On the distribution of statements in Pascal programs

A. Schroeder

Rapports Techniques

N° 39

ON THE DISTRIBUTION OF STATEMENTS IN PASCAL PROGRAMS

Anne SCHROEDER

Juin 1984

ON THE DISTRIBUTION OF STATEMENTS IN PASCAL PROGRAMS

Anne SCHROEDER

INRIA - B.P. 105
78153 Le Chesnay (France)

Abstract

This note presents a few experimental results on both static and dynamic use of Pascal operators, and more specifically, of Pascal statements; these results are compared with some other published ones. The main conclusion concerns the important deviation existing between the static and dynamic distributions of operators and statements.

This work has partly been supported by the Agence de l'Informatique (France)

Résumé

Cette note présente quelques résultats expérimentaux sur l'utilisation, tant statique que dynamique, des opérateurs du langage Pascal, et, plus particulièrement des différentes instructions. Ces résultats sont comparés avec d'autres publiés par ailleurs. La principale observation que l'on puisse faire est celle de l'écart existant entre les distributions statiques et dynamiques.

Ce travail a été réalisé avec le concours de l'Agence de l'Informatique

ON THE DISTRIBUTION OF STATEMENTS IN PASCAL PROGRAMS

Anne SCHROEDER

INRIA - B.P. 105
78153 Le Chesnay (France)

1 – Introduction

This note is to present a few experimental results on the use of Pascal operators, and more specifically, of Pascal statements. Both static and dynamic measurements have been collected; the static distribution of Pascal operators in a given program is the set of the utilisation frequencies of all operators in the source text of the program; the corresponding dynamic distribution gives account of the actual number of utilisations of the operators at run-time. While the static distribution of operators is collected once for all, the dynamic distribution is collected during an execution and thus dependent of the data set generating this execution; in order to reduce this dependence, the dynamic distributions we consider in this paper are cumulative results on several executions.

There is not much literature on this subject. Concerning static statement distribution, there have been papers giving experimental results for different languages: Cobol [Al-Jarrah79], Pascal [Brookes82, Shimasaki80], Pascal and Fortran compared [Perrott81]. Also, models have been proposed for operator distribution [Zweben77 and 79]. In [De Prycker82], are presented tools to achieve such measurements together with experimental results on both static and dynamic statement distribution in Pascal and Algol.

The operator distribution of a given language is precious information for programming language designers and/or implementors. According to their purpose, they may be interested in the static distribution rather than in the dynamic distribution or vice-versa. For instance, implementors of a syntactic analyzer would organize their tables with respect to the operators frequencies in the texts of the programs they will analyze, that is the static distribution; on an other hand, people interested in an efficient implementation of the instruction set of a machine will primarily be concerned with the dynamic instruction distribution [McDaniel82, Sweet82, Wiecek82].

In the next paragraph, we shall give experimental static and dynamic distributions of the most frequently used Pascal operators; dynamic measurements have been collected on two programs (respectively 211 and 56 Pascal blocks), the 1st one having been run 4 times, the 2nd one 3 times. Only the bodies of the blocks have been measured, no headings nor declaration areas. In the two last paragraphs, only particular operators are considered, namely the statements; our results are compared with other published ones and an attempt is made to explain the discrepancy between static and dynamic behaviors through a number of static complexity metrics.

2 – Static and Dynamic Operator Distributions

In this paper, the definition of the set of Pascal operators is that in use in the programming environment Mentor [Donzeau80 and 83], in which our measurement tools have been developed; in Mentor, programs are represented as syntactic operator/operand trees, and, in our tools, operators are counted as the non-terminal nodes of such trees (for further detail on the definition and the implementation of the measurement tools, see [Schroeder84]).

Our program sample is divided into different clases, which are the following:

- CR : set of programs coming from a same research group, including as its main part, the above Mentor system;
- AS : statistical programs, some of them adapted from Fortran;
- VE : programs that are part of a relational data base system;
- FL : a program that generates structured "flips", or transparents, described as embedded boxes containing text or drawings;
- LP : a grammar analyser, part of a meta-compiler.

In Table 1, the static counts of Pascal operators observed on a total sample of 766 Pascal blocks is presented. Only the most frequent operators have been kept. In Table 2, both static and dynamic distributions (percentages) of the operators of two programs among the 16 initial programs are given. The two programs FL and LP have been chosen for the dynamic experiments, because of their large size and of the facility of generating different executions for each of them, and the four measured runs of LP correspond to the analysis of four different languages. At first sight, it can be noted that static and dynamic distributions may differ significantly, which emphasizes the necessity to perform run-time experiments to get serious information on the use of operators. We shall come back to this point in the 4th paragraph, when studying statement distributions.

3 – Static Statement Distribution

Several experiments on the particular case of statement distribution have been presented in the literature, most often concerning static counts. In Table 3, the results obtained on our sample are compared with several static statement counts given in the following references:

- In [Brookes82], results are given on 11 programs: P1 to P6 and S1 to S5, which the authors identify as scientific (Si's) and non-scientific (Pi's) programs, P6 being a compiler; in our tables Si corresponds to the mean value of the Si's, Pi to that of the Pi's except P6, and P6 to itself.
- In [Shimasaki80], 5 Pascal compilers have been studied; the two first ones (Sp and St in the quoted reference) seem to be an original compiler and a modified version of it, and the results given for them actually are very close; in our tables, their mean value appears, identified as "12". The two

following ones (S8I and S8L) are two parts of the same compiler, and again, their results being very close, we consider only their mean value ("23"). The last one (Sq in the reference, and "5" in our tables) is a sequential 7-pass compiler.

- In [Perrott81], the measured program is a simulation program adapted from a Fortran version and the results are used to compare the Fortran and Pascal implementations.

Table 4 approximately contains the same information as Table 3, augmented with a fourth comparison [De Prycker82], on a smaller set of statements in order to make this last comparison possible. In [De Prycker82], two Pascal programs are studied: one that generates the syntax tree of a program (Tree or "Tr") and one that is part of syntax analyser generator (Split or "Sp"); let us note that programs with the same kind of functionalities belong to the "CR" class of programs in our sample.

A few comments can be made on these tables:
- the two most frequents statements in source texts are assignments and procedure or function calls;
- the FOR loop is the most frequent, though WHILE loops are as frequent in two programs; also, four programs present a specially big number of FOR loops, and knowing the origin of three of them, we may assume they all have been written by ex-Fortran programmers;
- the frequencies of the IF test vary from a program to another, but the IF construct is, in all cases, far more used than the CASE.

A global view on the different use of statements in the different groups of programs is given, via Correspondence Analysis, on Figure 1. Very shortly speaking, Correspondence Analysis is a multidimensional data analysis method, that gives a good approximate low-dimension representation of a set of points originally situated in a high-dimension space; here, the data in Table 4 can be considered as the data of 8 points, representing 8 types of Pascal statements, in a 14-dimension space, the coordinates axes of which represent the 14 different origins the measured progams may come from. Thus, if two origins are close to each other on the representation, it means that they use statements in a similar way. Read in the other way, Table 4 can also be considered as the data of 14 points, representing different origins, in a 8-dimension space (8 kinds of statements). Correspondence Analysis allows a simultaneous representation of the two sets put into correspondence by a rectangular data array, such as that given in Figure 1; a measure of the quality of this representation is given by the sum of the percentages attached to the considered axes, which is the proportion of initial information given by the graph; thus, Figure 1, which gives a 3-dimension representation of Table 4, displays 90.6% (61.3 + 16.0 + 13.3) of the total information. (Further detail on Correspondence Analysis may be found in [Hill74, Schroeder78])

The main results derived from Figure 1 are the following :

- the use of FOR loops is the first discriminating feature between programs; the programs in which FOR loops are used more frequently correspond to scientific or Fortran-like applications (AS,Pe,Si);

- there is a group of programs (VE,LP,FL,Pi,34,5) in which CALL statements are more frequent; they are highly modular programs;

- the use of test statements (IF and CASE) is more frequent in three of the origins (CR,12,P6), in which REPEAT loops also more are frequent.

These results indicate different programming styles, related to the application and to the programmer, at the same time.

4 – Dynamic Statement Distribution

In Table 5, static and dynamic statement distributions in four different programs can be compared. As in Section 2, it can be noted that these distributions may considerably differ, without any systematic pattern appearing. In the case of the two programs from our sample, we know they are rather complex programs, in terms of commonly accepted complexity indices:

- the maximum nesting depth of control statement is 7 in FL and 6 in LP,

- the cyclomatic numbers (equivalent to the number of control statements) [McCabe76, Schroeder84] of the Pascal blocks in the programs may be as large as 33 and 47 in FL, and as 25 to 30 in LP, while their normalised value (i.e. divided by the block size expressed in terms of the total number of operators and operands used in that block) may get to 13% in FL and to 20% in LP.

In order to precise the influence of such static complexity metrics on both static and dynamic behaviors, we analysed the corresponding data on the two programs LP and FL. An other Correspondence Analysis has thus been performed on the 223x8 array, each line of which corresponds to a Pascal block (176 in LP, 47 in FL, that have actually been executed), while the 8 columns correspond to 8 static complexity metrics, namely:

- Nin, the number of statements in the body of the block;

- TT, the size of the body of the block, expressed as the total number of tokens (operators and operands) needed to write it;

- Otd, the number of distinct operators used in the body of the block;

- Ond, the number of distinct operands (variables and constants) used in the body of the block;

- Voc, the vocabulary used, that is Ond + Otd;

- Prf, the depth of the syntax tree of the body of the block, which measures the logical complexity of a program in terms of statement and expression nesting;

- cmd, the cyclomatic number of the block (see above);

- Imx, the maximum nesting depth of statements in the block.

(For precise definition and properties of these metrics, cf. [Schroeder84])

Also, for each block, both static and dynamic statement distributions are known, which may add 16 columns to the data array (static and dynamic frequencies of 8 different kinds of statements). On Figure 2, is displayed the approximate representation of the 223x8 array (the 223 Pascal blocks are represented by dots, the static complexity metrics are written in straight characters); the static and dynamic statement distributions have not been taken into account in the computation of the projection axes, but they have been projected afterwards on these axes, being themselves points in the 223-space indexed by the blocks (each one is written in italics, with a postfix s or d according to whether it represents a static or a dynamic frequency). Thus, what can be interpreted in the display of Figure 2 is, on one hand, the general complexity structure of the blocks, and, on the other, the relationships between complexity and static versus dynamic use of statements.

The main results derived from Figure 2 are the following:

- the complexity of the blocks lays in three aspects: the dominant one is the size of the block; next come, at the same level of importance, the complexity due to the vocabulary and the variables used, and that due to statement and expression nesting;

- the only information about GOTO statements is that, in the 223 measured blocks, though there were a few GOTO's in the programs, none of them was actually executed at run-time (GOd placed at the origin on the graph); it is expected that GOTO's appear in particular sections such as error handling; also, GOTO's appear more frequently in those blocks in which the nesting is deeper;

- as for the other statements, REPEAT's and CASÉs are the only ones the static and dynamic frequencies of which apreciably differ; all the others have similar frequencies in program texts and at run-time;

- programs with a high vocabulary complexity are more likely to execute CASÉs and REPEAT's more often that expected in the program text;

- programs with a high nesting complexity are more likely to present WHILÉs. IF's and FOR's, in similar proportions statically and at run-time.

These results indicate the possibility of some kind of prediction of the dynamic use of statements from both their static use and the complexity structure of the block.

Conclusion

The main result that can be derived from this experimental study is the important deviation existing between the static and dynamic distributions of operators and statements. It shows that they are not equivalent when used in an implementation purpose and that one must clearly identify which one is to be considered.

Also, further analyses of the relationships between statement distribution and complexity, indicate a possible prediction of run-time behavior and some typical programming styles.

References

[Al-Jarrah79]
M. M. Al–Jarrah and I. S. Torsun
An Empirical Analysis of Cobol Programs
Software–Practice and Experience, Vol. 9, No. 5, 1979. pp. 341–359

[Brookes82]
G. R. Brookes, I. R. Wilson and A. M. Addyman
A Static Analysis of Pascal Program Structures
Software–Practice and Experience, Vol. 12, 1982. pp. 959–963

[De Prycker82]
M. De Prycker
On the Development of a Measurement System for High Level Language Program Statistics
IEEE Transactions on Software Engineering, Vol. C–31, No. 9, September 1982. pp. 883–891

[Donzeau80]
V. Donzeau–Gouge, G. Huet, G. Kahn and B. Lang
Programming Environments Based on Structured Editors: the MENTOR Experience
Rapport de Recherche INRIA, No. 26, 1980.

[Donzeau83]
V. Donzeau–Gouge, G. Kahn, B. Lang, B. Mélèse and E. Morcos
Outline of a Tool for Document Manipulation
IFIP'83, Paris , 1983.

[Hill74]
M. O. Hill
Correspondence Analysis: a neglected multivariate method,
Applied Statistics, Vol. 23, No. 3, 1974. pp. 340–354

[McCabe76]
T. J. McCabe
A Complexity Measure
IEEE Transactions on Software Engineering, Vol. SE–2, No. 4,Deember 1976. pp. 308–320

[McDaniel82]
G. McDaniel
An Analysis of a Mesa Instruction Set Using Dynamic Instruction Frequencies
SIGPLAN Notices, Vol. 17, No. 4, April 82. pp. 167–176

[Perrott81]
R. H. Perrott and P. S. Dhillon
An Experiment with Fortran and Pascal
Software–Practice and Experience, Vol. 11, 1981. pp. 491–496

[Schroeder78]
A. Schroeder
Proceedings of the CPEUG Meeting, Boston, October 1978.

[Schroeder84]
A. Schroeder
Integrated Program Measurement and Documentation Tools
7th International Conference on Software Engineering, Orlando, Florida (USA),March 1984.

[Shimasaki80]
M. Shimasaki, S. Fukaya, K. Ikeda and T. Kiyono
An Analysis of Pascal Programs in Compiler Writing
Software–Practice and Experience, Vol. 10,, 1980. pp. 149–157

[Sweet82]
R. E. Sweet and J. G. Sandman Jr.
Empirical Analysis of the Mesa Instruction Set
SIGPLAN Notices, Vol. 17, No. 4, April 82. pp. 167–176

[Wiecek82]
C. A. Wiecek
A Case Study of VAX-11 Instruction Set Usage for Compiler Execution
SIGPLAN Notices, Vol. 17, No. 4, April 82. pp. 177–184

[Zweben77]
S. H. Zweben
A Study of the Physical Structure of Algorithms
IEEE Transactions on Software Engineering, Vol. SE–3, No. 3, May 1977. pp. 250–258

[Zweben79b]
S. H. Zweben and K. C. Fung
Exploring Software Science Relations in Cobol and APL
Proceedings of the COMPSAC 79, IEEE Cat. No. CH1515, 1979. pp. 702–

	Total	CR	AS+ VE	FL	LP
# blocks	766	354	144	57	211
PROGRAMME	16	4	10	1	1
PROCEDURE	520	234	97	35	164
FUNCTION	230	126	37	21	46
LEXP	8034	2999	3390	408	1238
CALL	5336	2331	1595	393	1017
ASS	4975	2539	1221	365	850
INDEX	3100	968	1713	86	333
LSTAT	2700	1184	867	174	475
DOT	2081	812	605	95	569
IF	1911	1175	425	90	221
EQL	957	561	243	30	123
COLON	857	514	173	84	87
LCST	852	508	173	84	87
PLUS	648	124	339	74	111
FOR	513	116	323	1	61
NEQ	470	295	97	26	52
UPSTEP	457	97	290	13	57
MINUS	287	131	93	29	34
SETOF	244	189	6	34	15
AND	202	120	52	6	24
LELEM	172	120	5	34	13
WHILE	159	60	25	19	55
FORMAT	150	23	108	2	17
OR	132	89	24	3	16
LSS	128	62	40	15	11
NOT	125	74	17	6	28
GOTO	125	99	24	2	0
GTR	123	44	46	14	19
CASE	106	66	7	17	16
IN	106	61	5	25	15
WITH	103	66	8	0	29
L VARBL	102	66	8	0	28
LCOLON	101	63	7	17	14
LDEFID	98	0	98	0	0
MULT	92	2	71	18	9
LEQ	90	74	8	1	7
UMINUS	78	70	3	0	5
LABSTAT	76	63	11	2	0
REPEAT	72	30	21	6	15
LPARAM	53	0	53	0	0
GEQ	45	28	13	1	3

Table 1

	FL		LP	
	static	dynamic	static	dynamic
# blocks	57		211	
PROGRAMME	1		1	
PROCEDURE	35		164	
FUNCTION	21		46	
LEXP	18.1	18.5	20.8	20.5
CALL	17.4	14.8	16.9	4.0
ASS	16.2	18.4	14.1	15.9
INDEX	3.8	6.7	5.5	9.2
LSTAT	7.7	9.9	7.9	9.0
DOT	4.2	2.3	9.4	14.4
IF	4.0	8.2	3.7	7.0
VIDE	3.1	2.9	3.6	.5
EQL	1.3	1.0	2.0	4.4
COLON	3.7	1.3	1.4	3.0
LCST	3.7	1.3	1.4	3.0
PLUS	3.3	4.3	1.8	4.5
FOR	.6	.2	1.0	.2
NEQ	1.2	.4	.9	1.2
UPSTEP	.6	.4	.9	.2
MINUS	1.3	1.2	.6	1.2
SETOF	1.5	2.1	.2	.3
AND	.3	.2	.4	1.0
LELEM	1.5	2.1	.2	.3
WHILE	.8	.5	.9	.9
FORMAT	.1	.0	.3	.0
OR	.1	.0	.3	1.0
LSS	.7	.2	.2	1.1
NOT	.3	.1	.5	.8
GOTO	.1	.0	.0	.0
GTR	.6	.2	.3	1.9
CASE	.8	.3	.3	.3
IN	1.1	2.1	.2	.4
WITH	.0	.0	.5	.3
L VARBL	.0	.0	.5	.3
LCOLON	.8	.3	.2	.3
LDEFID	.0	.0	.0	.0
MULT	.8	.2	.1	.2
LEQ	.0	.0	.1	1.1
LABSTAT	.1	.1	.0	.0
REPEAT	.3	.1	.2	.2
LPARAM	.0	.0	.0	.0
GEQ	.0	.0	.0	.0

Table 2

							Brookes			Shimasaki			Perrott
	Total	CR	FL	VE	AS	LP	P6	Pi	Si	12	34	5	
# blocks	766	354	57	119	25	211	192	90	173	251	414	484	37
PROGRAM	16	5	1	7	3	1	1	5	5	2	2	7	1
PROCEDURE	520	224	35	78	19	164	182	75	137	242	398	465	36
FUNCTION	230	126	21	34	3	46	9	10	31	7	14	12	0
CALL	5336	2331	393	1379	216	1017	1176	975	906	2773	3836	2518	77
std. pr.	1119	363	175	254	147	180	242	782	401	295	311	216	--
user pr.	4217	1968	218	1125	69	837	934	193	505	2478	3525	2302	--
:=	4975	2539	365	860	361	850	1887	760	1784	3122	3198	1333	357
IF	1911	1175	90	354	71	221	729	173	501	1628	1875	567	85
CASE	106	66	17	7	0	16	35	16	18	55	58	56	6
FOR	513	116	13	157	166	61	57	75	358	56	81	61	99
WHILE	159	60	19	11	14	55	59	36	38	99	139	27	23
REPEAT	72	30	6	20	1	15	31	16	14	94	79	49	4
GOTO	125	99	2	11	13	0	25	7	106	26	46	0	0
WITH	103	66	0	8	0	29	177	62	17	286	368	193	33
lab. st.	76	63	2	2	9	0	23	3	70	20	28	0	0

(NB) std. pr. = standard procedures
user pr. = user procedures
lab. st. = labelled statement

Table 3

						Brookes			Shimasaki			Perrott	Prycker	
	CR	FL	VE	AS	LP	P6	Pi	Si	12	34	5		Tr	Sp
# prgs.	5	1	7	3	1	1	5	5	2	2	7	1	1	1
CALL	36.3	43.4	49.3	25.7	45.5	29.4	47.4	24.3	35.3	41.2	54.6	11.8	33.1	25.6
:=	39.6	40.3	30.7	42.9	38.0	47.2	36.9	47.9	39.8	34.3	28.9	54.8	51.2	47.4
IF	18.3	9.9	12.6	8.4	9.9	18.2	8.4	13.4	20.7	9.4	12.3	13.1	8.7	9.6
CASE	1.0	1.9	.3	.0	.7	.9	.8	.5	.7	.6	1.2	.9	.0	.0
FOR	1.8	1.4	5.6	19.7	2.7	1.4	3.6	9.6	.7	.9	1.3	15.2	4.6	8.8
WHILE	.9	2.1	.4	1.7	2.5	1.5	1.7	1.0	1.3	1.5	.6	3.5	.6	8.0
REPEAT	.5	.7	.7	.1	.7	.8	.8	.4	1.2	.8	1.1	.6	1.2	.8
GOTO	1.5	.2	.4	1.5	.0	.6	.3	2.8	.3	.5	.0	.0	.6	.0

Table 4

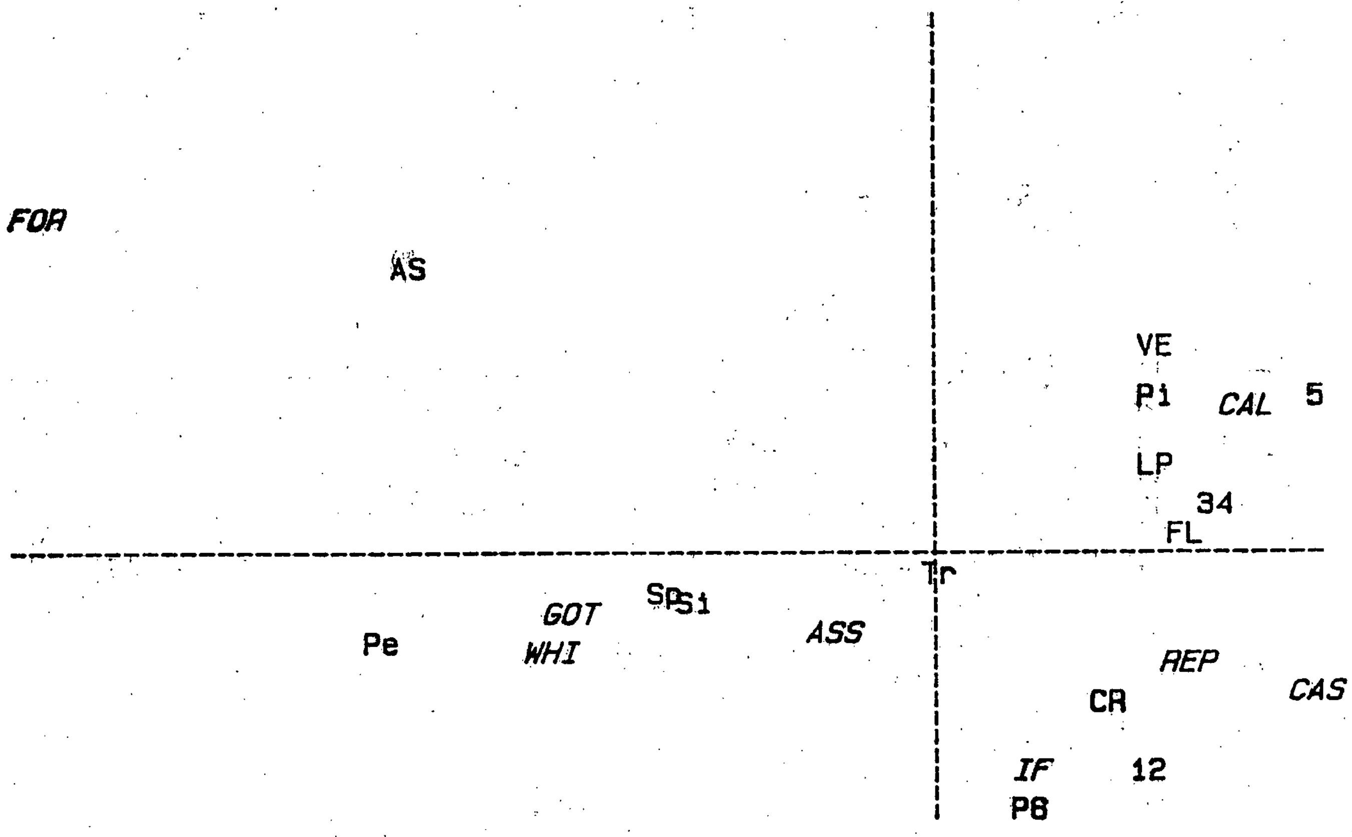
	FL		LP		DePrycker SPLIT		DePrycker TREE	
# executions	3		4		3		2	
# executed statements	66852		4273877		162018		204372	
	static	dynamic	static	dynamic	static	dynamic	static	dynamic
CALL	43.4	34.8	45.5	13.9	25.6	15.5	33.1	23.9
:=	40.3	43.4	38.0	55.8	47.4	74.7	51.2	50.1
IF	9.9	19.3	9.9	24.7	9.6	7.4	8.7	21.7
CASE	1.9	.6	.7	1.0	.0	.0	.0	.0
FOR	1.4	.4	2.7	.7	8.8	1.4	4.6	3.7
WHILE	2.1	1.1	2.5	3.2	8.0	.9	.6	.0
REPEAT	.7	.3	.7	.6	.8	.0	1.2	.8
GOTO	.2	.0	.0	.0	.0	.0	.6	.0

Table 5

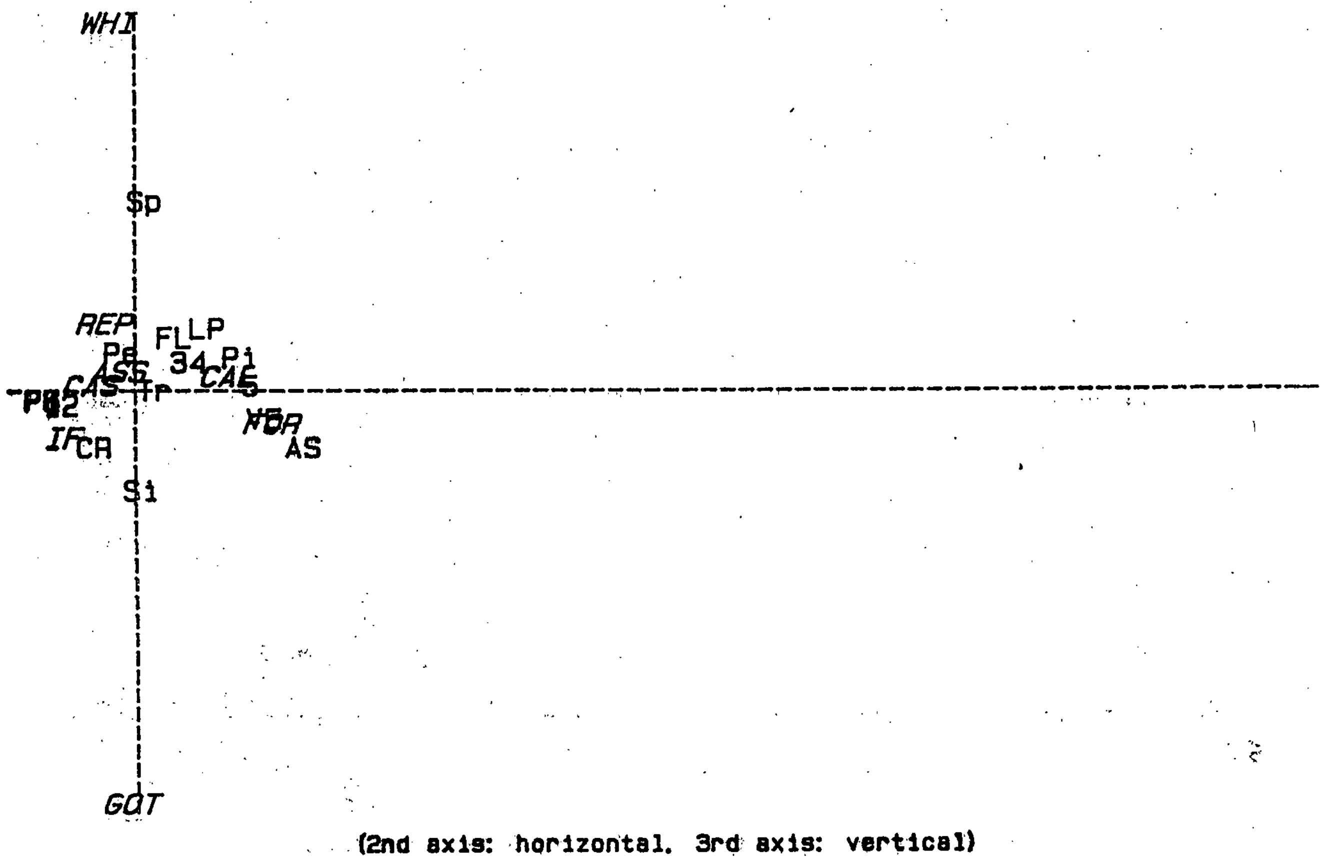
Correspondence Analysis

(8 statement types, 14 origins)

1st Axis :	61.3%
2nd Axis :	18.0%
3rd Axis :	13.3%
4th Axis :	4.2%



(1st axis: horizontal, 2nd axis: vertical)



(2nd axis: horizontal, 3rd axis: vertical)

Figure 1

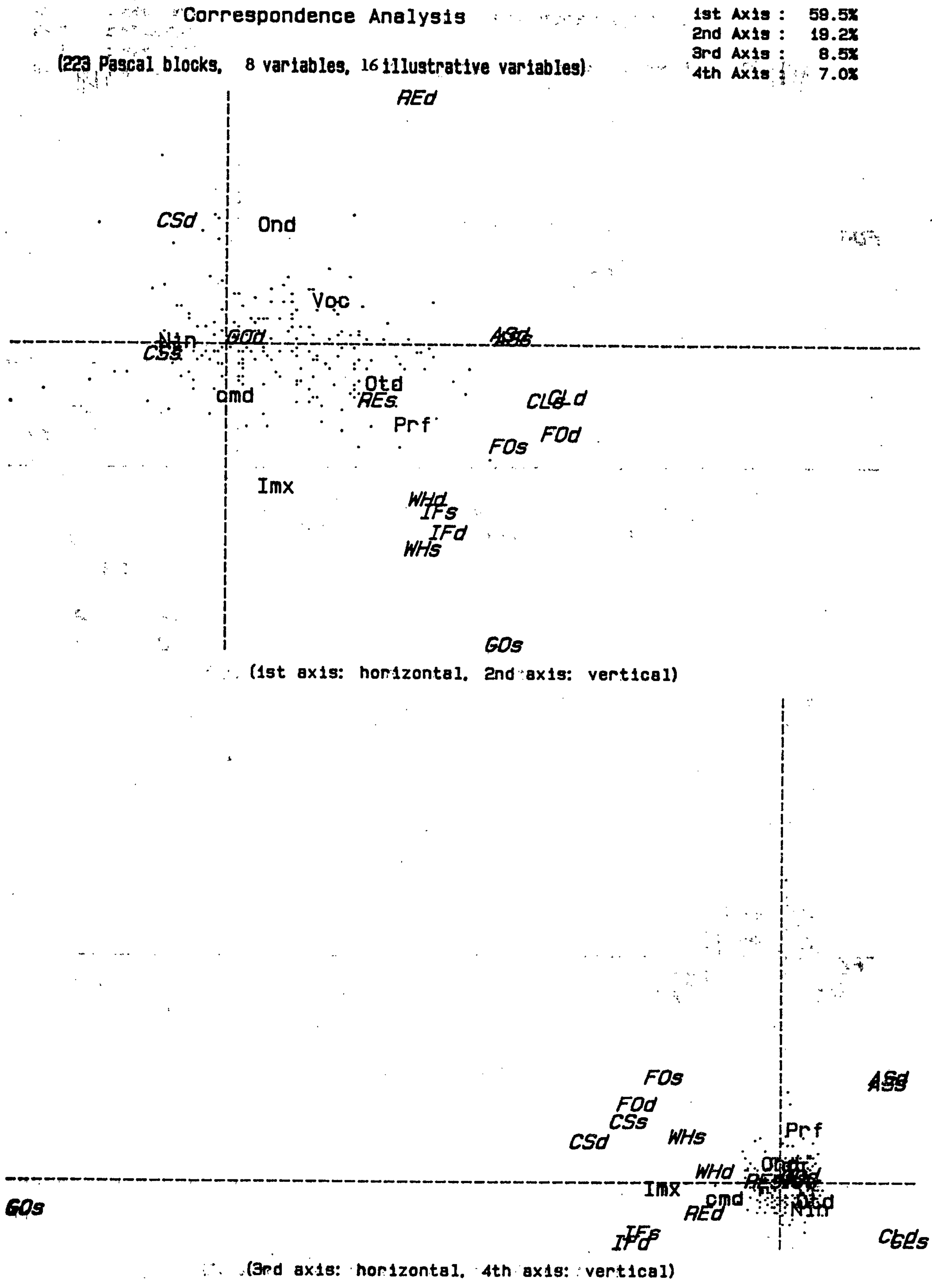

Correspondence Analysis
(223 Pascal blocks, 8 variables, 16 illustrative variables)
1st Axis : 59.5%
2nd Axis : 19.2%
3rd Axis : 8.5%
4th Axis : 7.0%
REd
CSd
Ond
Voc
Nin
CSs
GOd
ASd
ASs
Otd
cmd
REs
Prf
CLd
CLs
FOd
FOs
Imx
WHd
IFs
IFd
WHs
GOs
(1st axis: horizontal, 2nd axis: vertical)
FOs
ASd
ASs
FOd
CSs
CSd
WHs
Prf
WHd
Otd
GOs
Imx
cmd
REd
Nin
IFs
IFd
CLd
CLs
(3rd axis: horizontal, 4th axis: vertical)

Figure 2

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique