During the Symposium in Rome from March 26th to 31st prof. van der Poel asked the present editor to accept the responsibility of the secretary of the Working Group. To this I answered that this I would accept provided that I might expect the basic support of some of the more influential workers in the field, particularly professors F.L. Bauer and K. Samelson (both members of the original ALGOL committee, initiators of the ALCOR Group, prof. Bauer is the DARA representative to IFIP Tech. Comm. 2.). Prof. van der Poel consented in this condition and kindly accepted to put the question forward to professors Bauer and Samelson. The result of this mission was a flat refusal on their part to support me on the grounds that in their view the questionaire of AB 14 is biased.

In view of these developments it is clear that the responsible bodies of IFIP, in establishing the Working Group, deliberately have chosen to ignore the existence of the ALGOL Bulletin and the information and opinions expressed in it. It is further clear that the attempt on the part of prof. van der Poel at establishing a working collaboration between the Working Group and the AB in an informal manner meets with an opposition which would make this collaboration ineffective in practise. Now, in encouraging the ALGOL community to make use of the ALGOL Bulletin for expressing their views the editor must feel convinced that the views contained therein will indeed be taken properly into account when official action is taken. The developments mentioned above and the meeting in Rome have annihilated this conviction. Consequently the ALGOL Bulletin must cease to exist.

THE REPLIES TO THE AB 14 QUESTIONAIRE.

BRIEF NAMES, GROUP MEMBERS, TRANSLATORS.

NPL, England
M. Woodger, Mathematics Division, National Physical Laboratory, Teddington, Middlesex, England.

Zeiss, Germany
J. O. Kerner, VEB Carl Zeiss, Jena Germany.

SMIL, Sweden
Torgil Ekman, Leif Robertsson, Avd. for Numerisk Analys, Lunds Universitet, Lund, Sweden.
Translator for SMIL: March 1962; 1.5 man years; true subset; not recursive procedures, not arrays of variable length.

Syst.Dev.Corp., USA
Harold Isbitz, J. Schwartz, H. Bratman, System Development Corporation, JOVIAL Compiler Staff.
JOVIAL translators for Q7, Q32-V, IBM 7090, Philco 2000, CDC 1604: Dec. 1961; 80 man years; dialect; extended variable definition, variable modifiers, no recursive procedures, no own variables, I-O operators, string manipulation.

Royal McBee, USA
Arthur C. Housman, Royal McBee Corp., Research and Development.

Mayoh, USA
B.H. Mayoh, Univ. of Illinois.

IDA-Princeton, USA
Edgar T. Irons, Institute for Defense Analyses, Princeton, New Jersey, Von Neumann Hall.
Translator for CDC 1604: Nov. 1961; 2 man years; true subsets; no numeric labels, Fortran-like input-output.

Facit Group, Sweden
I. Dahlstrand, S. Laryd, Facit Electronics AB, ALGOL Group Box 13072, Gothenburg 13.
Translator for Facit EDB 2: Nov. 1961; 3.5 man years; true subset; not recursive procedures, not own; restrictions on variable index bounds, expressions called by name, and length of identifiers and strings.

Rehn, Finland.
Rafael E. Rehn, Maanmittaushallitus (General Survey Office), Helsinki, Finland.

Eng.El.Atomic, Eng.
B. Randell, L.J. Russell, F. Ford, A.P.Relph, E. M. P. Allsop, The Atomic Power Division, The English Electric Co. Ltd., Whetstone, Leicestershire, England.
Translator for English Electric DEUCE Mk 2A: Nov. 1961; 0.5 man years; true subset; no conditional expressions, no dynamic arrays, no switches, no booleans, no recursive use of procedures, single letter identifiers.

Elliott ALGOL, Eng.
C.A.R. Hoare, R.L.Cook, J. Hoare, J. Hillmore, Elliott Brothers (London) Ltd.
Translator for National Elliott 803: Mar. 1962; 3 man years; true subset; restrictions on switches and procedure parameters. Only simple uses of recursion and own arrays.

ALPHA, USSR
A. P. Ershov, Institute of Mathematics, Siberian Division of the USSR Academy of Sciences, Novosibirsk 72.
Translator under way: expected Nov. 1962; 10 man years; ALGOL 60 + (vectors, matrices, etc. as simple variables, complex, chains of inequalities, superscripts, initial values, functions yielded by expressions).

Buchholz, Germany
G. Buchholz, Funkwerk Dresden Rechenburo.

Oak Ridge, USA
Manuel Feliciano and A.A.Grau, C.J.Atta, L.L.Bumgarner, Oak Ridge National Laboratory, Programming research group of Mathematics Panel.
This reply is also supported by the Programming Research Dept., Lockhead Missiles and Space Co., California.
Translator for ORACLE: Jan. 1961; 4 man years; true subset; no switches, no procedures, no blocks. Added feature: tape files, input output facilities.

SSW-ZEF, Germany
Dr. E. Nuding, mr. Nees, mr. H.P.Wolf, miss Warmbold, miss Hecht, Siemens-Schuckertwerke AG. Erlangen.

Dupont, USA
Robert Hunn, E.I. DuPont de Nemours and Co., Inc., Eng. Dept., Wilmington 98, Delaware.

Rutishauser, Switzerland
H. Rutishauser, Zurich (private opinion).

Kidsgrove, England
F.G.Duncan, D.H.R.Huxtable, E.N.Hawkins, J.S.Green, A.G. Price, The English Electric Co., Ltd., Data Processing and Control Systems Division, Kidsgrove, Stoke-on-Trent, England.
The group uses the translator for DEUCE written by the Eng.El. Atomic (see above).

MNA-group, Sweden
G. Ehrling, A. Bring, Swedish Board for Computing Machinery.
The group uses the translator for FACIT EDB 2 written by the Facit Group (see above).

IAM Bonn, Germany
C.A. Petri, W.D. Meisel, G. Schroder, K.H. Bohling, Institut fur Angewandte Mathematik, Universitat, Bonn.

Math.Cent., Holland
E. W. Dijkstra, A. van Wijngaarden, J. A. Zonneveld (the two latter not responsible for the answers given), Stichting Mathematisch Centrum, 2de Boerhaavestraat 49, Amsterdam, Netherlands.
Translator for Electrologica NV X1: Aug. 1960; 2 man years; true subset; own implemented as suggested in Reformulation 23, but own arrays only with constant bounds; order of declarations at the beginning of a block somewhat restricted.

Dutch PTT, Holland
Prof. Dr. W.L. van der Poel, Dr. G. van der Mey, Dr. Neher Laboratory of the Netherlands Postal and Telecommunications Services, Leidschendam, Holland.

San Diego, USA
C.L. Perry, E. Ferguson, R. Mitchell, Univ. of California, San Diego, La Jolla, California.
The group has the NELIAC translator for the CDC 1604 available. NELIAC is a dialect, described in book by M. Halstead.

Leeds Univ., England
Dr. G.B. Cook, M. Wells, Electronic Computing Laboratory, University of Leeds, Leeds 2, United Kingdom.

ALCOR PERM, Germany
G. Seegmuller, F. Peischl, W. Urich, H. R. Wiehle, Rechenzentrum der Technischen Hochschule, Munchen.
Translator for PERM: Sept. 1961; 1.5 man years; true subset; ALCOR conventions (see ALCOR Z22 below).

ALCOR Z22 Germany
F. L. Bauer, Paul, Baumann, Witzgall, Musstopf, Institut fur Angewandte Mathematik der Universitat Mainz.
Translators for a) Zuse Z22: July 1961; 0.75 man years; b) Zuse Z22 R: Oct. 1961; o.25 man years; true subset; ALCOR conventions: not while, not own, no conditional Boolean or designational expressions, no recursive procedures, restrictions on type procedures to prevent side effects.

ALCOR 2002, Germany
K. Samelson, Hill, Langmaack, Mathematisches Institut der Universitat Mainz.
Translator for Siemens 2002: Dec. 1961; 1.5 man years; true subset; ALCOR conventions (see ALCOR Z22 above).

RCA-EDP, USA
R. Hux, R. Dash, K. Brons, A. Grace, Radio Corporation of America, Electronic Data Processing.

Bjork, Sweden
Harry Bjork, Inst. of Mathematics, Uppsala University.
Uses the translator for Facit EDB 2 (see Facit group above).

Moore School, USA
Peter Zilahy Ingerman, Harry A. Freedman, Mrs. Irene Cotton, Dr. Saul Gorn, Mechanical Languages Projects, Room 302 Moore,
University of Pennsylvania, Philadelphia 4, Pennsylvania.

IPM Darmstadt, Germany
Dr. W. Borsch-Supan, W. Barth, D. Stephan, J. v. Peschke, Institut fur Praktische Mathematik, Technische Hochschule, Darmstadt.
Translator for DERA: Jan. 1961; 0.5 man years; true subset; FORTRANSIT-like restrictions, read and write are delimiters, not procedures.

Computer Ass., USA
Kirk Sattley, Thomas E. Cheatham, Jr., Gene F. Leonard, Robert M. Shapiro, Computer Associates, Inc., 44 Winn St., Woburn, Mass.
The group has contracted for delivery of an Algol translator during the summer of 1962.

Regnecentralen, Denmark
P. Naur, J. Jensen, P. Mondrup, Regnecentralen, Copenhagen.
Translator for DASK: Oct. 1961; 4 man years; true subset; no recursive procedures, no own arrays, no value arrays, no integer division, no implication.

Univ.N.Carolina,USA
John W. Carr, III, Miriam G. Shoffner, Robert B. DesJardins, Peter J. Brown, Computation Center, University of North Carolina.

RRE, England
J. M. Foster, D.P.Jenkins, S.N.Higgins, Royal Radar Establishment, Malvern, England.

Stanford, USA
Harold R. Van Zoeren, George E. Forsythe, John G. Herriot, James Ortega, Beresford Parlett, Computer Science Division, Stanford University, Stanford, California.
The group uses the dialect BALGOL for Burroughs 220: no arrays of variable size, no own, no recursive procedures, no conditional expressions, no nested block structures.

RCA-LAB, USA
Allen H. Simon, Radio Corporation of America Laboratories.
Has translator for CDC 1604 available: no own arrays, no integer labels.

ARF, USA
R. W. Floyd, B. Mittman, R.R. Steck, Armour Research Foundation, 10 West 35th Street, Chicago 16, Illinois.
Translator for UNIVAC 1105: Mar. 1962; 2 man years; true subset; no procedures, 1 and 2 dimensional arrays, read and print added to basic words, produces USE assembly language.

NDRE, Norway
Jan V. Garwick, O.-J. Dahl, Norwegian Defense Research Establishment and Institute for Atomic Energy.
Translator for Mercury for the dialect MAC.

XTRAN-project, USA
Rainer Kogon, Martin Weitzman, IBM, 112 E. Post Road, White Plains, New York.
Translator for IBM 7090: Mar. 1962; 2-3 man years; dialect; ALGOL 60 except where implementation in the XTRAN source language indicated reasons for deviation.

Hockney, England
R. Hockney, English Electric Co., Ltd., Atomic Power Department Systems Office.
Has translator for DEUCE (see Eng.El.Atomic above) available.

NBS, USA
J.H. Wegstein, W.W. Youden, National Bureau of Standards

Tubingen, Germany
K. Zeller, F. Schwenkel, Mathematisches Institut (Abteilung Rechenzentrum), Tubingen.
Have translator for Siemens 2002 available.

Siemens, Germany
W. Heise, Froehr, Walter, Siemens und Halske AG, Munich.
Have the translator for Siemens 2002 available (see ALCOR 2002 above).

Standard El., Germany
Dr. A. Wilhelmy, Dipl.-Math. W. Heydenreich, Standard Electric Lorenz AG, Stuttgart-Zuffenhausen, Germany.
Translator for ER 56: Level 1 (excluding procedures): Febr. 1962, Level 2 (procedure version): under testing; true subset; based on ALCOR conventions, supplied by studies on recursive procedures.

AFCALTI, France
F. Genuys, Messrs. Nolin, Lehtin, Nivat, Picard, Pitrat, Broise.

Burroughs, USA
J.N. Merner, Don Knuth, L.D. Turner, F. Gerbstadt, R.B. Waychoff, ALGOL Group of Automatic Programming, Burroughs Corporation.
Translator for Burroughs 220: Sept. 1960: 5 man years; dialect; ALGOL 58 with I/O, without DO, dynamic arrays, but allowed arrays with partially filled subscripts.

Saarland, Germany
Dr. W. Handler, H.J. Schneider, D. Jurksch, Rechenzentrum der Universitat des Saarlandes.
Have the translator for Zuse Z22 available (see ALCOR Z22 above).

Remington Rand, USA
C.W. Dobbs, P.A. Smethurst, Systems Programming Dept., UNIVAC Division, Sperry Rand, Inc., and A.E. Roberts, General Kinetics, Inc.,
Translator for UNIVAC 1107 expected in June 1962.

Cambridge, England
D. F. Hartley, J.H. Matthewman, University Mathematical Laboratory, Cambridge.
These notes and replies on the questionaire are expressions of opinion and under no circumstances may they be used in any form of voting. We reserve the right to change our opinions or to be influenced by future events.

Wegner, England
Peter Wegner, London School of Economics, Houghton Street, London W. C. 2.

TABLE OF ACTIVITY REPORTS AND TRANSLATOR SPEEDS.

Brief name of group	No. of members	5: Man years	Teaching - percent -	Progr. - percent -	Impl. - percent -	12: Fraction per-cent	13,14: Pages written	publ.	Name of machine or system, with times for 100, 1000, and 10 000 instructions in minutes			
NPL, England	1	0.1	100	0	0	0	0	7				
Zeiss, Germany	1	3	70	20	10	-	10	-				
SMIL, Sweden	2	0.2	85	10	5	-	-	1	SMIL	0.4	4	-
Syst.Dev.Corp., USA	3	(75)	10	60	30	Note 1		0	JOVIAL	0.2	2	20
Royal McBee, USA	1	0	-	-	-	0	0	0				
Mayoh, USA	1	0.7	0	10	90	100	25	3				
IDA-Princeton, USA	1	2	5	5	90	5	30	0	CDC 1604	0.5	1	2
Facit Group, Sweden	2	3.5	5	5	90	Note 2		-	FACIT EDB 2	6	7	-
Rehn, Finland	1	0.2	50	50	0	90	40	0				
Eng.El.Atomic, Eng.	5	4	20	20	60	15	0	0	DEUCE Mk 2A	0.1	1.5	-
Elliott ALGOL, ENG.	4	2	5	5	90	55	-	3	Elliott 803	0.2	2.0	
ALPHA, USSR	1	(5)	10	10	80	-	-	-				
Buchholz, Germany	1	0.2	30	70	0	30	30	-				
Oak Ridge, USA	4	5	10	10	80	80	200	4	ORACLE	-	4.5	-
SSW-ZEF, Germany	5	0.5	50	50	0	10	100	0				
Dupont, USA	1	0.2	25	75	0	1	100	0				
Rutishauser, Switzer.	1	-	-	-	-	-	-	-				
Kidsgrove, England	5	3	10	5	85	-	-	-	DEUCE	-	1.5	-
MNA-group, Sweden	2	3	0	0	100	5	50	0	FACIT EDB 2			
IAM Bonn, Germany	4	1.5	90	10	0	5	100	-				
Math.Cent., Holland	3	5	20	40	40	Note 3		0	Electr. X1	0.5	3	-
Dutch PTT, Holland	2	2	5	0	95	Note 4						
San Diego, USA	3	1	25	75	0	5	-	4	CDC 1604			
Leeds Univ., England	2	0	-	-	-	2	20	0				
ALCOR PERM, Germany	4	3	17	33	50	95	750	0	PERM	1	3	-
ALCOR Z22, Germany	4	-	-	-	-	95	Note 5	-	Zuse Z22	0.8	6.2	-
ALCOR 2002, Germany	3	-	-	-	-	90	Note 5		Siemens 2002	0.33	3.2	-
RCA EDP, USA	4	-	-	-	-	-	25	2				
Bjork, Sweden	1	0	-	-	-	50	5	-	FACIT EDB			
Moore School, USA	4	3	60	30	10	50	30	3				
IPM Darmstadt, Germ.	4	1	10	40	50	Note 6		0.5	DERA	12	-	-
Computer Ass. USA	4	2	0	20	80	50	50	2				
Regnecentralen, Denm.	3	4	12	10	78	80	100	10	DASK	1.5	5	-
Univ.N.Carolina, USA	4	1	0	0	100	75	25	0				
RRE, England	3	0.5	10	20	70	50	50	0				
Stanford, USA	5	1	10	90	0	Note 7		25				
RCA-LAB, USA	1	0.1	70	30	1	1	10	-	CDC 1604			
ARF, USA	3	2.5	2	2	96	0	0	1	UNIVAC 1105	0.2	1.3	-
NDRE, Norway	2	2	0	-	100 -	70	150	0.5	Mercury	0.1	1	10
XTRAN Project, USA	2	2	25	25	50	100	250	-	IBM 7090	1	2	20
Hockney, England	1	0	Note 8		-	0	0	0	DEUCE			
NBS, USA	2	2	50	50	0	80	10	2				
Tubingen, Germany	2	1	-	-	-	10	50	0	Siemens 2002			
Siemens, Germany	3	-	-	-	-	-	-	-	Siemens 2002			
Standard El., Germ.	2	1	-	-	100	0	30	0	ER 56	1	10	
AFCALTI, France	7	-	-	-	-	-	-	-				
Burroughs, USA	6	5	10	2	88	100	100	2	Burroughs 220	0.3	2	20
Saarland, Germany	3	2	50	0	50	15	100	-	Zuse Z22			
Remington Rand, USA	3	6	0	17	83	-	-	-	UNIVAC 1107			
Cambridge, England	2	Note 9										
Wegner, England	1	0	100	0	0	5	1	0				

Notes to table of activity and translator speeds.

Note 1. Qu. 12: 99 percent. Qu 13: 1000 pages, does not include operational programs, these approximate 10000 pages. The JOVIAL compilers were written in JOVIAL.

Note 2. The ALGOL Group as such seldom writes ALGOL programs. About 50 percent of the programs in our company are written in ALGOL.

Note 3. Qu. 12: Too vague to be answered. Practically all service computations and all numerical experiments in ALGOL 60. Machine coding for non-numerical activities (linguistic investigations, algebraic translators) and the construction of machine coded procedures (a library for the benefit of the ALGOL user). Qu. 13: hundreds and hundreds, if not thousands.

Note 4. Qu. 12: 5 percent. Qu 13: 200 pages. The fraction of programs written in ALGOL is growing fast. The translator being developed is described in ALGOL.

Note 5. No count of the number of pages written in ALGOL is possible.

Note 6. Qu. 12: 20 percent. Qu. 13: 150 pages. This does not mean that we do not like ALGOL. Most programs are written for the IBM 650. Since there is no ALGOL compiler for that machine most programs are written in FORTRANSIT.

Note 7. The have been 1000 pages of BALGOL written by people who are also writing ALGOL 60. Other people at Stanford have written something like 50 000 pages of BALGOL.

Note 8. The absense of work reflects only the absense of a reasonably fast translator. We avait ALGOL on KDF 9. Deuce ALGOL is too slow except for very small problems.

Note 9. Statistics relating to teaching, programming and implementation of ALGOL 60 in this laboratory would be misleading. A translator for ALGOL is at present under construction.

TABLE OF REPLIES ON SIDE EFFECTS AND REFORMULATIONS.

	QUESTIONS 19 20 21	REFORMULATION (blank means reply = a) 10 12345	10 6789	10+ 12345	20 6789	20+ 12345	30 6789
NPL, England	abc		bb	b b	c	b	b
Zeiss, Germany	aca	b					
SMIL, Sweden	aac		b b b	b	b b		
Syst.Dev.Corp., USA	b-a	c				c	
Royal McBee, USA	caa	b	b	b			
Mayoh, USA	aac	d	c				b b
IDA-Princeton, USA	aaa		b				b
Facit Group, Sweden	baa	bc	b	c	c c	cc	c b c
Rehn, Finland	aaa						
Eng.El.Atomic, England	aac			c b			
Elliott ALGOL, Eng.	cbc	b		b b			b c
ALPHA, USSR	bac	c	bb	cc	c	bc	
Buchholz, Germany	ab-	c	c c	cb c	c	b	c b b
Oak Ridge, USA	baa	cbc	b c b	b c		bcc	c bcb
SSW-ZEF, Germany	aad	b	bb	d			c d
Dupont, USA	aab						
Rutishauser, Switzerland	b-b	c cc	h bhc	bc	c c	bcc	b cc
Kidsgrove, England	aac		b	b		ac	c
MNA-group, Sweden	aaa			c			
IAM Bonn, Germany	aaa						
Math.Cent., Holland	hff		h h		h		h h
Dutch PTT, Holland	b-f	c		db			b c
San Diego, USA	aa-	b	- h	-	b	h	--
Leeds Univ., England	abb						
ALCOR PERM, Germany	bcc	cbcc	c c b	bcc	c	ccc	c bcc
ALCOR Z22, Germany	bcc	cbcc	c c b	bccb	c c	ccc	c bcc
ALCOR 2002, Germany	bcc	cbcc	c c b	bccb	c c	ccc	c bcc
RCA EDP, USA	baa	c					
Bjork, Sweden	aaa						
Moore School, USA	abc	b	b			b	c
IPM Darmstadt, Germany	bcc	bcbcc	d c b	bccb	c c	ccc	c bcb
Computer Ass., USA	aab		c	c			
Regnecentralen, Denmark	aaa						
Univ.N.Carolina, USA	aaa						
RRE, England	bab						
Stanford, USA	baa						
RCA LAB, USA	aaa	b	c	b bd		dd	d d
ARF, USA	bbc	b cc	c cb	cc	c	b c	b c
NDRE, Norway	aac		b				
XTRAN project, USA	abc	bb d	bb		b	b	b b
Hockney, England	bbb						
NBS, USA	abc	bc	cb		dccd	b	b cb
Tubingen, Germany	caa			bb			
Siemens, Germany	bcc	c	d c b	c	c	b c	b c
Standard El., Germany	aac	c	c	bcc	c	bc	b c
AFCALTI, France	aa						
Burroughs, USA	bgg	c		b b	bbb b	b	b c
Saarland, Germany	aaa	c				c	
Remington Rand, USA	fcc	cc c	b c	c cb	b	bcc	c c
Cambridge, England	ab		c				c
Wegner, England	abc	b b b	bfb	bb		c bb	c c

f means a/b, g means (b+c)/2, h means reference to additional remarks (below).

Note to table of replies on side effects and reformulations.

Hockney, England: As to the reformulations I am indifferent to the exact manner of removing ambiguities. Far too much attention has been paid to these questions, which I consider to be of little importance compared with changes and extensions.

COMMENTS ON SIDE EFFECTS.

	QUESTION 20 Solution by definition		QUESTION 21 Solution by restriction		
	a or b Accept	c Oppose	a or b Accept	c Oppose	d Do not underst.
Totals:	40	6	25	19	1
QUESTION 19:					
a: Prefer definition Total 29	28	1	13	11	1
b: Prefer restriction Total 17	9	5	10	7	
c: Indifferent Total 3	3		2	1	

Alternative proposal for solution by restriction from Eng.El.Atomic, England:

Procedures called by function designators must not change the values of non-local variables, or contain go to statements leading out of the procedure body.

Alternative proposal for solution by restriction from ALPHA, USSR:

If a function designator calls a procedure declaration the body of which dynamically contains an assignment of a value to a global variable then the value of the variable is undefined outside the body.

If a function designator calls a procedure declaration the body of which dynamically contains a go to statement leading out of the procedure body then the transfer involved is undefined only if it does not lead to the program exit.

Comment from Rutishauser, Switzerland:

It cannot be denied that side effects of function designators are something that was not originally intended by the ALGOL-committee; indeed, section 3.3.3 clearly excludes them at least as formal parameters are concerned. Accordingly any attempt to make side effects legal by a change of 3.3.3 and defining the order of evaluation of primaries is a deep-carving change of ALGOL 60.

Comment from ALCOR Z22, ALCOR 2002, Germany:

The real problem touched by questions 19.20.21 is the following: Shall (A) the definition of the fundamental concept of expression be made to conform to the procedure concept introduced in ALGOL 60, or (B) should not this be the other way round.

The discussion in 2.1. assumes alternative (A) to be accepted: cf. last sentence on page 2 of questionaire 'these effects will make a more strict description necessary'.

Page 3, first paragraph after list of 'troubles': 'Of these 3 troubles those of nos. 1, 2, 5, 6, 7, and 8 may be cured in a fairly obvious way'.

The QUESTION (2.1) is formulated with considerable bias, since two proposals
(a) additional specification of the meaning
(b) restrictions on the use of the language
are presented on an equal footing. In fact, however, even (b) is less restrictive than the present status according to the ALGOL 60 report.

Section 3.3.3 says:
An arithmetic expression is a rule for computing a numerical value. . . For variables it the actual numerical value is the current value (assigned last in the dynamic sense)...

The first sentence implicitly excludes any assignment of values to actual or global parameters of procedures called by function designator in expressions. Furthermore it excludes jumps out of procedures called by function designators since in this case no value is defined. The second sentence excludes change of values of variables in the expression by function designators in the same expression since in this case 'assigned last in the dynamic sense' is undefined.

Therefore '2.1.2. Proposal for solution by restriction' means a change of the report, since it would e.g. permit use of one function designator leading to a jump out of the expression.

Proposal from RCA LAB, USA:

I propose the following compromise: A procedure may be declared to have nonexplicit side effects by replacing procedure by libertine procedure in the procedure declaration. If a procedure is to be used as a function designator, and is not declared to be libertine then it must not change the value of any nonlocal identifiers or go to any nonlocal labels. Nonlibertine procedures have the important property that all their effects are explicitly indicated by their actual parameter list. Adding this definition to the defining report requires the following changes
Section 5.4.1 CHANGE TO READ

<procedure declaration>::= procedure<procedure heading><procedure body> |
<type>procedure<procedure heading><procedure body> |
libertine procedure<procedure heading><procedure body> |
<type>libertine procedure <procedure heading><procedure body>STOP

ADD to TEXT

5.4.7 libertine procedures.

A procedure must be declared to be libertine if it is to be used as a function designator in an expression which should not be rearrangeable (cf. section 3.3.3) because the procedure contains nonlocal identifiers or labels. If this rule is violated the expression is undefined. A procedure should be declared to be libertine if it can be used as a function designator and either changes the values of identifiers nonlocal to the procedure or can go to labels nonlocal to the procedure. With this restriction it becomes possible to detect the conditions under which an expression is rearrangeable, and hence to have more efficient compiled programs. STOP
Also libertine must be added to the index. With this definition in mind, I am now ready to make my counter proposal.

I would change the proposal for solution by definition by adding the following

ADD TO TEXT

An arithmetic expression or a primary is said to rearrangeable if the order of evaluation of its primaries makes no difference other than that due to the finite accuracy of computer arithmetic. Equivalently an arithmetic expression or primary is rearrangeable if (a) it contains no libertine procedures (cf. Section 5.4.7) and (b) it contains no identifiers which are also used in a function designator as an actual parameter called by name and (c) it does not contain two function designators each of which has a parameter position specified to be a label or switch identifier.

The actual order of evaluation of a rearrangeable arithmetic expression or primary is undefined. Individual compilers may take advantage of this to improve the efficiency of compiled programs. As an example the arithmetic expression

$(A + Y)\uparrow 2 + A + Y) \times P(Y)$

is rearrangeable if the argument of procedure P is called by value and P is not a libertine procedure. In either case it contains the rearrangeable primary

$(A + Y)\uparrow 2 + A + Y)$ STOP

Reason for rereformulation. As I see it the major arguments expressed in section 2.1 of the questionaire are:

a) Side effects are good in some circumstances. It may be true that they were not originally envisaged, but they seem useful, and will probably grow more useful as experience with them increases.

b) Because of side effects the order of evaluation of expressions can make an important difference. Because of this the compiler cannot choose the order of evaluation so as to make the object programs more efficient. The price we must pay for the ability to have side effects namely ieffficient programs, is simply too high.

The above compromise still allows one to use side effects with the full generality desired by the proponents of side effects. In addition it is made easier to detect conditions which are sufficient to allow rearrangement of expressions (by implication of section 3.4.3 similar rules hold for Boolean expressions). In almost all cases of interest it becomes possible to do what ever rearrangement is possible, even if it is only possible for part of an expression.

Webster defines libertine as 'one who is without restraint'. If any one can think of a better word I would be happier.

I am afraid it may be desirable to insist on specifications. If a is a formal parameter called by name then $A\uparrow 8$ is not rearrangeable to $A\uparrow 2\uparrow 2\uparrow 2$, since A may be a function designator with no arguments.

Comment from Burroughs, USA:

Question 19.

In many cases it is wise to take restrictions out of a language, so the user needn't have to spend much time learning what he can't do. In this case, though, it seems the advantage of allowing expressions which can have different meanings because of the order of evaluation is slight indeed. If someone wants to use this feature, it will not be clear to others what he is doing. It seems the rules for the order of evaluation are more difficult to learn than the restrictions. And it is difficult to explain to someone what good he will be getting after taking the trouble to learn all the rules. Furthermore, this will slow down the machine language programs in all expressions, while the actual number of times when this will be used to advantage is very very small. The choice here seems to be clear: if we allow expressions which are ambiguous unless order is specified, we must specify an order; if we don't allow such expressions, we needn't specify an order. The order mentioned in this proposal is the order we had interpreted already from the original ALGOL 60 report. But, we think, people who would like to see these fancy rules of order put in the language tend to think of ALGOL as a language to play arround with and theorize about but not as a tool. Let's not say, 'Well, suppose the man writes such-and-such; how nice. We haven't ruled this out. Now what is the most general definition we can give to this construct.' Rather let us stick to useful things.

Question 20-21.

Although we much prefer the solution by restriction, we have answered '(b+c)/2' because we are shocked that 'kind 2' function designators are allowed.

To quote your report, 'Function designators define single numerical or logical values.' If a function designator contains a 'go to' leading out of the procedure, it has defined no value at all. This is not then a function designator. Allowing such seems also to be out of the spirit of the report, for in an expression the 'if-then' always must be followed by 'else' so that a single value is always defined.

Thus, the solution by restriction we favor is simply restriction (1) here; function designators of kind 2 are patently illegal.

Comment from Remington Rand, USA:

Although we have answered Q19 as a) and b) we really feel that some additional comment is needed here.

After initial difficulty in reading the ALGOL report, we found the Backus normal form to be an excellent means for defining syntactical rules, but that, in general, the semantic explanations were too concise to be really helpful in difficult situations. For example: sentence 1 in section 5.2.5.

Our feeling is that ambiguities and obscurities in the language should be resolved, as far as possible, by alterations to the syntax, and only when this is found to be impossible should additional definitions or restrictions be employed.

We are aware that many groups interested in ALGOL are opposed to this view and regard the ALGOL syntax as a sort of holy writ which should be altered only as a last resort. We consider that this view ignores the fact that the ALGOL syntax is not perfect and allows, by default, many features in the language which were not intended. We hope to prove this point in some counter proposals to the reformulations suggested in the questionaire.

Comment from Cambridge, England:

We are strongly opposed to any language that allows side effects in the evaluation of expressions. However, in order to achieve these principles we feel that drastic revisions to ALGOL 60 would be and are necessary. Our replies to further questions must therefore be taken as opinions on ALGOL 60 itself and not as proposals for a future language.

Comment from Wegner, England:

The note facility (Strachey and Wilkes) should probably have been mentioned in connection with side effects. The note facility may be used either as indicated by Strachey and Wilkes to indicate greater generality where necessary, or as indicated by Dijkstra to indicate lesser generality. Strachey and Wilkes regard the most common usage as 'normal' and therefore not requiring a note, whereas Dijkstra prefers to regard the most general usage as 'normal'. As pointed out by Dijkstra, the former approach is subjective and leads to floating semantics as common usage changes. I would therefore agree with Dijkstra in supporting a policy of treating the most general case as the normal one, and in indicating by means of notes, restrictions that permit greater efficiency of implementation.

COMMENTS ON REFORMULATIONS 1 - 30.

Reformulation 1: Verbal definition of block and program.
a: 42, b: 8, c: 1, d: 0.

Alternative proposal from the Facit Group, Sweden, regarding
REFORMULATION 1, PART 1
. . . Each declaration is attached to and valid for one block. A program is a self-contained block, i.e. a block which is not contained within another statement and makes no use of other statements not contained within it. STOP
Reason: See reformulation 9.

Alternative proposal from IPM Darmstadt, Germany:

The sentence proposed in part 1 states that 'sequences of statements may be combined into .. blocks by insertion of statement brackets'. This is not true since more than inserting statement brackets is necessary to create a block. We therefore propose the following reformulation:
Part 1: section 1, end of 3rd paragraph
ADD:
(i.e. the basic symbols begin and end). STOP
Part 2: section 2, end of 4th paragraph.
CHANGE TO READ:
... defining a function. Declarations are combined with sequences of statements to form a block by using statement brackets. Each declaration is valid for the block it is attached to only.
A program . . STOP
We agree to your reformulation of the 5th paragraph.

Comment from RCA LAB, USA:
I would add to the text of the reformulation
ADD TO TEXT Such a compound statement or block is syntactically equivalent to a statement. STOP
Reason for change. The definition of a program is unclear unless one knows this.

Comment from Remington Rand, USA:

Part 1: Obviously the original wording in the ALGOL report is correct as it stands, although it is not complete. However, the reformulation is incorrect as no sequence of statements can be combined into a block by insertion of statement brackets.

Part 2: Why not: - 'A program is a self-contained block etc.', or is this too restrictive for some. See also Reformulation 9.

Comment from Wegner, England:

Reformulate definition of 'program' as a single sentence at the end of the fourth paragraph:

'A statement which is not contained in another statement is called a program'.

Reason: The definition suggested in the questionaire is inacurate in that it does not take global symbols (sin, cos) into account. It is also muddling since it contains both essential and inessential information.

Reformulation 2: The comment conventions.
a: 34, b: 4, c: 13, d: 0.

Comment from Rutishauser. Switzerland:

This would be a change of the AR, but still fails to cover cases like (⟨ and ⟩ stand for string quotes)

printext (⟨ comment nonsense⟩) end;

On the other hand the change is quite unneeded, since it is of course understood that a program is read from left to right; therefore if in doing so one comes to one of the symbols comment or end or ⟨, then the reader has to disregard (or to take special action in case of a string) what follows until the corresponding terminating symbol. Thus the examples given by P. Naur (and M. Woodger) are all unambiguous.

Alternative suggestion by W.W. Youden, NBS, USA:

I can't help feeling that the following is what was intended for the 'comment' convention and that it is preferable to the syntax change recommended by M. Woodger.

Reformulation 2 Section 2.3, last paragraph.

ADD TO TEXT:

and conversely, that any of the three sequences of symbols shown in the left-hand column may, in any occurrence outside of strings or outside of <any sequence not containing . >, be replaced by the symbol shown in the right-hand column without any effect on the action of the program.

STOP

In other words, when in the course of scanning an ALGOL program from left to right, and either '; comment' or 'begin comment' is scanned, thereafter everything including comment, begin, end and else is ignored until a ';' is reached. Similarly, when 'end' is scanned, thereafter everything is ignored until either 'end' or ';' or 'else' is reached.

If one uses the above reformulation, the three ambiguous examples can only be interpreted as follows:

Example	Interpretation
begin comment begin comment A; P; Q end	begin P; Q end
; comment begin comment A; P;	; P;
end begin comment A; P;	end; P;

Suggestion from Zeiss, Germany:
Change to read:

The sequence of basic symbols:		is equivalent to
; comment <any sequence not containing	begin or ;>	;
	<begin or ;>	
begin comment < - - -	begin or ;>	begin
	<begin or ;>	

STOP.

with these conventions the examples supplied by Woodger are in each case reduced to
begin P; Q end
; P ;

With the third convention given in reformulation 2 the example supplied by Dijkstra is reduced to the sequence end begin Klaus; and this is wrong because the next symbol after an end must be an else or an end or a ; I don't know how to correct this.

Alternative proposals from the Facit Group, Sweden regarding
Section 2.3, last paragraph, CHANGE TO READ:

By equivalence is here meant that the action of a program is unchanged if the following transformation is made: Read the program from beginning to end and replace any occurence outside of strings of any of the three structures of the left hand column by the symbol shown in the same line of the right hand column. STOP
Reason: The comment conventions are quite good as they stand and only a clarification is necessary.
Section 2.3, line beginning with end, CHANGE TO READ:

end <letter string>	end STOP

Reason: It is a frequent error to forget the semicolon (;) after an end, thus destroying the following statement by mistake. In fact, we have been forced to let our compiler give error printouts on all other end-comments than letter strings.

Counter proposal from the Dutch PPT, Holland:
Section 2.3, last paragraph
After .. . action of the program
ADD TO TEXT: The 'comment' situation encountered first in the text when reading from left to right is having precedence over later 'comment' situations contained in the sequence to be disregarded. STOP
Reason for reformulation 2: This does not change the original text. It ly supplies the missing precedence rule.

Comment from Burroughs, USA:

This does not seem like a reasonable solution. First, people are writing comments using other than 'basic symbols.' For example, look at example 2 at the close of the ALGOL report, where Y_k' is used. (Several algorithms have even let semicolons slip into the comment.)

One needn't prohibit the word 'begin', merely say each comment runs to its next delimiter. In syntax,

<semicolon>::= ; | ; comment <string of symbols except ;> ;

<begin>::= begin | begin comment <string of symbols except ;> ;

Then use <semicolon> and <begin> as constructs in the remainder of the report. This part of the report has been incompatible with the rest anyway, and we think should have been put into syntax form originally.

There doesn't appear to be any reason to keep 'begin' from the words following 'end'. However we don't see the need for such general comment conventions following 'end' anyway, since it seems to hurt syntax checking. A man leaves out a semicolon after the word end, and he's lost a whole statement with no way of knowing it. We strongly would prefer

<end>::= end | end <letter string>

or at most end<identifier>.

Comment from Remington Rand, USA:

We see no reason for these changes, especially the third sequence. We suggest that the original comment conventions are entirely adequate.

Reformulation 3: Verbal definition of scope.
a: 42, b: 7, c: 1, d:1.

Comment from Wegner, England:

a. The term 'quantity' is usually associated with a numerical magnitude, and seems to be the wrong word for denoting objects which may be non-numerical. The term 'object' or 'value' would seem to be more appropriate. Whichever term is used, its meaning should be explicitly defined e.g. in section 2.4.3.

Suggested Reformulation: Add at end of first paragraph of section 2.4.3.

Identifiers are symbolic names which may designate either numerical quantities (simple variables, array elements), or non-numerical program constituents (labels, switches, procedures, formal parameters). The term 'quantity' is used to denote any object designated by an identifier.

b. Scope should be associated with identifiers rather than with the quantities designated by identifiers.

Suggested Reformulation:

2.7. Identifiers, Declarations and Scopes.

An identifier is introduced into a program by means of a declaration. Declarations define the 'scope' of the identifier associated with a given quantity.

The scope of an identifier is the set of statements over which the identifier associated with a given declaration can be used. An identifier is said to be defined in statements within the scope of the identifier and undefined elsewhere.

Reformulation 4: Evaluate subscripts from left to right.
a: 39, b: 0, c: 12, d: 0.

Comment from ALPHA, USSR:
This reformulation is implied only by side effects. If they will be deleted from the language in the spirit of AB 14.2.2.2. then this stipulation will become unnecessary.

Comment from Remington Rand, USA:
We object to this on the following grounds. An expression is a rule for computing a value. A formal language fails in its purpose if an expression can conceal a statement. We do not feel that the suggestions in 20 or 21 are satisfactory solutions to the problem. We realize that alterations to the syntax and semantics of function designators, procedure statements and procedure declarations which could resolve the problem, would require considerable time and effort. We feel strongly, however, that this approach should be taken.

Reformulation 5: Parameters called by name in function designators.
a: 42, b: 2, c: 6, d: 1.

Comment from ALCOR, Germany:
The original reading of the report shows clearly the intention.

Comment from Wegner, England:
Reformulate as follows:
... when applied to the actual parameter part of the function designator given in the expression.

Reformulation 6: Definition of the integer divide.
a: 37, b: 5, c: 5, d: 2.

Comment from SMIL, Sweden:
$$a \div b = \text{sign}\,(a/b) \times \text{entier}\,(\text{abs}(a/b))$$
If b is a factor of a, the above arithmetics could give an unwanted result. As a/b is an expression of type real, this implies that a/b only approximates an integer (cf. 3.3.6). Consequently the result could be wrong by one unit.

Comment from Rutishauser, Switzerland:
That division by 0 is undefined is trivial and need not be mentioned in the AR.

Comment from Dijkstra, Math. Cent., Holland:
I should like to have added a warning that in this particular definition the 'a/b' represents the exact, mathemetical quotient.

Comment from Remington Rand, USA:
We do not object to this change but don't really see the reason for it. Bjork's comment seems trivial - why is it important that a and b are called by value? Surely the standard functions won't by 'sneaky'.

Reformulation 7: Definition of relation.
a: 48, b: 1, c: 1, d: 0.

Comment from RCA LAB, USA:

I favor instead the suggestion of AB 12.5. With so much fuss about removing minor restrictions I see no reason for adding another one. I propose instead the following amplification of section 3.4.3 of the defining report. It is more in the spirit of section 3.3.3 and reformulation 12.

CHANGE TO READ ... given for arithmetic expressions in section 3.3.3. In particular in the more general Boolean expressions which include if clauses, one out of several simple Boolean expressions is selected on the basis of the actual values of the Boolean expressions of the if clauses. This selection is made as follows: The Boolean expressions of the if clauses are evaluated one by one in sequence from left to right until one having the value true is found. The value of the original Boolean expression is then the value of the first Boolean expression following the Boolean which was found to be true. (The largest Boolean expression found in this position is understood). The construction

else <simple Boolean expression>

is equivalent with the construction

else if true then <simple Boolean expression> STOP

Comment from Remington Rand, USA:

We entirely agree. This kind of change greatly improves ALGOL and is not 'restrictive', see Reformulation 12.

Reformulation 8: Switch designator with subscript outside range.
a: 39, b: 4, e: 8, d: 0.

Comment from Rutishauser, Switzerland:

The wording

' .. have no other effect than the evaluation of expressions ..' is certainly sufficient and understandable for both the general user and those who like side effects.

Comment from SMIL, Sweden:

As, owing to reformulation 4, section 3.1.4.2 has changed, section 3.5.4 should not refer to 3.1.4.2. Section 3.5.4. ought to be formulated so that it does not refer to section 3.1.4.2.

Alternative proposal from Oak Ridge USA.

Change 4.3.5 to read

A go to statement is undefined if the designational expression is undefined.

Comment from Wegner, England:

Change to read:

4.3.5. Go to an undefined switch designator.

If the value of the designational expression is not an integer in the range 1 to n, where n is the number of entries in the switch list, then the only effect of the go to statement will be that which might have been induced by the evaluation of expressions.

Reformulation 9: Syntactic definition of program.
a: 38 1/2, b: 7.5, c: 4, d: 0.

Comment from Rutishauser:
I am strongly opposed to the proposed reformulation, on grounds of the policy that vested interests in ALGOL 60 should be protected. We have always understood that in the introduction of the AR 'compound statement' includes also blocks and therefore built our compiler on the basis that a program begins with begin, possibly preceded by a label. I propose
<program>::= <compound statement>|<block>
Incidentally solving a problem which can be described by a single statement needs no computer at all and therefore I do not see why such an effort should be undertaken to make programs not beginning with begin possible.

Alternative proposal from the Facit Group, Sweden:
<program>::= <block>
Reason: The proposed change <program>::= <statement>; STOP would make all present ALGOL programs illegal, since they do not contain a final semicolon (;). Moreover, it seems unnecessary in practice to have programs that do not contain declarations.

Alternative proposal from ALPHA, USSR
<program>::= <unlabelled block>
Reason: more precise formulation.

Alternative from SSW-ZEF, Germany:
<program>::= <statement> STOP

Comment from Kidsgrove, England:
Surely this should read:
ADD TO TEXT: <programme>::= <unlabelled statement>; STOP
in order to conform to Reformulation 10.

Comment from Dijkstra, Math. Cent., Holland:
I prefer
<program>::= <unlabelled block>|<unlabelled compound>
Motivation: By forcing the program to start with the symbol 'begin' and to end with the corresponding 'end' we have a uniform rule to establish the lexicographical extent of the program. Otherwise it is difficult to distinguish between, say,
'if 3<4 then cos(5)' and 'if 3<4 then cos(5) else cos(6)'
This becomes a little bit more marked if we replace 'cos' by the procedure identifier of the kind 'print'. The last sentence of Reformulation 10 can then be omitted (viz. 'The statement of a program must be unlabelled since it has no embracing block.').

Remark from San Diego, USA:
The definition 9 does not appear to be consistent with reformulation 1 in that it does not specify that the statement is self contained. Both reformulation 1 and reformulation 9 restrict the expansion of a program at execution time.

Comment from Remington Rand, USA:
We propose:
<program>::= <block> STOP

Counter proposal from Cambridge, England:
<program>:= <unlabelled block>|<unlabelled compound>
Reason: Avoids contradiction with Reformulation 10.

Comment from Wegner, England:
A program is an unlabelled statement - see Reformulation 10.
Note: I would much prefer the possibility of a program being labelled, so that it can be referred to by another program. However, this is against current ALGOL 'scope' philosophy.

Reformulation 10: Local behaviour of labels.
a: 35, b: 13, c: 3, d: 0

Comment from SMIL, Sweden:
As we can see, the first sentence to the reformulation (Labels behave as though they were declared in the head of the innermost embracing block in which they occur attached to a statement)still leaves obscure the question about the scope of a label.
Example: begin real r1; ...; L1:L2: begin real r2; ... end .. end
According to the reformulation L2 should behave as though it were declared in the block L1:L2: begin real r2; ... end.
We suggest that the definitions in 4.1.1. should be changed in such a way that a labelled block is a compound statement:
<compound statement>::= <unlabelled compound> |
<label>:<compound statement> |<label>:<block>
<block>::= <unlabelled block>

Alternative from ALPHA, USSR:
(1) add to text ... innermost embracing block or compound statement ...
(2) delete the last sentence.
Reason: We think it would be more logical.

Remark from Oak Ridge, USA:
There is no reason for not permitting programs to be labeled. This may be desirable for programmer's information.
Remark: Define 'attached to a statement.'

Alternative from SSW-ZEF, Germany:
2) ADD INSTEAD:
... Labels behave as though they were declared on the head of the innermost embracing block in which they occur attached to a statement. In this context the body of a procedure declaration as well as the statement following a for-clause will act as a block, whether it has the form of a block or not. Labels of a program are without consequences for the run of the program. STOP

Remark from H. Rutishauser, Switzerland.
I am opposed to reformulation 10 but would support it if the last sentence were deleted. The reasoning that a label cannot be in front of a program (this incidencially would also touch vested interests) has a dangerous parallelism in the case of procedures which are not declared in the outermost block of a program, which might be forbidden with the same reasoning. But just such procedures which are declared outside a program play a very important and useful role in ALGOL.

Remarks from Computer Ass., USA:

Our only objection is to the last sentence of the addendum, concerning labelling the entire program. For the publication of algorithms, this point has little relevance. From the point of view of including ALGOL within a larger structure of programming languages, there are advantages to naming a program by attaching a label to its entire statement (which presumably is a block). Operationally, this serves to assign a name to the program, which enables the larger system to refer to it. Conceptually, the labelled program appears as a sub-block within (rather than some kind of appendage to) some 'universal' block. At the time of translation, the 'universal' block is the one in which the library procedures are 'declared'; at the time of execution, the 'universal' block is the scope of the computer's control program. Semantically, a go to the program name from within the program must, for consistency, represent a new (recursive) initial entry into the program, with whatever (re-)initialization activities are required.

Comment from Wegner, England:

The current reformulation is an improvement. However, section 4.1.3 as a whole is still rather obscure and could be improved. The last paragraph is particularly cryptic. The following paragraph would be clearer:

When a statement within a block is itself a block the rules which determine scope are quite subtle. Consider for instance a block A embedded in a block B. An identifier which is declared in block A is local to block A and non-local to the enclosing block B. If an identifier of the same name is declared in block B, the two identifiers are completely disjoint in their scope. The identifier declared in B has a scope that is lexicographically non-compact, since the block A creates a 'hole' in the scope.

Reformulation 11: Go to into compound statements are allowed.
a: 44, b: 2, c: 3, d: 2.

Remark from Engl.El.Atomic, Eng.:

Unnecessary.

Comment from ALPHA, USSR:

We object to this reformulation because of the correction to reform. 10.

Remark from Tubingen, Germany:

For the sake of clarity one might add

within a block
following
statements.

Comment from Wegner, England:

This addition is welcome. However, the distinction between blocks and compound statements in this respect should be noted also in a less specialised section, e.g. as part of Reformulation 10 or as part of the introductory discussion at the beginning of section 4.

Reformulation 12: Precedence of conditions and for clauses.
a: 31, b: 17, c: 3, d: 0.

Comment and alternative proposal from ALPHA, USSR:

We object this reformulation because we think it is not in the spirit of ALGOL. The matter is that we obtain a rule of syntactic analysis which is not implied by metalinguistic formulae only as we have in all other syntactical constructions. We now more incline to the Woodger's proposal.

Remark from Oak Ridge, USA:

We oppose only the wording.

Comment from Kidsgrove, England:

No doubt this is correct and expressed concisely, but it is not at all elegant. Why not adopt the so-called restrictive solution. This imposes only a rule of notation, destroys none of the pwer of the language, and has the great advantages of simplicity and clarity.

Comment from Computer Ass., USA:

So far as we know, this is the only place where the syntax of ALGOL is ambiguous, in the sense that a single well-formed statement of the source program might be decomposed, according to the syntactic rules, in two essentially different ways (with correspondingly different semantic effect). Esthetically, we would like to see the syntax amended to remove this ambiguity, rather than adding a patch to the prose. If this constitutes too 'major' a rewriting, the ad hoc rule will do, and the present formulation is acceptable.

Alternative from RCA LAB, USA:

I would change the reformulation as follows:

CHANGE TO READ

...matching begins and ends. In other words then and else are analogous to parentheses. A then is like an open parenthesis, and either an else or a semicolon is the associated closed parenthesis. In determining the else or semicolon associated with a given then one follows the usual rules for hierarchies of parentheses with one difference. One ignores any else or semicolon contained between matching begins and ends. STOP

Reason for reformulation. Improved clarity, and closer analogy to other rules of the language.

Remark from Tubingen, Germany (translated from German):

It seems to me that reformulation 12 makes the counting process difficult; I would therefore prefer that brackets (begin end) were required.

Suggestion from Prof. Harry, E. Goheen, in collaboration with G.A. Bachelor, D.W. Digby, P.H. Hartman, and S.P. Ogard, Oregon State University, Corvallis Oregon:

We suggest the following changes in syntax:

1. In sections 4.1.1 and 4.5.1, change the definition of
<unconditional statement> to read:
<unconditional statement>::= <basic statement>|
<unconditional for statement>|<compound statement>|<block>,
2. In section 4.5.1, also change the definition of
<conditional statement> to read:
<conditional statement>::= <if statement>|
<if statement>else<statement>|<conditional for statement>
3. In section 4.6.1, omit the definition of <for statement> and replace it with the following two definitions:
<unconditional for statement>::= <for clause><unconditional statement>|
<label>:<unconditional for statement>
<conditional for statement>::= <for clause><conditional statement>|
<label>:<conditional for statement>

In the semantics, it is understood that the term 'for statement' refers to both conditional and unconditional for statements.

Comment from Burroughs, USA:

This rule is clumsy as stated, and it would be better to make the syntax agree with this rule. One way to do this is:

<unconditional statement>::= <basic statement>|
<for statement 1>|
<compound statement>|<block>|
<if statement> else
<unconditional statement>
<conditional statement>::= <if statement>|<for statement 2>|
<if statement> else
<conditional statement>
<for statement 1>::= <for clause><unconditional statement>
<for statement 2>::= <for clause><conditional statement>

This seems to give all the generality you are looking for while also giving all the speed we are looking for.

Comment from Remington Rand, USA:

We strongly object to this proposal (see initial comments) and recommend the adoption of the proposal in 10.1.3.2. Moreover we do not agree that the proposal in the questionaire 'has the advantages of analogy with other rules of the language and a minimum of changes to the wording and examples of the report.' The proposal in 10.1.3.2 inserts the word 'unconditional' and the delimiters begin and end - three minor changes. (begin and end must also be inserted in the first for statement following the label BB in example 2 of the report).

Our main reason for supporting the proposal in 10.1.3.2 is this: the report insists that a statement following a then should be unconditional for obvious reasons. However, the report defines a for statement to be unconditional. Unfortunately the for statement as defined is either conditional or unconditional depending on the statement following the do. We feel that the proposal of 10.1.3.2 is not 'restrictive', is sensible, and is in the spirit of sections 4.5.1 and 4.1.1. Moreover, we feel that this change is much more important than that proposed in Reformulation 7.

Comment from Wegner, England:
It is not at all clear from the report why the substitution of statements for S1, S2, S3 or S4 should give rise to ambiguity. Please illustrate with an example.

Reformulation 13: Side effect of empty conditionals.
a: 42, b: 1, c: 8, d: 0.

Alternative from SMIL, Sweden:
The second form of a conditional statement is <if statement> else <statement> according to 4.5.1. To avoid misunderstanding we suggest that the reformulation should be: In the case of the second form of conditional statement if none of the Boolean expressions B1, B2 or B3 of the if clauses is true, the whole conditional statement will have no effect other than that which might be induced by the evaluation of the Boolean expressions.

Alternative from Rutishauser, Switzerland:
I would support 'In the case of the second form of the conditional statement, if none of the Boolean expressions of the if-clauses is true, the whole conditional statement will have no other effect than the evaluation of Boolean expessions.'

Remark from Computer Ass., USA:
A quibble about phraseology: The wording 'second form of conditional statement' could be taken to refer, not to the second illustrated form at the beginning of the paragraph, but rather to the second alternative definiens: <if statement> else <statement>, in which case the 'clarification' suggested is false. Suggest rewording equivalent to: 'In the case of a conditional statement in the form of the second illustration above.'

Reformulation 14: The definition of the step-until element.
a: 35, b: 5, e: 10, d: 0.

Alternative from Eng.El.Atomic, England:
As in Questionnaire, with last sentence
CHANGE TO READ
.. same type as V and S2 of the same type as B. STOP
Reason for re-Reformation.
Saves unnecessary type transfers.

Alternative proposal from Oak Ridge, USA:
Change to read:

```
    v:= A;
    s1:= B;
    s2:= C;
L1: if (v-s2)×sign(s1) ≤ 0 then
    begin
        S;
        v:= v + s1;
        go to L1
    end;
```

where v is .. in the program. The evaluation of A, B, and C has no effect on v.

Counterproposal from the MNA-group, Sweden:
CHANGE TO READ:
a) If V is a simple variable or a formal parameter corresponding to a simple variable as actual parameter (a formal parameter called by value being regarded as a simple variable):

```
S1:= V:= A;
S2:= B;
S3:= C;
L1: if sign (S2)×(S1-S3) > 0 then
    go to Element exhausted;
    Statement S;
    V:= S1:= S1 + S2;
    goto L1;
```

b) if V is a subscripted variable
V = D[I1, ...In]
or formal parameter called by name corresponding to a subscripted variable:

```
    Z1:= I1;
    ..
    Zn:= In;
    S1:= D[Z1 , ..  Zn]:= A;
    S2:= B;
    S3:= C;
L1: if sign(S2)×(S1-S3) > 0 then
    go to Element exhausted;
    Statement S;
    D[Z1, ... Zn]:= S1:= S1 + S2;
    go to L1;
```

Here S1,S2,S3 and Z1, ..Zn are auxiliary variables. The type of S1, S2 and S3 is = type of V in both cases and Z1, . Zn are of type integer. V is the controlled variable .. in the program.
Reason for change: A feeling that in many cases changes in the values of B,C,V and the subscripts of V induced by the execution of statement S are unwanted and should be suppressed. If changes are wanted the more general while-element can be used. In this way the step-until-element can also more easily be given a fast implementation.

Comment from RCA LAB, USA:
Rather than S1 and S2 I prefer identifiers with mnemonic value, say VTEMPORARY and BTEMPORARY.

Comment from Burroughs, USA:
It seems a shame that S2 is of type real. This is very inefficient on a machine, if the transfer functions have to be applied. For example consider the most common case 'for i:= 1 step 1 until n', where i, n are integer, it necessitates 3 transfer functions each time through. Was this decision reached because the type of a call-by-name expression, or of 'i↑j', are not defined at compilation time. It doesn't seem to solve those problems anyway. We suggest S2 has the same type as B.

It would also be lots more efficient if the sign of B had to be evaluated only once. Therefore, we would replace this code with the effect of the following code:

```
switch SW:= if S1 < C then Element exhausted else L, L, if S1 > C
     then Element exhausted else L;
       S1:= V: = A;
       S2:= sign (B);
       go to SW[S2 + 2];
L: Statement S;
       S1:= V:= V + B;
       go to SW[S2 + 2];
```

Comment from Remington Rand, USA:

We oppose this only on the grounds that it is pointless. We would like to see an ALGOL in which

sign (B)×(V - C) = (V - C)×sign (B)

(see comments on Reformulation 4).

Also Bring's comments seem to be needlessly pedantic. I think most readers interpret the ALGOL statements in section 4.6.4.2 as descriptive in the same way as the copy process for procedures. As such, the original statements are more concise.

Reformulation 15: The equivalent effect of a procedure statement.
a: 45, b: 4, c: 1, d: 1.

Comment from Remington Rand, USA:

The new description is hardly different from the old and does not answer Bottenbruch's question. Either leave the original sentence unchanged or describe in detail the workings of a recursive procedure.

Reformulation 16: The order of value assignment.
a: 42, b: 1, c: 8, d: 0.

Comment from Burroughs, USA:

You seem to have an implicit rule which ought to be made explicit; everything in the value part must be specified in the specification part. (We like this rule.) It would be clearer if one wrote, e.g.,

'real value X; integer array value Y; label value L,' etc.,

don't you think.

Reformulation 17: Types of value parameters
a: 46, b: 4, c: 0, d: 1.

Comment from SMIL, Sweden:

The last sentence of section 4.7.3.1 is somewhat unclear when considering formal parameters being labels called by value.

Comment from Burroughs, USA:

Here that implicit rule we just mentioned seems to be implicit again, so we hope you add that rule to section 5.4.5.

Comment from Remington Rand, USA:

You might make an implicit rule explicit here, and/or in section 5.4.5 by adding.

'All formal parameters called by value must be included in the specification part.'

Reformulation 18: Strings as actual parameters.
a: 46, b: 1, c: 3, d: 0.

Counterproposal from NPL, England:
CHANGE TO READ:
4.7.5.1. Strings supplied as actual parameters in procedure statements can only be used by procedure bodies expressed in non-ALGOL code (cf. section 4.7.8). STOP
Reason for counterproposal: Section 4.7.5.1. was false (see AB 13.2 Woodger). Reformulation 18 stated that the string parameters must be used, and and was unnecessarily lengthy.

Alternative from Dijkstra, Math.Cent., Holland:
'.. in further procedure statements or function designators, because eventually they can only be used.. '
Motivation: The following procedure is crazy, but it should be legitimate:

```
'procedure FUNNY(a,b); string b; real a; a:= a + 1 '
```

Comment from Burroughs, USA:
Change to 'in further procedure statements or function designators', etc.
There seems to be no way for the translator to tell when the procedure body is written in code. We suggest the declarator 'code procedure' for this.

Reformulation 19. Strings can only be called by name.
a: 49, b: 1, c: 1, d: 0.

Reformulation 20: Call of designational expression by value.
a: 41, b: 2, c: 7, d: 1.

Suggestion from SMIL, Sweden:
Expressions cannot be called by value.

Comment from ALPHA, USSR:
We strongly object this proposal. ALGOL has no mechanism to assign a value of a designational expression to any identifier.
Example:

```
begin
  procedure GO TO (1); value 1; begin go to 1 end;
  GO TO (R); R:
end of example
```

How can you explain the work of the program after the replacement of the procedure statement by the procedure body in terms of basic statements.
Incidently, there is a contradiction with the text of reformulation 17 because formal parameters which are designational expressions and called by value are not given any type.

Comment from Rutishauser, Switzerland:
Section 4.7.3.1 is cristalclear since it requires an explicit assignment which would be impossible for labels. Thus formal parameters corresponding to labels cannot be called by value. Anything else would be a change of ALGOL 60.

Comment from Burroughs, USA:
Add a statement that if a designational expression called by value reduces to a switch whose subscript is out of bounds it is undefined.

Reformulation 21: Verbal characterization of declarations.
a: 50, b: 0, c: 1, d: 0.

Comment from Wegner, England:
Quantities are normally associated with numerical magnitude. Some other term such as 'objects' should be used if a modification is introduced.
However, the original formulation seems preferable. Declarations serve to associate properties with identifiers rather than with the objects named by identifiers.

Reformulation 22: .. identifiers lose their local significance.
a: 50, b: 1, c: 0, d: 0.

Comment from Burroughs, USA:
This doesn't seem to explain it any better than before. How about '... all identifiers which are declared for the block lose the significance they had in that block.'

Reformulation 23: The meaning of own.
a: 32, b: 11, c: 6, d: 1.

Comment and question from ALPHA, USSR:
We support the idea but we do not understand one point: 'Whether they remain accessible depends on whether the exit is made to a place within the scope of the identifiers'.
Example:

```
begin integer n,m;
  n:= 3;
  begin own integer n;
    n:= 5
  end;
  m:= n
end of example
```

Is m equal to 5. If so it is bad, if not we don't understand the sentence mentioned.

Remark from Rutishauser, Switzerland:
The main problem, namely whether different calls of the same procedure (having own-declared variables in its body) define the same or different sets of own variables, is still not quite settled by the present lengthy wording (I would prefer the same set of own-variables).

Comment from Kidsgrove, England:
Although we have, for the purposes of compilers, accepted the definitions given here as they stand (except for dynamic own arrays), we are very unhappy about the whole thing. The new proposal is something which is not in the original intention of the report, seems to be justified by no practical applications (I refer to own in recursions), but is merely a system which can be implemented. We think that the idea of own needs much more close examination; we would not like to see it abandoned altogether.

Comment from Computer Ass., USA:

We thoroughly approve of the doctrine as stated concerning own variables. However, the sentence: 'Thus every entry into a block .. will make the same set of values of own variables of this block accessible' still allows the absurd interpretation facetiously proposed by Ingerman. It should perhaps be amended to make clear that 'every' means 'every time, within a single execution of a single instance of the program'.

Comment from Remington Rand, USA:

We do not see how an exit from a block can be made to a place which is within the scope of the identifiers of the block. We feel that this section should be re-written in a more positive fashion with the sort of example found in Bottenbruch's primer. Much of the semantic difficulty in this passage and in the ALGOL report can be traced to the use of phrases such as 'with regard to' and 'with respect to'.

Comment from Wegner, England:

Section 5, fourth paragraph. Change to read:

Declarations for simple variables and arrays may be marked with the additional declarator own. Variables marked with own have their values preserved between successive activations of the block in which the declaration occurs, whereas values of non-own variables are lost between successive entries to the block. Local non-own variables must be recomputed from non-local variables during each entry to the block, whereas own variables have their values carried over from previous activations of the block. When a block is called recursively the values of own variables are transmitted between successive levels of activation.
See also note to QUESTION 37: DELETION OF OWN FROM LANGUAGE.

Reformulation 24: Admit non-constant array bounds in outermost block of program
a: 36, b: 3, c: 11, d: 1.

Comment from Burroughs, USA:

You haven't mentioned explicitly in the report that standard procedures other than analytic and transfer functions may exist, have you. This should be mentioned.

Proposal from Saarland, Germany:

Delete:
'Consequently'

Comment from Remington Rand, USA:

Although a programmer may wish to use parameters from an input medium to specify subscript bounds, we do not see the practicality of using function designators to read these values. We would prefer to see the original sentence left in the report, even though it does 'restrict' the language. We cannot honestly see the point of Woodger's device. Surely a programmer would require the subscript bounds elsewhere in his program and would normally employ a procedure statement to assign the input values to variables. The array could then be declared in an inner block.

Comment from Wegner, England:

Section 5.2.4.2 is unsatisfactory when the deletion has been made. The following sentence could be added:

However, subscript expressions may be made variable even in the outermost block by the use of global identifiers of code procedures (e.g. procedures associated with input and output).

Reformulation 25: Evaluate bound expressions in order.
a: 40, b: 0, c: 10, d: 0.

Suggestion from San Diego, USA.
We suggest adding:
... in the order in which they appear (left to right).

Comment from Remington Rand, USA:
Opposed on the same grounds as Reformulation 4.

Reformulation 26: Own arrays.
a: 26, b: 10, c: 12, d: 1.

Remark from Oak Ridge, USA:
We would prefer a reformulation which makes the values of components undefined whenever the bounds are changed.

Remark from Kidsgrove, England:
This is bound up with 23, above. In our versions we do not allow dynamic own arrays.

Comment from Burroughs, USA:
We do not see any justification for treating own arrays with varying size. It is true that this facility is difficult to obtain using only the other features of ALGOL, but we think the process is so inefficient, a man who wishes to use this should think up a better algorithm so he doesn't need to waste the computer time doing this. We went to see this paragraph cut down to the following:
'The program is only defined if the values of the subscript bounds evaluated at the second and following entries are the same as those evaluated at the first entry.' STOP

Comment from Cambridge, England:
We see no justification for the restriction imposed by the last two sentences of Reformulation 26 and would have them removed.

Comment from Wegner, England:
Delete the second half of the suggested addition, i.e. the section: In the case of ... entries into a block.
It seems perfectly feasible that a recursive block might require arrays whose dimensions differ between successive activations. Complete symmetry should be preserved between non-recursive calling of a block and recursive calling of a block. I do not see any difficulties in extending this principle to own arrays.
See also note to QUESTION 37: DELETION OF OWN FROM LANGUAGE.

Reformulation 27: Meaning of switch declaration
a: 48, b: 2, c: 0, d: 0.

Reformulation 28: Procedure body as block.
a: 43, b: 7, c: 1, d: 0.

Alternative proposal from the Facit group, Sweden:

The procedure body (together with the specifications of the formal parameters) always acts like a block, whether it has the form of one or not. Consequently the scope of any label attached to a statement within the body or to the body itself can never extend beyond the procedure body. In addition, the identifier of a formal parameter must not be declared anew or attached as label to a statement within the procedure body. It may well be declared or used as a label within a subblock of the procedure body. STOP

Reason: There is no sense in declaring a formal parameter in such a way as to make it altogether inaccessible.

Comment from Remington Rand, USA:

We object to the concept of formal parameters being attached as labels to statements as this can serve no useful purpose. We also object to the concept of declaring formal parameters within a procedure body. This concept is completely opposed to the purpose of formal parameters. Also, what happens to the value of a formal parameter which corresponds to an actual parameter called by value.

Reformulation 29: Type procedure called by procedure statement.
a: 38, b: 1, c: 12, d: 0.

Remark from Rutishauser:

The effect of placing a function designator as procedure statement is not defined by the AR. Thus 'making an implicit rule explicit' can only mean that we state : 'procedures which are declared with an additional type-declarator in front, cannot be called through a procedure statement'.

Comment from Burroughs, USA:

We think it illogical to allow this case. Consequently, 'a type procedure may not be called as a procedure statement' is the rule we accept. This is only common sense, isn't it.

Comment from Wegner, England:

Is this rule really intended. It would seem that the use of a procedure statement to define the value of a procedure identifier is a natural one, although it cannot be easily implemented within the framework of an anonymous stack. Perhaps procedure statements should be abolished altogether, since they tempt the user into erroneous usage.

Reformulation 30: Types for name parameters.
a: 34, b: 5, c: 8, d: 2.

Remark from Dijkstra, Math. Cent., Holland

I should prefer a transfer function to be invoked.

Alternative from Dutch, PTT, Holland:

ADD TO TEXT: However, if specifications for parameters called by name are included, the values of the actual parameters will be transformed into the types as given in the specification by means of the transfer functions. STOP

Reason for reformulation 30: An actual parameter called by name does not need to have the same type as the specified type (cf. AB 14.2 Nagao).

ADDITIONAL AMBIGUITIES.

Reformulation 31. THE FATE OF THE CONTROLLED VARIABLE IN THE FOR STATEMENT.

Proposal from Eng.El.Atomic, Eng.:

Section 4.6.5. Second Paragraph.

DELETE:

If the exit .. .... undefined after the exit. STOP

Reason for reformation.

Ambiguous in the case where V is a subscripted variable, whose subscripts might be altered by evaluation of B, C, the controlled statement, or even V itself.

Comment from Kidsgrove, England:

Am I right in thinking that section 4.6.4.2 (subject to Reformulation 14) and section 4.6.4.3 are intended to provide rules for the interpretation of for statements? In other words, that a for statement is in general an abbreviation for other ALGOL statements of the form indicated.

If this is so, then I do not see the need for section 4.6.5. The first sentence in any case needs rewriting because of possible side effects in the evaluation of the designational expression in the go to statement. The second sentence would seem to allow an inaccurate implementation which did not agree with that defined by 4.6.4.2 and 4.6.4.3; if this sentence were suppressed, what harm could come.

And what is meant by 'controlled variable'? If we have something like

```
for A[i]:= . ... do
    begin . . ; i:= . .... ; ... end;
```

what is it that is undefined on exhaustion of the for list?
The whole array A? The elements A[i] corresponding to the values taken by i during the execution? The A[i] corresponding to the original value of i?

I am in favour of stating explicitly that 4.6.4.2 (revised) and 4.6.4.3 define the action of a for statement, and of suppressing 4.6.5 altogether.

Comment from IPM Darmstadt, Germany:

In section 4.6.3., the meaning of 'advance' is not quite clear in case of a for list after the last assignment and the corresponding execution of S have been done. In case of a step-until- or while-element the assignment after the last (intended) assignment is clearly defined, but not in case of a for list. Of course this is irrelevant according to 4.6.5 since, after that, 'test' finds that the for list is exhausted. Nevertheless we feel that this point should be clarified.

In the following sentence the term 'last assignment' apparently means the assignment after the last intended assignment and the corresponding execution of S. But we feel that the unbiased reader will misunderstand this point.

We therefore propose the following reformulation:

DELETE: Test .. done. STOP

ADD INSTEAD:

However, if the for list has been exhausted, 'advance' merely transmits this fact to 'test'. 'Test' determines if the for list is exhausted. STOP

Reformulation 32: ACTUAL FORMAL CORRESPONDENCE OF TYPE.

Proposal from the Facit Group, Sweden.

4.7.5.2 ADD TO TEXT: Often integer and real parameters can be used interchangeably. However, if there in the procedure body occurs an assignment to a formal parameter, specified to be of type real, the corresponding actual parameter must be of type real, not integer. If a formal parameter, specified to be of type integer, is used in the procedure body in other ways than left part variable, then the corresponding actual parameter must be of type integer.

Reason: It is not clear to what extent real and integer parameters may be used interchangeably. From 4.7.5.5 and AB. 14, reformulation 30, one might get the impression that they may not be used interchangeably at all; on the other hand it is expressly stated in 3.2.4 that the standard functions operate indifferently on actual parameters of both types. The other extreme would be to permit interchangeable use freely, even in the cases forbidden above; in that case the running program must be able to invoke transfers from real to integer dynamically. The above proposal is to be taken as a compromise. It would, for instance, rule out the program in AB. 14.2.

Reformulation 33: SCOPES AND PROCEDURE STATEMENTS.

Proposal from NPL, England:

Section 4.7.6, second sentence.

CHANGE TO READ:

A procedure statement written outside the scope of any quantity which is non-local to the procedure body, or to the bodies of procedures called directly or indirectly from within the procedure body, is undefined. STOP

Reason for reformulation: Improved accuracy (cf. AB 13.5 Woodger).

Note from Burroughs, USA:

Section 4.7.3.2.

ADD SENTENCE. These 'systematic changes' do not apply to identifiers which are non-local to the procedure body. They do, however, apply to all local switches and labels which conflict with switches or labels which can be associated with an actual-parameter switch.

Reason: Improved accuracy, makes switches more worthwhile parameters (see Knuth and Merner ACM Comm. June 61, footnote 3)

Reformulation 34: FORMAL PARAMETERS AS INDEX BOUNDS.

Proposal from the Facit Group, Sweden:

5.2.4.2 ADD TO TEXT:

The formal parameter of a procedure declaration may not enter into bounds of array declarations in its procedure body (but well into array declarations of subblocks within the procedure body). STOP

Reason: At present it seems that formal parameters called by value are local to the procedure body, and hence must not be used in bounds, whereas parameters called by name may be used; we question whether this difference is really intended and propose that it is removed.

Reformulation 35: ADMIT own array.

Proposal from RCA LAB, USA:
Section 5.2.1 of defining report.
CHANGE TO READ
<array declaration>::= array<array list>|
own array<array list><local or own type>array<array list> STOP
Reason for reformulation. Corrects an aparent oversight of the original report.

Reformulation 36: NON-LOCALS IN PROCEDURE BODIES.

Proposal from IPM Darmstadt, Germany:
Section 5.4.3., end of paragraph:
ADD TO READ:
. . appears. Procedure declarations which contain global parameters in its body and appear outside the scope of any of these globals, are undefined, even if it is used only in procedure statements inside the scope of those globals. STOP
Reason: Why troubling the compiler at the time when a procedure declaration is processed.

Reformulation 37: SPECIFICATIONS OF NAME PARAMETERS.

Proposal from the Dutch PTT, Holland:
Reformulation of 5.4.5, last sentence
CHANGE TO READ: In this part no formal parameter may occur more than once. STOP
Reason for reformulation 35: When specifications may be included this means that they need not be included. Then the sentence that formal parameters called by name may be omitted is just saying again what has already been said.

Proposal from RCA LAB, USA:
Section 5.4.5 of defining report.
CHANGE TO READ
... may be omitted together. In particular if and only if all formal parameters are called by name the entire specifications part may be omitted. STOP
Reason for reformulation. It is not clear under what conditions the specification part may be dropped. I get the impression that many people feel the specifications part may be dropped at will. If this were so then every general ALGOL compiler would have to be able to operate in the absence of specifications. But then what would we need specifications for. Even if the latter interpretation is intended the defining report seems to be consistent with the above reformulation, so some clarification is necessary.

Reformulation 38: THE ASSIGNMENT TO THE IDENTIFIER OF TYPE PROCEDURES.

Proposal from the Facit Group:
5.4.4. there must, within the procedure body, occur an assignment of a value to the procedure identifier
CHANGE TO READ:
there must, within the procedure body, occur one (or more) assignments of a value to the procedure identifier STOP
Reason: From the original wording one may get the impression that only one assignment were permitted.

Proposal from Eng.El.Atomic, England:
Part 1 Section 5.4.4.
CHANGE TO READ:
For a procedure declaration to define the value of a function designator, there must, within the procedure body, occur one or more assignment statements, at least one of which must be obeyed, which assign a value to the procedure identifier ..... STOP
Part 2 Section 4.2.1
CHANGE TO READ: <left part>::= <variable>:= |<procedure>:=
STOP
Reason for reformation.
Clarifies the phrase 'assignment of a value to the procedure identifier', and the sentence 'Any other occurence . . activation of the procedure'.

Proposals from Elliott ALGOL, England:
Section 5.4.4 end of first paragraph.
ADD TO TEXT:
However, this assignment may not be made by writing the procedure identifier as an actual parameter called by name. STOP
Reason for addition. To prevent the situation in which certain replaced occurrences of the parameter inside the procedure body should activate a recursive call, while others merely cause an assignment of a value to the function designator.
Section 5.4.4
'occur an assignment'
CHANGE TO READ
'occur at least one assignment' STOP
Reason for reformulation: The revised text explicitly permits more than one assignment.

Comment from Kidsgrove, England:
The syntax of 4.2.1 and 3.1.1 should be extended to allow assignment to a function designator. (And the semantics amended accordingly.) The wording of 5.4.4 is not explicit enough. Is it intended that (dynamically) only one assignment to the procedure identifier should take place. Can alternative assignments be written. What is the meaning if the assignment is in a repeated loop.
The second sentence surely applies only to occurrences within expressions - that is, not on the left hand sides of assignments.

Reformulation 39: NEW METALINGUISTIC OPERATOR.

Proposal from Zeiss, Germany:
I suggest to add a new metalinguistic operator:
<any string> means that there is not standing <any string>. Then I suggest to write
3.3 Arithmetic expressions
...
<factor>::= ↑<primary> | <factor>↑<primary>

<term>::= <mult.op><factor> | <term><mult.op><factor>

<simple arith.exp.>::= <add.op.><term><mult.op.> | <empty><add.op>
<term><mult.op.> | <simple arith.exp.><add.op>
<term><mult.op.>

<arith.exp>::= <simple arith.exp.><add.op.> | ...
and similar for 3.4 Boolean expressions.
This mode of expression makes clear the rules of precedence and does not allow the wrong
syntactic units b + c, b + c/d
of the expression a×b + c/d.

Reformulation 40: EXPRESSIONS WHOSE TYPES CAN ONLY BE KNOWN AT RUNNING TIME.

Question from Facit Group, Sweden:
In the program example:
Boolean b; integer i1, i2, i3; real r;
i1:= if b then i1 else r; i3:= i1↑i2;
the type of the conditional expression can only be decided at running time. The same holds for the expression with exponentiation (integer or real, according as i2 is positive or negative).
In this a correct interpretation of the ALGOL 60 report.
If so, what is your reaction to defining expressions of these kinds as type real.

Reformulation 41: MEANING OF else IN EXPRESSION.

Proposal from Burroughs, USA:
Section 3.3.3:
CHANGE LAST LINE TO READ
else if true then <simple arithmetic expression> else
<arithmetic expression> STOP
Reason: The example given was not well-formed according to the syntax.

Reformulation 42: THE EXAMPLE OF A FOR STATEMENT.

Proposal from Burroughs, USA
Section 4.6.2:
CHANGE TO READ
.. for j:= I + 6, 1 step 1 until N, C + D do
V1:= A[k, j]:= B[k, j] STOP
Reason: The example given was confusing because it gets into an unending loop, unless V1 $\geq$ N originally, or if N, I, G, L, C, D or V1 is a function designator which changes the value of V1.

Reformulation 43: THE EXAMPLE OF A DUMMY STATEMENT.

Proposal from Burroughs, USA:
Section 4.4.2:
CHANGE LAST LINE TO
begin I:= 1; JOHN: end
Reason: Previous example using three dots was not in keeping with the other examples which were valid ALGOL constructs.

Reformulation 44: LEXICOGRAPHICAL ORDER.

Suggestion from Wegner, England:
Section 4.3.3, second line:
Replace 'write-up' by 'lexicographical order'.

TABLE OF REPLIES ON SUBSETS, CHANGES AND EXTENSIONS, AND OFFICIAL ADOPTION.

	SUBSETS			CHANGES AND EXTENSIONS								OFFICIAL ADOPTION								
	25	29	30	31	32	33	34	35	36	37	38	39	40 a	40 b	40 c	40 d	40 e	41	42	43
NPL, England	no	a	a	no	b	b	b	b	yes	a	b	g	2	1	3	4	5	a	a	b	a
Zeiss, Germany	no	a	a	no	b	b	a			a	b		2	1	3	4	5	a	b	c	a
SMIL, Sweden	no	b	b	no	-	-	c	c	no	b	c	c	x					c	c	c	b
Syst.Dev.Corp., USA	no	b	a	no	-	a	-	a	-	-	-	-	2	1				a	b	b	-
Royal McBee. USA	no	a	a	no	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	a
Mayoh, USA	no	a	a	-	-	a	c	a	b	b	b	b	1	2	3	5	4	b	a	b	a
IDA-Princeton, USA	no	b	a	no	c	b	a	c	no	b	b	-	1	2	3	4	5	a	a	c	a
Facit Group, Sweden	yes	a	a	no	b	b	b	b	yes	a	b	a	1	2	3	4	5	a	a	a	b
Rehn, Finland	no	a	a	no	b	b	b	b	-	b	c	-	1	2	3	4	-	a	a	b	a
Eng.El.Atomic, Eng.	yes	a	c	yes	c	c	c	c	no	b	b	c	2	3	1	4	-	a	a	d	a
Elliott Algol, Eng.	yes	a	a	yes	b		b	c	no	b	c	c	1	2	3	4	5	a	b	c	b
ALPHA, USSR	no	a	a	yes	-	b	c	c	yes	a	b	c	3	1	2	4	5	b	b	b	a
Buchholz, Germany	no	a	a	no	b	-	b	b	no	b	b	-	3	2	1	4	-	b	b	c	b
Oak Ridge, USA	no	a	b	no	b	b	c	b	yes	a	c	c	4	2	1	5	3	-	-	-	b
SSW-ZEF, Germany	no	a	a	no	b	b	b	b	no	b	a	b	2	1	3	4	5	b	c	c	a
Dupont, USA	-	a	a	-	-	b	a	a	yes	b	b	-	3	2	4	1	5	b	b	a	b
Rutishauser, Switzerl.	-	-	-	-	-	b	b	c	-	-	c	c	-	-	-	-	-	-	-	d	b
Kidsgrove, England	yes	a	a	no	a	a	c	f	yes	b	b	c	2	3	4	5	1	a	a	c	b
MNA-group, Sweden	no	a	a	no	b	b	b	b	no	b	b	b	1	2	3	4	5	a	a	b	b
IAM Bonn, Germany	yes	b	a	no	b	b	b	c	yes	a	c	-	4	1	3	2	-	a	a	d	a
Math.Cent, Holland	no	c	c	no	c	g	c	c	no		b	c	1	2	3	4	5	a	-	-	b
Dutch PTT, Holland	no	a	a	no	b	a	a	c	no	b	a	c	2	3	1	4	-	a	c	b	b
San Diego, USA	no	a	a	no	a	a	b	c	yes	b	b	-	2	3	1	-	-	b	g	b	b
Leeds Univ., Eng.	no	a	a	no	a	b	a	b	yes	b	c	b	2	1	3	4	-	b	c	b	b
ALCOR PERM, Germany	yes	a	a	no	c	b	c	c	no	a	b	b	5	1	5	5	5	b	b	d	b
ALCOR Z22, Germany	yes	a	-	yes	c	b	c	c	no	a	b	a							b	d	g
ALCOR 2002, Germany	yes	a	-	yes	c	b	c	c	no	a	b	a							b	d	g
RCA-EDP, USA	no	a	a	no	-	b	a	-	yes	b	b	b	2	3	1	-	-	a	b	b	b
Bjork, Sweden	no	a	a	no	-	-	-	-	-	-	-	-	2	1	3	4	5	a	a	-	a
Moore School, USA	no	a	a	no	b	b	a	b	yes	b	b	b	4	2	3	1	5	a	b	b	a
IPM Darmstadt, Germany	yes	a	a	no	b	b	c	b	no	a	b	a	1	2	-	-	-	a	b	d	b
Computor Ass., USA	no	a	a	yes	b	a	f	a	no	b	b	a	2	1	4	3	-	b	b	b	a
Regnecentralen, Denmark	yes	a	a	yes	c	c	b	c	yes	a	c	c	3	1	4	5	2	b	c	d	a
Univ.N.Carolina, USA	no	a	a	yes	b	b	b	-	yes	-	c	c	4	2	1	3	-	f	f	b	b
RRE, England	no	a	a	no	b	b	c	a	yes	b	c	c	2	1	3	4	-	a	a	b	a
Stanford, USA	no	a	a	no	b	a	b	b	yes	b	c	b	2	1	3	4	-	a	b	a	a
RCA LAB, USA	no	a	a	no	a	b	b	a	c	b	b	c	3	1	2	4	-	b	b	b	b
ARF, USA	no	b	a	no	c	c	b	c	yes	a	b	c	5	1	4	2	-	a	b	d	a
NDRE, Norway	no	b	a	no	b	a	a	c	yes	a	b	c	4	3	1	2	5	a	-	b	a
XTRAN project, USA	no	a	a	no	b	a	a	a	yes	a	b	a	3	2	1	4	-	b	c	a	b
Hockney, England	no	a	a	no	a	a	d	d	-	-	d	-	3	1	2	4	-	a	a	b	a
NBS, USA	no	c	b	yes	b	b	c	c	no	b	c	c	3	2	1	4	-	a	c	b	a
Tubingen, Germany	no	a	a	no	b	b	c	b	no	a	b	a	2	1	3	4	5	b	b	c	b
Siemens, Germany	yes	a	b	no	b	b	c	c	no	a	b	b	3	4	2	1	-	b	b	c	b
Standard El., Germany	yes	c	a	-	-	-	b	c	no	a	b	-	x					a	a	c	b
AFCALTI, France	no	a	-	no	a	a	c	a	a	a	-	-	1	4	2	3	5	-	-	-	b
Burroughs, USA	-	-	b	no	-	b	c	b	yes	b	c	h	5	2	4	3	1	a	e	b	a
Saarland, Germany	yes	b	b	no	c	c	c	c	-	-	b	c	1	2	3	4	5	b	c	d	b
Remington Rand, USA	yes	a	b	no	c	c	c	c	yes	a	b	c	5	1	4	3	2	a	b	a	b
Cambridge, England	no	b	-	no	b	b	c	-	yes	b	c	-	-	-	-	-	-	-	-	-	b
Wegner, England	no	c	b	no	b	b	b	a	a	a	a	f	-	-	-	-	-	c	-	b	b

Special codes: f = a/b, g = b/c, h = a/c

COMMENTS ON SUBSETS.

Question 25, MEMBERSHIP OF SUBSET-GROUP:
Yes: 14, No: 34.
5 groups exist:

Group 1: members: Facit Group, Sweden; Regnecentralen, Denmark.

Group 2: members: Kidsgrove, England; Eng.El.Atomic, England. Name of subset: KDF9 ALGOL. Full ALGOL 60 except 1. Dynamic Own Arrays. 2. Integer labels. 3. Optional specifications.

Group 3: members: ECMA TC 5 (incl. Elliott ALGOL, England).

Group 4: ALCOR, members: IAM Bonn, Germany; ALCOR PERM, Germany; ALCOR Z22, Germany; ALCOR 2002; IPM Darmstadt, Germany; Siemens, Germany; Standard El., Germany; Saarland, Germany. Described in Elektronische Rechenanlagen 3 (1961) 206-212 and following articles.

Group 5: members: C.W. Dobbs, UNIVAC, Philadelphia; R. Belscamper, UNIVAC, St. Paul. Name UNIVAC Standard ALGOL. Characteristics as yet undefined, tentatively 1107 ALGOL.

QUESTION 29, THE IDEA F RECOMMENDED SUBSETS
a: 37, b: 8, c: 4.

Comment from Burroughs, USA:
As it stands now, we wager nobody will implement ALGOL 60 completely. This is because of two major cases:
1) Recursive procedures which use non-local variables which are local to other recursive procedures.
2) Implementing 'procedure a(b); procedure b; 2: b(2).' In this example 2 can be used as a label, number, or both by the procedure b. These seem to be obstacles nobody has overcome.
So everybody will pick some subset or other, leading to a somewhat chaotic state. This is unfortunate; it would be much better to have as ALGOL a language which every fairly large computer can implement (and efficiently too). Then a single recommended subset for the small computers which are incapable of handling too extensive a language should be given as a guide to reduce the chaos somewhat.

Comment from Remington Rand, USA:
In our opinion, the definition of subsets provides partial solutions to some of the problems of inter ALGOL processor compatibility. We hope in UNIVAC to establish hardware representation compatibility by this method.

Comment from Wegner, England:
Surely the principal strength of ALGOL lies in its being a standardised language. Subsets seem to be due principally to a lack of understanding of the principles of ALGOL implementation. It has been shown by Dijkstra that the implementation of virtually complete ALGOL is straightforward even on a small machine. The question of subsets should therefore be reconsidered.

QUESTION 30. THE REACTION TO THE TWO SPECIFIC SUBSETS, BASIC ALGOL 60 and SMALGOL.
a: 36, b: 8, c: 2.

Comment from Remington Rand, USA:

We suggest that once the ambiguities and obscurities have been cleared up in ALGOL 60, the question of subsets may be left to the individual user. Provided these were all subsets and not dialects, we see no reason why the number of subsets should be restricted to two.

It seems likely that some features might be added to Basic ALGOL and SMALGOL in the light of programming and implementation experience.

Comment from Cambridge, England:

We do not wish to commit ourselves on this point.

Comment from Wegner, England:

Basic ALGOL 60 and SMALGOL were motivated by an incomplete understanding of ALGOL implementation. If any given subset is developed it should be based on the greater understanding of ALGOL implementation that has recently become available through the publications of Dijkstra. However subsets are probably not necessary at all.

COMMENTS ON CHANGES AND EXTENSIONS.

QUESTION 31: Familiarity with ABS 14 (the report by Ershov, Kozhukhin, and Voloshin).
Yes: 9, no: 38.
Comment from ALPHA, USSR:

We believe that some of the Input Language constructions are very desirable in ALGOL, specially: complex, internal dimensions, forming and composing operators, initial values, chains of inequalities, functions yielded by expressions and time superscripts.

Comment from ALCOR Z22 and ALCOR 2002:

The axiomatic system is overextended. It should be possible to give definitions of new elements of the language in terms of simpler ones within the language (like procedure declarations) instead of adding new definitions to the basic system.

Comment from Computer Ass., USA:

We have translated the introduction of this report, and followed some of the more puzzling proposals some distance into the body of the report; we have not studied it sufficiently to give a responsible detailed criticism of the proposals. None of them seen impossible to implement; some are quite interesting; most of them seem, in the light of the present feelings on 'freezing' ALGOL, to be in the nature of serious extensions. We will be pleased to comment upon them in the next questionaire. We have doubts, though, that the promised compatibility with ALGOL 60 will hold if the proposals for allowing 'output' parameters to be specidied by value is included.

Comment from Regnecentralen, Denmark, to QUESTIONS 31, 32, 33, 38:

We cannot support the inclusion of additional special facilities in ALGOL. We want the language to include such general features which permit the user to add arbitrary special mechanisms to it at will. For this reason we feel that the procedure concept is the most important one in ALGOL 60. It has not yet been fully explored. Further developments should generalize this concept, e.g. by permitting the user to specify, not only the meaning, but also the form, of procedure calls. In this manner most of the above special extensions would become unnecessary we feel.

Question from Remington Rand, USA:

When will English translation be available (see APIC Bulletin no. 8, p. 38, 8A.005.)

QUESTION 32: THE HOCKNEY PROPOSAL.
a: 6, b: 24, c: 8.

Comment from ALPHA, USSR:

The Hockney proposal is good in idea, but has not been developed very well. Our proposals cover the Hockney ones.

Comment from Rutishauser, Switzerland:

The Hockney proposal contains actually 4 different proposals and therefore cannot be dismissed simply by one three-branch question. First, it proposes a complex-declaration which I would welcome since several prospective users of ALGOL complain its nonexistence. Second, it introduces a new notation for arrays of arrays which certainly has its merits but should be discussed carefully before it is voted upon. Third, a notation for the value of an array that allows to give the components of an array numerically, and fourth, the introduction of matrix calculus in ALGOL, which I strongly oppose.

Comment from IPM Darmstadt, Germany:

Array arithmetic seems doubtful to us. But complex arithmetic is very desirable. We would like to see double precision arithmetic included too. Eventually, one should introduce facilities for defining arbitrary (not standardized) types, which are chosen by the programmer.

Comment from Computer Ass., USA:

The Hockney proposal reached us only after the middle of February. The proposals seem mostly to overlap a part of the proposals of Ershov et al. We could not say at this time which we prefer. The inclusion of facilities like these seems an obvious way to extend ALGOL when the time for extensions arrives.

QUESTION 33: STRING MANIPULATION.
a: 11, b: 28 1/2, e: 5 1/2.

Remark from Oak Ridge , USA:

But not necessarily as exactly formulated.

Comment from SSW-ZEF, Germany:

According to the proposal by Wegstein and Youden, in a string declaration the length of the string must be declared by the programmer. This is quite uncomfortable, and, in our opinion, not necessary, as the strings are represented by threaded lists.

QUESTION 34: THE NOTATION FOR CONDITIONAL EXPRESSIONS AND STATEMENTS.
a: 9 1/2, b: 17 1/2, c: 20.

Comment from Computer Ass., USA:
The restrictions are unnecessary for Expressions at present (and we shall not require them in our implementation), and would become unnecessary for Statements upon a proper reformulation of the syntax. This is not urgent, but when the first 'changes' to ALGOL 60 are made, these should be among them.

QUESTION 35: SYNONYM ASSIGNMENT.
a: 9 1/2, b: 13 1/2, c: 21, d: 1.

Remark from Computer Ass., USA:
But not the way Thacher proposes.

Remark from AFCALTI, France:
In particular, Mr. Nolin would wish that it be possible to change the names of variables in the course of execution (for instance, for flip-flop works).

Remark from Burroughs, USA:
We favor a synonym assignment of the type <identifier> :: <string> meaning substitute string for every future occurrence of this identifier.

Comment from Cambridge, England:
We are indifferent.

QUESTION 36: LACK OF INITIALIZATION OF OWNS IS A SERIOUS DEFECT
Yes: 21, no: 18.

QUESTION 37: DELETION OF OWN FROM LANGUAGE.
a: 20, b: 23, c: 0.

Comment from Dijkstra, Math. Cent., Holland:
At present the definition of the concept 'own' is unsatisfactory, everybody knows that. But to favor the suggestion to delete it from the language means that we have given up hopes to improve the definition. In this connection we should not close our eyes for the fact that the 'dynamic own array' is something which is not expressible by any other means already provided by the language. On the whole I think it should be retained in the sense of Reformulation 23. With regard to Reformulation 26 I think that I fail to see the reason to restrict 'adjustment of the bounds' to the entry of an 'outermost' activation and I therefore suggest this restriction to be removed from the Reformulation 26.

QUESTION 38: COMBINATION WITH COBOL.
a: 3, b: 29, c: 14, d: 1.

Remark from AFCALTI, France:
We should like a certain unity of style between ALGOL and Cobol (punctuation rules priority, etc. . .).

Comment from Burroughs, USA:
We believe anyone who really understands both ALGOL and COBOL would realize that they are quite incompatible and that an idea to combine them is absurd.

Comment from Wegner, England:
Data description and input-output form an important part of any language for computation and I feel that there is an urgent need for the introduction of such facilities, although not necessarily along the lines suggested by Sammet.

QUESTION 39: THE PROPOSALS F STRACHEY AND WILKES.
a: 8, b: 10, c: 19.

Remark from Computer Ass., USA:
Our feeling is that an ALGOL translator should, in principle, accept the language in its full generality, and, when efficiency becomes important, it should be able to detect when 'special features' (which might prevent the free use of efficiency techniques) are not present, to take advantage of their absence. This is perhaps somewhat idealistic, and, so, rather than restrict the specification of the language to match compiling techniques, we prefer the approach of providing in the source program, some hints and promises to the translator. The Strachey-Wilkes proposals are mostly in this vein, and we find most, but not all, of their proposals acceptable.

Remark from Regnecentralen, Denmark:
The influence of present machine designs should be confined to the design of processors for them. Since the demands of powerful languages are the most important incentives towards better machine designs it would be disastrous if the limitations of present day machines were allowed to limit the power of expression of the languages.

Comment from Burroughs, USA:
We did not care for the proposals of Strachey and Wilkes regarding functions, and result of, but their other suggestions are fine.

Comment from Cambridge, England:
We agree with these proposals in principle and would bring about a corresponding major change in ALGOL rather than inclusion in the existing language.

ADDITIONAL CHANGE OR EXTENSION: INPUT-OUTPUT.

Comment from San Diego, USA:
We consider it highly desirable that input-output definitions be incorporated in ALGOL at an early date. The data division of COBOL would be a good starting point. Input-output capability is an essential part of working programming systems.

ADDITIONAL CHANGE OR EXTENSION: THE POWER OPERATOR.

Suggestion from Burroughs, USA:

Define a↑b where a and b are type integer to be always of type integer with the value to be (b negative)

1 ÷ (a×a× . . a)

This is conventional with most other compilers and it means a good savings in object program efficiency. Otherwise the type of a↑b varies at running time.

Proposal from Remington Rand, USA:

(Editor's note: this proposal is a complete revision of section 3.3.4.3 of the ALGOL 60 Report, doing essentially the same thing as the above suggestion from Burroughs. The end of the proposal follows).

Reason for proposal: Makes sure that the type of an expression does not depend upon the value of a variable. (See Comm. ACM, June 1961, Knuth and Merner). This is, however, only a partial solution to the problem. Consider the following example:

real a,b; integer i,j; Boolean A;
a:= j + (if A then i else b)

Is the arithmetic expression in parenthesis to be considered of integer type if A is true and of real type otherwise.

ADDITIONAL CHANGE OR EXTENSION: SWITCHES.

Suggestion from Burroughs, USA:

Restrict the components of switches to be merely labels. The statement go to S[n] would be a dummy statement only if n=0, would be undefined if n ≤ -1 or if n is too large. This change is motivated by the fact that the more complicated designational expressions have not proved to be efficient or particularly useful, and when they are used the algorithm becomes difficult to follow.

A facility which actually seems to be wanted most often is a 'procedure switch' or a 'return-jump switch' which would be something like

return switch S:= A,B,C

where A,B,C are procedure names, and the statement

S[n] (X)

would cause A (X) to be called if n = 1, etc. This appears to be a more frequent occurrence than the need for a switch of the present type.

ADDITIONAL CHANGE OR EXTENSION: THE MODULUS OPERATOR.

Sug estion from Burroughs, USA:

Introduce a new <multiplying operator>, mod, defined for at least integer arguments. We would have

a mod b equivalent to a-(a ÷ b)×b

ADDITIONAL CHANGE OR EXTENSION: ONLY SIMPLE FOR-VARIABLE.

Proposal from Remington Rand, USA:

In section 4.6.1 change third definition to read:

<for clause>::= for <simple variable>:= <for list> do

Reason for reformulation: We feel that subscripted variables were not intended to be used as the controlled variable in for statements, otherwise the discussion in 4.6.5 would have surely have given them particular mention. If they were intended to be used in this manner, we question the usefulness of the feature.

ADDITIONAL CHANGE OR EXTENSION: ABANDON CONCEPT OF LOCAL.

Proposal from Remington Rand, USA:

In section 2.4.3, second paragraph. Change to read:

'The same identifier cannot be used to denote two different quantities.'

Reason for reformulation: To relieve the utter boredom of reading any more nonsense about this feature. More seriously, I think this feature can be interpreted in two separate ways depending on what the introduction to Section 5 actually means - the use of such words as 'significance' and 'meaning' help to confuse the issue. Most writers on ALGOL use the interpretation given by Bottenbruch (p. 23-24 - Structure and Use of ALGOL 60), but this interpretation could not be proved to be true from the ALGOL report. Another and more useful interpretation could be given by rewriting the last two sentences of the second paragraph in section 5 to read.

'If these identifiers had already been defined by other declarations outside, they are for the time being given a new significance. On exit from the block these identifiers will resume their old significance but any values they might have had before entry into the sub-block will be lost. Identifiers which are . . . . meaning.'

This has the effect of saving storage, whereas the other interpretation produces nothing but confusion.

Naturally the question of values only applies to variables and to take care of procedure identifiers, etc. more explanation is needed. Therefore, we propose that identifiers should be unique.

ADDITIONAL CHANGE AND EXTENSION: BLOCK STRUCTURE.

Comment from Wegner, England:

In my opinion, the principal shortcoming of ALGOL 60 is the lack of facilities for expressing a large problem in terms of a number of lexicographically independent subroutines. These objections are elaborated in an appended note entitled 'Blocks in FORTRAN and ALGOL'. (Editor's note: These 5 pages have been omitted as being outside the scope of the AB).

COMMENTS ON OFFICIAL ADOPTION.

QUESTION 40: AUTHORITATIVE BODY.

Number of replies with preference no:	1	2	3	4	5
a. ad-hoc committee (authors)	10	15	9	5	4
b. AB + U.S.Maintenance	18	17	6	2	0
c. IFIP	10	5	18	7	1
d. ISO	3	3	5	23	5
e. Other	2	2	1	1	17

Remark from Rutishauser, Switzerland:

The correction of the AR is of course up to those who are responsible for the defects. It would therefore be welcomed if the ALGOL-committee could assemble again in order to remove the ambiguities of the AR. However, this body should not have power to change ALGOL where the AR is clear.

Comment from The Facit Group, Sweden:

We strongly suggest that clarifications and changes in ALGOL are worked out and adopted by a committee, composed of the authors of the ALGOL 60 report and such persons as they wish to add to it. The votes cast by the U.S.Maintenance Group and the ALGOL Bulletin readers will serve as proposals to this committee and no more.

Two reasons are:

1. Obviously nobody but the authors of a report has the right to change that report, unless the authors delegate that right to another body.
2. The method above - a committee guided by the discussions of the ALGOL Bulletin and the Comm.ACM - was used to produce the ALGOL 60 report, and it worked remarkably well. By contrast the method of mailed questionaires and voting among the readers has not produced convincing results.

Remark from Kidsgrove, England:

We feel that excellent as the ALGOL Language is it has not received the universal support it deserves. The prime reason for this appears to be that the ALGOL language maintenance and revision is in the hands of a team of competent amateur enthusiasts without official status. The main requirement is to preserve the team with its enthusiasm and competence, but to give it more of an official stature. The situation might well change for the better if the ALGOL effort were to receive the support, encouragement, and blessing, from IFIP.

Remark from San Diego, USA:

We consider that the various ALGOL groups have done excellent work. The ALGOL work should have the official sanction of an international computation organization if that is possible. This is the reason we recommend that the adoption be by an ad-hoc committee or by the U.S. Maintenance Group and ALGOL Bulletin as part of IFIP.

Remark from ALCOR Z22, ALCOR 2002:

Clarifications and subsets ISO, extensions IFIP. Final approval by ISO.

Comment from Cambridge, England:
ALGOL 60 is the property of the committee that formulated the language. Changes can only be made by them or by some body appointed by them. This, of course, does not apply to any future, different language and in view of this we leave the answer to question 43 open.

QUESTION 41: TIME FOR ADOPTION OF CLARIFICATIONS OF ALGOL 60.
a. Jan. - June 1962: 26 1/2
b. July - Dec. 1962: 15 1/2
c. Jan. - June 1963: 2

Comment from ALCOR Z22, ALCOR 2002:
Removal of ambiguities and inconsistencies only: as soon as possible.

Comment from Computer Ass., USA:
As soon as feasible, without recklessness.

Comment from Burroughs, USA:
We should like to see the clarifications come out as soon as possible.

QUESTION 42: TIME FOR ADOPTION OF SUBSETS.
a. Jan. - June 1962: 13 1/2
b. July - Dec. 1962: 20
c. Jan. - June 1963: 8 1/2
d. July - Dec. 1963: 0
e. Later: 1

Comment from Burroughs, USA:
The subsets should not be decided officially until some experience is gained trying present subsets.

QUESTION 43: TIME FOR ADOPTION OF CHANGES AND/OR EXTENSIONS.
a. During 1962: 5
b. - 1963: 20
c. - 1964: 10
d. Later: 10

Remark from ALPHA, USSR:
A partial alternative.
We propose to discuss the question about variables in the period of July - Dec. 1962. If the deletion of own variables from the language were generally adopted it would be possible to do it immediately.

QUESTION 44: REACTION TO THE QUESTIONAIRE.
a. Useful, adequate for final decisions 21
b. Useful, but not adequate for final decisions 28
c. Unwanted or harmful 1 (1/2 + 1/2)

Remark from Remington Rand, USA:

Surely many of these ambiguities were known immediately after publication of the ALGOL report and could have been promptly corrected. It is amazing that two years after publication we are still debating whether to define a program or not.

In essence, the questionaire is useful but we doubt its efficacy. One objection we have is that the phrasing of certain questions shows a less than impartial attitude, e.g., Reformulation 12.

Comment from Burroughs, USA:

We are sure this questionaire was a very good idea and that it will certainly serve a very useful purpose.

Comment from Wegner, England:

This questionaire serves a very useful purpose since, by answering it conscientiously one is forced to make decisions on a number of controversial points that have arisen in the last two years. I have found it particularly useful from an educational point of view and it has given me a better understanding of the ALGOL language.

However, I do not feel that it is the only method of reaching decisions about ALGOL. Open discussions, committee meetings and private enterprise all have their place. In this connection, the principal criterion is the quality of the end result rather than the means by which it is reached.

Regarding further stages in the evolution of international algebraic languages, I feel that there could be two separate parallel stages of development.

1. An agreed set of reformulations and revisions based on replies to section 2 of the questionaire could be circulated and further debated for a period of about six months. A set of resulting modified revisions should then be incorporated into the report, and a second edition of the report, possibly entitled 'ALGOL 63' could then be published early in 1963.

2. At the same time, the accumulated experience with ALGOL 60, COBOL, LISP and other languages, should be used in the formulation of a new international language, without unduly worrying about compatibility. ALGOL 60 has contributed a tremendous amount to an understanding of the requirements of international languages for computers. However, if ALGOL 60 is regarded as sacred, and is accorded the hallowed status of an elder statesman, it might well stand in the way of progress.