GIER ALGOL Translator
General Pass Mechanism
3. May 1962.

Peter Naur

THE TRANSLATION PROBLEM IN MACHINES WITH A NON-HOMOGENEOUS STORE.

The translation of an ALGOL program consists basically in the combination of two bodies of information, viz. the information supplied by the ALGOL source program and that of the translation program (the translator). ~~Unless the~~ In a machine ~~has~~ with a ~~sufficiently large fast~~ non homogeneous store ~~to hold both of these bodies simultaneously~~ some division of these bodies becomes necessary. Two ways of making this division suggest themselves: In the ~~one~~ sequential method we attempt to perform ~~a~~ one purely sequential scan through the source program, taking a small section at a time. In the ~~other~~ we divide the translator into several functional subunits and let these operate on the complete source program or the partially translated intermediate versions of it in turn. ~~This latter method is then a multiscan process.~~

the sequential method and the multiscan method

multiscan method

\a

/e

In a machine which has a sufficiently large fast store to hold the complete translator and its tables the sequential method is clearly preferable. However, if the fast store is not large enough to hold the translator the multiscan process is better. This is because in a sequential process the order in which references are made to the various parts of the translator is unknown, since this order depends on the source program. Therefore, in ~~a sequential process~~ this method we are forced to ~~make~~ perform simultaneous a linear scan of the source program and ~~almost random~~ any number of references to all parts of the translator in an almost random order. In a multiscan process, on the other hand, the part of the translator used during a particular scan at some time is fixed and the only reference to a large body of information is the purely linear scan through the partially translated program. For these reasons the GIER ALGOL translator ~~will~~ uses a multiscan method.

If regarded in this manner the potential superiority of the multiscan method becomes clear. In fact

It

[es /i

It is interesting to note that on GIER this choice is entirely compatible with a high translation speed. In fact, the time for transferring a progran occupying 150 drum tracks into the core store and back again, as it will be necessary during each pass, takes only about 6 seconds. If 10 passes are necessary the total drum transfer time will thus be about 1 minute for this rather large program. It must be expected that the actual translation process will require considerably more time than this, and thus the simultaneous drum transfer facility in GIER will work to our full advantage.

INTERMEDIATE LANGUAGES.

In a multi~~pass~~ scan translator the choice of intermediate languages becomes of prime importance. The following conflicting factors have to be considered: From the point of view of the individual scans (or translator 'passes') great flexibility and a great multitude of structures of varying length is desirable. From the point of view of the programs which have to perform the packing and unpacking of information, which is necessary during each scan, uniformity of structure is desirable. Uniformity is also desirable for translator checking purposes.

scan
e

In view of these considerations the following compromize has ~~been~~ adopted: All intermediate languages are expressed in terms of basic units of information each of 10 bits. This unit is called a 'byte'. Thus the input to each pass and the output from it will always consist of a uniform string of bytes. Within each intermediate language any amount of structure within this byte string may be employed in order to communicate more intricate structures. Thus, for example, a number in the original

/w

ALGOL program will in most of the intermediate languages be represented by 5 consecutive bytes, the first being a destinct 'number mark' while the remaining four supply the actual value of the number. However, from the point of view of the general packing and unpacking program the various intermediate languages are undistinguishable, since they all consist of just a uniform byte string. /i

By this method the programs which will perform the packing and unpacking of the intermediate versions of the program and which will take them from the drum and store them on the drum will be the same for all passes. In addition since the output from each pass is always of the same form the same test output program can be used for all passes and can be coupled to the common pass administration. ~~This will probably facilitate the checking of the translator greatly.~~

REVERSE SCANS, ~~AND ERROR REMOVAL.~~

In a multipass translator it is highly advantageous to let some of the scans be reverse scans, i.e. scans which start at the end of the program and move towards the begin. First of all the problem of forward references clearly is solved complete in this manner. In addition it becomes easy to eliminate syntactically incorrect sections of the program during the translation. This is important because it is highly desirable that an error which has been detected does not prevent the translation process from continuing to check the rest of the program. /e ely

~~The way of doing this in the GIER ALGOL translator is as follows: During the syntactical check of the program (pass 3) a special byte 'statement start' is output at every point where an ALGOL statement starts in the text. If a syntactical error is found in a statement a special byte 'trouble' is output and the rest of the symbols of the statement up till the first following semicolon (;) or end are skipped completely. The following scan is a backward scan. Every time this scan finds the byte 'trouble' it will skip all bytes back to the first 'statement start' point. In this way the complete incorrect text is removed. At the same time various parameters describing the state of the translation are reset to appropriate values to enable the translation to continue to translate the following statement correctly.~~

Two of the 9 passes of GIER Algol are reverse scans.

~~ERROR SIGNALING.~~

~~The translator includes an extensive checking of the formal correctness of the source program. Every time an error is found an error description will be produced in the output from the translator. This will include the location and kind of the error. The location will be indicated by the number of the line in the original ALGOL test where it occurs. In order to facilitate the identification of lines the first pass of the translation will produce a copy of every 10'th line of the program in its output, with the line number inserted at the beginning of the line. The kind of the error will be described by an appropriate error message.~~

STORAGE OF THE PARTIALLY TRANSLATED PROGRAM.

According to the description given above all passes use the same general pass administration. As far as the individual pass programs are concerned the general pass administration is a subroutine with two entries, one for input and one for output. In fact these two entries are almost completely independent since they use two indenpendent buffers in their communication with the drum. The general pass administration uses 4 sections of 40 words each in the core store. Two of these are used as buffers for the input to the pass programs, the other two for the output from the pass program. Normally one of the input buffers holds the bytes of that section of the program which is presently being processed by the pass program while the other input buffer holds the next input drum track. When all the bytes on the active buffer have been used the transfer from drum of the next following drum track is initiated. At the same time the pass program can proceed to process the input bytes waiting in the other input buffer. A similar buffering technique is used on the output side.

The bytes are packed into the GIER words with 4 bytes in a word. Thus one drum track holds 160 bytes. The unpacking (on the input side) and packing (on the output side) of the bytes into the words are performed by the general pass administration. This turns out to be a more time- consuming process than the corresponding drum transfers. In fact the average unpacking time per byte is about 220 microseconds, which means that the unpacking of the 160 bytes on a track will take about 35 milliseconds. This again means that except for collisions between drum transfers called from the input and output side the time for drum transfers will be negligible, owing to the parallel operation during drum transfers in GIER. The time for packing on the output side being about 260 microseconds per byte, the total time for the administration of a pass becomes about 77 milliseconds per drum track plus the time wasted due to collisions of drum transfers, about 5 milliseconds on the average. The processing time per byte of course varies greatly, the minimum being about 80 microseconds (direct copying from input to output). At an early stage of the development we guessed that an average byte processing time of 500 microseconds, corresponding to about 10 instruction execution times, might be expected. On this assumption ~~while a normal figure of perhaps 500 microseconds may be expected. If this holds~~ the pass administration and processing times are comparable and the total time for 10 passes will be about 2 seconds per drum track. This is proved to correspond closely to the normal performance.

On the drum the tracks holding the partially translated program are used in a cyclic manner. ~~Probably~~ About half the drum ~~will~~ is available for this, while the rest of the drum ~~will~~ holds the translator and the running system programs. Suppose that the available tracks are numbered from 1 to N. The first pass will then place its output in tracks no. 1 to M, say, where clearly M ≤ N. The second pass will take its input from tracks no. 1, 2, etc. and will place its output in tracks M+1, M+2, etc. When output to track no. N has been made the administration will continue to output into track no. 1, 2, etc. which presumably have now been released by the input side administration. This process is continued smoothly from one pass to the next, except for the case that the direction of scanning is reversed. This does not cause any difficulty however, since the cyclic use of the drum tracks may with equal ease take place in either direction. Only in the programs which perform the packing and unpacking a few changes are necessary.

MACHINE ARRANGEMENT OF THE GENERAL PASS ADMINISTRATION.

ALGOL descriptions of the general pass administrations are found below. The following notes show the storage of the translator on the drum and the use of the core store during translation.

Drum: 320 tracks (40 words each)

SLIP address		
0-31	SLIP (symbolic input program)	
32-	Running system	
1014	Forward pass	see core store
2014	Backward pass	16 - 55
3014	testprint, print, new line, entry to message (1 track total)	
4014	endpass (1 track)	
5014	message (output program, 1 track)	
6014	texts for message (1 track)	
7014-	Pass n	Translator pass programs
-	Pass n-1	Translator pass programs
	. . .	Translator pass programs
-	Pass 1	Translator pass programs
-	Partially translated ALGOL program (tracknumbers FIRST to LAST)	
LAST+1 -319	Table of strings, formed during pass 1	

Core store: 1024 words

Absolute address		
0-15	Reserved for SLIP	
16-54	Drum and packing administration of partially translated program (forward or backward).	
55	Base in = e13	
56-95	Input buffer 1	Also used for texts by message
96	Buffer stop	Also used for texts by message
97-136	Input buffer 2	Also used for texts by message
137	Base out	
138-177	Output buffer 1	Also used for message
178	Buffer stop	Also used for message
179-218	Output buffer 2	Also used for message
219	Buffer stop	
220-259	testprint, print, new line, entry to message, universal translator parameters	
261- 999	Translator pass programs and their tables	
960-999	Also used for endpass	
1000 -1023	Reserved for SLIP	

Taking the tracks of the drum in order, this gives a general survey of the meaning of the various parts:

SLIP is the input program used for reading the translator into the machine. When the translator is completed this will of course become unnescessary and the complete translator be shifted forward on the drum.

The running system is the set of administrative programs and standard procedures used by the running ALGOL program, when the translation is completed. See GIER ALGOL Running System.

Forward pass and backward pass are the programs performing the packing, unpacking and drum transfers described above in the section on STORAGE OF THE PARTIALLY TRANSLATED PROGRAM. One of these tracks will be placed from 16 to 54 in the cores. Two seperate programs have been written in order to make sure that this central process runs at the highest possible speed. The pass program appropriate to each separate pass is transferred to the cores by endpass.

The track holding the programs testprint, print, new line, and entry to message, in addition to certain universal translator parameters, is permanently held in the core store 220-259. Entry to message is used for transferring the message printing program from drum. The message program will be placed in that one of the two output buffers which is not currently being used for collecting output. This output buffer will always have been transferred to drum before it is overwritten by message and therefore need not be saved. On exit message will only have to make sure that the output buffer is left with the correct marks on all words.

The next track holds the program endpass which is used to perform the transition from one pass to the next. When used this program is transferred to 960-999 in the core store. Endpass does the following: 1) The last output buffer is filled up with dummy bytes until it is transferred to the drum. 2) Information about the pass which has just been completed is printed. This will always include the number of used tracks. 3) According to a table held by endpass itself the direction of scan is reversed if necessary, the appropriate pass program (forward or backward) is transferred to 16-54 in the cores, and the program for the pass itself is transferred to locations 261 and following.

The message program is a text printing program. As described above it will be placed in one of the two output buffers when it is needed. It starts by transferring a track of text information from the following track to that one of the two input buffers which has last been transferred from the drum and which has therefore not yet been processed. At the end of its work the message program must restore the input track which was overwritten by the text track.

The next section of the drum holds the translation programs proper. This is followed by the working section of the drum (see the section on STORAGE OF THE PARTIALLY TRANSLATED PROGRAM above).

The very top end of the drum is used for holding a table of strings in the ALGOL program. This is formed during the very first pass and will reserved as much of the working section of the drum as it needs. This is possible during pass 1 because the cyclic use of the working tracks has not yet had the chance of moving to the top of store (unless the program is too big for the machine).

ALGOL PROGRAMS FOR THE PASS ADMINISTRATIONS.

The following ALGOL descriptions follow the machine codes sufficiently closely to be of help in desiphering these. It should be noted, first of all, that the input and output procedures are highly 'sneaky' since they use and change a number of nonlocal variables. These variables are:

```
integer output word address, output byte number, number of used tracks,
        number of available tracks, output track, last track, input word
        address, byte address, input track;
boolean in testmode;
integer array BYTE BUFFER[1:5];
integer array WORD BUFFER[0:164];
```

The WORD BUFFER is the section in the core store from 55 to 219 (page 4). The five words placed in between and at the ends of the buffers proper are used to control the counting through their marks in GIER. The marks placed on the words in the WORD BUFFER are as follows:

	Marks	Use of buffer
WORD BUFFER 0	3	
1-40	0	Input 1
41	2	
42-81	0	Input 2
82	3	
83-122	0	Output 1
123	2	
124-163	0	Output 2
164	3	

In addition to the non-local variables a number of non-local procedures are used. Hopefully those of them which are not declared may be understood from their identifier.

```
procedure output(byte); value byte; integer byte; comment This performs
        the packing and drum transfers of the output from each pass in
ri      turn. The algo‸thm given works only for a forward pass. The one
        for backward passes is so similar that it will not be reproduced.
        For further notes, see the section on STORAGE OF THE PARTIALLY
        TRANSLATED PROGRAN on page 3;
begin switch packing:= pack first, pack second, pack third, pack fourth;
        if in testmode then testprint(byte);
        output byte number:= output byte number + 1;
        go to packing[output byte number];
pack first: output word address:= output word address + 1;
        WORD BUFFER[output word address]:= byte;
        go to output done;
pack second: WORD BUFFER[output word address]:=
             WORD BUFFER[output word address]×2↑10 + byte;
        go to output done;
pack third:  WORD BUFFER[output word address]:=
             WORD BUFFER[output word address]×2↑10 + byte;
        go to output done;
```

```
pack fourth: WORD BUFFER[output word address]:=
             WORD BUFFER[output word address]×2↑10 + byte;
      output byte number := 0;
      if marks of(WORD BUFFER output word address) > 0 then
           begin number of used tracks:= number of used tracks + 1;
                if number of used tracks > number of available tracks
                    then alarm({<program overflow>, 'stop');
                byte:= WORD BUFFER[output word address];
                output track:= output track + (if output track = lasttrack
                                    then 1 - available tracks
                                    else 1);
                TRANSFER TO DRUM(output track)from:(output word address - 40);
                if marks = 3 then output word address:= output word address - 81;
                go to if this is last then exit from pass else pack first
           end;
output done;
end output;

integer procedure input; comment This performs the drum transfers and
unpacking used for input to all passes except the first which reads the
paper tape. In the machine code this is coded as an open subroutine of
the following 2 long orders:
         ARS (el) t + 1     or     PMX (el) t + 1
         HS   e2  LA               HS   e2  LA;
begin byte address:= byte address + 1;
      R:= BYTE BUFFER[byte address];
      if R = 'nonsensebyte' then  UNPACK BYTES;
      input:= R
end input;

procedure UNPACK BYTES; comment This is called every time an input word
must be unpacked, i.e. once for every 4 bytes input by input;
begin integer marks;
      input word address:= input word address + 1;
      marks:= marks of(WORD BUFFER[input word address]);
      if marks > 0 then
          begin input track:= input track +
                  (if input track = last track then 1-available tracks else 1);
               TRANSFER FROM DRUM(input track)to:(input word address - 40);
               if marks = 3 then input word address := input word address - 81;
               number of used tracks:= number of used tracks - 1
          end;
      word:= WORD BUFFER[input word address];
      R:= BYTE BUFFER[1]:= first part(word);
      BYTE BUFFER[2]:= second part(word);
      BYTE BUFFER[3]:= third part(word);
      BYTE BUFFER[4]:= fourth part(word);
      comment BYTE BUFFER[5] permanently holds the value 'nonsensebyte';
      byte address := 1
end UNPACK BYTES;
```

```
procedure TRANSFER TO DRUM(track number) from:(buffer location); code;
prodedure TRANSFER FROM DRUM(track number) to:(buffer location); code
comment These procedures transfer the 40 words of a track to or from the
40 words held in the WORD BUFFER from WORD BUFFER[buffer location] and
onwards;
```

The following procedures use some further non-local parameters:

```
integer rest of line, pass number, CRcounter, information 1, information 2,
                                    pass number;
boolean first print in pass, no running, this is last;

procedure testprint(n); value n; integer n;
begin rest of line:= rest of line - 1;
      if rest of line = 0 then new line;
      skrv(⟨dddd⟩, n, skrvml(2));
end testprint;

procedure print(n); value n; integer n;
begin rest of line:= 0;
      skrv(⟨dddd⟩, n, skrvml(2));
end print;

procedure new line;
begin skrvvr; skrvvr;
      if first print in pass then
         begin first print in pass:= false;
               skrv(⟨d⟩, pass number);
               skrvtekst(⟨<.⟩);
         end
      else skrvml(3);
      rest of line:= 10
end new line;

procedure message(n, kind); value n, kind; integer n, kind;
begin boolean give up;
      switch action:= hopeless, serious error, error, line number, no line number;
      TRANSFER FROM DRUM('message track') to:(if input word address > 41 then 1
                                                                          else 42);
      new line;
      go to action kind;
hopeless: give up:= true;
serious error: no running:= true;
      red output;
error: printtext(⟨<error⟩);
line number: printtext(⟨<line⟩);
      print(CRcounter);
no line number: printtext(n); comment printtext is a procedure which prints
      that text in the text list which has the number given as parameter;
      TRANSFER FROM DRUM(track in) to:(if input word address > 41 then 1 else 42);
      black output;
wait: if drum transfer in progress then go to wait;
      if give up then stop
end message;
```

se manuskript

```
end pass:
      output(0);
      working boolean:= in testmode;
      in testmode:= false;
L:    output(0); output(0); comment This fills the remainder of the output
      track until it has been transferred to drum. output jumps to 'exit
      from pass' when a drum transfer has been completed and 'this is last'
      is true;
      this is last:= true; go to L;

exit from pass: new line;
      print(used tracks);
      if information 1 ≠ 0 then print (information 1);
      if information 2 ≠ 0 then print (information 2);
      pass number:= pass number + 1;
      in testmode:= working boolean;
      if in testmode then wait;
      first print in pass:= true;
      marks:= marks of(passinformation[pass number]);
      R:= pass information[pass number];
      comment The table 'pass information' tells whether the direction of
      scan should be reversed (marks > 2). Also four addresses are given
      telling where to find the new pass program on the drum and where to
      enter into it;
      if marks > 1 then exchange (input track, output track);
      TRANSFER FROM DRUM(input track) to:(1);

begin integer track, store, first track, exit;
      track:= part 1(R);
      first track:= part 2(R);
      store:= part 3(R);
      exit:= part 4(R);
      TRANSFER FROM DRUM(if marks = 0 ∨ marks = 2 then forward track
                         else backward track) to:(pass mechanism);
more: track:= track - 1; store:= store - 40;
      TRANSFER FROM DRUM(track) to:(store);
      if track > first track then go to more;
L:    if input = 0 then go to L; comment This eliminates the filler zeroes;
      byte address:= byte address - 1; comment In this way the last byte
                will be repeated;
      go to instruction[exit];
end;
```

SYMBOLIC NAMES FOR THE WHOLE TRANSLATOR (e-NAMES).

Entry points and parameters held as addresses in instructions:

e1	byte address
e2	inaddress (input word address and entry to UNPACK BYTES)
e3	output
e4	(see below)
e5	entry to message
e6	testprint
e7	newline and print
e8	newline
e9	print
e10	
e11	
e12	outaddress (output word address)

General parameters for translator (are stored at the end of the track holding print etc. and are initialized to the values indicated when reading that track from drum). e4 must be defined right at beginning of loading.

	holds	initial value
e4	CRcounter	QQ 0 t1
1e4	number of used tracks	QQ 0
2e4	information 1 for output of	QQ 0
3e4	information 2 statistics	QQ 0
4e4	last track - 1	QQ e20 - 1
5e4	available tracks - 1	QQ e20 - e19
6e4	input track	QQ e19
7e4	output track	QQ e19
8e4	no running (>0 = <u>false</u>)	QQ 1
9e4	pass number	QQ 1
10e4	first track + 1	QQ e19 + 1