COLOPHON

The attached paper, although on a Computer Science topic, has no connection with my current research in Digital Documents. It is another in a series of experiments to see how long it takes me to re-build electronic versions of my published early papers as properly re-typeset 'PDF Normal' rather than just as a bitmap scan.

The attached paper appeared in the Proceedings of a Conference called "Applications of Algol 68" which was held at the University of East Anglia in March 1976.

The text of this conference, to the best of my knowledge, is not available online. This paper was acquired by scanning the paper from the original Proceedings and then using Omnipage OCR on the resulting TIFF files. The paper was then re-typeset using UNIX *troff* suite to set up the correct typeface (Courier) and to get the line and page breaks as accurate as possible.

Despite the fact that Courier normally scans in well under OCR the pages in the typewritten Proceedings were of very variable quality. Some pages had faded badly leading to poor recognition accuracy and the need for some re-keying. The time taken to rebuild this paper was about 4 hours.

ALGOL 68 DIALECT CONVERSION—A SYNTACTIC APPROACH

D.F. Brailsford and R.D. Knott
Department of Mathematics
University of Nottingham
Nottingham NG7 2RD

Summary

A syntax directed package for converting Revised Algol 68 programs into Algol 68-R form, (where possible) is being developed at Nottingham. The package makes use of J.M. Foster's Syntax Improving Device (SID) [1]. The experience gained has underlined the value of a syntactic approach to problems of this sort. A far wider range of constructs can be translated than would ever be possible by using *ad hoc* methods. In many respects the difficulties encountered are those of conventional compiler writing, but some intriguing new problems arise when, as in this case, the source language and target language differ relatively little in philosophy and appearance.

1. Introduction

Since the earliest days of high level computer languages a wide variety of dialects has sprung up often due to lack of hardware or software facilities for implementing some of the language features and sometimes, more simply, because the implementers take issue with the language designers over fundamental design decisions. In the main, however, Algol 68-R tries to implement Algol 68 as defined in the original Report, subject to restrictions required for one-pass compilation. Thus, the conversion task is approximately that of converting Revised Algol 68 to original Algol 68.

When dialects of a language differ only at the level of external representation it is possible to do a great deal of conversion work by very straightforward methods [2]. However, the moment one strays into syntactic differences between dialects the need to compile the source text, in some sense, makes itself apparent. It is precisely at this stage that the writer of a conversion program baulks at the task of constructing a goodly part of a new compiler.

Fortunately, due to some joint research work between ourselves and R.R.E. Malvern, we were able to use a syntax analyser produced by SID as the basis for our converter. SID was originally developed by J.M. Foster [1] and it has been much used as a tool for compiler writing at Malvern — including the Coral 66 and Algol 68-R compilers. We have also benefitted directly and indirectly from work done by P.M. Woodward and A. Putley [3] on the SARA system; in particular for the extraction of the lexical analyser from the ALGOL 68-R compiler.

SID itself is written in Algol 68-R and, given a set of syntax rules for a grammar, it will transform them, if possible, into a one-track form. If these transformations are successful. then the output from SID is some ALGOL 68-R text for a fast one-track analyser. The analyser will be capable of parsing all permissible 'sentences' as defined by the original input grammar. One can embed semantic 'actions' into the syntax rules to indicate where the resulting analyser should call these 'actions' when parsing a given string. A concise description of how SID works, and the way that compiling actions can be interspersed with

the syntax is given in reference [3].

2. Syntax for Revised Algol 68

An excellent starting point for,a syntax is the chart of Watt, Peck and Sintzoff [4]. Ultimately this has to be expressed in Backus Normal Form (BNF) in order to be acceptable to SID, but at this point we benefitted from R.D, Knott's special pre-processor for SID, called SIDTAX [5], which, among other, things, enables rules to be expressed in something like the form defined in the syntax chart. These forms are then converted, automatically, into BNF by SIDTAX.

For example, in SIDTAX notation, a Revised Algol 68 loop clause becomes

(FOR ID, **Φ**)(FROM UNIT, **Φ**)(BY UNIT, **Φ**)(TO UNIT, **Φ**)
(WHILE ENQUIRYCLAUSE, **Φ**) DO SERIALCLAUSE OD

Alternatives for any part of a construct are expressed as a list separated by commas and enclosed within round brackets. **Φ** denotes the empty alternative.

Unfortunately, the final syntax deviates from the elegant starting point, mainly due to some re-writing so that SID can get the syntax into a one-pass form. This problem becomes especially acute when the compiling actions are interspersed with the syntax.

In principle, the type of grammars amenable to SID treatment require only a knowledge of the next symbol in the source text in order to decide, unambiguously, which route to follow through the syntax. The initiated will be aware that Algol 68 does not satisfy this criterion and, for example, the symbol "(" could be an abbreviated form of a

begin, **case** or **if** symbol. We may wish to take different actions depending on which construction is intended. In such cases one has to cheat a little with SID and arrange for a look-ahead along the source string. The amount of look-ahead that is necessary will be discussed in a later section.

It will not be possible to describe all the conversions that have to take place but the next sections illustrate the more interesting points.

3. Problems encountered.

3.1. Representations and notational changes.

Problems of this sort are the easiest to deal with and call be handled, in a variety of ways.
For example, the lexical analyser can be modified to accept a variety of different representations, or, in some cases an appropriate library operator can be written. In either of these ways one could, for example, make Algol 68 accept the symbol % for integer division as well as '/'.
There are also one or two straightforward notational differences between Algol 68-R and Revised Algol 68 which are easily dealt with, such as the change from **elsf** to **elif**, from **charput** to **file**, from **plus** to **plusab** and so on.
Features of the standard prelude not available in Algol 68-R can also, to some extent, be provided by library procedures and operators.

3.2. Syntactic differences between dialects.

It is in this area that *ad hoc* methods run into great difficulties, due to a lack of any syntactic appreciation of what is going on. The problems vary in difficulty and

perhaps the simplest change to effect is to convert the Revised Report's **do** into **do begin**, and **od** into **end** thereby converting

do serial clause **od**
into **do begin** serial clause **end**

which is sufficient to satisfy Algol 68-R's syntactic requirement for **do** unitary clause. In a similar way one can enclose the unitary clauses of Revised Algol 68 procedure-bodies within round brackets or **begin end**, to make them acceptable to Algol 68-R, but here, of course, we have to identify where the unit ends in order to place the closing bracket correctly.

Perhaps the next most difficult of these straightforward problems is to convert casts i.e.

fdec ENCLOSED-clause

(which is a primary in Revised Algol 68) into

(fdec **val** unitary clause)

in Algol 68-R. Here fdec is an abbreviation for formal mode declarer and the extra round brackets are necessary in the Algol 68-R version so that the whole construction becomes a primary once again. In this conversion we have had to perform a short look ahead in the Revised Algol 68 source text in order to direct the analysis along the right lines. The problem is that a fdec can signal the start of an identity declaration, a denotation for a parameterless procedure, or a cast. The lack of an identifier or a colon immediately following the fdec rules out the first two possibilities and one can then proceed to convert the cast.

The symbols **loc** and **heap** in generators can be handled

fairly easily because one only has to omit **heap** to obtain the required effect in Algol 68-R. But in variable declarations the **loc** prefix, which is optional in Revised Algol 68, must not be given in Algol 68-R.

Thus **loc int** i := 2 converts to **int** i := 2. Algol 68-R, additionally, does not permit the **heap** form of variable declaration so **heap int** i := 2 has to be expanded to **ref int** i = **int** := 2.

Finally let as consider two problems of bracket matching. Algol 68-R does not have the **ouse** elision but it is a reasonably straightforward problem to replace it by **out case** and to position the closing **esac** correctly. In similar vein, the easiest way to get round the Algol 68-R restriction that serial clauses containing declarations must begin with a declaration is to note, on entry to every serial clause, whether or not it starts with a declaration. If not, then an extra set of **begin** and **end** brackets will have to be correctly positioned if a subsequent declaration is encountered. Hence the Revised Algol 68

```
begin print ("start"); int i :=2; print ("finish") end
```

converts to the Algol 68-R form

```
begin print ("start"); begin int i := 2; print ("finish")
                       end end
```

This technique fails to give the right effect only when the program uses **goto** to skip over declarations. Such cases are either abominable programming, for which one has no sympathy, or else are actually forbidden by the Revised Report.

4. Example

The following example illustrates the conversion of some of the features mentioned in the previous section. Apart from this it is neither useful nor meaningful. The Revised Algol 68 source is given first, followed by the Algol 68-R version produced by the converter.

Revised Algol 68

```
proc example = (ref ref int i, int i) void:
      case i plusab 1
           in for k to i do print ("hooray") od,
         ref int (i) := heap int := 2
ouse j in
           print ("yes"),
           out print ("no")
esac
```

Algol 68-R

```
 proc example = (ref ref int i, int j) void:
      (case i plus 1
         in for k to to i do (print ("hooray")),
           (ref int val (i)) := int := 2;
      out case j in
                print ("yes"),
                skip
      out print ("no")
      esac esac)
```

5. Is mode analysis strictly necessary?

From a purely syntactic point of the answer to this question is "no", for Revised Algol 68 has been designed for

mode-independent parsing. But, of course, a knowledge of modes can be invaluable in minimising the amount of look-ahead that is necessary in a given situation. A good example is

```
(b| unit, unit|unit)
```

where mode analysis of b immediately preceding the stick symbol will indicate at once whether this is a conditional clause or a **case** clause. In the absence of this information one has to look much further ahead, to detect two units in the **in** part in order to reach the same conclusion.

Mode analysis, and indeed knowledge of position strengths, is necessary for actually performing the conversion since modes are sometimes treated differently in the two languages. For example, string denotations of 4, 8 or 12 character in Algol 68-R are compiled as **bytes**, **long bytes** and **long long bytes** respectively if the context is not strong. This situation has now been immortalised by one of our colleagues in the phrase "dogged by **bytes**", For example

```
if "fish" [3] = "s" …
```

which happily yields **true** in Revised Algol 68 has to be converted to

```
if ([] char val "fish")[3] = "s" …
```

in Algol 68-R.

6. Difficult points.

There are many places where a meaningful conversion from Revised Algol 68 to Algol 68-R is difficult, if not impossible, to achieve. An important subset of these is caused by the scoping differences between the two languages so that

```
int i := 2;
if int i := 6; true then print (i) fi
```

will print 6 in Revised Algol 68 and 2 in Algol 68-R.

A solution to this problem for conditional clauses is to embed the whole construction inside a closed clause in the Algol 68-R translation, together with a re-positioning of the **if**. For example we have in Revised Algol 68:

```
if preamble; condition
    then action 1
    else action 2
fi
```

and in Algol 68-R:

```
begin
preamble ;
if condition
    then action 1
    else action 2
fi
end
```

A similar device will also cope with translation of **case…esac** clauses, but **do** do clauses have to be translated as follows:-

Revised Algol 68:

```
while preamble; cond do thing od
```

Algol 68-R

```
while preamble ;
if cond
    then thing; true
     else false
```

fi

do skip

Even with the above tricks, problems still remain. If, for example, the Algol 68-R translation of the conditional were to be the source in an assignment, then one has to remember the Algol 68-R constraint on values delivered from serial clauses which constitute declarations [6].

Clearly, a point is reached where translation is going to involve much re-writing of the source program, and it is questionable just how far one should go in this direction. For these reasons we insist, at present, that the source text conforms to the following requirements (among others)

- Defining occurrences of indicators fist precede their applied occurrence in the text.
- Mutually recursive procedures and operators are not allowed.
- Field selection from a row of structures to yield an array is not allowed.

If any of these forbidden features is encountered then translation is not attempted, but instead a warning can usually be printed.

7. Conclusions.

Our experience with the converter has shown that we can successfully convert a much wider range of constructs than would ever be possible by non-syntactic methods but there still remain some difficult problems, exemplified in the previous section. Apart from these points it is possible to achieve a conversion in more than 1 but less than 2

passes, the fractional part representing an allowance for the look-aheads that have to take place.

Interestingly, many of the difficulties experienced occur precisely because the two languages are so similar, in form and intent, that the bulk of the text passes over with no change whatsoever. The corollary to this is that when a given construct has an identical appearance in both languages, but a different meaning, then large scale re-writing is often necessary. Perhaps this should not be too surprising because, as with most quirks of artificial languages, we can find parallel situations in natural languages. In our case a comparison of English and American would be a good analogy for Revised Algol 68 and Algol 68-R (not necessarily in that order!). We all know the difficulties that arise, due to the same words having different meanings, so that, in an attempt not to confuse, an Englishman might have to say "I put on my vest and pants this morning and so far as I am concerned these are both undergarments". Or consider the verb "to service" which has connotations in America that are undreamed of in England. Indeed, a British Computer manufacturer is reported to have reduced American computer professionals to fits of helpless laughter by taking a full-page back cover advertisement on the rear of an American trade journal proudly proclaiming "We'll service you like you've never been serviced before".

Acknowledgements.

We thank the S.R.C. and R.R.E. Malvern for financial support for R.D.K., and the members of the Computing and Software staff at R.R.E. Malvern for much help and encouragement

References.

[1] J.M. Foster, Computer Journal, Vol. 11, p.31, May 1968.

[2] A. Maybrey, Proceedings of Liverpool Conference on Uses of Algol 68, March, 1975.

[3] P.M. Woodward, SARA—RRE Internal Document.

[4] J.M. Watt, J.E.L. Peck and M. Sintzoff, Algol Bulletin No. 17, June, 1974.

[5] R.D. Knott, M.Sc. Dissertation, University of Nottingham, 1975.

[6] "Differences between Algol 68-R and 68" RRE Internal Document (2nd. Edn. June, 1973).