stichting
mathematisch
centrum

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 109/79 MEI

D. GRUNE

SOME STATISTICS ON ALGOL 68 PROGRAMS

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

MS classification scheme (1980): 68A15,68A20

ACM-Computing reviews-categories: 4.6,4.22,4.12

Some Statistics on ALGOL 68 Programs*)

by

Dick Grune

ABSTRACT

An attempt is made to assess some static and dynamic properties of ALGOL 68 programs, which are useful for optimization decisions. The results indicate that slicing and assignation are the most important candidates for optimization, and that optimization efforts need to be directed to the simple cases only.

KEY WORDS & PHRASES: compiler construction, optimization, ALGOL 68

---

*) This report will be submitted for publication elsewhere.

1. THE PROBLEM

For the design of the code generator of the MC ALGOL 68 Compiler we are interested in the frequency of language constructs in normal run-of-the-mill ALGOL 68 programs [1, 2]. Knowledge of these frequencies can guide us on what to optimize, or, if we do not want to optimize now, at least prevent us from making decisions which would rule out useful optimizations later on.

The 'frequency of language constructs in normal programs' is not a very precise notion and it is not easy to determine. There is no good definition of a 'normal program' and we need a full parser to identify and count 'language constructs'.

We can, however, try to get an approximation. Rather than defining 'normal programs' and a distribution, we can take a number of existing real-world programs. ALGOL 68 is used extensively at our installation (Control Data Cyber 72), where 7 % of all compilations are ALGOL 68, so we have the opportunity.
And rather than tinkering with the existing compiler (which we cannot do) we can do statistical analysis on the texts of the programs and try to interpret the results.

Much of the philosophy developed by Knuth in his study of FORTRAN programs [3] applies to this work as well.

Similar investigations have been done for ALGOL 60 [4], PL/I [5] and COBOL [6].

2. THE STATIC BEHAVIOUR

2.1. Simplifying transformations

We collected 53 real-world user programs (in total 8131 lines) by asking users. These programs were subjected to the following transformations (through editing, UNIX-commands and devious means):

1. comments and pragmats were deleted;
2. mode- and priority-declarations were removed;
3. all tags were replaced by 'tag',
   all denotations by 'denotation',
   all user operators by 'user_operator',
   all user mode indications by 'user_mode' and
   all colons by 'label_token', 'colon_token' (in specifications), 'up_to_token' (in rowers and in trimmers) or 'routine_token', as appropriate;

4. SKIP and NIL were replaced by 'denotation';
5. parentheses in parameter-packs in calls were recognized;
6. brackets were split in indexers and rowers;
7. symbols that come in pairs or triples were taken together (like (), [], IF THEN FI, etc.);
8. all different representations of the same operator were taken together (e.g. +:= and PLUSAB), except those for = and EQ.

2.2. Counting symbols

The symbols were then counted and sorted in descending frequency, which yields the following table.

Table I, Symbol Count

<table>
<tr><td>17209</td><td>tag</td><td>104</td><td>ABS</td></tr>
<tr><td>5916</td><td>denotation</td><td>101</td><td><</td></tr>
<tr><td>5200</td><td>,</td><td>81</td><td>ELIF THEN</td></tr>
<tr><td>3457</td><td>;</td><td>77</td><td>CASE IN ESAC</td></tr>
<tr><td>2362</td><td>( )</td><td>76</td><td><=</td></tr>
<tr><td>1892</td><td>:=</td><td>72</td><td>AND</td></tr>
<tr><td>1850</td><td>indexer</td><td>69</td><td>-:=</td></tr>
<tr><td>1488</td><td>=</td><td>65</td><td>STRING</td></tr>
<tr><td>1242</td><td>call</td><td>65</td><td>~</td></tr>
<tr><td>1061</td><td>user_mode</td><td>59</td><td>OR</td></tr>
<tr><td>961</td><td>-</td><td>55</td><td>IS</td></tr>
<tr><td>793</td><td>OF</td><td>54</td><td>**</td></tr>
<tr><td>714</td><td>*</td><td>53</td><td>BY</td></tr>
<tr><td>662</td><td>REAL</td><td>48</td><td>ISNT</td></tr>
<tr><td>626</td><td>DO OD</td><td>46</td><td>|: |</td></tr>
<tr><td>618</td><td>+</td><td>41</td><td>*:=</td></tr>
<tr><td>572</td><td>INT</td><td>41</td><td>>=</td></tr>
<tr><td>547</td><td>up_to_token</td><td>39</td><td>label_token</td></tr>
<tr><td>540</td><td>REF</td><td>37</td><td>@</td></tr>
<tr><td>501</td><td>routine_token</td><td>37</td><td>CHAR</td></tr>
<tr><td>499</td><td>TO</td><td>22</td><td>/:=</td></tr>
<tr><td>480</td><td>FOR</td><td>20</td><td>LOC</td></tr>
<tr><td>461</td><td>|</td><td>13</td><td>FILE</td></tr>
<tr><td>413</td><td>rower</td><td>12</td><td>%</td></tr>
<tr><td>396</td><td>IF THEN FI</td><td>12</td><td>+=:</td></tr>
<tr><td>361</td><td>UPB</td><td>11</td><td>OUT</td></tr>
<tr><td>355</td><td>/</td><td>11</td><td>STRUCT</td></tr>
<tr><td>326</td><td>PROC</td><td>10</td><td>SIGN</td></tr>
<tr><td>278</td><td>user_operator</td><td>9</td><td>ELEM</td></tr>
<tr><td>228</td><td>FROM</td><td>8</td><td>MOD</td></tr>
<tr><td>211</td><td>+:=</td><td>6</td><td>ENTIER</td></tr>
<tr><td>202</td><td>ELSE</td><td>6</td><td>%:=</td></tr>
<tr><td>197</td><td>VOID</td><td>6</td><td>ROUND</td></tr>
<tr><td>150</td><td>LWB</td><td>4</td><td>BITS</td></tr>
<tr><td>148</td><td>HEAP</td><td>4</td><td>EXIT</td></tr>
<tr><td>147</td><td>BEGIN END</td><td>4</td><td>UNION</td></tr>
<tr><td>145</td><td>OP</td><td>2</td><td>FLEX</td></tr>
</table>

145	WHILE	1	GOTO
143	>	1	ODD
132	/=	1	OUSE IN
121	colon_token	1	REPR
108	BOOL		

This table gives rise to some observations.

The meaning of some symbols is very unclear. Prime example is the = , which may be a dyadic operator or an is-defined-as-token; only profound analysis can tell the difference.

The first two items in the list correspond to loading a value, which can also be considered part of the operator that uses the result; and the next three items are not connected to any semantic action at all in a reasonable implementation. It is true that the semicolon signifies 'voiding' which technically would amount to discarding a result, but in practice no code needs to be generated. The first to require real action is the := . So it might be useful to weed from the list all symbols that are not directly connected to a run-time action (however, the above list does not contain the "invisible" actions involved in coercions). This yields:

Table II, Action Count

<table>
<tr><td>1892</td><td>:=</td><td>72</td><td>AND</td></tr>
<tr><td>1850</td><td>indexer</td><td>69</td><td>-:=</td></tr>
<tr><td>1488</td><td>=</td><td>65</td><td>~</td></tr>
<tr><td>1242</td><td>call</td><td>59</td><td>OR</td></tr>
<tr><td>961</td><td>-</td><td>55</td><td>IS</td></tr>
<tr><td>793</td><td>OF</td><td>54</td><td>**</td></tr>
<tr><td>714</td><td>*</td><td>53</td><td>BY</td></tr>
<tr><td>618</td><td>+</td><td>48</td><td>ISNT</td></tr>
<tr><td>499</td><td>TO</td><td>46</td><td>|: |</td></tr>
<tr><td>461</td><td>|</td><td>41</td><td>*:=</td></tr>
<tr><td>413</td><td>rower</td><td>41</td><td>>=</td></tr>
<tr><td>396</td><td>IF THEN FI</td><td>37</td><td>@</td></tr>
<tr><td>361</td><td>UPB</td><td>22</td><td>/:=</td></tr>
<tr><td>355</td><td>/</td><td>20</td><td>LOC</td></tr>
<tr><td>278</td><td>user_operator</td><td>12</td><td>%</td></tr>
<tr><td>228</td><td>FROM</td><td>12</td><td>+=:</td></tr>
<tr><td>211</td><td>+:=</td><td>10</td><td>SIGN</td></tr>
<tr><td>150</td><td>LWB</td><td>9</td><td>ELEM</td></tr>
<tr><td>148</td><td>HEAP</td><td>8</td><td>MOD</td></tr>
<tr><td>145</td><td>WHILE</td><td>6</td><td>ENTIER</td></tr>
<tr><td>143</td><td>></td><td>6</td><td>%:=</td></tr>
<tr><td>132</td><td>/=</td><td>6</td><td>ROUND</td></tr>
<tr><td>104</td><td>ABS</td><td>1</td><td>GOTO</td></tr>
<tr><td>101</td><td><</td><td>1</td><td>ODD</td></tr>
<tr><td>81</td><td>ELIF THEN</td><td>1</td><td>OUSE IN</td></tr>
<tr><td>77</td><td>CASE IN ESAC</td><td>1</td><td>REPR</td></tr>
</table>

76 <=

It is tempting to put percentages into this list and say that "13 % of all semantic actions are assignations", but this is meaningful only if all the symbols given above correspond to actions of the same complexity, which is, of course, not true. Our objective is to find constructions which merit our attention in optimization; it is clear that assignations and slicing are the great winners.

Other constructions can be identified which do not show up directly in the tables. One is the 'boolean-enquiry-clause'; its frequency can be found by adding those of IF-THEN-FI, ELIF-THEN, WHILE and a percentage of | (which may represent THEN, ELSE, IN or OUT), and of |: | (which may be ELIF-THEN or OUSE-IN). If we make the only reasonable but totally unwarranted assumption that the brief symbols occur in the same ratio as the bold symbols, we find that 270 |'s are THEN's and 45 |:'s are ELIF's.

Another construction is 'standard-operator', which can be identified but is of doubtful use: the field is too wide for determined optimization. On the other hand, they are so numerous that not identifying them would also give a false impression. We then arrive at the following table.

Table III, Summary

4420	standard_operator	228	FROM
1892	:=	148	HEAP
1850	indexer	128	CASE IN ESAC
1488	=	55	IS
1242	call	53	BY
937	boolean_enquiry	48	ISNT
793	OF	37	@
499	TO	20	LOC
413	rower	2	OUSE IN
278	user_operator	1	GOTO

The main constructs of interest are assignations, slices and calls. A further analysis (through more editing etc.) is given in the following tables ('simple' means 'identifier or denotation', s.slice means 'slice with simple indexers only', s.selection means 'selection on an identifier' and s.formula means 'formula with one standard operator and one or two simple operands').

Assignations.

destination:		source:	
simple:	71 %	simple:	45 %
s.slice:	15 %	s.slice:	5 %
s.selection:	4 %	s.selection:	5 %
		s.formula:	8 %
rest:	10 %	rest:	37 %

Slices.

primary:		indexer:	
simple:	89 %	one, simple:	58 %
s.slice:	4 %	more, simple:	20 %
s.selection:	4 %	trimmer:	8 %
rest:	3 %	rest:	14 %

Calls.

primary:		parameters:	
simple:	100 %	one, simple:	22 %
		more, simple:	19 %
		'print' etc:	17 %
		rest:	42 %

All this suggests very strongly that it is most efficient to direct the optimization effort to the simple cases only.

2.3. Denotations

The denotations extracted from the text in point 3 in paragraph 2.1 were distributed as follows,

3912	int	194	nil
813	real	148	skip
596	string	18	format
233	bool	2	bits

whereas the integral-denotations were classified thus:

value (range)	freq.
0	601
1	1628
2:3	663
4:15	664
16:255	299
256:4095	53
>4095	6

One conclusion from this is that a reasonable implementation on the IBM 370 may put integers smaller than 4096 in the instruction (LA) and use horrible code for the rest.

2.4. Identifiers

The distribution of identifier-lengths was as follows:

freq.	length	freq.	length
6539	1	29	14
3200	2	11	15
1985	3	12	16
1792	4	14	17
1345	5	2	18
690	6	2	20
731	7	3	21
190	8	3	24
350	9	2	27
168	10	4	34
129	11	1	42
138	12	1	50
52	13	1	52

or, if we consider different identifiers only:

freq.	length	freq.	length
26	1	13	14
295	2	6	15
232	3	4	16
270	4	6	17
168	5	2	18
153	6	1	20
102	7	2	21
61	8	2	24
88	9	1	27
69	10	2	34
43	11	1	42
35	12	1	50
14	13	1	52

This may provide trade-off information for the identifier-table algorithm.

The 10 most frequent identifiers were:

976	i	359	r
581	n	339	s
564	k	324	b
558	a	305	x
376	j	302	newline

3. THE DYNAMIC BEHAVIOUR

All the above measurements pertain to the static text of the program. We would, however, like to get some insight in the dynamic importance of the various constructs. Now such results are hard to come by and have a inherently large inaccuracy. We therefore decided to accept a static (textual) analysis of the innermost do-parts as a reasonable estimate of the dynamic behaviour of the program, on the (not too well founded) assumption that these parts are the most heavily executed pieces of code.

The same process as above yields the following tables:

Table IV, Symbol Count in Inner Do-parts

<table>
<tr><td>4021</td><td>tag</td><td>13</td><td>CASE IN ESAC</td></tr>
<tr><td>916</td><td>indexer</td><td>10</td><td>ELIF THEN</td></tr>
<tr><td>907</td><td>denotation</td><td>9</td><td>AND</td></tr>
<tr><td>879</td><td>,</td><td>9</td><td><</td></tr>
<tr><td>504</td><td>:=</td><td>8</td><td>OR</td></tr>
<tr><td>406</td><td>;</td><td>7</td><td>IS</td></tr>
<tr><td>355</td><td>( )</td><td>7</td><td>/:=</td></tr>
<tr><td>269</td><td>call</td><td>7</td><td>~</td></tr>
<tr><td>222</td><td>*</td><td>6</td><td>ELEM</td></tr>
<tr><td>193</td><td>OF</td><td>6</td><td>ISNT</td></tr>
<tr><td>166</td><td>-</td><td>6</td><td><=</td></tr>
<tr><td>159</td><td>+</td><td>6</td><td>rower</td></tr>
<tr><td>131</td><td>=</td><td>5</td><td>@</td></tr>
<tr><td>113</td><td>+:=</td><td>4</td><td>UPB</td></tr>
<tr><td>92</td><td>up_to_token</td><td>4</td><td>>=</td></tr>
<tr><td>86</td><td>/</td><td>3</td><td>ENTIER</td></tr>
<tr><td>83</td><td>IF THEN FI</td><td>3</td><td>%</td></tr>
<tr><td>74</td><td>|</td><td>3</td><td>PROC</td></tr>
<tr><td>68</td><td>user_mode</td><td>3</td><td>|: |</td></tr>
<tr><td>44</td><td>-:=</td><td>2</td><td>BEGIN END</td></tr>
<tr><td>39</td><td>REF</td><td>2</td><td>MOD</td></tr>
<tr><td>37</td><td>INT</td><td>1</td><td>BITS</td></tr>
<tr><td>35</td><td>user_operator</td><td>1</td><td>CHAR</td></tr>
<tr><td>33</td><td>REAL</td><td>1</td><td>OUT</td></tr>
<tr><td>30</td><td>**</td><td>1</td><td>%:=</td></tr>
<tr><td>28</td><td>/=</td><td>1</td><td>ROUND</td></tr>
<tr><td>27</td><td>ELSE</td><td>1</td><td>STRING</td></tr>
<tr><td>24</td><td>ABS</td><td>1</td><td>STRUCT</td></tr>
<tr><td>22</td><td>*:=</td><td>1</td><td>UNION</td></tr>
<tr><td>22</td><td>></td><td>1</td><td>+=:</td></tr>
<tr><td>18</td><td>HEAP</td><td>1</td><td>routine_token</td></tr>
<tr><td>14</td><td>colon_token</td><td></td><td></td></tr>
</table>

Table V, Action Count in Inner Do-parts

916	indexer	10	ELIF THEN
504	:=	9	AND
269	call	9	<
222	*	8	OR
193	OF	7	IS
166	-	7	/:=
159	+	7	~
131	=	6	ELEM
113	+:=	6	ISNT
86	/	6	<=
83	IF THEN FI	6	rower
74	|	5	@
44	-:=	4	UPB
35	user_operator	4	>=
30	**	3	ENTIER
28	/=	3	%
24	ABS	3	|: |
22	*:=	2	MOD
22	>	1	%:=
18	HEAP	1	ROUND
13	CASE IN ESAC	1	+=:

Table VI, Summary of Counts in Inner Do-parts

987	standard_operator	35	user_operator
916	indexer	18	HEAP
504	:=	13	CASE IN ESAC
269	call	7	IS
193	OF	6	ISNT
146	boolean_enquiry	6	rower
131	=	5	@

Although the overall picture remains the same, certain shifts in emphasis can be discerned. The slice is now clearly the most important construct, but assignation is still a powerful second. The call has lost much of its weight.

Analysis of slice and assignation gives:

Slice in Inner Do-parts.

primary:		indexer:	
simple:	86 %	one, simple:	62 %
s.slice:	7 %	more, simple:	22 %
s.selection:	4 %	trimmer:	0 %
rest:	3 %	rest:	16 %

Assignations in Inner Do-parts.

destination:		source:	
simple:	50 %	simple:	32 %
s.slice:	43 %	s.slice:	9 %
s.selection:	2 %	s.selection:	9 %
		s.formula:	37 %
rest:	5 %	rest:	13 %

We see that the assignations tend to have simpler sources now, which again suggests that optimizing the simple cases only will lead to considerable gain. The slices themselves show no real difference.

4. CONCLUSION

The main candidates for optimization efforts are slices, assignations and calls; there are indications that the first two are the most important from a dynamical point of view.

Optimization efforts need to be directed to the simple variants of the above constructions only.

This conclusion is in full agreement with the results obtained by Knuth for FORTRAN [3].

5. ACKNOWLEDGEMENT

I wish to thank Lambert Meertens for posing the problem and for numerous comments and improvements.

6. REFERENCES

[1] C.H. Lindsey and S.G. van der Meulen, Informal Introduction to ALGOL 68 Revised, North Holland Publ. Comp., Amsterdam, 1977.

[2] A. van Wijngaarden et al., editors, Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica, 5, 1-236, 1975; MC Tract 50, Mathematical Centre, Amsterdam, 1976; SIGPLAN Notices, 12, 5, 1-70, 1977.

[3] D.E. Knuth, An Empirical Study of FORTRAN Programs, Software-Practice and Experience, 1, 105-133, 1971.

[4] B.A. Wichmann, A Comparison of ALGOL 60 Executive Speeds, National Physics Laboratory, Central Computer Unit, Report 3, 1969.

[5] J.L. Elshoff, A Numerical Profile of Commercial PL/1 Programs, Software-Practice and Experience, 6, 505-525, 1976.

[6] M.M. Al-Jarrah and I.S. Torsun, An Empirical Analysis of COBOL Programs, Software-Practice and Experience, 9, 341-359, 1979.