, the requirement for a **compound-statement**. Most modern languages have followed this lead except that, being without any sense of humour, they reject the easily memorable ***fi*** in favour of ***end if***, or ***endif***, or even plain ***end***. (How *do* you remember which one it is?)

### 2.3.2 Arrays

A 2-dimensional array can be regarded in two ways. First, it can be an ***array*** of ***array***s, and this was the view taken in both [Wirth 1965] and [MR 76]. Thus, one might expect to declare

```
loc [1:4][1:5] int a4a5;
```

and subscript it by *a4a5[i][j]*, and the type of *a4a5* would be described as **'row of row of integral'** (indeed that example is legal in [RR]). A consequence is that *a4a5[i]* is then a row-vector, but there is no way to construct a column-vector. WG 2.1 understood and approved this at St. Pierre, Bauer remarking, "It is not necessary for our two-dimensional arrays to have all the characteristics of matrices." But they did ask that "*][*" be written as "," allowing *a4a5[i, j]* after all (but titter ye not! for this is exactly the situation now in Pascal, and also in ALGOL W as implemented [Wirth 1966c]).

In [W-2], however, we find genuine 2-dimensional ***array***s:

```
loc [1:4, 1:5] int a45;
```

subscripted by *a45[i, j]* and with type **'row row of integral'** (also legal in [RR]). Semantically, this was modelled in [R] as a linear sequence of elements together with a descriptor composed of an origin $c$, and two each of lower bounds $l_i$, upper bounds $u_i$, and strides $s_i$. The element at *a45[i, j]* is then found in the sequence at $c + (i-l_1) * s_1 + (j-l_2) * s_2$. This suggests an obvious implementation. Moreover, it now allows all sorts of **slices**, such as

| | |
|---|---|
| *a45[2, ]* | row 2 |
| *a45[ , 3]* | column 3 |
| *a45[2:3, 3]* | part of column 3 |
| *a45[2:3, 2:4]* | a little square in the middle. |

This was accepted with enthusiasm; it is simple to implement and does not hurt those who do not use it; I cannot understand why other languages (excepting, in some degree, PL/1) have not done the same. It also provides the possibility to extract diagonals, transposes and the like.

The next facility added, in [MR 88], was flexible arrays. These allowed the size of an ***array*** variable to be changed dynamically.

```
loc flex [1:0] int array;      # initially empty #
array := (1, 2, 3);            # now it has bounds [1:3] #
array := (4, 5, 6, 7, 8)       # now it has bounds [1:5] #
```

That seems nice, but every student who has ever tried to use the facility has wanted to extend the *array* with bounds *[1:3]* by adding *5* extra elements at the end, leaving the existing values in *array[1:3]* untouched. But that goodie is *not* on offer, and what *is* on offer can be achieved in other ways anyway.

Van Wijngaarden has explained to me that flexible arrays were provided simply to facilitate ***string***s, which are officially declared as

```
mode string = flex [1:0] char;
```

But ***string***s *can* be extended

```
loc string s := "abc";
s +:= "defgh";     # now s = "abcdefgh" #
```

It would have been better, in my opinion, to omit the ***flex*** feature entirely and to provide ***string***s as a primitive type. In [RR] ***flex*** was made a property of the ***ref***erence to the ***array*** rather than of the ***array*** itself, thus bringing ***flex***ibility within the strong typing. This avoided a run-time check required in [R] to prevent the preservation of ***ref***s to no-longer-existent elements of ***flex array***s.

It was a fundamental principle that every value in the language should be expressible by some external denotation. Hence the **row-display** (the *(4, 5, 6, 7, 8)* in the example) has been present since [MR 76], although the corresponding **structure-display** did not appear until after Tirrenia. Unfortunately, this does not provide a convenient way to initialize the whole of an array to some same value (Ada has better facilities in this respect), and another problem, pointed out by Yoneda, is that although *()* and *(1, 2)* are valid **row-displays**, *(1)* is not (it is just a *1* with *( )* around it), and if you want to initialize your array with it you have to rely on the rowing coercion (2.3.3.5). Oddly, this bug was also present in the Preliminary Ada Reference Manual—*Plus ça change ...* .

Operators ***lwb*** and ***upb*** were provided in [MR 99] to determine the actual bounds of any given array. These replaced the less useful **flexible-bounds** used up to then in **formal-parameters** (as in *[1:**int** upper] **int** a*). However, bounds could still be specified, optionally, in **formal-parameters**. This possibility was removed in [RR] because, so it was said, there was nothing useful that implementors could do with that information. I now think this was a mistake, since to have such preconditions (even if all they trigger is a run-time check) can simplify writing the body of a routine.

### 2.3.3 Coercion

Although coercions had existed in previous programming languages, it was ALGOL 68 that introduced the term and endeavoured to make them a systematic feature (although it is often accused of a huge overkill in this regard). They existed in [W-2] (not by that name) but reached their full fruition in [MR 93], provoking three quarters of a day's discussion at Tirrenia. Coercion can be defined as the implicit change of the type (a priori) of an **expression** to match the type (a posteriori) required by its context.

There were eight different coercions in [R], diminishing to six in [RR]. These will now be examined, starting with the least controversial.

#### 2.3.3.1 *Widening*

You can 'widen' from ***int*** to ***real*** and from ***real*** to ***compl***. This is quite natural, and most languages (Ada is the notable exception) allow it. Note that there is no 'narrowing' in ALGOL 68 as there is in FORTRAN and PL/1. Thus *realvariable := intvalue* is allowed, but not *intvariable := realvalue*.

#### 2.3.3.2 *Dereferencing*

This changes a ***ref***erence into the thing ***ref***erred to. Thus, having declared *y* as a **real-variable**, so that its type is ***ref real*** (2.3.4.3), it is necessary to dereference *y* in *x := y*. The term 'dereferencing' is now used by many languages (even those that call their references 'pointers' and those that use an explicit operator for the purpose).

#### 2.3.3.3 *Deproceduring*

This is the method whereby parameterless ***proc***s are called (the alternative used by some languages is to write *p()*). A coercion is necessary to distinguish the case requiring the value returned by

the **proc** (as in **loc real** *x := random*) from that requiring the **proc** value itself (as in **loc proc** *anotherrandom := random*).

#### 2.3.3.4 *Uniting*

Any language with **union**s needs this coercion, which can turn, for example, an a priori **int** into an a posteriori **union(int, real)**.

These four coercions are uncontroversial, either they, or some syntactic equivalent, being obviously necessary to perform their respective functions. The next four are all peculiar in one way or another.

#### 2.3.3.5 *Rowing*

This turns a value of some type into an **array** of that type, but it is not, as might be expected, a means to initialize a whole **array** to some given value. Rather, it produces an **array** with just one element, and the only reason it exists is to overcome the Yoneda ambiguity described in 2.3.2. Consider

```
[] int zero = (), one = (1), two = (1, 2);
```

*()* and *(1, 2)* are **row-displays** yielding **array**s of, respectively, 0 and 2 elements. *(1)* however is a **closed-clause** and its a priori value is an **int**, which must be 'rowed' to make an **array** of 1 element to suit the context. The same thing arises with **string**s:

```
string zero = "", one = "A", two = "AB";
```

since *"A"* is, a priori, a **character-denotation** whereas *"AB"* is a **string-denotation**.

The rowing coercion was clearly a mistake. There just has to be some other way (some new kind of brackets perhaps) to solve this problem.

#### 2.3.3.6 *Proceduring*

It was agreed at St. Pierre that the elaboration of **operands** should be in parallel, to the discouragement of side effects (2.3.8). However, at Warsaw McCarthy raised the issue of **or** and **and**, where it might be considered reasonable not to elaborate the second **operand** if the first had already determined the outcome (as in **true or** *doesitmatter*). Although McCarthy pressed strongly, it was agreed to leave matters alone in the interest of semantic consistency.

What did appear, instead, was the proceduring coercion, as in the following example taken from [R 7.5.2].

```
op cand = (bool john, proc bool mccarthy) bool:
        if john then mccarthy else false fi;
```

Now, in *p* **cand** *q*, *q* is 'procedured' from **bool** to **proc bool** and, according to the definition of **cand**, the resulting **proc** will never be called unless *p* turns out to be **true**. The proceduring coercion can also be used to make Jensen-like operations look more Jensen-like when **formal-parameters** are parameterless **proc**s.

Now the proceduring coercion complicated the syntax considerably (it is necessary to avoid cycles of proceduring and deproceduring) and it was a pain to implement. Moreover, it did not do the job it was intended to do, for in

```
p cand (a := b; q)
```

it is only *q* that gets procedured, and *a := b* is elaborated whether *p is* ***true*** or not. For these reasons, proceduring was removed entirely from [RR].

#### 2.3.3.7 *Hipping*

**Jumps**, ***skip***, and ***nil*** have no type a priori, but they can occur in some contexts where a value is (syntactically) expected (for example, *x := if p then y else goto error fi*). In [R] they were said to be 'hipped' to the required mode. The same things are acceptable in [RR], but are not described as coercions.

#### 2.3.3.8 *Voiding*

Voiding is a trick to satisfy the syntax when, upon encountering a ';' (as in *x := y; a := b*), the value of the **unit** just elaborated (*x := y*) is to be discarded. The user need not be aware of it.

The real complication of the coercions lay in the rules that governed how they might be cascaded. The removal of proceduring simplified things considerably, as may be seen by comparing the Coercion Chart on page 208 of [Lindsey 1971] with that on page 196 of [Lindsey 1977].

### 2.3.4 The Type System (or Records, References, and Orthogonality)

Orthogonality, as a guiding principle of design, had been introduced by Van Wijngaarden in [MR 76]. The idea was that if a range of features (say type constructors) exists along one axis, and another range (say types) along another, then every meaningful application of the one to the other should be present in the language. Whether WG 2.1 had accepted it as a guiding principle is another matter.

The effect of this principle on the language is nowhere more apparent than in the case of parameter passing and records—features that are superficially unrelated.

#### 2.3.4.1 *Parameter Passing*

It is said that an Irishman, when asked how to get to some remote place, answered that if you really wanted to get to that place, then you shouldn't start from here. In trying to find an acceptable parameter-passing mechanism, WG 2.1 started from ALGOL 60 which, as is well known, has two mechanisms—call-by-value (which is well understood) and call-by-name (with which is associated Jensen's Device, and which is nowadays regarded as obsolete). Starting from ALGOL 60 was a mistake. All subsequent proposals were measured by how well they stood up to the classic *Innerproduct* and other such Jensen paradigms. ([R 11.2, 3, 4] contains no less than three *innerproduct* examples.)

Moreover, they still used the term 'name' even when discussing alternatives that might replace it. It was soon realized that there were two cases of call-by-name; where the **actual-parameter** was an **expression** and a 'thunk' had to be generated (sometimes termed 'call-by-full-name'), and where the **actual-parameter** was a **variable** to be assigned to (which was thus termed 'simple-name').

Here is the classic example of Jensen's device in ALGOL 60, the *Innerproduct* example from [Naur *et al.* 1962]:

```
procedure Innerproduct(a, b) Order: (k, p) Result: (y);
    value k;
    integer k, p; real y, a, b;
```

```
begin real s; s := 0;
    for p := 1 step 1 until k do s := s+a*b;
    y := s
end Innerproduct
```

and here is a call:

```
real array x1, y1[1:n]; integer j; real z;
Innerproduct(x1[j], y1[j], n, j, z);
```

Observe that the **formal-parameters** *a* and *b* are called-by-(full-)name, and *p* and *y* are called-by-(simple-)name. See how *j*, the **actual-parameter** for *p*, is also used within the **actual-parameters** for *a* and *b*. The semantics of call-by-name then require that the **for-statement** behave as if it had been written

```
for j := 1 step 1 until k do s := s+x1[j]*y1[j];
```

In [MR 76] **formal-parameters** could be marked as ***val***, ***loc*** or ***var***, which might nowadays be rendered as ***in***, ***out*** and ***in out***. Things declared ***loc*** or ***var*** could be used on the left of an **assignation**, and things declared ***val*** or ***var*** within **expressions** on the right.

```
proc f = (real val x, real loc y, real var z) real: . . . ;
```

In a **call** of *f*, the coercion rules allowed

- an ***int*** or ***real*** value for *x*;
- a ***real*** or ***compl*** variable for *y*;
- only a ***real*** variable for *z*.

**Actual-parameters** must be compatible with their **formal-parameters** as regards ***val***, ***loc***, or ***var***, so that the accident of assigning to an **expression**, which could happen (and be detected only at run-time) in ALGOL 60, is avoided. If call-by-full-name is required, it must be indicated by ***expr:*** at the point of call. Here is *innerproduct*:

```
proc innerproduct = (real val a, b, int val k, int loc p, real loc y) void:
    begin real var s := 0;
        for p := 1 step 1 until k # ALGOL 60 loop semantics #
        do s := s+a*b;
        y := s
    end;
```

and here is a **call**:

```
loc [1:n] real x1, y1; loc int j; loc real z;
innerproduct(expr: x1[j], expr: y1[j], n, j, z);
```

Seegmüller [1965b] proposed that **formal-parameters** for call-by-full-name should be explicitly declared as parameterless ***procedure***s (that is, explicit thunks), with the corresponding **actual-parameter** indicated at the point of call (as in [MR 76]). Simple-name parameters, however, would be explicitly declared ***reference*** (and ***reference*** variables were also to be allowed, but ***reference reference*** types were forbidden). An explicit ***ref*** operator was provided to indicate that a reference to some variable, rather than to its contents, was to be taken (if you like, an anti-dereferencing operator like the '*&*' of C).

```
int reference ii; int i;
ref ii := ref i;        # to assign a reference to i #
ii := i;                # to assign the int in i to the place indicated by ii #
```

Here is *innerproduct*:

```
proc innerproduct =
        (proc real a, b, int k, int reference p, real reference y) void:
    begin loc real s := 0;
        for p := 1 step 1 until k # ALGOL 60 loop semantics #
        do s := s+a*b;
        y := s
    end;
```

And here is the **call**:

```
loc [1:n] real x1, y1;  loc int j;  loc real z;
innerproduct(expr: x1[j],  expr: y1[j],  n,  ref j,  ref z);
```

One can see how Seegmüller's proposal derived from EULER [Wirth 1966a], but EULER was a dynamically typed language so that ***reference***s could refer to values of any type. EULER was the first language, so far as I am aware, to introduce the terms 'call-by-reference' and 'call-by-procedure'.

At St. Pierre, Hoare presented a concept of 'value/anti value', which appears to have been what we now call 'copy/copy back'. At any rate, this is what appeared in [Wirth 1966b] alongside an untouched call-by-name as in ALGOL 60 and parameterless ***procedure*** parameters (to be matched by **statements** or **expressions** in the **actual-parameters**). Here is *innerproduct* in ALGOL W:

```
procedure innerproduct
        (real a, real procedure b, integer value k, integer p, real result y);
    begin y := 0; p := 1;
        while p≤k do begin y := y+a*b; p := p+1 end
    end;
```

Just to be contrary, I declared *a* by-name and *b* by-procedure. The effect is the same. Here is the **call**:

```
real array [1:n] x1, y1;  integer j;  real z;
innerproduct(x1[j],  y1[j],  n,  j,  z);
```

This then was the State-of-the-Art of parameter passing in October 1965, and its total confusion may be contrasted with the limited set of options to which modern programming language designers confine themselves, and with the simplicity and elegance of the system that presently emerged for ALGOL 68. At the end of the St. Pierre meeting they voted against "Seegmüller's references," but everything else was left undecided.

### 2.3.4.2 *Records*

A concept of 'trees' had been introduced at Princeton (they seem to have been somewhat like the 'lists' of EULER) and had consequently been included in both [Wirth 1965] and [MR 76]. Shortly before St. Pierre, however, Hoare [1965b] had proposed 'records', each being of some named 'record ***class***' and with local ***ref*** *(some class)* variables to locate them, and also ***ref*** fields within the records themselves. Records of a given class could be created, on demand, on the heap (but there were to be *no* local records on the stack). With these, one could create all manner of lists, trees, graphs, and so

forth. This technique is quite familiar to us now, but it seemed like a revolution at the time, and the WG accepted it with enthusiasm.

The idea of records in fact derived from several sources. McCarthy [1964] had proposed a type constructor ***cartesian*** (also a rather elaborate ***union***), Naur [1964] had proposed similar 'structures', but without ***class***es, and of course COBOL had used records as the basic unit to be transferred to/from files. But it was AED-0 [Ross 1961] that first showed how complex structures could be built up using records and ***ref***s. The difference between AED-0 and Hoare's scheme is that the latter was strongly typed; thus if you had a ***ref*** you knew which of the fields within its record were themselves ***ref***s, and thus you could do garbage collection (although Hoare did have an explicit *destroy* operation as well). AED-1, which appeared subsequently, was also strongly typed. In the absence of garbage collection, a record would disappear when the **block** in which its ***class*** was defined was exited. (This feature is also present in Ada, but it is only appropriate in a language with a name-equivalence rule for its types (2.3.4.7).)

Hoare also proposed some optional features, including the possibility of specifing a default initialization for a ***class*** (this useful feature was resurrected for Ada) and discriminated ***union***s (with even a **conformity-clause** (2.3.4.6) such as eventually appeared in [RR]). He also wrote a further paper [Hoare 1966] which included, by way of example, Dijkstra's well-known algorithm for finding the shortest path between two nodes of a graph.

In fact, the records actually introduced into ALGOL 68 were known as 'structures', and introduced by the word ***struct***.

### 2.3.4.3 *Orthogonality vs Diagonality*

Seegmüller [1965b] had ***ref***s for call-by-reference (which the WG had rejected). Hoare [1965b] had ***ref***erences for accessing records (which the WG had accepted). By Kootwijk, Van Wijngaarden had brought these two concepts together, and everything fell into place (at least that was his view, which Seegmüller was happy to share at that time).

According to the principle of orthogonality

- A record ***class*** was a type (just as an ***array*** was a type).

  ∴ there should be record variables, record parameters, and record results.

- ***ref x*** was a type, for any ***x*** (whether a record or not).

  ∴ ***ref ref x*** was a type.

- ***ref***s could be used for parameters (for call-by-simple-name).

  ∴ ***ref***s should be able to refer to any variable of suitable type, even (especially) local variables. In fact, the concepts of variable and reference had become synonymous, and the type of a ***real*** **variable** *x*, as declared in ***loc real x***, is actually ***ref real***.

- ***proc***s could be used for parameters (for call-by-full-name).

  ∴ there should be ***proc*** variables, ***proc*** parameters, and ***proc*** results.

  ∴ also there should be syntax for constructing anonymous ***proc***s (see 2.3.5, where the final version of *innerproduct* will also be found).

- The left hand of an **assignation** had customarily been a **variable**.

  ∴ Now it would be any ***ref*** valued **expression**.

In fact, the only major feature altered from Seegmüller's scheme was that a dereferencing **operator** ***val*** took the place of the anti-dereferencing **operator** ***ref***. And because of the automatic coercion, ***val***

was hardly ever necessary. At the last moment, in [MR 100], ***val*** was abolished in favour of the newly invented **cast**.

Hoare was horrified. In his Turing Award lecture [Hoare 1981] he describes his dismay at the "predominance of references" and other complex features being added to the language at that time. In his opinion, records had been provided for one specific purpose, and he had therefore deliberately eschewed local records and ***ref***s to local variables [Hoare, 1965b]. As he said at Warsaw:

> In the last two years I spent a tremendous amount of energy trying to persuade people not to have any indirect addressing in programming languages. There was a discussion in Princeton and Grenoble and the Committee was against indirect addresses. I wanted to have references only in connection with the records. I do not understand what all these general references are about. I think an ordinary programmer will have a tremendous scope for mistakes.

Throughout the meeting he strove to have the facilities of references limited to the things he considered safe (this being dubbed the "diagonal" approach).

> If you kept references to records and parameter-passing mechanisms apart, we could save ourselves a tremendous amount of confusion.

The discussion lasted for the best part of a day, but Van Wijngaarden was able to demonstrate that his system was self-consistent, and the only technical objection that stuck was the possibility that a ***ref*** to a local variable might still exist after exit from the **block** where it was declared, and this was fixed by a rule forbidding **assignations** that could lead to this possibility; usually this rule could be enforced at compile-time, although run-time checks would be needed in some circumstances. Hoare remained unhappy about this, although my experience of using the language with students is that violations of this rule arise very seldom in actual programming, and it is not a serious practical issue.

Thus the battle within WG 2.1 was won by Orthogonality. Moreover, most modern programming languages have tended to follow this lead, judging by the emphasis that is customarily placed upon all types of values being "first class citizens" of the language.

The type system of ALGOL 68 has been adopted, more or less faithfully, in many subsequent languages. In particular, the ***struct***s, the ***union***s, the pointer types, and the parameter passing of C were influenced by ALGOL 68 [Ritchie 1993], although the syntactic sugar is bizarre and C is not so strongly typed. Another language with a related type system is SML [Milner 1990], particularly with regard to its use of ***ref*** types as its means of realizing variables, and C++ has also benefitted from the ***ref*** types [Stroustrup 1996]. Even the Intuitionistic Theory of Types [Löf 1984] uses essentially the same type system.

#### 2.3.4.4 *The Bend*

There was an additional feature inherent in allowing the left side of an **assignation** to be any ***ref*** **expression**, and this was that ***ref***s to individual fields within records, or elements within ***array***s, had to be permitted, for how else could you write

*age* ***of*** *tom := 16;* or *a[i] := 3.142;* ?

Orthogonality then demanded that such ***ref***s (to fields within records) could be passed as parameters and preserved in variables. Getting this correct in the syntax of **selections** and **slices** nearly drove the authors "round the bend"; hence, the title given to it.

Now although modern languages tend to have adopted the orthogonal approach (Pascal is more orthogonal than ALGOL W, and Ada is more orthogonal than Pascal), very few of them (except for

C) allow their ***refs*** to do this, or to point to locals, so is there any benefit to be gained apart from satisfying some philosophical principle?

Now the claim for orthogonality must be that, by providing a very clean and systematic language, serendipitous benefits, not foreseen at the time of language design, will arise during actual use. The technique, which became known as the "3-***ref*** trick," and was only discovered after the necessary features were already in place [R 10.5.1.2.b], illustrates this point.

Here is a recursive program to insert a new item at its correct place in a binary tree. This is the classic example to illustrate the benefits of recursion in programming languages.

```
mode node = struct (int val, ref node left, right);
ref node nonode = nil;
loc ref node start := nonode;

proc insert = (int v, ref ref node place) ref node:
    if place :≠: nonode
    then    if v<val of place then insert (v, left of place)
            else #v≥val of place# insert (v, right of place)
            fi
    else place := heap node := (v, nonode, nonode)
    fi;
```

Now the recursion here is tail recursion, and it is well known that tail recursion can always be changed into iteration. It would be a strange language in which this could not be done and indeed, in this case, it is straightforward and one can imagine a mechanical tool to do the transformation automatically.

```
proc insert = (int v) ref node:
    begin loc ref ref node place := start;
        while place :≠: nonode
        do  if v<val of place then place := left of place
            else #v≥val of place# place := right of place
            fi
        od;
        ref ref node(place) := heap node := (v, nonode, nonode)
    end;
```

Observe that the ***ref ref node*** **formal-parameter** *place* has become a ***loc***al ***ref ref node*** **variable**. (The type of *place* is therefore ***ref ref ref node***; hence the term "3-***ref*** trick.") Observe also that *place* is required to point at both a local variable (*start*) and at a field of a ***struct*** variable (*left* ***of*** *place*).

Now the first version of *insert* can easily be written in Pascal (*place* will be a ***var*** parameter) or in Ada, but try to write the second version in either of those languages. It just cannot be done. The only other modern language in which this works is C, and there you have to be exceedingly careful to watch the types of everything.

### 2.3.4.5 *Variable-declarations*

The history of how variables are created is interesting. As explained in 2.3.4.3, a **'real'** variable is the same thing as a **'reference to real'** constant. A variable is created by means of a **generator**:

***loc real*** or ***heap real***

and a **variable-declaration** is equivalent to a **constant-declaration** containing a **generator**. Thus:

| | | |
|---|---|---|
| ***loc real*** *x;* | means the same as | ***ref real*** *x* = ***loc real;*** |
| ***heap real*** *x;* | means the same as | ***ref real*** *x* = ***heap real;*** |

Indeed, in [R] the **variable-declaration** was defined by that equivalence. Now in [MR 88] ***loc*** was not a symbol writable by the user, but from [MR 93] the user could write his own **local-generators** (as well as **heap-** ones), which was a pity because they are painful to implement and not particularly useful. And in the **heap-generator** the ***heap*** was optional, which was a pity because it made it difficult to parse. Also, the explicit ***loc*** was not then permitted in the **variable-declaration** (you could only write ***real*** *x;* just as in ALGOL 60).

In [RR] we made the ***heap*** compulsory in **heap-generators** and introduced the optional ***loc*** into the **variable-declaration** (too late to make it compulsory, but it made things more or less symmetrical). An explicit ***loc*** in a **variable-declaration** has considerable advantages didactically (readers will observe its consistent inclusion in this paper), and it prevents any possible confusion between

| | |
|---|---|
| ***real*** *e* := *2.718;* | a **variable-declaration** for a variable *e*, and |
| ***real*** *e* = *2.718;* | a **constant-declaration** for a constant *e*. |

If only the ***loc*** had been present and compulsory all along, there would have been an enormous simplification of implementation (see 2.7.1 for a full discussion), but the ALGOL 60 influence was still too strong.

### 2.3.4.6 *Unions*

Languages with Hoare-style record handling can benefit greatly from having ***union***s as well [Hoare 1965b]. [W-2] actually had a type ***free***, effectively the union of all possible types. However, after discussions at Warsaw, Hoare's ***union***s were incorporated in [MR 88] in its place.

To determine the actual type in residence, there was the **conformity-relation**, which could test whether the actual type was suitable for assignment to a given **variable**, and even assign it if asked.

```
mode man = struct(string hisname, ...), woman = struct(string hername, ...);
mode person = union(man, woman);
loc man tom; loc woman mary; loc person body;
tom ::= body;        # assigns and returns true iff body is actually a man #
```

There was also a **conformity-case-clause**

```
case tom, mary ::= body in hisname of tom, hername of mary esac
```

However, in [RR], both of these were replaced by a **conformity-clause**

```
case body in
        (man m): tom := m; hisname of tom,
        (woman w): mary := w; hername of mary
esac
```

As originally proposed, ***union(real, int, bool)***, ***union(int, real, bool)*** and ***union(real, union (int, bool))*** were different types, but at Tirrenia, Yoneda asked for ***union***s to be both commutative and accumulative, in the interests of mathematical tidiness (although it must be said that there is quite a price to be paid in the implementation). The members pounced on the idea, saying that a W-Grammar

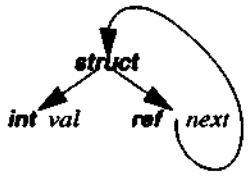
could not express it, but the next day Van Wijngaarden produced the syntax—a weird and wonderful constructive use of syntactic ambiguity which now appears in [R 7.1.1cc–jj].

#### 2.3.4.7 *Type Equivalence*

In Hoare's record proposal, it was assumed that every record would belong to a ***class***, and that the ***class*** would be identified by its name (so two ***class***es would be distinct even if they had identical structures, and hence the possibility existed to destroy all instances of a ***class*** once the **block** where it was introduced had been exited). In ALGOL 68, however, the naming of a ***struct*** type is optional (though usually to be recommended) and a rule of structural equivalence of types applies, so that even ***a*** and ***b*** in the following are of the same type:

```
mode a = struct (int val, ref a next);
mode b = struct (int val, ref struct (int val, ref b next) next);
```

both being descriptions of the following graph:




This caused much furore at Tirrenia, not because of structural equivalence as such (the existence of the distinction was never mentioned), but because of the infinite protonotion that arose in describing it. However, that is a matter of the description method, and will be discussed in (2.5.1.4).

Pascal (eventually) and Ada have a rule of name equivalence, and with hindsight maybe ALGOL 68 should have done the same. (It would have made the future introduction of abstract data types more secure, although it is awkward for generic or polymorphic type schemes, which is why recent languages such as SML have reverted to structural equivalence.)

### 2.3.5 Procedures

A proposal in [Naur 1966] to amalgamate the **specification** (of the type) of **formal-parameters** into the same line as the **parameters** themselves was accepted with enthusiasm (such that to modern eyes the separation of these items in ALGOL 60 seems strange). The idea of letting the value of the last thing in a **block** be the value of the whole **block** (and hence the way to show the result of a ***proc***) comes from [Naur 1964], although it also featured in EULER [Wirth 1966a].

Since in [W-2] ***proc***s could be **formal-parameters**, and hence by orthogonality could be anything else, it was natural that a means to construct anonymous values of ***proc*** types should exist (essentially a lambda-expression, but here known as a **routine-text**). The idea came from Samelson [1965] where it was conceived more as a way of cleaning up Jensen's device. Here is *innerproduct* taken from [RR 11.2]:

```
proc innerproduct1 = (int n, proc (int) real x, y) real:
        begin long real s := long 0;
            for i to n do s +:= leng x(i) * leng y(i) od;
            shorten s
        end;
```

And here is the **call**, with **routine-texts**:

```
loc [1:n] real x1, y1;  loc real z;
z := innerproduct1(n, (int j) real: x1[j], (int k) real: y1[k]);
```

Now that **procs** were first class citizens it was hoped that full functional programming would become possible, but the following example, produced by Landin at Tirrenia, shows the snag.

```
proc curryplus = (real u)proc(re al)real:  (real v)real: u+v;
proc(real)real addthree = curryplus (3);
addthree (5)          # should yield 8 #
```

The **routine-text** *(real v)real: u+v* yielded by *curryplus* has built into it the **identifier** *u*, itself a **formal-parameter** of *curryplus*, and in any reasonable stack-based implementation the value of *u* is no longer around by the time *addthree* comes to be called. There is therefore an extent restriction to forbid this usage [RR 7.2.2.c].

After publication of [R], Bekic made strong pleas at Habay-la-Neuve, and subsequently, to have this restriction lifted, but it would have implied a fundamental change to the underlying philosophy (namely, that ALGOL 68 was a stack-based language). At Vienna I was able to show that the same effects could be achieved by introducing partial parametrization into the language (as was later published in [Lindsey 1976], although never implemented to my knowledge). Modern functional languages such as SML do not have this restriction, but in consequence they pay an extra run-time overhead.

### 2.3.6 Overloading

This was McCarthy's pet topic. Seemingly he had raised it at Baden in 1964; Naur [1964] suggested essentially the same thing, and apparently Hoare had also proposed it at a NATO Summer School. But it was not among the features agreed on at Princeton, so the next time McCarthy appeared at a meeting, at Warsaw, he raised it again. By sheer persistence, he persuaded a somewhat reluctant WG to let Van Wijngaarden put it in.

The feature allows the user to declare his own **monadic-** or **dyadic-operators**, together with a **priority-declaration** if the **(dyadic-)operator** has not a priority already (**monadic-operators** always have the highest priority regardless).

```
prio max = 9;
op max = (int a, b)int: if a>b then a else b fi;
op max = (real a, b)real: if a>b then a else b fi;
```

Observe that the body of ***max*** is just a **routine-text**, so it might be said that we have just a fancy new way of calling **procs**, but observe also that we have separate definitions of ***max*** for the cases ***int-int*** and ***real-real*** (and ***int-real*** and ***real-int*** ought to have been defined also). Thus ***max*** has been 'overloaded', and to see which version of ***max*** is intended in any particular context the compiler simply has to look at the types of its two **operands**. Contrast this with the situation in Ada (which along with C++ [Stroustrup 1996], has adopted this overloading) in which, not only the types of the **operands**, but also the type required for the result must be taken into account. This puts a hugely increased burden on the compiler, with little real practical benefit that I can see.

Now it is the coercion of the **operands** that provides the interest in overloading. Clearly, in *x* ***max*** *y* the coercion of *x* and *y* must not be allowed to include widenings, but it should include the 'firm' coercions such as dereferencing (for *x* and *y* might well be **variables** of type ***ref int***, say). We have

also modified the rules of block-structure, for a **block** can now contain several definitions of the same **operator**, but not so that more than one of them can be legitimately chosen in a given **formula**. There has to be a rule forbidding the coexistence of two definitions the types of whose **operands** are 'loosely related' [R 4.4.2.c] or 'firmly related' [RR 7.1.1]. (Checking this property is quite hard work, and in the sublanguage promulgated in [Hibbard 1977] there is a less permissive 'meekly related' rule.)

Incorporating overloading into the Report was a major upheaval. Not only did the coercion rules and context conditions have to be overhauled, but all the existing "built-in" **operators** had to be taken out of the syntax and replaced in the **standard-prelude**. At Zandvoort, Yoneda asked why could not ***proc***s be overloaded also? Because we expect to be able to do widening coercions to **actual-parameters** (note that Ada can overload its ***procedure***s because it has no widening coercion). Hoare [1968] wanted subscripting, '*:=*' and even '*;*' to be defined as overloaded **operators**, but these would have needed user access to the descriptors of ***array***s (dangerous), right-associative **dyadic-operators**, and other such complications.

### 2.3.7 Labels and Switches

ALGOL 60 had the most horrendous collection of **labels**, **designational-expressions**, and **switch-declarations**. Trouble was that people got used to using them. (This was long before such things had been declared "harmful" [Dijkstra 1968b]), and as a result some really horrid constructs were proposed (some even made their way into the final language). Things actually started by getting worse. Duncan [1964] proposed ***label*** as a type, with ***array***s of them instead of ***switch***es, and the ability to jump *into* a **block**. Naur [1964] realized that an environment has to be a part of every ***label*** value, rendering jumps into **blocks** inappropriate.

The solution to the ***switch*** problem came with [Hoare 1964], which proposed a **case-clause** of the (eventual) form

***case*** *integer expression* ***in*** *first, second, third* ***esac***

but without any ***out*** (or ***otherwise*** option). Clearly, this would be implemented by a table of jumps (in contrast to the **case-clause** in more recent languages where each alternative is preceded by a possible value of the *expression*). This was well received, and was the end of the ALGOL 60 ***switch*** (but it still left the possibility of ***label*** values and ***array***s thereof).

At St. Pierre, Wirth [1965] would not allow jumps *into* a syntactic entity, nor any **designational-expressions** (these were accepted), nor ***label***s as parameters (which was not accepted). Van Wijngaarden then asked whether **declarations** might not be labelled (apparently so that one could jump back to them to change the size of ***array***s). Surprisingly, and in the face of grim warnings from Hoare, the meeting agreed to let him look into this (but fortunately the idea never came to fruition).

***label*** variables and parameters had been a feature of [Seegmüller 1965b], but following St. Pierre the only way to pass a ***label*** around was to encapsulate it as a **jump** inside a ***proc***. So Seegmüller complained that to pass *label* to a procedure *p* he would have to write something like *p(****expr: goto*** *label)*. Van Wijngaarden promised to look into it. The result was that, in [MR 88], first a **jump** in a suitable context was automatically turned into a ***proc***, allowing *p(****goto*** *label);* and second the ***goto*** in a **jump** was made optional (to the great inconvenience of compiler writers, as it transpired), allowing *p(label)*. But worse! Van Wijngaarden was now able to exhibit his pride and joy—his pseudo-switch [R 8.2.7.2].

```
[ ] proc void switch = (e1, e2, e3); # e1, e2 and e3 are jumps to labels #
switch[i];
```

or even

***loc** [1:3] **proc void** switch := (e1, e2, e3);*
*switch[2] := e1;*

To my shame, I must admit that this still works in [RR], although implementations tend not to support it.

### 2.3.8 Parallel and Collateral

At St. Pierre there was discussion as to whether the order of elaboration of the two **operands** of a **formula**, or of the **actual-parameters** of a **call**, or of the two sides of an **assignation**, should be prescribed. In spite of some protest from Naur, it was agreed that it should be explicitly undefined (and all subsequent languages seem to share this view).

Meanwhile, [MR 76] contained a proposal for explicit parallelism in which the elementary actions in the parallel branches were to be merged in an undefined order. This was agreed (again over the protests of Naur) but with the proviso that a serial elaboration should be among the legal interpretations (hence there was no idea of fairness). Questions of synchronization and resource sharing were hardly discussed, it being supposed that interlocks could easily be programmed (that this was not so was shown by [Dijkstra 1968a], of which earlier drafts were available in 1965, but apparently not known to the WG). However, [MR 76] also suggested that elaboration of **operands** of a **formula** and so on should be implicitly in 'parallel', thus going further than merely undefining the order. Of course the reason was to allow the implementor as much leeway as possible to introduce optimizations, and it later turned out that this proposal also permits the elimination of common sub-expressions, even when they contain side effects.

Certainly there was a demand for genuine parallelism (see for example [Dahl 1966]), and by the time the issue was discussed at Zandvoort, [Dijkstra 1968a] was well known; hence [MR 88] provided for the symbol ***elem*** in front of a **closed-clause**, the whole of which was to be treated as an elementary action. Van Wijngaarden seemed to think this sufficient to enable Dijkstra's semaphores to be written (but I cannot see how he could have avoided using spin locks). Randell then proposed that semaphores should be included as an explicit feature. This led to a bad attack of cold feet concerning whether parallelism should be in at all, and if so whether it was to support multiple processes on one or more processors, or was just a device to undefine the order of elaboration. The trouble was that [MR 88] did not distinguish between these two usages.

The outcome was that in [MR 93] the term 'collateral elaboration' was used to indicate where the implementor might do his optimizations, and an explicit construction

***par begin** process1, process2, process3 **end***

was introduced for 'parallel elaboration', within which Dijkstra's semaphores (written as ***up*** and ***down***) could be used [R 10.4; RR 10.2.4]. The ***elem*** symbol was still present, but it had disappeared by [MR 101]. With hindsight, it is debateable whether such a low-level feature as the semaphore was a proper feature of such a high-level language. At North Berwick, even Dijkstra was expressing doubt on this point.

### 2.3.9 Transput

The report of the I/O subcommittee established at St. Pierre [Merner 1966] was in the form of an addition to [Wirth 1965]. It provided separate procedures for converting numerical values to/from ***string***s, as opposed to transmitting them to external media (an explicit request of the WG at St. Pierre). There was an elaborate system of formats (represented internally as ***string***s), largely derived from [Knuth *et al.* 1964]. At Zandvoort, the WG asked for an explicit ***format*** type (one benefit of which was the possibility of introducing **dynamic-replicators**). However, this was about the only action on transput taken by anybody up until October 1967, when Koster returned from his military service and serious work began.

Koster's main contribution was the 'rectangular book' model in which an external document was conceived as composed of so many pages, each of so many lines, each of so many characters, it being the responsibility of the system to keep track of the 'current position' and to institute action if the boundaries were exceeded. (ALGOL 68 is the only major language to provide this service.) Actually, the prime reason for adopting this particular model was to ensure implementability on IBM mainframes, whereon the alternative idea of a file as an undifferentiated stream of characters (with perhaps some newlines thrown in) would be quite unimaginable. (Just think—there might then be more than 80 characters between newlines.) IBM was not to be given any such excuses.

The conversion and transmission of values were now recombined so as to provide a usable system, although the explicit conversions to/from ***string***s were still present (but in a not very satisfactory form, so that in the revision new functions *whole*, *fixed*, and *float* were provided in their place). For reasons of orthogonality, the transput of ***struct***s and ***array***s had appeared (as 'straightened' sequences of their primitive components). A nice feature was the ability to gather a sequence of disparate values and layout procedures into one procedure **call**:

```
print((newpage,
        "Some Title", newline,
        "index = ", someinteger, space, somestruct, ", ", somereal, newline));
```

With the removal of the type ***free*** after Warsaw, this had to be done by a fictitious ***union*** of all transputable types, and it required some trickery to fit it into the language framework. The ***proc*** *read* was made to be, so far as possible, complementary with *print*. The ***format***s were derived from those of [Merner 1966] with the addition of **dynamic-replicators**, and **alignments** to control line and page endings. Considerable effort was devoted to making the action of ***format***s on input complement that on output.

[MR 93] was the first that anyone had seen of the detailed transput specifications, but serious discussion was lost in the general furore about other matters. Following discussions at Tirrenia, 'on' routines to handle exceptions such as line and page overflow, and facilities for *open*ing named files were added in [MR 95]. However, many detailed problems remained. If you took it too literally (and evidently you were intended to take the rest of the Report literally) you found that all sorts of strange behaviours were mandated.

The changes in the revision were mostly minor, to remove such unintended behaviours. They were concerned with the fact that the old model assumed every line to be padded with spaces to some fixed width and with a presumption that the **particular-program** was the only program running in the whole universe. Various problems with ***format***s were fixed (for example, that the effect of the **format** *$ 5z $* differed from that of *$ zzzzz $*). The only new features were the conversion routines *whole*, *fixed*, and *float*, and the facility to *associate* a ***file*** with a ***[]char*** in place of an external ***book***.

**TABLE 2.4**

WG 2.1 members active in the revision of ALGOL 68

Of the members already listed in Table 2.3, the ones who continued to be active in the revision were Bauer, Bekic, Goos, Koster, Lindsey, Mailloux, Paul, Peck, Van der Poel, Samelson, Sintzoff, Van Wijngaarden, and Yoneda.

Newcomers to the scene were:

| | |
|---|---|
| Steve Bourne | Bell Labs, Murray Hill, NJ |
| Henry Bowlden | Westinghouse Research Labs, Pittsburgh, PA [Secretary of WG 2.1] |
| Ian Currie | RRE, Malvern, UK |
| Paul Branquart | MBLE, Brussels |
| Peter Hibbard | University of Liverpool, UK |
| Lambert Meertens | Mathematisch Centrum, Amsterdam |
| Sietse van der Meulen | Rijks Universitaet, Utrecht |
| Stephen Schuman | IRIA, Paris |
| Robert Uzgalis | UCLA, Los Angeles, CA |

## 2.4 HISTORY OF THE REVISED ALGOL 68 REPORT

### 2.4.1 Dramatis Personae

Many who had earlier been prominent in WG 2.1 resigned from the group after Munich (most moving over to the newly formed WG 2.3), although Turski, Duncan, and Hoare remained in WG 2.1 for the time being. Table 2.4 lists some new members active in the revision.

The style of WG meetings, under the new chairman, Manfred Paul, was very different (and less confrontational). Much of the work was done by setting up subcommittees, each with a Convenor who could invite members from both within WG 2.1 and outside.

### 2.4.2 Improvements

#### 2.4.2.1 *Habay-la-Neuve to Fontainebleau*

Habay-la-Neuve (July 1970) was the week after a TC2-sponsored conference on ALGOL 68 Implementation in Munich [Peck 1971]. Although the implementations by Goos at Munich and Branquart at MBLE had been underway since before the finalization of [R], the race had been won by an outsider, a team from the Royal Radar Establishment at Malvern in England, who had implemented a dialect they named ALGOL 68R. This had caused quite a stir at Munich. It was simple, and it worked, even though it was not quite ALGOL 68. There had also been discussion at Munich about features that had caused implementation problems, and there had been suggestions for sublanguages that would avoid them.

At the meeting, there was a wish-list, prepared by Van Wijngaarden and myself, of things we might like to change. As the meeting progressed, this list was augmented until it was three times its original size. A selection of these items for serious consideration was chosen, and an ad hoc subcommittee on "improvements" hammered them into shape. It was clearly envisaged that there was going to be a Revised Report, and there was tension between doing it later (to make it as complete as possible, an option strongly advocated by Samelson) and doing it earlier (so that fewer implementations would

be affected). There was, however, a firm decision that, when the time came, there would be just one revision "once and for all."

The pattern established was for the two subcommittees established at this time to meet between and during WG meetings and to produce reports for consideration by the full WG. After each meeting, some official message from the WG to the Computing Community would be published in the *ALGOL Bulletin*, together with edited versions of the subcommittee reports. So after the Manchester meeting (April 1971), a letter to the readership of the *ALGOL Bulletin*, quoting from the Covering Letter (Appendix A) the remark about having subjected ALGOL 68 "to the test of implementation and use," announced that there would presently be a single Revision but that, in the meantime, the changes under consideration were being published "for their critical appraisal [WG 2.1, 1971c].

By the time of Novosibirsk (August 1971) it was felt possible to fix a definite timetable, and a formal resolution [WG 2.1, 1972a] laid down that the definitive list of changes would be published in the *ALGOL Bulletin* after the next meeting, at which time editors would be commissioned to prepare the Revised Report, which was to be approved and published before the end of 1973. This schedule was known to be extremely tight, and Sintzoff pointed out that two further meetings of the Maintenance subcommittee would be required before the next full meeting.

Fraser Duncan, who had edited the *ALGOL Bulletin* since shortly after the beginning of WG 2.1, had announced his intention to resign, and at a moment when I was absent from the meeting attending to some business or other I was elected to take his place, a post I held until the *ALGOL Bulletin* finally expired in 1988.

At Fontainebleau (April 1972), the Editors commissioned were Van Wijngaarden, Mailloux, and Koster, together with Sintzoff and Lindsey (the convenors of the two sub committees). Peck, who was now Professor at Vancouver, did not become an editor officially until the following meeting. Our brief, as expressed in the formal resolution, was to "consider, and to incorporate as far as practicable" the points from the two subcommittee reports and, having corrected all known errors in the Report, "also to endeavour to make its study easier for the uninitiated reader" [WG 2.1 1972d].

#### 2.4.2.2 *Maintenance and Improvements*

This subcommittee, successor to the ad hoc improvements subcommittee at Habay, was officially constituted at Manchester with Sintzoff as convenor (although I acted as the "scribe," keeping the texts of the reports on paper tape and editing them on a Flexowriter). It met at Amsterdam (August 1971), Malvern (November 1971), and Brussels (February 1972), as well as during WG meetings, and each time it passed over the accumulated proposals together with comments received, and everything that was deemed "safe" by the full WG was published [WG 2.1 1971b, 1972b, and 1972e].

The report presented at Fontainebleau contained 53 specific suggestions for change, which were discussed, re-examined, and voted upon every which way. Many points that had seemingly been decided at earlier meetings were reopened and revoted, not always with the same result. By the time [WG 2.1 1972e] was published in the *ALGOL Bulletin*, implementors of the language had a pretty good idea of what was going to happen, and could bend their implementations accordingly.

After that the responsibility for improvements effectively passed to the Editors, who brought further items to the Vienna meeting, resulting in [WG 2.1 1973a], and to Dresden and Los Angeles, resulting in [WG 2.1 1973b].

In the event, the agreed changes amounted to a thorough tidying up of minor ambiguities, oversights, and inorthogonalities in the original language. Various redundancies, most notably proceduring (2.3.3.6) and formal bounds (2.3.2), were removed. ("The language is too fat," as Bauer remarked at Novosibirsk.) ***void*** became a type, ***flex*** became part of the type (2.3.2), a new **confor-**

**mity-clause** replaced the old **conformity-relation** (2.3.4.6), and various matters of representation were tidied up (for example, you could no longer write ***if*** *c* | *a* | *b* ***esac***). The only major language change that might have happened, the lifting of the extent restriction on ***proc***s advocated by Bekic (2.3.5), was left to a possible future extension.

#### 2.4.2.3 *Data processing and Transput*

This subcommittee was established at Habay with myself as convenor. It met the week before Manchester and considered many small changes to tidy up the transput. However, its main consideration was a concept of 'record transfer' in which complex structures could be sent to backing store and later retrieved exactly as they had been written. There was a concept of a 'masskey', which was to backing store what a ***ref*** was to ordinary store, and there was a proposal for 'modals'—a facility for generic/polymorphic procedures. These things continued to be discussed for some considerable time, but never did make it into the revision.

A further meeting was held in Amsterdam (August 1971), after which transput discussions tended to happen at the same time as the Maintenance subcommittee. The reports of this subcommittee were [WG 2.1 1971a, 1972c, and 1972f]. At Fontainebleau, we presented 32 specific suggestions for change but we ran out of time before these could be discussed. (It has always been a problem within the WG to get people who have a strong understanding and concern for language issues to discuss transput seriously.) The transput proposals were therefore all left for decision until Vienna, where they were mostly accepted and included in [WG 2.1 1973a]. The changes were mostly minor, and have already been described in (2.3.9).

### 2.4.3 Revision

Of the Editors appointed, Van Wijngaarden immediately announced that he would leave all the hard work to the rest of us, and in the event, Koster, who was in the process of becoming Professor at Berlin, did not play an active role either. The rest of us were geographically well separated: myself in Manchester, Mailloux in Edmonton, Peck in Vancouver, and Sintzoff in Brussels.

For keeping the text, machine assistance was needed, and we decided to keep it on the MTS system at Edmonton using a line editor called *ed* and a formatting program known as *fmt*. Edmonton had an AM typesetter for which we had a special disc made. It was the craziest mixture of fonts and symbols ever found on a typesetter disc, which may explain why it took them over two years to produce it. Mailloux was in charge of the text, and wrote a typesetting program compatible with *fmt*.

#### 2.4.3.1 *Vancouver (July 1972)*

John Peck invited us all to spend three weeks in Vancouver during July. There we all became familiar with MTS, *ed*, and *fmt*, and agreed how to divide the task. I was to be responsible for Chapters 1 and 2, which introduced the syntax notation and the semantic model (2.5.2). Peck was to be responsible for the syntax, and Sintzoff for the semantics. Mailloux was to be Keeper of the Text. We invented (or discovered) production trees, predicates (2.5.1.2), **'NEST'**s (2.5.1.3), and environs (2.5.2), and established a new 'structured' style for the semantics (2.5.3).

Sintzoff returned home via Edmonton, and as he sat in Mailloux's office the telephone rang. There was a temporary nine-months teaching post available, and did he (Mailloux) know anyone who could fill it? Well of course he did! Thus it came about that the centre of gravity of the whole operation moved to Canada. It was particularly fortunate that Vancouver and Edmonton ran the same MTS operating system.

#### 2.4.3.2 *Vienna (September 1972)*

The next WG meeting was in Vienna. (The Editors had a private meeting beforehand.) We had available samples of what we had written, in particular the **'NEST'** syntax (2.5.1.3), and also a large number of detailed issues to raise on the improvements. The meeting gave first priority to transput, whose discussion had been curtailed at Fontainebleau, and then proceeded to discuss and vote on the various improvements.

But then started the biggest attack of cold feet I have ever witnessed (brought about, perhaps, by the extent of the changes we were proposing to the method of description, and by the **'NEST'** syntax in particular). A "Petition for the Status Quo of ALGOL 68," signed by five members and five observers, claimed that the Revised Report had become a pointless undertaking (beyond a few corrections to ambiguities or inconsistencies) and that the idea should be abandoned. This was debated on the very last day of the meeting. The motion was that the revision should consist of [R] plus an addendum, and the petitioners produced a small list of the changes they conceded should be allowed (and clearly there was no time left at that meeting to re-vote all the detailed decisions already made).

We pointed out that some errors (for example, the infinite mode problem (2.5.1.4)) were almost impossible to correct in the old framework; we pointed out that implementors seemed content with the intentions as published and that only three implementations were anything like complete (and Malvern already incorporated many of the changes); we pointed out that, to remove troublesome points so as to have a basis for upwards-compatible future extensions, we would need at least twice the list proposed by the petitioners; we pointed out that it would then require an addendum half the size of the Report, and that it was crazy to define the changes to be made by the length of an addendum—we were getting nowhere, and now it was time for the vote.

It had long been a Working Group tradition to phrase straw votes in the form "Who could live with . . .?" rather than "Who prefers . . .?" Therefore, to let everybody see the full consequences before taking the formal vote, I proposed the questions, "Who could live with the state if the resolution was passed?" and "Who could live with the opposite state?" The results were 12-5-1 (for-against-abstain) and 12-1-5.

Now Van Wijngaarden was always a superb politician. Having observed who had actually voted, he now pointed out that passing the resolution "would cause the death of five of the six editors. That seems a most unproductive situation." And when the formal motion was put the vote was 6-6-6, and we were through.

#### 2.4.3.3 *Revision by Mail*

For the next nine months we worked separately on our pieces. Each circulated his texts to the others, and we all checked each other's work. The text of the syntax was maintained in Vancouver and the rest in Edmonton. (Periodically, I sent long *ed* scripts to Edmonton on paper tape—and surprisingly they usually worked without error.) This method of collaboration proved to be exceedingly fruitful. You don't launch a text over such distances until you are quite sure it is really what you intend to say. We survived together, accepting and appreciating each other's strengths and weaknesses, because that was the only way that could work.

It became apparent, right from Vancouver, that a major rewrite of the Report was necessary if it was to be made more accessible to the uninitiated reader. From time to time we were shocked and surprised by some outrageous consequence of the original definition, and each time, rather than patching it up, we stood back, identified *why* the misunderstanding had arisen, and modified the method of description so that it was no longer possible to express problems of that class. (Examples are the removal of extensions (2.5.4.1) and of infinite modes (2.5.1.4).) We treated the Report as a

large programming project, and consciously applied the principles of structured programming and good system design. The syntax and semantics are now written so that it is generally easy to check that all possible cases have been accounted for, and wherever there was any doubt about this we generated a proof of the doubtful property, even incorporating the proof in the pragmatics [RR 7.3.1]. That we succeeded in our aims is evidenced by the fact that, disregarding the transput section (which is another story), the total number of bugs now known in [RR] can be counted on the fingers of one hand (2.6.2.1).

During August of 1972 I had met Koster in Manchester, where we worked over the transput. However, there was no obvious person available actually to do the work. Transput, as usual, was receiving less attention than it should. Eventually, in February 1973, Richard Fisker, a student at Manchester, came to me looking for a project for his Masters thesis. He embarked upon it and had a small sample to show at Dresden, but the work was barely complete by the deadline in Los Angeles.

We circulated a draft two months before the next WG meeting in Dresden (April 1973), and I wrote an explanation of how the descriptive method had changed. This time there were no significant calls for abandonment, and many people gave their opinion that the new document was clearer to read, especially Sintzoff's semantics. A few further improvements were discussed and voted on.

#### 2.4.3.4 *Edmonton (July 1973)*

We had obtained funding from the NATO Science Committee that enabled the Editors to meet in Edmonton. Lambert Meertens, who had been playing an increasingly active role in monitoring the progress of the revision, was sent by Van Wijngaarden as his representative; Fisker was also present, and thus we had now reached our final complement of eight editors. The texts of the various parts were now merged into one document, and a hectic three weeks ensued.

As before, we discussed, argued, and compromised, but always retaining our mutual respect. The technique used if one of us thought his opinion was being shouted down was to say to the others, "Eat!" (these arguments often took place at mealtimes); by established convention the others then kept silent while the point was made; the paradigm is specified formally in [RR 3.3]. Finally, we produced a text for circulation to the WG in advance of the next meeting that was complete except for the pragmatics in the transput section (and much of these were included in a further edition provided at the meeting itself).

The Los Angeles meeting (September 1973) considered the Report, proposed some minor changes, and authorized its submission to TC2 subject only to "editorial polishing and explanatory material which make no additional substantive changes." The Editors were instructed to make it available to the subscribers of the *ALGOL Bulletin* and to submit it for publication in appropriate journals. WG 2.1 also established a standing subcommittee on ALGOL 68 Support (convened by Robert Uzgalis, and then by Van der Meulen from 1978) "to act as a point of contact with users and implementors of the language, and to serve their needs by preparing complementary specifications and enhancements."

### 2.4.4 The Aftermath

The Editors met once again in Manchester to discuss the polishing, and Fisker and I continued to tidy up the transput and complete the missing pragmatics. Since the typesetting disc still had not arrived, the Report was eventually issued to the *ALGOL Bulletin* readership reproduced from a lineprinter listing [TR 74-3].

#### 2.4.4.1 *Cambridge (April 1974)*

The newly formed Support subcommittee held its first meeting in Cambridge. A close study of the transput section, mainly in Amsterdam, had revealed a number of bugs and inconsistencies, and these formed the main topic of discussion. Now this meeting was, in my experience, the first time that any group of people sufficiently concerned with transput and competent to discuss it had ever assembled in one place, and the result was a very thorough review leading to a considerable cleanup (2.3.9). Whether we were entitled to make such changes as were shortly to be published in AB 37 as errata to [TR 74-3] is a legal nicety, but we certainly did, and the language is all the better for it. For a report of this meeting see [King 1974].

#### 2.4.4.2 *Typesetting*

The WG met in Breukelen in August 1974, but our special typesetting disc still had not been delivered. A further set of errata to [TR 74-3] appeared in AB 38.

The disc finally appeared early in 1975, so that at the Munich meeting of WG 2.1 (August 1975) the Editors spent all of their free time reading through the galleys, spotting all the things that were not quite right. [Van Wijngaarden 1975] just made it by the year's end, and a final set of errata in AB 39 brought [TR 74-3] into line.

### 2.4.5 Postscript

What have we learned about designing programming languages? First, a small group of people (four or five) must do the actual work, and geographical separation (as between Amsterdam/Brussels in [R] or Manchester/Edmonton/Vancouver in [RR]) is a great encouragement to careful work. (The advent of e-mail may undo this in the future.) The editors need to have very tidy and pernickety minds (which does not necessarily qualify them as good designers of language features); moreover, constructing the formal definition at the same time as the language is being designed is a sure way to avoid obscure corners—a simultaneous *prototype* implementation would be even better.

A large committee is a good place for brainstorming to get initial ideas, but it must recognize its limitations, the most important of which is that it is incapable of attending to detail. A large committee is also necessary to give democratic approval to the work of the editors. (In spite of the fact that this arrangement was observed to be unstable, I still do not know how to dampen it.) Moreover, the large committee will often take decisions which the editors *know* to be technically flawed—but that is in the nature of democracy.

Finally, there is much to be said for doing the whole job twice, but in any event it always takes twice as long as you think (even when you think you have allowed for this factor). Any attempt to rush it is doomed to failure (although it must be admitted that our revision did, bar a few items, meet its deadline—but it was hard work). My opinion is that in this Revision we did get it just about right, and what I have observed of other language design efforts since that time only serves to confirm this view.

## 2.5 THE METHOD OF DESCRIPTION

The word "on the street" is that ALGOL 68 is defined by an "unreadable" document. Certainly [MR 93], when it first came out, was a tough nut to crack (I should know!), but [R] was already a big improvement and we tried strenuously to attack this problem for [RR]. But unfortunately, much of the mud slung at [MR 93] is probably still sticking.

The language is defined by its Report in four parts: the Syntax, the Semantic Model, the Semantics proper, and the **Standard-prelude**. These will now be examined in turn.

## 2.5.1 Syntax

### 2.5.1.1 *W-Grammars*

The notation now known as a W-Grammar seems to be the main stumbling block to people approaching the Report for the first time. When this completely new notation first appeared, its readers had no previous model on which to build. Today there are several similar notations (for example, Attribute Grammars and Prolog) with which people are familiar. So a good way to explain it to a modern audience is to start from Prolog. This will also help to illuminate both the power and the limitations of W-Grammars.

That a W-Grammar can be explained in terms of Prolog is not surprising when it is realized that Prolog actually stems from some attempts at natural language processing by Colmerauer. His first system [Chastellier 1969] actually used W-Grammars. This developed into the more structured Q-Systems, and finally into Prolog [Colmerauer 1996].

Here is a rule written in Prolog:

***assignation(ref(MODE)) :-***
***destination(ref(MODE)), becomes_symbol, source(MODE).***

This is Prolog, so *MODE* is a variable (it begins with an upper case letter) and *ref* is a functor (with no inherent meaning). The meaning is that we have a goal "do we see an ***assignation*** here?", and to answer this we must test the subgoals "do we see a ***destination*** here?", followed by "do we see a ***becomes_symbol*** here?", etc. (I have cheated slightly by regarding the source text as an implicit variable). Prolog will keep backtracking if it fails, and if it gets into a loop, one just says, "Ah! but that was just the procedural meaning—the declarative meaning clearly expressed the right intention." The nice thing about Prolog is that the values of variables such as *MODE* may be deduced from the subgoals or imposed with the question; and again, from the declarative point of view, the distinction does not matter—the value of the variable just has to be consistent throughout.

Here now is the corresponding rule in a W-Grammar [R 8.3.1.1.a].

**reference to MODE assignation :**
**reference to MODE destination, becomes symbol, MODE source.**

The difference is that in Prolog the parameters of goals are well-formed expressions built out of functors, atoms and variables, whereas in a W-Grammar they are just free strings of characters, or 'protonotions', and the variables, or 'metanotions', stand for other strings of characters as produced by a context-free 'metagrammar'. Some possible 'metaproductions' of the metanotion **'MODE'** are

**'real', 'integral', 'reference to integral', 'row of reference to integral'**

and so on. So if you substitute **'integral'** for **'MODE'** you get

**reference to integral assignation :**
**reference to integral destination, becomes symbol, integral source.**

(observe the consistent substitution for **'MODE'** throughout), which will eventually produce familiar things like *i* := *99*. But observe how we have insisted that the type (or mode) of the **destination** *i* must

be ***ref int*** because the type of the **source** *99* is ***int*** (or *99* must be ***int*** because *i* is ***ref int***—you can argue both ways round, just as in Prolog).

The unstructured and potentially ambiguous strings of characters that form the rules of a W-Grammar give it great power; indeed, Sintzoff [1967] shows that it can produce any recursively enumerable set, and hence is equivalent to a Chomsky Type-0 Grammar. One can say things that simply cannot be expressed using the well-formed expressions of Prolog. The Bad News is that it is quite impossible to write a parser based on unrestricted W-Grammars. If you want to do that, you must introduce some more structure, as in Prolog, or as in the Affix Grammars described in Koster 1971.

Here is an example that cannot be written in Prolog (although it was possible in Q-Systems). First, some metagrammar:

**MOOD :: real ; integral ; boolean ; etc.**
**LMOODSETY :: MOOD and LMOODSETY ; EMPTY.**
**RMOODSETY :: RMOODSETY and MOOD ; EMPTY.**

Hence, in the rule [R 8.2.4.1.b],

**one out of LMOODSETY MOOD RMOODSETY mode FORM :**
**MOOD FORM; ....**

**'LMOODSETY MOOD RMOODSETY'** can metaproduce a sequence such as **'real and integral and boolean'** in such varying ways that the **'MOOD'** can be chosen as any of the three (for example, choose **'real and integral'** for **'LMOODSETY'**, **'EMPTY'** for **'RMOODSETY'**, and thus **'boolean'** for **'MOOD'**).

Thus a W-Grammar is a much more powerful notation than Prolog; but this power must be used by report editors with the utmost discretion, if the result is to be intelligible.

Here is another rule [R 8.3.0.1.a]:

**MODE confrontation : MODE assignation ; MODE conformity relation ;**
**MODE identity relation ; MODE cast.**

With this you can try to produce a **confrontation** for any **'MODE'**—so how about a **real-confrontation**? But the rule already given for **'assignation'** will produce only a **reference-to-MODE-assignation**. Likewise, the only rule for **'conformity relation'** is for a **boolean-conformity-relation**. In fact, the only alternative on the right side of the rule that has productions for every possible **'MODE'** is **'MODE cast'**. All the other attempts at a **real-confrontation** lead to 'blind alleys'. Duncan was much concerned about blind alleys, especially as to how the reader could recognize one without an exhaustive search through the whole syntax, and he was not satisfied by their indication by a '–' in the cross references. Nevertheless, blind alleys turn out to be a very powerful tool in the hands of the grammar writer. Blind alleys occur also in Prolog.

In [RR] we made three major changes to the syntax, as will now be described.

### 2.5.1.2 *Predicates*

In the Revision we discovered new and helpful ways to use a W-Grammar. Consider the following: First the metagrammar

**NOTION :: ALPHA ; NOTION ALPHA.**
**ALPHA ::**
**a ; b ; c ; d ; e ; f ; g ; h ; i ; j ; k ; l ; m ; n ; o ; p ; q ; r ; s ; t ; u ; v ; w ; x ; y ; z.**
**NOTETY :: NOTION ; EMPTY.**

and now the ordinary rules

**where true : EMPTY.**
**where (NOTETY) is (NOTETY) : where true.**

So if I ask for productions of **'where (abc) is (abc)'** I will get the terminal production **'EMPTY'**, but if I ask for productions of **'where (abc) is (def)'** I am forced into a blind alley, and I get no terminal production at all.

There is also a rule of the form

**unless (NOTETY1) is (NOTETY2) : ...**

which produces **'EMPTY'** if **'NOTETY1'** and **'NOTETY2'** are different, and a blind alley if they are the same. This is quite a bit harder to write than the **'where'** case; for the full story see [RR 1.3.1].

Rules that can produce only **'EMPTY'** or a blind alley are known as 'predicates', and their use greatly simplifies the number and complexity of rules required in the Report. Here is an example of their use [RR 6.1.1.a]. The intention is to forbid deproceduring for a certain subclass of **FORMs**.

**strong MOID FORM coercee :**
**where (FORM) is (MORF), STRONG MOID MORF ;**
**where (FORM) is (COMORF), STRONG MOID COMORF,**
**unless (STRONG MOID) is (deprocedured to void).**

### 2.5.1.3 *Context Conditions vs NESTs*

In [R] a whole chapter is devoted to 'context conditions', which seek to ensure, for example, that each applied occurrence of an **identifier** 'identifies' its correct defining occurrence. In other languages these conditions are often referred to as 'static semantics'. So in [R 4.1.2.a], for example, there is a fairly straightforward set of Steps for starting at an applied occurrence and searching in ever wider **blocks** for its defining occurrence. But this is deceptively straightforward; people will quote it at you to show you how simple it all is. What they do not show you is the corresponding rule [R 4.3.2.b] for identifying **operators**, with this alternative form for the Step 3 of [R 4.1.2.a].

> Step 3: If the home contains an operator-defining occurrence **O** {, in an **operation-declaration** (7.5.1.a,b),} of a terminal production **T** of **'PRAM ADIC operator'** which is the same terminal production of **'ADIC indication'** as the given occurrence, and which {, the identification of all descendent **identifiers, indications** and **operators** of the **operand(s)** of **F** having been made,} is such that some **formula** exists which is the same sequence of symbols as **F**, whose **operator** is an occurrence of **T** and which is such that the original of each descendent **identifier, indication** and **operator** of its **operand(s)** is the same notion as the original of the corresponding **identifier, indication** and **operator** contained in **F** {, which, if the **program** is a proper **program**, is uniquely determined by virtue of 4.4.1.a}, then the given occurrence identifies **O**; otherwise, Step 2 is taken.

Every word of that (except for the pragmatic remarks between {...}) is essential for its correct interpretation. I know that it is correct because I have, in my time, studied it carefully, and I have seen all the half dozen incorrect versions that preceded it. But how can one have confidence with mechanisms requiring such turgidity?

In the Revision, therefore, we removed all the context conditions and brought the whole identification process into the syntax. The tools required to do this are still complex, but once one has understood them they hardly intrude. And their formality ensures that it is always possible to work through any particular case and to see whether and why it is or is not allowed.

Briefly [RR 1.2.3], there is a metanotion **'NEST'** that metaproduces a sequence of **'LAYER'**s (each corresponding to a **block**). Each **'LAYER'** metaproduces a sequence of **'DEC'**s (each corresponding to a **declaration** in the **program**). The **'NEST'** of each **block** contains one more **'LAYER'** than its parent **block**, and the syntax of **declarations** enforces that the **'DEC'**s of that extra **'LAYER'** correspond to the things declared, and moreover that they are 'independent' of each other (for example, that the same **identifier** is not declared twice) [RR 7.1.1]. Each syntax rule carries a **'NEST'**, so that the new rule for **assignation** [RR 5.2.1.1.a] is

**REF to MODE NEST assignation :**
**REF to MODE NEST destination, becomes token, MODE NEST source.**

Now, if the **source** is, or contains, an **applied-identifier**, the syntax fishes its **'DEC'** out of the **'NEST'** [RR 7.2.1] (it is required to be there) and checks its **'MODE'**. Of course, all of this machinery makes heavy use of predicates.

### 2.5.1.4 *Infinite Modes*

Consider the following:

***mode language = struct (int** age, **ref language** father);*
***loc language** algol;*

In [R] the mode of *algol* is **'reference to [language]'**, where **'[language]'** stands for **'structured with integral field [age] and reference to [language] field [father]'**. In other words, the string of characters describing this mode is infinite. Why did this need to be so? Consider:

*algol := father **of** algol .*

If the mode of *algol* had been represented finitely, by some such notation as **'reference to structured with integral field [age] and reference to *language* field [father]'**, then the mode of *father **of** algol* (after coercion) would be ***'language'***. Substituting these in the right side of the syntax rule for **'assignation'** gives

**reference to structured with integral field [age] and**
**reference to *language* field [father] destination,**
**becomes symbol, *language* source.**

And this is not allowed, since the **'MODE'** in the rule has not been substituted consistently. With an infinite string the substitution is consistent, provided only that you believe that, if $\mathbf{X}^{\infty}$ stands for **...XXXX**, then $\mathbf{X}^{\infty}$ **X** is no different from $\mathbf{X}^{\infty}$.

Infinite modes caused much discussion at Tirrenia and after. The strange thing is that the fuss had not started earlier, as infinite modes had been present since [MR 88]. Maybe the one obscure pragmatic remark in [MR 88] had been overlooked, whereas a second more obvious remark in [MR 93] had invited attention.

Duncan was particularly concerned (he claimed he could not read the Report beyond a certain point because he was still unfolding some infinite production). The question asked was "how could an infinite sequence be written on a finite piece of paper?", to which Van Wijngaarden's famous reply was that you wrote the first character on the first half of the sheet, the second character on half of the remaining part of the sheet, the third on half of what remained after that, and so on. More seriously, he claimed that (from the viewpoint of a constructivist mathematician) it was the mechanism that produced the infinite sequences that was important, not the sequences themselves. However, [R] did

not discuss these mathematical niceties, and it was not at all clear that the procedure was mathematically sound.

Meertens [1969] (quoting a letter from Tseytin) and Pair [1970] were the first to suggest that there might be flaws, but these doubts were expressed so vaguely, or so abstractly, that they went unnoticed. It was not until Boom [1972] exhibited an actual ambiguity that we were forced to take note. Here is Boom's example:

```
mode n = struct(ref n a, c);
mode hendrik = struct(ref struct(ref n a, b, c) c);  # 1 field #
mode boom = struct(ref n b, c);                      # 2 fields #
```

Now if you write out (in imagination) the full modes for ***hendrik*** and ***boom***, you will find that they cannot be distinguished under any reasonable understanding of infinity, certainly not one that regards $X^{\infty}$ X as equivalent to $X^{\infty}$, yet their modes are clearly supposed to be different.

For [RR], therefore, we resolved that any **program** should be producible in only a finite number of moves. The mode of *algol* is now **'reference to mui definition of structured with integral field [age] reference to mui application field [father] mode'** (read **'mui'** as a kind of label). However, there are many other ways of 'spelling' that same mode, just by expanding the **'mui application'** a few more times, but all these spellings are equivalent (in fact, a mode is now an equivalence class). The equivalence is defined by means of a predicate [RR 7.3.1] that systematically compares two (possibly infinite) trees, essentially following the algorithm of [Koster 1969]. Writing the syntax for this predicate was hard work. We all had a go at it, first to get a version that worked, and then to get a version tidy enough to present. It is, admittedly, also hard work to read (in spite of copious pragmatics), but the Report is written so that it does not obtrude, and the naive reader need hardly be aware of it.

### 2.5.2 The Semantic Model

Any definition of a programming language needs to define very carefully the "domain of discourse," or model, in terms of which everything is to be explained. Even today, many language definitions do not do this (they suppose that the reader can infer the meaning "obviously" intended), or they scatter this important information throughout the text. But McCarthy had explained this need to the WG as early as 1963, leading to its adoption by the Vienna school [Lucas 1969]. Following an explicit request from the Subcommittee at Kootwijk, [R2] was therefore devoted to describing the rules of the 'hypothetical computer'.

This particular hypothetical computer is well removed from "reality," giving little comfort to the man who wants to relate it to concepts with which he is already familiar (variables, addresses, and the like). Essentially, it is a graph of 'objects' (including 'external' constructs as produced by the syntax, and 'internal' values). Objects may have attributes (for example, values have types and scopes). The arcs of the graph are 'relationships' (both static and dynamic) between objects (for example, 'to access', 'to refer to', 'to be newer than', and 'to be a subname of').

The terminology used is often arcane and different from common usage, as already pointed out in Section 2.1.3. For example, the attribute 'scope' would nowadays be termed 'extent'. (It will be noted that I have been using such present-day terminology throughout this paper.) The worst example was the term 'name'. Names are allowed 'to refer to' values, and from time to time (during **assignations**) they may be made to refer to other values. If you read 'name' as 'variable', and 'to refer' as 'to contain', then you might suddenly begin to understand things a whole lot better. And you must not confuse

(internal) names with (external) **identifiers**, although the latter may, in appropriate circumstances, 'access' the former.

The 'actions' available to the hypothetical computer are the creation of new objects and the changing of dynamic relationships. The semantics for the various constructs of the language prescribe the actions that comprise their 'elaboration'.

One of the actions provided by the original Report was to take a copy of some construct, to replace some of its identifiers systematically by other identifiers, and then to elaborate it. This technique was in the tradition of the semantics of ALGOL 60, although other language definitions, such as [Lucas 1969], had already abandoned it. If carefully applied, it does give the expected and well-known semantics of block-structured languages, but its operation is counter intuitive as regards the way programmers normally think of a running program, and it is exceedingly difficult to ensure that it has indeed been "carefully applied" throughout. Having been caught out several times, therefore, we abandoned this concept and introduced a system of 'environs' and 'locales'.

An 'environ' comprises a 'locale' (which models the objects accessible locally within some **block**) together with an older environ (whose locales model objects that were declared outside of that **block**). Every elaboration is deemed to take place within some identifiable environ, and the chain of locales comprising that environ corresponds to the 'dynamic stack' familiar to implementors. (A separate chain of locales is defined for the 'static stack'.)

### 2.5.3 The Semantics

The semantics of ALGOL 68 is an operational semantics. In [R] the semantics were described in sequences of Steps, with frequent jumps to other Steps (a program full of ***goto***s, in effect). Also, there was some very turgid phraseology that we were able to remove by defining terminology in the semantic model more carefully.

In [RR], Sintzoff set out to adhere to the principles of structured programming, with 'If's, 'Case's, and nice indentation. The new semantics is therefore much shorter (only 12 pages total, which for 12 months work might not seem much). Here is the semantics of assigning a value to a name in both the old and the new styles.

First, from [R 8.3.1.2.c]:

> An instance of a value is assigned to a name in the following steps:
>
> Step 1: If the given value does not refer to a component of a multiple value having one or more states equal to **0** {2.2.3.3.b}, if the scope of the given name is not larger than the scope of the given value {2.2.4.2} and if the given name is not **nil**, then Step 2 is taken; otherwise, the further elaboration is undefined;
>
> Step 2: The instance of the value referred to by the given name is considered; if the mode of the given name begins with **'reference to structured with'** or with **'reference to row of'**, then Step 3 is taken; otherwise, the considered instance is superseded {a} by a copy of the given instance and the assignment has been accomplished;
>
> Step 3: If the considered value is a structured value, then Step 5 is taken; otherwise, applying the notation of 2.2.3.3.b to its descriptor, if for some **i**, $i=1, \ldots, n$, $s_i=1$ ($t_i=1$) and $l_i$ ($u_i$) is not equal to the corresponding bound in the descriptor of the given value, then the further elaboration is undefined;
>
> Step 4: If some $s_i=0$ or $t_i=0$, then, first, a new instance of a multiple value **M** is created whose descriptor is a copy of the descriptor of the given value modified by setting its states to the corresponding states in the descriptor of the considered value, and whose elements are copies of elements, if any, of the considered value, and, otherwise, are new instances of values whose mode is, or a mode from which is united, the

mode obtained by deleting all initial **'row of'**s from the mode of the considered value; next **M** is made to be referred to by the given name and is considered instead;

Step 5: Each field (element, if any,) of the given value is assigned {in an order which is left undefined} to the name referring to the corresponding field (element, if any,) of the considered value and the assignment has been accomplished.

And now from [RR 5.2.1.2.b]:

A value **W** is "assigned to" a name **N**, whose mode is some **'REF to MODE'**, as follows:

It is required that

- **N** be not **nil**, and that
- **W** be not newer in scope than **N**;

Case A: **'MODE'** is some **'structured with FIELDS mode'**:

- For each **'TAG'** selecting a field in **W**,
- that field is assigned to the subname selected by **'TAG'** in **N**;

Case B: **'MODE'** is some **'ROWS of MODE1'**:

- let **V** be the {old} value referred to by **N**;
- it is required that the descriptors of **W** and **V** be identical;
- For each index **I** selecting an element in **W**,
- that element is assigned to the subname selected by **I** in **N**;

Case C: **'MODE'** is some **'flexible ROWS of MODE1'**:

- let **V** be the {old} value referred to by **N**;
- **N** is made to refer to a multiple value composed of

  (i) the descriptor of **W**,

  (ii) variants {4.4.2.c} of some element {possibly a ghost element} of **V**;

- **N** is endowed with subnames {2.1.3.4.g};
- For each index **I** selecting an element in **W**,
- that element is assigned to the subname selected by **I** in **N**;

Other Cases {e.g., where **'MODE'** is some **'PLAIN'** or some **'UNITED'**}:

- **N** is made to refer {2.1.3.2.a} to **W**.

On the face of it that looks straightforward enough, but if you peer closely you will observe that no less than 20 apparently ordinary words are being used with a very precise meaning, as defined in the semantic model. Here they are:

value; name; mode; is; require; nil; newer than; scope; select; field; subname; refer to; descriptor; index; element; make to refer to; multiple value; variant; ghost element; endow with subnames.

Thus much of the improved readability of [RR] came from a careful choice of new technical terms and abbreviations—the art of choosing such abbreviations lay in making them "suggest" to the reader the same "obvious" meaning that he found when he actually looked them up. But the price paid was

the piling of more and more concepts and definitions into the semantic model, their purpose and necessity not always being immediately apparent. It is probably fair to say that the model is too big. (It certainly occupied more space than the semantics proper.)

Although the semantics is ostensibly written in English, it is in effect a formal notation ("syntax-directed English," as Mailloux described it). It is therefore proper to ask why we did not, in the revision, go the whole way and make it entirely formal. This would clearly have been possible, but the growth of our understanding of the new semantics and its model did not happen overnight, but indeed occupied all the time available to us, so the possibility could not be considered seriously. Moreover, it would have, in my opinion, added nothing to the rigour achieved, and the formalization of the large semantic model itself would have been a major task, not necessarily resulting in a clearer product.

### 2.5.4 Standard-prelude

It is always tempting to define parts of a language in terms of other parts already defined, but there are dangers, as I shall explain. This was done in two places in [R], in the 'extensions' and in the **standard-prelude**, especially in the transput.

#### 2.5.4.1 *Extensions*

Having defined a core language (which was actually about 90% of the total), [R 9] purported to define the rest by providing replacements for certain sequences of symbols. For example, according to [R 9.2.c] you could replace

***struct*** ***s*** = (***int*** *a, b*), ***struct*** ***t*** = (***real*** *x, y*); by ***struct*** ***s*** = (***int*** *a, b*), ***t*** = (***real*** *x, y*);

and you could replace

*[1:n]****real*** *a1, [1:n]****real*** *a2;* by *[1:n]****real*** *a1, a2;* .

Effectively, these extensions were defining both syntax and semantics at the same time. So it turned out that the first example also allowed you to produce, by extension,

***struct*** ***s*** = (***int*** *a, b*), ***t*** = ***real;***

which is nonsense (this is a syntactic problem), and the second example mandates that, given *[1:n]****real*** *a1, a2;* it is quite in order for the side effects of *n* to happen twice (this is a semantic problem).

WG members had expressed doubts about the safety of this mechanism as far back as Zandvoort. We now found so many further shocks and surprises that our confidence in the mechanism entirely evaporated and we abandoned it, incorporating these features (where they were retained at all) into the main syntax. Extensions seemed like a good idea at the time, but they are only really appropriate for a language where the core really is small (say 30 percent), and even then some separate method of specifying the syntax should be used.

#### 2.5.4.2 *Transput*

Defining the built-in **operators** in the **standard-prelude** by means of **operation-declarations** turned out to be quite satisfactory, but doing the same thing for the transput routines was a grave mistake.

The underlying model of the transput system has already been described (2.3.9). To write this model and the ***proc***s using it in ALGOL 68 was a substantial programming task, running to 68 pages in [RR 10.3]. This was akin to writing an actual implementation, insofar as we had to add numerous

**declarations** and so forth with which to 'implement' the model. But it still was not (and was not intended as) an implementation that you could run (you were simply meant to read it in order to see what requirements were being defined), and care was taken to leave many things undefined.

The net result was that you could not see the forest (the model) for the trees (the details of its implementation), and it was difficult to show that it was free of bugs and inconsistencies (which it was not). Most of the problems we had in the maintenance phase (2.6.2.2) lay in sorting out the resultant mess. If I were doing this job over again, I would certainly define the meaning of the transput ***proc***s in a more axiomatic style—by providing the preconditions and postconditions on the state of the model that each was supposed to satisfy, for example.

### 2.5.5 Style

[R] was written in a very pedantic style. This was necessary because the more formal your document is seen to be, the more people will actually believe what you actually wrote, as opposed to what you intended. Van Wijngaarden's English followed a very set style, with punctilious punctuation (as may be seen from the examples already quoted). In the revision, we tried to adhere to the same high standards (I believe I learned to write a good pastiche of Van Wijngaarden's style), and we often spent much time discussing what was and what was not correct English. For example, we had great arguments as to the proper form of the negative subjunctive ("It is required that the value *be not* nil") and I remember a full 15 minutes devoted to whether *well-formedness* should be hyphenated.

The successive drafts of [R] gradually established the style. In [MR 76] the metanotions consisted of just one upper-case letter. (Hence there were just 26 of them.) There was no explanatory material and no examples, and that document really was unreadable (in addition to being buggy), so that I find it hard to imagine why WG 2.1 agreed to go with it. The meeting at Kootwijk agreed that the metanotions should be English words with suggestive meanings. By [MR 88] 'pragmatic remarks' (comments enclosed within {. . .}) had appeared and by [MR 101] had been elevated to an art form. The choice of words used for metanotions and nonterminals was initially too arbitrary. (One can recognize the overzealous use of Roget's Thesaurus in the sequence **'adapted'**, **'adjusted'**, **'fitted'**, **'peeled'** which preceded the now familiar **'strong'**, **'firm'**, **'weak'**, **'soft'**.) It took much pressure from the WG, in Zandvoort and Tirrenia, to persuade Van Wijngaarden to include more motivations, more cross references, different fonts for syntactic objects, and so on. In the revision, we tried to take this policy even further. The additions to the syntactic tools enabled us to use fewer nonterminals, and we systematically introduced each section with pragmatics that set out the goods to be described (indeed, you can almost discover the language from the pragmatics alone).

If a reader approaches some piece of mathematical formalism with the wrong preconceived idea of what is to be presented, he can waste hours before his internal model becomes corrected. I therefore regard the principle purpose of pragmatics to be to "preload" the reader with the appropriate model. This of course introduces redundancy into the document, but this is no bad thing (as seems to have been accepted when this came up at Tirrenia), for if the formal and pragmatic descriptions are found, after close scrutiny, to disagree, this may well alert the reader to the presence of a bug in the formalism; better to recognize this (and to seek clarification from the proper authorities) than to implement the bug.

I therefore offer the following guidelines, based on our experience.

- Pragmatic remarks should be copious, but well delineated from the formal text. Their purpose is to educate the reader (in the sense of the Latin *educare*).

- Motivation should be given for *why* things are as they are. Redundancy is no bad thing, except within the strictly formal parts.
- Syntax should be fully cross referenced, both forwards and backwards; semantics likewise.
- Each syntax rule should be accompanied by examples.
- There should be copious glossaries, indices, and summaries.
- The Report should not take itself too seriously. With the best will in the world, it is not going to be perfect. Our Report contains some good jokes, many quotations from Shakespeare, Lewis Carroll, A. A. Milne and others, and even a picture of Eeyore. The various translations of it have been made in the same spirit.
- Above all, the Report should be fun to write.

Unfortunately, Standards Bodies and Government Departments do not always encourage these practices in their guidelines. I invite the reader to consider the extent to which they have been followed, or otherwise, in more recent programming language definitions, together with the perceived success of those definitions. On that basis, I will rest my case.

## 2.6 THE MAINTENANCE PHASE

After 1974, WG 2.1 proceeded to other things and left most ALGOL 68 activity to its Support Subcommittee. This considered enhancements to the language, together with the various bugs and problems that were reported. It met whenever there was a meeting of the full WG, but its main activities took place in independent meetings, as listed in Table 2.5.

**TABLE 2.5**

Meetings of the Support Subcommittee.

| Date | Meeting place | Meeting | Topics discussed |
|---|---|---|---|
| Apr 1974 | Cambridge | SC | Transput (2.4.4.1); Sublanguage. |
| Aug 1974 | Breukelen | WG, SC | |
| Jan 1975 | Boston | SC | Modules; Standard Hardware Representation; Sublanguage; Partial Parametrization. |
| Aug 1975 | Munich | SC, WG | TUM-10 mechanism; acceptance of Sublanguage, Standard Hardware Representation, and Partial Parametrization. |
| Sep 1976 | St. Pierre de Chartreuse | WG, SC | Sublanguage still being polished. |
| Aug 1977 | Kingston | SC | Modules; Bugs and problems; TP task force formed. |
| Dec 1977 | Oxford | TP, SC, WG | Implementation Model; Transput problems; 1st Commentaries released. |
| Aug 1978 | Amsterdam | TP | Transput problems; Implementation Model. |
| Aug 1978 | Jablonna | WG, SC | Acceptance of Modules and TORRIX; 2nd Commentaries released. |
| Dec 1978 | Cambridge | TP | Transput problems; Implementation model. |
| Apr 1979 | Summit | TP, SC, WG | Acceptance of Test Set and Implementation Model; 3rd Commentaries released. |

WG = full Working Group; SC = Support Subcommittee; TP = Transput task force.

### 2.6.1 Enhancements

#### 2.6.1.1 *The Sublanguage*

Although a Subcommittee on Sublanguages had been formed at Manchester, it never produced anything concrete, and it was an independent effort by Peter Hibbard, implementing on a minicomputer with only 16K words at Liverpool, which eventually bore fruit. Hibbard's sublanguage, first introduced to the WG at Vienna, was intended for mainly numerical applications, and the features omitted from it were intended to simplify compilation. It was discussed by the WG and the Support Subcommittee on several occasions, and finally formulated as an addendum to the Report and released for publication as an IFIP Document at Munich. However, the final polishing lasted for some time until it eventually appeared as [Hibbard 1977]. A more informal description will be found in Appendix 4 of [Lindsey 1977].

#### 2.6.1.2 *The Standard Hardware Representation*

ALGOL 68 was conceived in the days when every computer had its own character code. Even with the advent of ASCII things were not entirely settled, since many printers still did not support lower case letters. The conclusion reached, and sold to the Support Subcommittee by Wilfred Hansen and Hendrik Boom, was that one could easily transliterate between two character sets, provided that implementations restricted themselves to a set of 60 'worthy characters', each representable by a single character in each set. This still left the problem of 'stropping' (2.1.3) and the solution adopted was for *POINT* stropping, which would always be enabled (as in *.REAL X*) with a choice of *UPPER* stropping (as in *REAL x*) or *RES*erved stropping (as in *REAL X*) under control of a **pragmat**.

The document formalizing this [Hansen 1977] was also released for publication as an IFIP Document at Munich. For an informal treatment, see Appendix 5 of [Lindsey 1977].

#### 2.6.1.3 *TUM-10*

The hassle of getting a document approved through the IFIP hierarchy was considered too great for every small enhancement to the language, and a simpler mechanism was agreed on at Munich. The Support Subcommittee was to scrutinize proposals to ensure that they were consistent, upwards-compatible, and useful, and release them "to encourage implementors experimenting with features similar to those described ... to use the formulation here given, so as to avoid proliferation of dialects," and the full WG would then authorize publication in the *ALGOL Bulletin* with this wording. This principle was enshrined in a document numbered TUM-10 [WG 2.1 1975].

#### 2.6.1.4 *Partial Parametrization*

After the failure of the Bekic proposal to relax the extent restriction on **proc**s (2.3.5), it was agreed that this problem ought to be solved by partial parametrization. At Boston the Support Subcommittee appointed a task force (Bekic, Foster, Hibbard, Meertens, and myself) which brought forward a proposal to incorporate this [Lindsey 1976], and it was adopted under the newly invented TUM-10 mechanism at Munich.

Here is an example:

```
proc compose = (proc(real)real f, g, real x) real:
                    f(g(x));
proc(real)real sqex = compose(sqrt, exp,   );
```

#### 2.6.1.5 *Modules and Separate Compilation*

The basic idea for program encapsulation using modules was first presented (so far as I am aware) by Steve Schuman at the Fontainebleau meeting of WG 2.1 (see [Schuman 1974] for a fuller account). The inclusion of such a facility within ALGOL 68 was discussed by the Support Subcommittee on various occasions, and finally agreed under the TUM-10 mechanism at Jablonna [Lindsey 1978]. The people involved at various times, in addition to myself and Hendrik Boom who wrote the final document, included Steve Bourne, Robert Dewar, Mike Guy, John Peck, and Steve Schuman.

Here is an example:

```
module stack =
    def  int stacksize = 100;
        loc [1:stacksize] int st, loc int stptr := 0;
        pub proc push = (int n)int:
            ((stptr+:=1)<=stacksize | st[stptr] := n | print("stack overflow"); stop);
        pub proc pop = int:
            (stptr>0 | st[(stptr-:=1)+1] | print("stack underflow"); stop);
    postlude
        (stptr/=0 | print("stack not emptied"); stop)
    fed;
...
access stack (push(1); push(2); print(pop); pop)
```

There was provision for separate compilation of **module**s at the global level, but there was also a second mechanism for separately compiling an **egg** in an environment known as a **nest**, which could be declared at any level within a **program**. It is interesting to note that Ada also has two separate compilation mechanisms, of a similar sort.

#### 2.6.1.6 *TORRIX*

TORRIX was an ALGOL 68 library for handling vectors and matrices, developed at the University of Utrecht [van der Meulen 1978]. At its Jablonna meeting WG 2.1 commended its use, using a variation of the TUM-10 wording.

#### 2.6.1.7 *The MC Test Set*

This was a suite of 190 ALGOL 68 programs, developed at the Mathematisch Centrum [Grune 1979], and designed to establish confidence that an ALGOL 68 compiler was correct with respect to the Report. It therefore contained many "pathological" examples as well as some more straightforward programs. The Summit meeting of WG 2.1 authorized a statement of commendation to be attached to it.

### 2.6.2 Problems

It was not long before people began to report problems with [RR]. The policy for dealing with them was hammered out by the Support Subcommittee at Kingston and Oxford, the principles being that

- Clerical errors and misprints in [RR] could be corrected by publication of errata.
- [RR] would not be changed in any substantive fashion.

- Commentaries would be published stating the opinion of the Support Subcommittee on problems that had been raised, but they were "not to be construed as modifications to the text of the Revised Report." Their definitive forms can be found in [WG 2.1 1978] and [WG 2.1 1979].

#### 2.6.2.1 *Language Problems*

Only three of the commentaries refer to the main body of the Report, that is, to the syntax and the semantics, and one of these is arguably discussing a non-problem. There are also a couple of problems with [RR 10.2] (the non-transput part of the **standard-prelude**). Allowing a margin for these uncertainties and for some problems that might be counted as two, it is still safe to say that the number of known problems in the main part of the Report can be counted on the fingers of one hand.

#### 2.6.2.2 *Transput Problems*

Unfortunately, the same cannot be said for [RR 10.3], the transput part of the **standard-prelude**. A continuous stream of problems was reported, notably by Hans van Vliet who had been trying to construct a machine-independent implementation model of the transput. At Kingston a task force, convened by Chris Cheney, was set up with a brief to review [RR 10.3] and to provide a reasonable interpretation for it.

Over a succession of meetings the task force decided to adopt Van Vliet's implementation model, adjusting both it and the Report to ensure that they were consistent (of course, proving the equivalence of two programs is not easy, but a very intensive study of [RR 10.3] had now been taking place, resulting in some confidence that we at last understood it, warts and all). Commentaries were prepared to establish the "approved" understanding of the Report, and the final version of Van Vliet's model [Van Vliet 1979] was released with a TUM-10-like wording.

## 2.7 IMPLEMENTATIONS

Shortly after publication of the ALGOL 60 Report there were implementations on a large variety of machines, written in universities and research establishments and by machine manufacturers. There was a natural expectation that the same thing would happen with ALGOL 68 and, for example, it was reported at Novosibirsk that 20 implementations were underway on 15 different machines. The sad fact is that the great majority of implementations that were started were never completed.

### 2.7.1 The Early Implementations

The implementations by Goos at Munich on a Telefunken TR4, and by Branquart at MBLE on an Electrologica-X8 were well under way by the appearance of [R]. Moreover, Mailloux had had a particular responsibility among the authors for ensuring that the language, as it was being defined, was implementable.

In his thesis, Mailloux [1968] considered examples such as the following:

```
begin real x;
    proc p = void:
        begin proc q = begin somelongexpression + x; ... end;
            abc x;
            ...
        end;
```

```
#either#    mode abc = ... ;
#or#        op abc = ... ;
```

When ***abc*** *x* is encountered, we do not yet know whether it declares a new **variable** *x* of an as-yet-undeclared mode ***abc***, or whether it is an application of an as-yet-undeclared **operator** ***abc*** to the previously declared x. So a first pass of the compiler is needed to detect all declarations of ***mode***s and ***op***s before it can be known what **identifiers** have been declared. But now, supposing the first pass shows ***abc*** to be a mode, we cannot tell on the second pass whether the *x* within ***proc*** *q* is of mode ***real*** or of mode ***abc***. Therefore, we cannot tell which overloaded version of the **operator** + is meant in ***proc*** *q* until the third pass, by which time it is too late to compile code to coerce *somelongexpression*. Hence a fourth pass is required to generate the code. Thus Mailloux established the classical 4-pass ALGOL 68 compiler, and they all said how clever he was and gave him his degree. What they ought to have said was how clever he was to have discovered a flaw in the language that could easily be fixed by a small syntactic change, such as a compulsory ***loc*** in front of each **variable-declaration** (2.3.4.5), thus saving an entire pass in every compiler.

But compile-time efficiency was not regarded as important although run-time efficiency was always a prime goal. As Van Wijngaarden said at North Berwick, ". . . yes, it does some extra work for the compiler but we should not be concerned with the efficiency of compiling. You do not lose any run-time efficiency, and that is what we are most concerned with." Strangely, Goos, who was well into writing his compiler at that time, seemed quite content with this philosophy. Goos' compiler in fact had six passes.

Branquart's compiler was a piece of research into compiler methodology rather than a serious production tool, but it did establish many features necessary to all compilers, particularly the storage of values and the layout of the stack, all documented in a series of MBLE Reports that were finally collated into [Branquart 1976]. The project consumed 20 man-years of effort, spread over the years 1967 to 1973.

Being implementations of the original language, neither of these compilers ever came into use outside their home bases. The Malvern ALGOL 68R compiler, on the other hand, restricted the language so as to permit 1-pass compilation and also made numerous small language changes, for convenience of implementation. Some, but not all, of these changes were subsequently blessed by inclusion within [RR]. Thus [R], [RR], and ALGOL 68R could be regarded as three vertices of an equilateral triangle. It was available from 1970 onwards on ICL 1900 series machines, and became the most widely used implementation, especially in Great Britain.

A TC2-sponsored conference on ALGOL 68 implementation was held in Munich in July 1970, and the proceedings [Peck 1971] contain papers on all the three compilers mentioned, together with several papers discussing garbage collection, showing this problem not to be as intractable as had been feared.

### 2.7.2 Implementations of the Revised Language

A full list of compilers available at the time can be found in [AB 52.3.1]. It is now apparent that implementing full ALGOL 68 requires an effort beyond what a university department can usually provide, which is why so many attempts failed. But by making some quite modest restrictions to the language the effort is reduced quite significantly, as Malvern showed, and such restrictions or omissions hardly affect the functionality or the orthogonality of the language. We therefore need to distinguish between full commercial implementations and small partial ones.

#### 2.7.2.1 *Full Implementations*

The most successful commercial implementation was by CDC Netherlands, first delivered in 1977 in response to a threat from several Dutch universities to buy only machines with ALGOL 68 available. It was an excellent compiler, but the parent company in the USA never showed any interest in it.

Next, Malvern produced their second attempt, ALGOL 68RS, for their own in-house machines and available from 1977. This was much closer to [RR] than their first product, it was written to be machine independent, and it has been ported to the ICL 2900 series, to MULTICS and to VMS Vaxen. It is still the best starting point, should anyone wish a new (almost) full implementation.

The final commercial-strength product, called FLACC and first shipped in 1978, was a checkout compiler for IBM machines. This was produced by two of Mailloux's ex-students who set up their own company. Again, this was an excellent product (not fast, being a checkout system), but it did not spread far because they completely misjudged the price the market would pay.

#### 2.7.2.2 *Partial Implementations*

Hibbard's Liverpool implementation, around which his sublanguage (2.6.1.1) was designed, was rewritten in BLISS for a PDP-11 when he moved to Carnegie Mellon, and was rewritten again by myself in Pascal and now runs on various machines. On the way, it acquired **heap-generators** (not in the original sublanguage), giving it nearly the functionality of the full language.

ALGOL 68C was a portable implementation originated by Steve Bourne and Mike Guy at Cambridge in 1975, and now available on IBM machines, DEC-20s (anyone still have one?), VMS Vaxen, Primes, Telefunken TR440, and even a CYBER 205. Its development continued for a while, but it was never completed (for example, it still has no garbage collector, although one was "almost working" at one stage). It was quite widely used, especially on IBM machines where it is much faster than FLACC.

ALGOL 68LGU is a system for IBM 360 clones, written by Tseytin and his colleagues at Leningrad State University (as it then was). As with other university products, it lacked features such as garbage collection, but it has now been adopted by Krasnaya Zarya, a "commercial" company in Leningrad, which has produced various products around it, including cross compilers and a portable version. Indeed, the former Soviet Union is the only place where there exists an official Standard for ALGOL 68 [GOST 27974/9-88].

Finally, an interactive implementation by Peter Craven of Algol Applications Ltd, originally developed for MS-DOS systems, is now available in the form of a public-domain interpreter, written in C.

## 2.8 CONCLUSION

### 2.8.1 Whatever Happened to ALGOL 68?

Well, one could say that it is alive and well and living in Pascal, or C, or C++, or SML, or Ada; and indeed this is true in part for all of these languages (see 2.3.1, 2.3.4.3 and 2.3.6 for some specific mentions).

The real question is why it did not come into more widespread use, and the answer here is simple enough: because it was not implemented widely enough, or soon enough. And the reason for that is that implementation was too hard, and the reason for that was on account of a relatively small number

of troublespots, not inherently necessary for the basic principles of the language and certainly not for its orthogonality. It was possible only to correct a few of these problems during the revision.

But it *was* in fact used in many places (and still is in some), especially in Great Britain where the ICL machines were popular. Universally, those who had once used it never wanted to use anything else, and would gleefully point out the shortcomings of any other language you might care to name. It was the orthogonality that they liked (and the troublespots were in such obscure corners that no-one beyond the implementors ever noticed them).

The most successful feature was the strong typing. It was a not at all uncommon experience for a program of significant size to work first time, once it had got past the compiler. Thus it was in actual use a very safe language. Another successful concept was orthogonality. No subsequent language has achieved it to quite the same degree, but it is now firmly fixed in every language designer's agenda.

Some features, strangely, have not made it into more recent languages. Among these are truly unlimited ***string***s (2.3.1), **slices** (2.3.2), and the transput model (2.3.9), none of which is particularly difficult to implement.

It was often used as a first teaching language. Students usually demand to be taught the language that they are most likely to use in the world outside (FORTRAN or C). This is a mistake. A well-taught student (that is, one who has been taught a *clean* language) can easily pick up the languages of the world, and he or she will be in a far better position to recognize their bad features as he or she encounters them. It still has a role in teaching because of the concepts that it can illustrate. It is still a widely talked-about language (even among those who have never used it) and no respectable book on Comparative Programming Languages can afford to ignore it.

Has it a future? The honest answer must now be "No." The world has moved on. To be accepted now, your language needs modules, exception handling, polymorphism, safe parallelism, and clean interfaces to everything else (not to mention a rich and powerful sponsor). Moreover, you need all these things *from the start*—this is certainly one place where Ada got it right.

Was it TOO BIG? It has often been so accused, but it is really quite a small language compared with PL/1, or Ada. What, for example, does it provide beyond Pascal? Dynamic ***array***s, **slices**, **formats**, overloading, full block-structure, and expression-orientedness. Put these things into Pascal, make it orthogonal where it is not, and you have ALGOL 68.

### 2.8.2 Whatever Have We Learned from ALGOL 68?

The chief legacy of ALGOL 68 must be what the world has learned from it. I, personally, have learned a lot, and a major purpose of writing this paper was in order that others might do the same, bearing in mind that the chief lesson of history is that we do not learn from history.

So, first I know more about the dynamics and instabilities (but also the benefits) of large committees, and against that I know what can be achieved by a small band of people dedicated to working towards a common aim (2.4.5).

Next, there is the requirement for absolute *rigour* (which is not the same as formality) in the description of programming languages. I claim that we have shown that it can be done, but examples of *real* programming languages whose official definitions match this ideal are few and far between, and I know of no others where the rigorous definition was not a retrofit. What we achieved we achieved by consciously applying principles of good program design (2.4.3.3). We also recognized that rigour must be counterbalanced by adequate motivations and commentaries (2.5.5). And we must also warn of the considerable price to be paid in time and effort in order to do the job properly.

Was the WG wrong to attempt too much innovation in what was intended as a language for widespread use? Possibly so. It may be that experimental features should be tried out in *small*

experimental languages. Instead, we found ourselves with a *large* experimental language on our hands. That might have been avoided, but only at the expense of more time.

Whether the ultimate benefit was worth all the effort and argumentation and heartbreak that went into it is another matter—and whether a good product can ever be made at all without some degree of heartbreak is also debatable.

## ACKNOWLEDGMENTS

I would like to acknowledge the help of several people who have read earlier versions of this paper and offered constructive comments, notably Tony Hoare, Kees Koster, John Peck, Brian Randell, Michel Sintzoff, Peter Lucas, and Henry Bowlden. I should also like to thank Wlad Turski for the excellent and almost verbatim minutes of WG meetings which he took down in longhand, and without which my task would have been impossible.

## REFERENCES

The *ALGOL Bulletin*, in which many of the following references appeared, was the official publication of WG 2.1. It is kept in a few academic libraries but, as the former Editor, I still have copies of most back numbers since AB 32, and could also probably arrange for photocopying of earlier issues.

[AB 28.1.1] News item—Tenth anniversary colloquium, Zürich, May 1968, *ALGOL Bulletin* AB28.1.1, Jul. 1968.

[AB 31.1.1] News item—Minority report, *ALGOL Bulletin* AB31.1.1, Mar. 1970.

[AB 52.3.1] Survey of viable ALGOL 68 implementations, *ALGOL Bulletin* AB52.3.1, Aug. 1988.

[Baecker, 1968] Baecker, H. D., ASERC—a code for ALGOL 68 basic tokens, *ALGOL Bulletin* AB28.3.5, Jul. 1968.

[Boom, 1972] Boom, H. J., IFIP WG 2.1 Working Paper 217 (Vienna 4), Sep. 1972.

[Branquart, 1976] Branquart, P., Cardinael, J.-P., Lewi, J., Delescaille, J.-P., and Vanbegin, M., *An optimized translation process and its application to ALGOL 68*, LNCS 38. New York: Springer-Verlag, 1976.

[Chastellier, 1969] De Chastellier, G., and Colmerauer, A., W-Grammar, *Proc. 24th National Conference*. New York: ACM Publication P-69, 1969.

[Colmerauer, 1996] Colmerauer A., and Roussel, P., The birth of Prolog, in these Proceedings.

[Dahl, 1966] Dahl, O-J, A plea for multiprogramming, *ALGOL Bulletin* AB24.3.5, Sep. 1966.

[Dahl, 1968] Dahl, O-J, Myhrhaug, B., and Nygaard, K., *The Simula 67 Common Base Language*. Oslo: Norwegian Computing Centre, Oslo, 1968.

[De Morgan, 1976a] De Morgan, R. M., Hill, I. D., and Wichman, B. A., A supplement to the ALGOL 60 revised report, *Comp. Jour.* 19:3, Aug. 1976, pp. 276–288; also *SIGPLAN Notices* 12:1, January 1977, pp. 52–66.

[De Morgan, 1976b] De Morgan, R. M., Modified report on the algorithmic language ALGOL 60, *Comp. Jour.* 19:4, Nov. 1976, pp. 364–379 ([Naur *et al.* 1962] as modified by [De Morgan 1976]).

[De Morgan, 1978] De Morgan, R. M., Hill, I. D., and Wichman, B., A. Modified ALGOL 60 and the step-until element, *Comp. Jour.* 21:3, Aug. 1978, p. 282 (essential errata to [De Morgan, 1976a] and Modified report [1976b]).

[Dijkstra, 1968a] Dijkstra, E. W., Cooperating sequential processes, In *Programming Languages*, F. Genuys, Ed., New York: Academic Press, 1968.

[Dijkstra, 1968b] Dijkstra, E. W., Goto considered harmful, letter to the Editor, *Comm. ACM* 11:3, Mar. 1968.

[Duncan, 1964] Duncan, F. G., and van Wijngaarden, A., Cleaning up ALGOL 60, *ALGOL Bulletin* AB16.3.3, May 1964.

[Garwick, 1965] Garwick, J. V., The question of I/O procedures, *ALGOL Bulletin* AB19.3.8, Jan. 1965.

[GOST 27974/9-88] *Programming language ALGOL 68 and ALGOL 68 extended*, GOST 27974-88 and GOST 27975-88, USSR State Committee for Standards, Moscow, 1989.

[Grune, 1979] Grune, D., *The Revised MC ALGOL 68 test set*, IW 122/79, Amsterdam: Mathematisch Centrum, 1979.

[Hansen, 1977] Hansen, Wilfred J., and Boom, Hendrik, The report on the standard hardware representation for ALGOL 68, *SIGPLAN Notices* 12:5, May, 1977; also *Acta Informatica* 9, 1978, pp. 105–119.

[Hibbard, 1977] Hibbard, P. G., A sublanguage of ALGOL 68, *SIGPLAN Notices* 12:5, May 1977.

[Hoare, 1964] Hoare, C. A. R., Case expressions, *ALGOL Bulletin* AB18.3.7, Oct. 1964.

[Hoare, 1965a] Hoare, C. A. R., Cleaning up the for statement, *ALGOL Bulletin* AB21.3.4, Nov. 1965.

[Hoare, 1965b] Hoare, C. A. R., Record Handling, *ALGOL Bulletin* AB21.3.6, Nov. 1965.

[Hoare, 1966] Hoare, C. A. R., Further thoughts on record handling AB21.3.6, *ALGOL Bulletin* AB23.3.2, May 1966.

[Hoare, 1968] Hoare, C. A. R., Critique of ALGOL 68, *ALGOL Bulletin* AB29.3.4, Nov. 1968.

[Hoare, 1981] Hoare, C. A. R., The emperor's old clothes (the 1980 ACM Turing award lecture), *Comm. ACM* 24:2, Feb. 1981, pp. 75–83.

[King, 1974] King, P. R., WG 2.1 subcommittee on ALGOL 68 support, *ALGOL Bulletin* AB37.3.1, Jul. 1974.

[Knuth *et al.*, 1964] Knuth D. (Chairman), Bumgarner, L. L., Hamilton, D. E., Ingerman, P. Z., Lietzke, M. P., Merner, J. N., and Ross, D. T., A proposal for input-output conventions in ALGOL 60, *Comm. ACM* 7:5, May 1964, pp. 273–283.

[Koster, 1969] Koster, C. H. A., On infinite modes, *ALGOL Bulletin* AB30.3.3, Feb. 1969.

[Koster, 1971] Koster, C. H. A., Affix-grammars, in *ALGOL 68 Implementation*, J. E. L. Peck, Ed., North Holland, 1971, pp. 95–109.

[Lindsey, 1968] Lindsey, C. H., ALGOL 68 with fewer tears, *ALGOL Bulletin* AB28.3.1, Jul. 1968.

[Lindsey, 1971] Lindsey, C. H., and Van der Meulen, S. G., *Informal Introduction to ALGOL 68*, North Holland, 1971.

[Lindsey, 1972] Lindsey, C. H., ALGOL 68 with fewer tears, *Comp. Jour.* 15:2, May 1972.

[Lindsey, 1976] Lindsey, C. H., Specification of partial parametrization proposal, *ALGOL Bulletin* AB39.3.1, Feb. 1976.

[Lindsey, 1977] Lindsey, C. H., and Van der Meulen, S. G., *Informal Introduction to ALGOL 68* Revised Edition, North Holland, 1977.

[Lindsey, 1978] Lindsey, C. H., and Boom, H. J., A modules and separate compilation facility for ALGOL 68, *ALGOL Bulletin* AB43.3.2, Dec. 1978; also IW 105/78, Mathematisch Centrum, Amsterdam, 1978.

[Löf, 1984] Martin-Löf, P., Constructive mathematics and computer programming, in *Mathematical logic and programming languages*, Hoare, C. A. R., and Shepherdson, J. C., Eds. New York: Prentice-Hall, 1985.

[Lucas, 1969] Lucas, P., and Walk, K., On the formal description of PL/1, in *Annual review in automatic programming* 6:3, Pergammon, 1969, pp. 105–182.

[Mailloux, 1968] Mailloux, B. J., *On the implementation of ALGOL 68*, Mathematisch Centrum, Amsterdam, 1968.

[McCarthy, 1964] McCarthy, J., Definition of new data types in ALGOL X, *ALGOL Bulletin* AB18.3.12, Oct. 1964.

[Meertens, 1969] Meertens, L. G. L. T., On the generation of ALGOL 68 programs involving infinite modes, *ALGOL Bulletin* AB30.3.4, Feb. 1969.

[Merner, 1966] Merner, J. M., Garwick, J. V., Ingerman, P. Z., and Paul, M., Report of the ALGOL X I-O subcommittee, IFIP WG 2.1 Working Paper 48 (Warsaw 3), July 1966.

[Milner, 1990] Milner, R., Tofte, M., and Harper, R., *The definition of standard ML*, Cambridge, MA: MIT Press, 1990.

[MR 76] Van Wijngaarden, A., Orthogonal design and description of a formal language, *MR 76*, Mathematisch Centrum, Amsterdam, Oct. 1965.

[W-2] Van Wijngaarden, A., and Mailloux, B. J., A draft proposal for the algorithmic language ALGOL X, IFIP WG 2.1 Working Paper 47 (Warsaw 2), Oct. 1966.

[MR 88] Van Wijngaarden, A., Mailloux, B. J., and Peck, J. E. L., A draft proposal for the algorithmic language ALGOL 67, *MR 88*, Mathematisch Centrum, Amsterdam, May 1967.

[MR 93] Van Wijngaarden, A., Ed., Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., Draft report on the algorithmic language ALGOL 68, *MR 93*, Mathematisch Centrum, Amsterdam, Jan. 1968.

[MR 95] Van Wijngaarden, A., Ed., Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., Working document on the algorithmic language ALGOL 68, *MR 95*, Mathematisch Centrum, Amsterdam, Jul. 1968.

[MR 99] Van Wijngaarden, A., Ed., Mailloux, B. J., Peck,J. E. L., and Koster, C. H. A., Penultimate draft report on the algorithmic language ALGOL 68, *MR 99*, Mathematisch Centrum, Amsterdam, Oct. 1968.

[MR 100] Van Wijngaarden, A., Ed., Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., Final draft report on the algorithmic language ALGOL 68, *MR 100*, Mathematisch Centrum, Amsterdam, Dec. 1968.

[MR 101] The first printing of [Van Wijngaarden 1969].

[Naur *et al.*, 1960] Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P. Ed., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Van Wijngaarden, A., and Woodger, M., Report on the algorithmic language ALGOL 60, *Numerische Mathematik* 2: 1960, pp. 106–136; also *Comm. ACM* 3:5, May 1960, pp. 299–314.

[Naur *et al.*, 1962] Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P. Ed., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Van Wijngaarden, A., and Woodger, M., Revised report on the algorithmic language ALGOL 60, *Numerische Mathematik* 4: 1963, pp. 420–453; also *Comm. ACM* 6:1, Jan. 1963, pp. 1–17; also *Comp. Jour.*, 5:1, Jan. 1963, pp. 349–367.

[Naur, 1964] Naur, P., Proposals for a new language, *ALGOL Bulletin* AB18.3.9, Oct. 1964.

[Naur, 1966] ———, The form of specifications, *ALGOL Bulletin* AB22.3.7, Feb. 1966

[Naur, 1968] ———, Successes and failures of the ALGOL effort, *ALGOL Bulletin* AB28.3.3, Jul. 1968.

[Naur, 1981] ———, The European side of the last phase of the development of ALGOL 60, in *History of Programming Languages*, Richard L. Wexelblat, Ed. New York: Academic Press, 1981.

[Pair, 1970] Pair, C., Concerning the syntax of ALGOL 68, *ALGOL Bulletin* AB31.3.2, Mar. 1970.

[Peck, 1971] Peck J. E. L., ed., *ALGOL 68 Implementation*, North Holland, 1971.

[R] See [Van Wijngaarden 1969].

[RR] See [Van Wijngaarden 1975].

[Ritchie, 1993] Ritchie, D. M., The development of the C language, in these Proceedings.

[Ross, 1961] Ross, D. T., A generalized technique for symbol manipulation and numerical calculation, *Comm. ACM*, 4:3, Mar. 1961, pp. 147–50.

[Ross, 1969] Ross, D. T., Concerning a minority report on ALGOL 68, *ALGOL Bulletin* AB30.2.3, Feb. 1969.

[Samelson, 1965] Samelson, K., Functionals and functional transformations, *ALGOL Bulletin* AB20.3.3, Jul. 1965.

[Seegmüller, 1965a] Seegmüller, G., Some proposals for ALGOL X, *ALGOL Bulletin* AB21.3.1, Nov. 1965.

[Seegmüller, 1965b] Seegmüller, G., A proposal for a basis for a report on a successor to ALGOL 60, Bavarian Acad. Sci., Munich, Oct. 1965.

[Sintzoff, 1967] Sintzoff, M., Existence of a Van Wijngaarden syntax for every recursively enumerable set, *Annales Soc. Sci. Bruxelles*, II, 1967, pp. 115–118.

[Schuman, 1974] Schuman, S. A., Toward modular programming in high-level languages, *ALGOL Bulletin* AB37.4.1, Jul. 1974.

[Stroustrup, 1996] Stroustrup, B., A history of C++, in these Proceedings.

[TR 74-3] The first printing of [Van Wijngaarden 1975], published as Technical Report TR74-3, Dept. of Computing Science, University of Alberta, Mar. 1974; subject to errata published in *ALGOL Bulletin* AB37.5 Jul. 1974, AB38.5.1 Dec. 1974, and AB39.5.1 Feb. 1976.

[Turski, 1968] Turski, W. M., Some remarks on a chapter from a document, *ALGOL Bulletin* AB29.2.4, Nov. 1968.

[Turski, 1981] Turski, W. M., ALGOL 68 revisited twelve years later or from AAD to ADA, in *Algorithmic Languages*, J. W. de Bakker and J. C. van Vliet, Eds., North Holland, 1981.

[Van der Meulen, 1978] Van der Meulen, S. G. and Veldhorst, M., *TORRIX—a programming system for operations on vectors and matrices over arbitrary fields and of variable size Vol. 1*, Mathematical Centre Tracts 86, Mathematisch Centrum, Amsterdam, 1978.

[Van der Poel, 1965] Van der Poel, W. L., extract from WG 2.1 Activity Report, *ALGOL Bulletin* AB21.1.1.

[Van der Poel, 1986] Van der Poel, W. L., Some notes on the history of ALGOL, in *A quarter century of IFIP*, H. Zemanek, Ed., North Holland, 1986.

[Van Vliet, 1979] Van Vliet, J. C., *ALGOL 68 transput, Pt 1: Historical review and discussion of the implementation model, Pt 2: An implementation model*, Mathematical Centre Tracts 110 and 111, Mathematisch Centrum, 1979.

[Van Wijngaarden, 1969] Van Wijngaarden, A., Ed., Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., Report on the algorithmic language ALGOL 68, *Numerische Mathematik* 14: 1969, pp. 79–218; also A. P. Ershov and A. Bahrs, transl., *Kybernetica* 6, 1969, and 7, 1970, bilingual; also I. O. Kerner (transl.), *Bericht über die algorithmische sprache ALGOL 68*, Akademie-Verlag, Berlin, 1972, bilingual; also J. Buffet, P. Arnal and A. Quéré (transl.), *Définition du langage algorithmique ALGOL 68*, Hermann, Paris, 1972; also Lu Ru Qian (transl.), 算法语言 *ALGOL 68* 报告, Beijing, Science Press, 1977.

[Van Wijngaarden, 1975] Van Wijngaarden, A., Mailloux, B. J., Peck, J. E. L., Koster, C. H. A., Sintzoff, M., Lindsey, C. H., Meertens, L. G. L. T., and Fisker, R. G., Revised report on the algorithmic language ALGOL 68, *Acta Informatica* 5:1–3, 1975; also *Mathematical Centre Tract 50*, Mathematisch Centrum, Amsterdam; also *SIGPLAN Notices* 12:5, May 1977; also I. O. Kerner, (transl.), *Revidierter bericht über die algorithmische sprache ALGOL 68*, Akademie-Verlag, Berlin, 1978; also A. A. Bahrs (transl.) and A. P. Ershov (Ed.), *Peresmotrenoye So'obszczeniye ob ALGOLE 68*, Izdatelstvo "MIR," Moscow, 1979; also Lu Ru Qian (transl.), 算法语言 *ALGOL 68* 修改报告, Beijing, Science Press, Aug. 1982.

[W-2] Van Wijngaarden, A., and Mailloux, B. J., A draft proposal for the algorithmic language ALGOL X, IFIP WG 2.1 Working Paper 47 (Warsaw 2), Oct. 1966.

[WG 2.1, 1964a] Report on SUBSET ALGOL 60 (IFIP), *Numerische Mathematik* 6: (1964), pp. 454–458; also *Comm. ACM* 7:10, Oct. 1964, p, 626.

[WG 2.1, 1964b] Report on input-output procedures for ALGOL 60, *Numerische Mathematik* 6: pp. 459–462; also *Comm. ACM* 7:10, Oct. 1964, p 628.

[WG 2.1, 1971a] Report of the subcommittee on data processing and transput, *ALGOL Bulletin* AB32.3.3, May 1971.

[WG 2.1, 1971b] Report of the subcommittee on maintenance and improvements to ALGOL 68, *ALGOL Bulletin* AB32.3.4, May 1971.

[WG 2.1, 1971c] Letter concerning ALGOL 68 to the readers of the ALGOL Bulletin, *ALGOL Bulletin* AB32.2.8, May 1971.

[WG 2.1, 1972a] WG 2.1 formal resolution: Revised report on ALGOL 68, *ALGOL Bulletin* AB33.1.1, Mar. 1972.

[WG 2.1, 1972b] Report of the subcommittee on maintenance of and improvements to ALGOL 68, August 1971, *ALGOL Bulletin* AB33.3.3, Mar. 1972.

[WG 2.1, 1972c] Report of the subcommittee on data processing and transput, August 1971, *ALGOL Bulletin* AB33.3.4, Mar. 1972.

[WG 2.1, 1972d] Report on the meeting of working group 2.1 held at Fontainebleau, *ALGOL Bulletin* AB34.3.1, Jul. 1972.

[WG 2.1, 1972e] Report on considered improvements, *ALGOL Bulletin* AB34.3.2, Jul. 1972.
[WG 2.1, 1972f] Proposals for revision of the transput section of the report, *ALGOL Bulletin* AB34.3.3, Jul. 1972.
[WG 2.1, 1973a] Further report on improvements to ALGOL 68, *ALGOL Bulletin* AB35.3.1, Mar. 1973.
[WG 2.1, 1973b] Final report on improvements to ALGOL 68, *ALGOL Bulletin* AB36.3.1, Nov. 1973.
[WG 2.1, 1975] IFIP WG 2.1 Working Paper 287 (TUM 10), Munich, Aug. 1975.
[WG 2.1, 1978] Commentaries on the revised report, *ALGOL Bulletin* AB43.3.1, Dec. 1978.
[WG 2.1, 1979] Commentaries on the revised report, *ALGOL Bulletin* AB44.3.1, May 1979.
[Wirth, 1965] Wirth, N., A proposal for a report on a successor of ALGOL 60, *MR 75*, Mathematisch Centrum, Amsterdam, Oct. 1965.
[Wirth, 1966a] Wirth, N., and Weber, H., EULER: A generalization of ALGOL, and its formal definition: Part II, *Comm. ACM* 9:2, Feb. 1966, pp. 89–99.
[Wirth, 1966b] Wirth, N., and Hoare, C. A. R., A contribution to the development of ALGOL, *Comm. ACM* 9:6, Jun. 1966, pp. 413–431.
[Wirth, 1966c] Wirth, N., Additional notes on "A contribution to the development of ALGOL," *ALGOL Bulletin* AB24.3.3, Sep. 1966.
[Wirth, 1968] ALGOL colloquium—closing word, *ALGOL Bulletin* AB29.3.2, Nov. 1968.

## APPENDIX A: THE COVERING LETTER

Working Group 2.1 on ALGOL of the International Federation for Information Processing has been concerned for many years with the design of a common programming language and realises the magnitude and difficulty of this task. It has commissioned and guided the work of the four authors of this first Report on the Algorithmic Language ALGOL 68, and acknowledges the great effort which they have devoted to this task. The Report must be regarded as more than just the work of the four authors, for much of the content has been influenced by and has resulted from discussions in the Group. Consequently, this Report is submitted as the consolidated outcome of the work of the Group. This does not imply that every member of the Group, including the authors, agrees with every aspect of the undertaking. It is however the decision of the Group that, although there is a division of opinion amongst some of its members, the design has reached the stage to be submitted to the test of implementation and use by the computing community.

The Group intends to keep continuously under review the experience thus obtained, in order that it may institute such corrections and revisions to the Report as may become desirable. To this end, it requests that all who wish to contribute to this work should do so both via the medium of the ALGOL Bulletin, and by writing to the Editor directly.

## APPENDIX B: THE MINORITY REPORT

We regard the current Report on Algorithmic Language ALGOL 68 as the fruit of an effort to apply a methodology for language definition to a newly designed programming language. We regard the effort as an experiment and professional honesty compels us to state that in our considered opinion we judge the experiment to be a failure in both respects.

The failure of the description methodology is most readily demonstrated by the sheer size of the Report in which, as stated on many occasions by the authors, "every word and every symbol matters" and by the extreme difficulty of achieving correctness. The extensive new terminology and the typographical mannerisms are equally unmistakable symptoms of failure. We are aware of the tremendous amount of work that has gone into the production of the Report, but this confirms us in our opinion that adequacy is not the term that we should employ of its approach. We regard the high degree of inaccessibility of its contents as a warning that should not be ignored by dismissing the problems of "the uninitiated reader." That closer scrutiny has revealed grave deficiencies was only to be expected.

Now the language itself, which should be judged, among other things, as a language, in which to *compose* programs [*sic*]. Considered as such, a programming language implies a conception of the programmer's task. We recognize that over the last decade the processing power of commonly available machines has grown tremendously and that society has increased its ambition in their application in proportion to this growth. As a

result the programmer of today and tomorrow, finding his task in the field of tension between available equipment and desired applications, finds himself faced with tasks of completely different and still growing scope and scale. More than ever it will be required from an adequate programming tool that it assists, by structure, the programmer in the most difficult aspects of his job, that is, in the *reliable creation* of sophisticated programs. In this respect we fail to see how the language proposed here is a significant step forward: on the contrary, we feel that its implicit view of the programmer's task is very much the same as, say, ten years ago. This forces upon us the conclusion that, regarded as a programming tool, the language must be regarded as obsolete.

The present minority report has been written by us because if we had not done so, we would have forsaken our professional responsibility towards the computing community. We therefore propose that this Report on the Algorithmic Language ALGOL 68 should not be published under IFIP sponsorship. If it is so published, we recommend that this "minority report" be included in the publication.

*Signed by:*
Dijkstra, Duncan, Garwick, Hoare, Randell, Seegmüller, Turski, Woodger.

(In a letter dated Dec. 23 1968, Jan. V. Garwick, who had not been present at the Munich meeting, requested that his name be affixed to this Minority Report.)

## TRANSCRIPT OF PRESENTATION

**C. H. LINDSEY:** (SLIDE 1) My story starts with IFIP, which is the International Federation for Information Processing. It is a hierarchical organization, as you see, with a layer of Technical Committees (TC1, TC2 and so on), and finally a layer of Working Groups (such as Working Group 2.1, Working Group 2.2 and so on). And here we have the authors of the original ALGOL 60 Report.

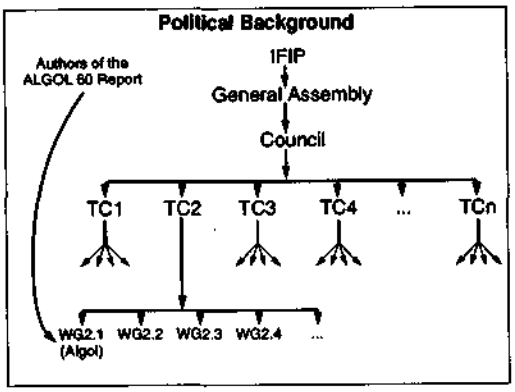
And, in 1962, these authors in fact became Working Group 2.1 of IFIP.

In due course Working Group 2.1 started out to produce a new language to follow ALGOL 60. Ideas for this future language, which became known as ALGOL X, were being discussed by the Working Group from 1964 onwards, culminating in a meeting at Princeton in May 1965. There was also a language ALGOL Y—originally it was a language which could modify its own programs, but in actual fact it turned out to be a "scapegoat" for features that would not fit into ALGOL X. At any rate, at Princeton they thought they were ready, and accordingly they solicited drafts of a complete language to be presented at the next meeting.

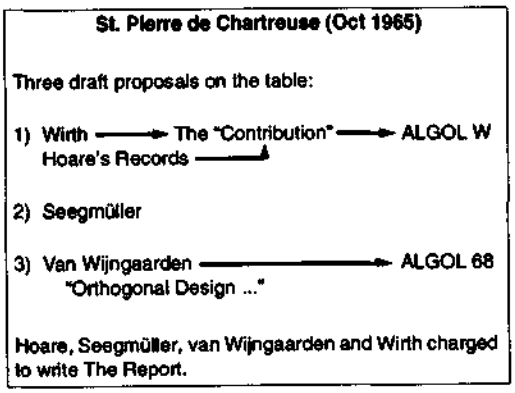
(SLIDE 2) The next meeting, at St. Pierre de Chartreuse, there were three drafts on the table officially, but observe that a fourth document has crept in, and this was Hoare's paper on Records, which Wirth immediately accepted as an improvement on his own. Henceforth we must regard the Wirth/ Hoare proposal as one, and it was eventually published in *CACM* as "A Contribution to the development of ALGOL," and in due course it became the language ALGOL W.

Seegmüller's document was not a serious contender, but it did contain some interesting ideas for call-by-reference, of which more anon. Van Wijngaarden's document was entitled "Orthogonal design and description of a formal language." Its chief innovation was a new 2-level grammar, which so impressed the Working Group that they resolved formally, "Whatever language is finally decided upon by Working Group 2.1, it *will* be described by Van Wijngaarden's metalinguistic techniques." I find this rather surprising because Van Wijngaarden's document, in its original form, was both unreadable and buggy. These four authors were now charged to write THE Report for THE language, with the clear intention that the whole project would be completed within the next year.

The history of what followed is very much concerned with parameter passing, so I propose to follow the development of this in some detail. It is said that an Irishman, when asked how to get to some remote place, answered that if you really want to get to that place, then you shouldn't start from




SLIDE 1




SLIDE 2

here. In trying to find an acceptable parameter-passing mechanism, Working Group 2.1 started from ALGOL 60 which, it is well known, has two parameter-passing mechanisms—call-by-value and call-by-name. This was a grave mistake.

(SLIDE 3) Here is call-by-name in ALGOL 60, and this example is straight out of the ALGOL 60 Report. There are two ways to use call-by-name:

- *y* must be called by name, because it is a result parameter,
- but *a* and *b* must be called by name because their corresponding actual-parameters in the call—*x1[j]* and *y1[j]*—are expressions that involve another actual-parameter, *j*,

  which corresponds to another formal-parameter *p*, also called by name,

  which is used in the loop counter of this for-statement,

  and the whole effect is then equivalent to this for-statement down at the bottom.

  And that *j* stands for "Jensen."

This technique was known as "Jensen's Device," and nowadays we should call it "Jensen's Hack." That is what it was—a Neat Hack. But Hacks, however neat, should never be applied outside of their original context, but this is just what the Working Group now tried to do.

(SLIDE 4) Well here is the Working Group's first attempt:

- the result parameter *y* is now distinguished by the reserved word **loc**,
- but the need for call-by-name is now indicated, not in the procedure heading, but *at the point of call*, here, by this word **expression**.

And the mysterious "*j*" for Jensen is still there, so this is just a minor tinkering with Jensen's Device.

(SLIDE 5) This is Seegmüller's proposal, and it is call-by-reference much as we now know it:

- *y* is now a **reference** parameter,
- *a* and *b* are explicitly declared as **procedures**,
- and at the point of call we have this word **expression**, here, to correspond to those **procedure** parameters.

And Jensen's mysterious *j* is still there as a parameter.

**Call-by-name in ALGOL 60**

```
procedure Innerproduct(a, b, k, p, y);
  value k;
  integer k, p; real y, a, b;
  begin real s; s := 0;
    for p := 1 step 1 until k do s := s+a*b;
    y := s
  end Innerproduct
```

Here is a call:

```
real array x1, y1[1:n]; integer j; real z;
Innerproduct(x1[j], y1[j], n, j, z);
```

Which is equivalent to:

```
for j := 1 step 1 until k do s := s+x1[j]*y1[j];
```

SLIDE 3

**Call-by-name before St. Pierre**

```
proc innerproduct =
      (real val a, b, int val k, int loc p, real loc y)
void:
    begin real var s := 0;
      for p := 1 step 1 until k
      do s := s+a*b;
      y := s
    end;
```

and here is a call:

```
loc [1:n] real x1, y1; loc int j; loc real z;
innerproduct(expr: x1[j], expr: y1[j], n, j, z);
```

SLIDE 4

(SLIDE 6) This is the scheme from the Hoare/Wirth proposal and the example is actually written in ALGOL W:

- here y is now what they called an "anti-value" parameter, or call-by-value-result, as we should now say,
- but this *x1[j]*, here, is just call-by-name, exactly as in ALGOL 60, so Jensen is still with us.

This then was the State-of-the-Art of parameter passing in October 1965, and its total confusion may be contrasted with the limited set of options to which modern programming language designers now confine themselves, and with the simplicity and elegance of the system that presently emerged for ALGOL 68.

(SLIDE 7) Now we need to look at Hoare's Records. The concepts here should now seem quite familiar:

- we have a **record class** with fields *val* and *next*,
- here we declare *start* as the start of an empty *link*ed list,
- and now, down here, when we say *start := link*, this is the *only* way to create new values of this *link* **record class**—and this means that *link*s or records always live on the heap,

**Call-by-reference (Seegmüller)**

```
proc innerproduct = (proc real a, b, int k,
                     int reference p, real reference y)
void:
  begin loc real s := 0;
    for p := 1 step 1 until k
    do s := s+a*b;
    y := s
  end;
```

And here is the call:

```
loc [1:n] real x1, y1; loc int j; loc real z;
innerproduct(expr: x1[j], expr: y1[j],
             n, ref j, ref z);
```

SLIDE 5

**Call-by-value/result (ALGOL W)**

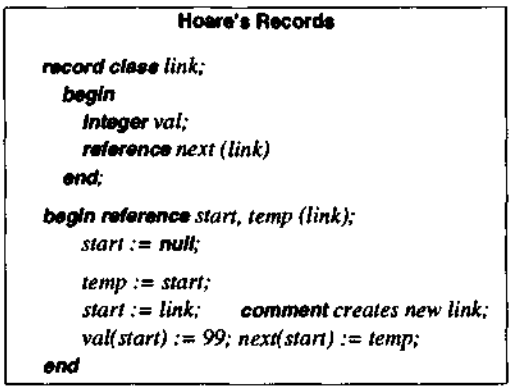
```
procedure innerproduct (real a, real b,
          integer value k, integer p, real result y);
  begin y := 0; p := 1;
    while p≤k do
      begin y := y+a*b; p := p+1 end
  end;
```

*a* is by-name and *b* by-procedure.

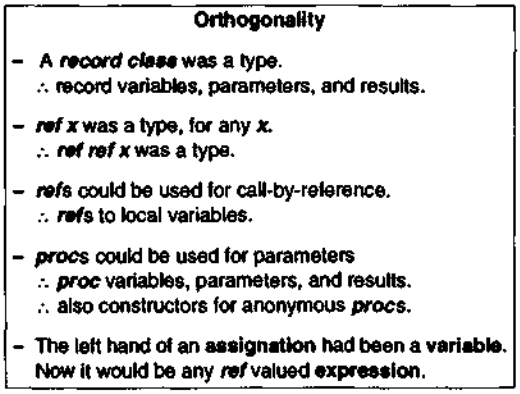
Here is the call:

```
real array [1:n] x1, y1; integer j; real z;
innerproduct(x1[j], y1[j], n, j, z);
```

SLIDE 6




SLIDE 7




SLIDE 8

- and these **references**, here and here, are *nothing at all* to do with Seegmüller's **references**—as they cannot point to local variables and they are certainly not to be used for call-by-reference.

Well, there should have been a full Working Group meeting in April 1966 to consider the report of the subcommittee, but the subcommittee Report was nowhere near ready. Instead, it was the subcommittee itself—Wirth, Hoare, Van Wijngaarden and Seegmüller—that met at Kootwijk.

Now, it had been agreed at St. Pierre that Van Wijngaarden should write the Draft Report for the subcommittee, using his formalism, but instead there were two documents on the table. The Wirth/Hoare one was called "A Contribution to the development of ALGOL," and indeed it bore more resemblance to the language that the Working Group intended, but it wasn't in Van Wijngaarden's formalism. Well, discussions proceeded amicably enough, many points were agreed, until they came to the parameter passing mechanism.

And at this point a complete split developed within the subcommittee. So, what was all the fuss about?

(SLIDE 8) Van Wijngaarden was pressing the parameter-passing mechanism that arose from his principle of "orthogonality." (Remember his original document "Orthogonal design and description of a formal language.") Now, according to the principle of orthogonality,

- A ***record class*** was a type; therefore we could have record variables, record parameters and record results. Nowadays we should say that a record was a "1st class citizen" of the language.
- Similarly, we have references, so ***ref x*** was a type, for any ***x***; therefore it follows that ***ref 'ref x'*** was a type.
- These ***ref***erences were values, so they could be used for call-by-reference, just as Seegmüller had proposed;
  therefore we needed to be able to have ***ref***erences to local variables, which Hoare did not allow for *his* references.
- Similarly, you could pass ***proc***edures as parameters;
  therefore a procedure was a 1st class citizen. You could have procedure variables, procedure parameters and procedure results, and also constructors for *anonymous* procedures.
- And now, you see, traditionally in the older languages, the left hand side of an **assignation** had been a **variable**. Now it would simply be any ***ref*** valued **expression**.

**Everything is call-by-value**

```
proc innerproduct =
        (proc (int) real a, b, int k, ref real y) void:
  begin loc real s := 0;
    for i to k do s +:= a(i) * b(i) od;
    y := s
  end;
```

And here is the call:

```
loc [1:n] real x1, y1;  loc real z;
innerproduct(
        (int j) real: x1[j],  (int k) real: y1[k],
        n, z);
```

**SLIDE 9**

(SLIDE 9) So Hoare's records and Seegmüller's references had now been coalesced into one grandiose scheme, in which *everything* was going to be called by value.

This example is now written in true ALGOL 68:

- *y* is declared as a **ref** parameter: the *value* we pass is simply a reference,
- *a* and *b* are now ***proc***edure parameters,
- and the corresponding actual parameter is now a constructed procedure, that is, a procedure of which *j* is just an ordinary formal-parameter. So Jensen is finally dead.

It was orthogonality, and the use (or misuse) of references, that caused the split in the subcommittee, and Wirth and Hoare could not accept them at all.

Of course all this came up again at the next meeting in Warsaw. The document Warsaw-2 had Van Wijngaarden as its only author. Could this be accepted as the document that had been commissioned from a subcommittee of four? Hoare was dubious at best and Wirth had resigned from the subcommittee and didn't even attend this meeting.

Now came the great "Orthogonality" vs "Diagonality" debate. The discussion lasted the best part of a day, but Van Wijngaarden was able to demonstrate that *his* system was entirely self-consistent.

The other thing which came up at Warsaw was McCarthy's scheme for overloaded operators. This was an entirely new topic, and it was resisted by Van Wijngaarden on the grounds that it was too late for such a major change.

Clearly, the document Warsaw-2 was not going to be finalized that week, as some had initially imagined. So these are the decisions that the Working Group took at Warsaw:

- First of all, the document was to be amended, with proper pragmatics. Note that "pragmatics" are to a Report as "comments" are to a program.
- Then the document was to be published in the *Algol Bulletin* for all the world to see.
- And implementation studies were to be incorporated in it.
- Van Wijngaarden was to be the sole editor.
- And at the next meeting they would get a chance to talk about ALGOL Y, and the meeting after that would take the final decision.
- Oh! and McCarthy's overloading could be incorporated, if Van Wijngaarden found it to be feasible.

The next meeting was at Zandvoort, and it was quite a constructive meeting (relatively speaking) although it didn't actually discuss much ALGOL Y. And there was no sign of rebellion.

McCarthy's overloading was now in "in all its glory." Here I should say that those who were eventually to urge rejection of the Report all shared an exasperation with Van Wijngaarden's style, and with his obsessional behaviour in trying to get his way. He would resist change, both to the language and to the document, until pressed into it.

- The standard reply to a colleague who wanted a new feature was first to say that it was too late to introduce such an upheaval to the document.
- Then that the colleague should himself prepare the new syntax and semantics to go with it.
- But then, at the next meeting, he would show with great glee how he had restructured the entire Report to accommodate the new feature and how beautifully it now fitted in.

McCarthy's overloading, which required an enormous upheaval to the Report, provides the finest example of this behaviour.

Well, the draft Report, as commissioned at Warsaw for publication in the *Algol Bulletin*, was dropped onto an unsuspecting public in February of 1968, and was the cause of much shock, horror, and dissent, even (and perhaps especially) among the membership of Working Group 2.1. At an ALGOL 58 Anniversary meeting at Zurich in May,

- it was attacked for its alleged "obscurity, complexity, inadequacy, and length,"
- and defended by its authors for its alleged "clarity, simplicity, generality, and conciseness."

And both Naur and Wirth resigned from the Working Group at this time.

The next meeting at Tirrenia was a stormy one, and the Report was the sticking point. As Randell said at that meeting,

> "From our discussions . . ., it seems to follow that there is reasonable agreement on the language, but there is the need for an investigation of alternative methods of description."

So what to do? Well, it was decided,

- First, that the present document could not go upwards to TC2;
- But second, nevertheless, it *was* the document that had been commissioned at Warsaw, and so the author had fulfilled his obligation.

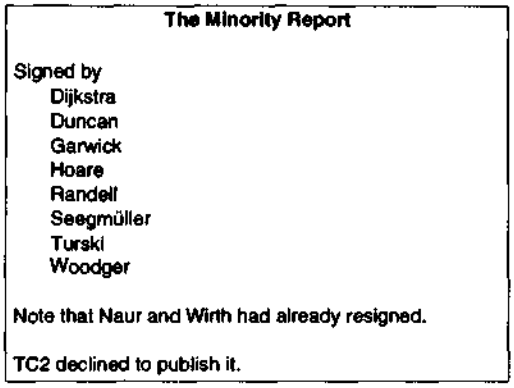
Now what? Well, in the last hours of the meeting, Van Wijngaarden offered to prepare one last edition for final acceptance or rejection in December 1968.

But there was one more meeting before that, at North Berwick, just before the IFIP Congress in Edinburgh. This was the first meeting I attended myself. It was a rude introduction into five days of continuous politics—Oh! there was one afternoon devoted to technical discussion. And the possibility of a Minority Report was being raised.

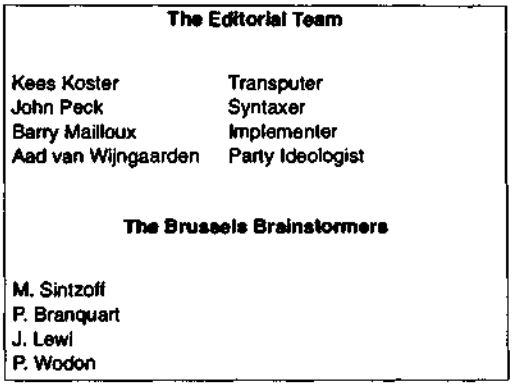
The next meeting at Munich was relatively calm. Of course we had a decision to reach. A covering letter was prepared and its tenor was that

> This Report is submitted as the consolidated outcome of the work of the Group. It is ... the decision of the Group that, although there is a division of opinion amongst some of its members, the design has reached the stage to be submitted to the test of implementation and use by the computing community.

(SLIDE 10) However, in the last hour of the meeting, some who were unhappy with the final form of the Covering Letter, and with interpretations which were being put upon it, produced a minority




SLIDE 10




SLIDE 11

report. The text of that minority report is in the paper, and as you see, it was signed by some very worthy names. It was clearly intended to be published alongside the full Report, and the decision by TC2 not to do so I can only describe as Unethical and Inexcusable.

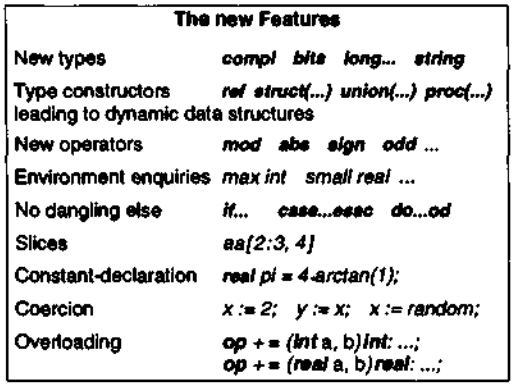
(SLIDE 11) Well, there is the team that actually wrote the Report:

- Kees Koster (transputer) was a student at the Mathematisch Centrum under Van Wijngaarden.
- John Peck (syntaxer) was on sabbatical leave from the University of Calgary.
- Barry Mailloux (implementer) had been Van Wijngaarden's doctoral student from the time of Kootwijk.
- And, of course, Aad van Wijngaarden himself (party ideologist) kept the whole thing together.

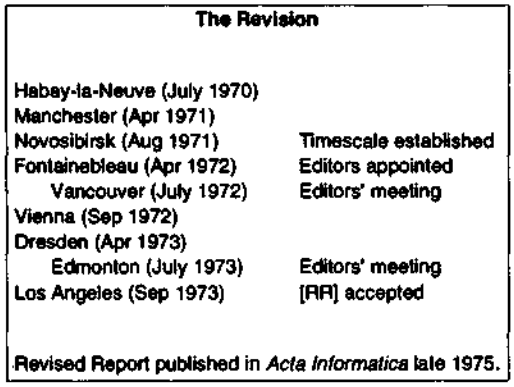
Now, during the preparation of the Report, drafts and comments were being circulated among an "inner circle," of whom the group of Branquart, Lewi, Sintzoff, and Wodon at MBLE in Brussels is especially to be noted. Effectively, you see, there were two teams: One in Amsterdam creating the text, and another in Brussels taking it apart again, and this mechanism was actually very fruitful.

(SLIDE 12) These are some of the new language features that appeared in ALGOL 68.

- There were many new types and type constructors (references, unions, and so on), leading to the possibility of dynamic data structures.
- There were lots of new operators.
- There were environment enquiries.
- The ***if*** was matched by ***fi***, and other similar things. That solved the "dangling ***else***" problem.
- There were decent multi-dimensional arrays with nice slicing facilities.
- There was a **constant-declaration**. As an example, here you could not assign to this *pi* as declared here because it is declared as a constant with this value.
- Coercions. The three coercions shown here were known as 'widening' (***int***eger to ***real***), 'dereferencing' (***ref***erence to ***real*** to ***real***), and 'deproceduring' (***proc***edure returning ***real*** down to ***real***).
- And there was overloading of operators such as these two versions of + declared here, one for ***int***egers and one for ***real***s.




SLIDE 12




SLIDE 13

(SLIDE 13) The style of Working Group meetings became much less confrontational after 1968. The first implementation of something like ALGOL 68, by the Royal Radar Establishment at Malvern, appeared early in 1970. It now became clear that there were many irksome features of the language that hindered implementation and upwards-compatible extensions, while not providing any user benefit, and thus a revision was called for.

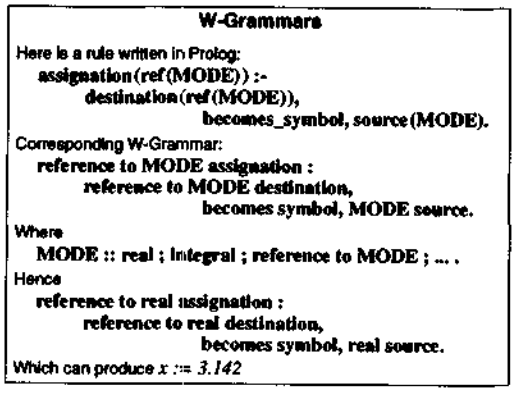
The slide shows the timescale up to the point where Editors were appointed in 1972. The new Editors included Michel Sintzoff, from Brussels, and myself, later to be joined by Lambert Meertens and Richard Fisker. Van Wijngaarden and Koster, on the other hand, withdrew from any active role. Our brief was

> to consider, and to incorporate as far as possible" the points from various subcommittee reports and, having corrected all known errors in the Report, "also to endeavour to make its study easier for the uninitiated reader.

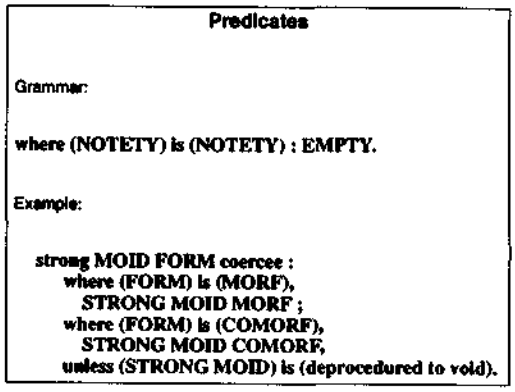
The new editors got together in Vancouver in July of 1972, and it soon became apparent that, to make the language definition both watertight and readable, a major rewrite was going to be necessary. I therefore intend to take a look at the method of description used in the Report, starting with the 2-level van Wijngaarden grammar.

(SLIDE 14) I am going to explain W-Grammars in terms of Prolog, since Prolog itself can be traced back to some early work by Colmerauer using W-Grammars.

- So this rule is written in Prolog. We have a goal "do we see an ***assignation*** here?", and to answer this we must test subgoals "do we see a ***destination*** here?", "do we see a ***becomes_symbol*** here?", and so on. And in Prolog, the values of variables such as *MODE* may be deduced from the subgoals, or they may be imposed with the question—in Prolog the distinction does not matter: the value of the variable just has to be consistent throughout.
- In the corresponding ALGOL 68 rule, a goal is just a free string of characters, in which variables such as **MODE** stand for strings produced by a separate context-free 'metagrammar'.
- So **MODE** can produce **real**, or **integral**, or **reference to** some other **MODE**, and so on (metagrammar is usually recursive).
- So by choosing for **'MODE'** the value **'real'** you get the following production rule:
  a **reference to real assignation** is a **reference to real destination**, a **becomes symbol** and a **real source**.
- Of course, this can produce things like *x := 3.142*.




SLIDE 14




SLIDE 15

But observe how we have insisted that the **MODE** of the **destination** *x* must be ***ref***erence to ***real*** because the **MODE** of the **source** *3.142* is ***real*** (or you could argue the other way round—*3.142* must be ***real*** because *x* is known to be ***ref***erence to ***real***—you can argue both ways round just as in Prolog).

(SLIDE 15) In the Revision, we found newer and cleaner ways to use W-Grammars, most notably through the use of "predicates." Here we have a rule to produce a thing called a **strong-MOID-FORM-coercee**. Now it so happens that **FORM**s can be various things—some of them are called **MORF**s, some of them are called **COMORF**s.

And the whole purpose of this rule is actually to forbid deproceduring of certain **COMORF**S, so

*If* the particular **FORM** we are dealing with happens to be a **MORF**

- this so-called "predicate" **where (FORM) is (MORF)** produces **EMPTY**, so then we go ahead and produce what we wanted to produce.

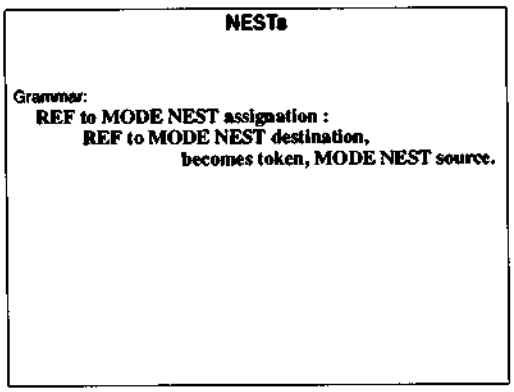
But if the **'FORM'** is a **'COMORF'**

- then we satisfy this predicate **where (FORM) is (COMORF)**,
- after which we also have to satisfy this one, which says that "**unless** the **'STRONG MOID'** happens to be **'deprocedured to void'**." And that is so arranged that *if* the **'STRONG MOID'** is *not* **'deprocedured to void'**, then it does produce **EMPTY**, but not so if the **'STRONG MOID'** *is* **'deprocedured to void'**.
- And in that case, since it doesn't produce anything at all, I am prevented from producing a **deprocedured-to-COMORF**, as intended.

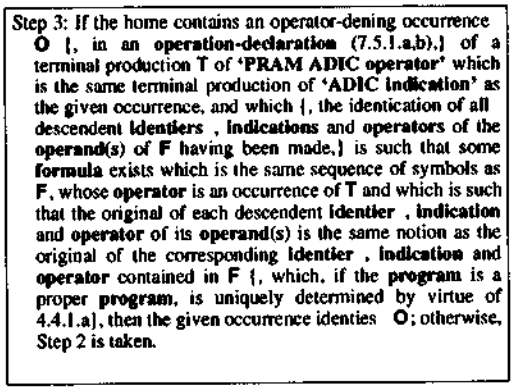
Well, the point is that, with predicates, you can do the most amazing things, once you have learned how to use them.

In the original Report, a whole chapter had been devoted to what we now call "static semantics." But we found too many bugs—shocks and surprises—in this chapter to have any confidence in its correctness. We were now determined that the Report should be written in such a way that we could successfully argue the correctness of any part of it. Hence the static semantics was put into the grammar, using predicates. Here is how it was done. (SLIDE 16)

- Here again is the syntax of an **assignation**. Now you see that the **assignation** is accompanied by a **'NEST'** containing all the declarations visible at that point in the program, and the **destination** is accompanied by the same **'NEST'**, and likewise the **source**.




SLIDE 16




SLIDE 17

And if the **destination** or the **source** contains any **identifiers**, then predicates in the grammar ensure that each **identifier** occurs properly in its **'NEST'**, and the syntax of **declarations**, of course, ensures that the **'NEST'** corresponds to the things that actually were declared. And suitable **'NEST'**s are propagated throughout the whole of the grammar, so that all the static semantics are incorporated. Well, that is the grammar, but the real problem with the original Report was *not* the W-Grammar; it was the style of the semantics, which was pedantic, verbose, and turgid.

(SLIDE 17) Here is a nice example from the Original Report, which is concerned with the static semantics of overloaded operators. Let me read it to you.

[Dramatic reading of the slide, with applause].

Now that text happens to be both necessary and sufficient for its purpose, but I hope you can see why we concluded that the static semantics would be more safely described in the Syntax.

(SLIDE 18) So we treated the Report as a large-scale programming project, consciously applying the principles of structured programming. Here, for example, is the original semantics of assignment.

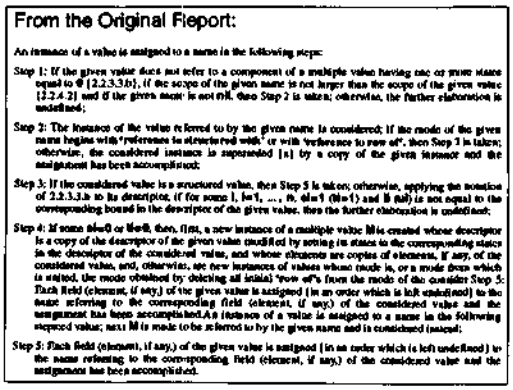
- It is made up of all these Steps, and it is clearly a program full of **gotos**, and it conveys a general feeling of wall-to-wall verbosity.

(SLIDE 19) Here is the revised semantics of assignment, and I hope you will believe, from its layout alone, that you could readily set about understanding it if you had to.

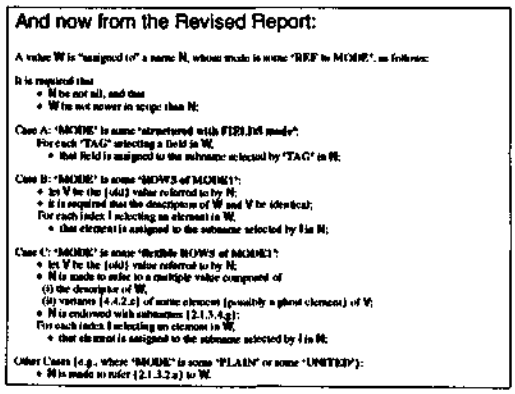
- You see it has nice indentation, cases, local declarations, and for-loops.

Well, we took some examples of the new style of syntax and semantics to the next Working Group meeting in Vienna, where they were greeted with horror by some members. A "Petition for the Status Quo of ALGOL 68" was presented, claiming that the Revised Report had become a pointless undertaking and that the idea should be abandoned. This was debated on the very last day of the meeting, the motion being that the Revision should merely consist of the Old Report plus an addendum.

- Well, we pointed out that some errors were almost impossible to correct in the old framework.
- We pointed out that, to have a basis for upwards-compatible future extensions, we would need at least twice the list of changes proposed by the petitioners.
- We pointed out that it would then require an addendum half the size of the Report.
- We were getting nowhere. Now it was time for the vote.




SLIDE 18




SLIDE 19

It had long been a Working Group tradition that straw votes were phrased in the form "Who could live with . . .?" rather than "Who prefers . . .?". So to let everybody see the full consequences before taking the formal vote, I proposed these questions.

"Who could live with the state if the resolution was passed?"

- 12 could live with it, 5 could not live with it, and 1 abstained.

"Who could live with the opposite state?"

- 12 could live with the opposite state, 1 could not, and 5 abstained.

Now van Wijngaarden was always a superb politician. Having observed who had actually voted, he pointed out that passing the resolution "would cause the death of five of the six editors, which seemed a most unproductive situation." And then the formal motion was put.

**[6-6-6]**

And we were through.

By the next meeting in Dresden, all the fuss had subsided, and people even complimented us on the improved clarity. The editors held a second three-week get-together in Edmonton, and the Revised Report was accepted, subject to polishing, in September 1973 in Los Angeles.

So here are my recommendations to people who essay to design programming languages.

- The work should be done by a small group of people (four or five is plenty).
- There is considerable advantage in geographical separation, as between Amsterdam and Brussels the first time, or between Manchester, Vancouver, and Edmonton the second time.
- The Editors need tidy and pernickety minds and, incidentally, good editors are not necessarily good language designers, nor vice versa.
- You should construct a formal definition of the language at the *same* time. I believe ALGOL 68 is the only major language in which the Formal Definition was not a retrofit.
- There is considerable benefit in doing the whole job twice. We had that advice this morning also.
- And we most certainly observed that large committees are unstable (although they may be necessary for democratic reasons).

And here are my recommendations for writing Reports.

- There must be copious pragmatic remarks, well delineated from the formal text. The purpose of these remarks is to *educate* the reader.
- Motivation should be given for *why* things are as they are. This means that we have redundancy, but redundancy is no bad thing.
- The syntax should be fully cross referenced, forwards, backwards, and every which-way; and the semantics likewise.
- And the Report should not take itself too seriously. The Report is not going to be perfect. Our Report contained some good jokes, many quotations from Shakespeare, Lewis Carroll, A. A. Milne and others, and even a picture of Eeyore.
- So, above all, the Report should be *fun* to write.

Unfortunately, Standards Bodies and Government Departments do not always encourage these practices in their guidelines. I invite you to consider the extent to which they have been followed, or otherwise, in more recent programming language definitions, together with the perceived success of those definitions.

On that basis I will rest my case.

## TRANSCRIPT OF QUESTION AND ANSWER SESSION

**HERBERT KLAEREN (University of Tubingen):** Could you explain the hipping coercion a bit. In your example, *x* := ***if*** *p* ***then*** *y* ***else goto*** *error* ***fi***, I would expect that *x* doesn't change if *p* is ***false***. How does the coercion come in?

**LINDSEY:** The hipping coercion was a dirty piece of syntax in the original Report. It is simply that, if you have got a conditional expression, one half of it ***if*** *some condition* ***then*** *some value*, then that presumably is the mode of the result. If the ***else*** part says ***goto*** *error*, what is the mode of ***goto*** *error*? So there was a fictitious coercion to make everything look tidy. It was syntactic tidiness.

**STAVROS MACRAKIS (OSF Research Institute):** Was the schedule of three meetings per year good or bad?

**LINDSEY:** It would average nearer two, except when things got hectic in 1968. The work has to be done off-line by a small group of four or five people. The function of a committee is simply to oversee. So two meetings a year is plenty. This matter might come up again with Colonel Whitaker this afternoon.

**MARTIN CAMPBELL-KELLY (University of Warwick):** Is ALGOL 68 completely dead, or just a dying language?

**LINDSEY:** It is a matter of common knowledge—it is not currently in widespread use, but, there are places that still use it. If you want a compiler, see me afterwards. Incidentally, there are some documents at the back, and one of them is a copy of the *Algol Bulletin* (AB 52, August 1988) listing all the implementations that currently exist, or did exist a few years ago.

**MIKE WILLIAMS (University of Calgary):** I once asked John Peck if ALGOL 68 was ever intended to be used or if it was an intellectual exercise.

**LINDSEY:** It was quite definitely intended as a language to be used. The reasons why languages do, or do not, get used bear little resemblance to the goodness or badness of the language. And which is

the most commonly used language at the moment, and is it a good one? I won't name it, but I am sure you all have your ideas. Yes, it was definitely intended to be used, and the reasons it wasn't are as much political as technical, I think.

**HERBERT KLAEREN (University of Tubingen):** ALGOL 68 has been criticized as being a committee-designed language, and looking that way. From your history of the original ALGOL 68 Report, I got the impression that it was much more Van Wijngaarden's language than it was a committee language. Could you comment on that?

**LINDSEY:** I think it was a well-designed language in spite of being designed by a committee. You say that a camel is a horse designed by a committee. ALGOL 68 was no camel. It was Van Wijngaarden's personality, I think, which kept it clean in spite of the committee. Nevertheless, many of the odd little features which are in, are in because the committee said—voted—"we want this X in like this"; and there it is if you look. Usually, where there are unfortunate little holes in the language, they are usually small little things—nothing to do with the orthogonality. Many of those, I think, were put in because suddenly the Working Group said "I want this," and they got it. Van Wijngaarden always reserved to himself the right as to how the thing was described. I am not sure whether this was necessarily a good thing or not. It was done in spite of a rather large and somewhat unruly committee. But, in a democracy, committees are necessary.

**HERBERT KLAEREN (University of Tubingen):** Why is the rowing coercion "clearly" a mistake? In connection with string processing, it would seem natural to coerce a character *"A"* into a one-element string, if necessary.

**LINDSEY:** Because its effect is counter-intuitive (my students were always surprised when it did not produce an ***array*** of some appropriate size with all the elements initialized to the given value). It was put in to fix a hole in the syntax. Indeed, it also turns a ***char*** into a one-element ***string***, but that could have been brought about in other ways. No other language has had to do it this way.

## BIOGRAPHY OF C. H. LINDSEY

Charles Lindsey was born in Manchester, UK, in 1931. He obtained his Bachelor's degree in Physics at Cambridge University in 1953, and his PhD in 1957 for a thesis exploring use of the newly invented ferrite cores for logical purposes rather than for storage. At that time, he was clearly an engineer, although he did write some programs for the EDSAC.

From 1957, he worked for Ferranti (now a part of ICL), becoming project leader for the Orion computer (that was actually the first commercial time-sharing computer, although it was never a commercial success). After that, he worked on "Design Automation" (or Computer Aided Design, as we should now say), and designed his first programming language for simulating hardware.

In 1967, he joined the staff of the recently created Department of Computer Science at the University of Manchester, where he immediately became interested in the language ALGOL 67 (as it then was) under development by IFIP Working Group 2.1. A paper "ALGOL 68 with fewer tears" written to explain the new language informally to the "uninitiated reader" brought him to the notice of WG 2.1, of which he soon became a full member. Enhancement and development of ALGOL 68 kept him fully occupied until about 1983.

This was followed by some work on extensible languages, and participation in the continuing work of WG 2.1, which was now centered around specification languages. He retired from the University in 1992, but took his computer home with him, so as to be able to continue working in Computer Science.