On the generation of AIGOL 68 programs involving infinite modes

L. Meertens

0. Introduction

Certain proper AIGOL 68 (particular-)programs, e.g.,

begin struct chain = (ref chain link); skip end;

are only be generated according to the rules given in the Report on the Algorithmic Language
AIGOL 68 (1), by producing, in an infinite number of steps, a mode of
infinite length. It has raised objections that this generation process is
not finite and, therefore, not constructive. Moreover, G.S. Tseytin has
shown (2) that the definitions in the AIGOL 68 report do not preclude an
interpretation of equality between infinite modes in which, e.g., the
modes specified by the mode-notations a and b defined by the declaration

mode a = proc (e, e) a, b = proc (e, e) a a are equal, although these are
clearly intended to be different.

The purpose of this note is to sketch a process that allows the
generation of such programs in a finite, constructive way, and yet without
need to change the syntax and the metaproduction rules in the AIGOL 68
report (with one annoying exception).

1. The stages of the generation process

The generation process is described in three stages (1.1 up to 1.3),
each stage yielding the material to be used in the next stage. This does
not imply that it is necessary to complete the first stage first, and next
the second stage, and so on; on the contrary: whenever the process cannot
be continued due to shortage of material, the current stage may be
interrupted in order to generate new material; it is even possible to
integrate the first two stages in the last stage, but this necessitates
quite some administration circumvented in the approach described here.

Before we start one change in the metaproduction rules of (1) has to
be made: rule 1.2.5.f is replaced by

NOTION: ALPHA; MODE; NOTION ALPHA; NOTION MODE.

It is a nuisance that this change introduces unnecessary ambiguities in
the process of generating the program (but not on the semantic level).
These ambiguities can be circumvented, but only in a cumbersome way;
we would have to write out:

NOTION: library prelude; library postlude; declaration prelude;
label; label sequence; etc.

1.1. Generation of "specific" metanotions and their
specific production rules

We proceed from the set of production rules of the metalanguage,
obtained in 1.1 of (1). (Actually we need only a finite subset).

A specific metanotion is a metanotion followed by the decimal notation
of a natural number; e.g., NODE17. Associated with a specific metanotion
is its specific production rule, obtainable from a production rule for
that metanotion by inserting after that metanotion, as it appears before
the colon, the number of that specific metanotion and some, arbitrarily
chosen, natural number after each metanotion appearing in the direct
production (the part after the colon). E.g., the specific production rule
of MODE17 might be MODE17: MOOD3.
and that of FIELD1:
FIELD1: MODE18 field TAG23.
In this example, MOOD3 is the direct production of MODE17. The set of specific metaproduction rules cannot, of course, contain both LOWER1: lower.
and LOWER1: upper., as only one of these can be the specific production rule for LOWER1. For any program only a finite number of specific metaproduction rules has to be generated.

1.2. Generation of "normal" production rules of the strict language

We proceed from the set of "unfinished" production rules of the strict language, as obtainable from 1.1.5 of (1) when 1.1.5.a.Step3 (i.e., replacing a metanotion by one of its terminal productions) is skipped. These unfinished production rules are turned into normal production rules (of the strict language) by inserting after each metanotion appearing in them a natural number, with the understanding that after all occurrences of a given metanotion in some rule the same natural number is inserted. So actual LOWER1 bound: strict LOWER1 bound. and actual LOWER2 bound: strict LOWER2 bound. both are normal production rules, but actual LOWER1 bound: strict LOWER2 bound. is not a normal production rule of the strict language. In contrast with the set of specific metaproduction rules, the set of normal production rules may contain both NOTION4096 option: NOTION4096. and NOTION4096 option: EMPTY1.

From a given normal production rule another one may be obtained by replacing one of the specific metanotions appearing in it by the direct production of that specific metanotion. If, in stage 1, we have generated the specific metaproduction rule LOWER1: lower., then we can obtain from the normal production rule two new ones, viz.
actual lower bound: strict LOWER1 bound. and actual LOWER1 bound: strict lower bound.
From each of these rules we may obtain yet another one:
actual lower bound: strict lower bound.
For any program only a finite number of normal production rules has to be generated.

1.3. Producing the program

Before each normal production rule obtained in stage 2, < is placed.
Each colon is replaced by ::= <, each comma by > < and each point by >. Next, one rule <empty>::= is added, and all occurrences of <> are replaced by <empty>. We thereby have obtained a Backus Normal Form grammar that produces, starting from the metalinguistic variable <program>, our program (where every symbol is still a metalinguistic variable, e.g., <begin symbol>). This syntax may, of course, be abbreviated by the convention of using | for alternative direct productions.
2. The context conditions

The generation of the program is only half the story; if the program is to be a proper one it has to satisfy the context conditions. The formulation of these conditions as given in (1) is not without more applicable to the treatment used here; the changes are, however, rather obvious. Instead of comparing 'reference to structured with reference ... (etc. ad infinitum) mode identifier' with some other notion, we may find ourselves in the position where we wonder whether <MODE17 mode identifier> happens to be the "same" as <structured with reference to MODE17 field TAG23 mode identifier>. This can be decided upon on the basis of the specific metaproduction rules obtained in stage 1, by means of an algorithm as has been given in (3), section 2.3.5, exercise 11.

3. Example

In the example given in the introduction, the crucial spot is the mode-declaration. The most important specific metaproduction rules and normal production rule (as modified in 1.3) directly related to that mode-declaration might be:

MODE17: MOCD17.
MOCD17: STOWED1.
STOWED1: structured with FIELDS1.
FIELDS1: FIELDI.
FIELD1: MODE18 field TAG23.
MODE18: MOCD18.
MOCD18: TYPE1.
TYPE1: reference to MODE17.

<mode declaration> ::= <mode symbol> <MODE17 mode indication>
<equals symbol>
<actual structured with reference to MODE17 field TAG23 declarer>

In this example, 'MODE17' and 'structured with reference to MODE17 field TAG23' stand for the same mode.

References:

