A note on Backus Naur form

By J. S. Rohl*

Backus Naur Form cannot describe itself. This note describes a modification which gives it this ability.

Backus Naur Form (BNF) has gained acceptance as a language for formally describing the syntax of programming languages, even to the extent of being used as input for certain compilers. Pure BNF (Naur, 1963) has generally been used in bottom-to-top algorithms. Top-to-bottom algorithms, on the other hand, require a reformulation of the metalinguistic formulae for their operation. Let us consider the syntax of the ALGOL ⟨for statement⟩. (Section 4.6.1 of the ALGOL Report).

⟨for list element⟩ ::= ⟨arithmetic expression⟩ |
⟨arithmetic expression⟩
step ⟨arithmetic expression⟩ **until** ⟨arithmetic expression⟩
| ⟨arithmetic expression⟩ **while** ⟨Boolean expression⟩
⟨for list⟩ ::= ⟨for list element⟩ | ⟨for list⟩ , ⟨for list element⟩
⟨for clause⟩ ::= **for** ⟨variable⟩ := ⟨for list⟩ **do**
⟨for statement⟩ ::= ⟨for clause⟩ ⟨statement⟩ | ⟨label⟩ : ⟨for statement⟩

For top-to-bottom recognition it is convenient to alter the definitions so that

(i) premature recognition is prevented,
and (ii) infinite loops are avoided.

In the definition of ⟨for list⟩ the alternatives need to be reversed so that the algorithm does not stop looking after one ⟨for list element⟩ is recognized, and the components of the (original) second alternative need to be reversed to stop the algorithm looking for a ⟨for list⟩ by looking for a ⟨for list⟩. The above definitions can be rearranged in this reversed BNF as below:

⟨for list element⟩ ::= ⟨arithmetic expression⟩
step ⟨arithmetic expression⟩
until ⟨arithmetic expression⟩|⟨arithmetic expression⟩
while ⟨arithmetic expression⟩|⟨arithmetic expression⟩
⟨for list⟩ ::= ⟨for list element⟩ , ⟨for list⟩
| ⟨for list element⟩
⟨for clause⟩ ::= **for** ⟨variable⟩ := ⟨for list⟩ **do**
⟨for statement⟩ ::= ⟨for clause⟩ ⟨statement⟩
| ⟨label⟩ : ⟨for statement⟩

This is the form that is used in the Compiler Compiler (Brooker *et al.*, 1963).

The above considerations aside, BNF suffers from the disadvantage that it cannot specify itself. This is because the metalinguistic formulae involve special metalinguistic symbols such as ::=, |, ⟨and⟩. This is not merely an inelegance. As we have indicated earlier, BNF is being accepted as an input by some compilers and there is a strong case for embedding the special purpose facilities in a common language rather than purpose facilities in the common language rather than producing special purpose languages. Thus we can expect that it might be useful to allow some facilities of BNF to be added to ALGOL. (They have already been added to Atlas Autocode (Brooker, Morris and Rohl, 1967)). As it stands ALGOL with BNF facilities would be capable of describing a host of languages including normal ALGOL but excluding itself.

There are at least two solutions. One, adopted in the Compiler Compiler, is to create special pseudo-metalinguistic variables such as ⟨meta stroke⟩, ⟨meta left bracket⟩ etc. which implicitly describe the special symbols. The appeal of this solution decreases as one tries to expand BNF by use of further symbols as has been done by, for example, Burkhardt (1965).

The second solution is to use brackets to surround not the metalinguistic variables but the basic symbols. (It is interesting to note that the English language uses both solutions: specific sentences, words and even letters are enclosed in quotes, but punctuation marks are generally described by their names, colon, comma etc.). In this case we need a symbol to indicate concatenation, and we have chosen the point. Thus the syntax of the for statements can be restated, using the reversed form:

for list element ::= arithmetic expression . ⟨**step**⟩ .
arithmetic expression.
⟨**until**⟩ . arithmetic expression | arithmetic expression.
⟨**while**⟩ . Boolean expression | arithmetic expression
for list ::= for list element.⟨,⟩. for list | for list element
for clause ::= ⟨**for**⟩.variable.⟨:=⟩.for list.⟨**do**⟩
for statement ::= for clause.statement | label.⟨:⟩.for statement

We have assumed in the above that ALGOL delimiter words are regarded as one symbol. This is convenient but not necessary since, for example, we could regard ⟨**for**⟩ as being the concatenation ⟨**f**⟩ . ⟨**o**⟩ . ⟨**r**⟩.*

* Alternatively we could allow the brackets to surround strings rather than symbols. This solution, however, implies the use of some arbitrary convention for dealing with strings containing a right bracket.

* *Department of Computer Science, The University, Manchester* 13.

The meta variables are more restricted than those of the original BNF since they may not contain the meta symbols. This is of minor importance since it does not restrict the language being described. In any case most users of BNF restrict themselves to purely alphabetic meta variables, as we will here.

We can now define the metalinguistic formulae themselves:

```
meta formula  ::= meta variable . ⟨::=⟩ . definiens
definiens     ::= construction . ⟨|⟩ . definiens
                | construction
construction  ::= component . ⟨ . ⟩ . construction
                | component
component     ::= meta variable | ⟨⟨⟩ . symbol . ⟨⟩⟩
```

The brackets are effectively a device for over-riding any metalinguistic significance of the symbol they enclose. Thus, there is no need to use the complicated metalinguistic symbols of BNF which were chosen because they were unlikely to occur in programming languages. We might, for example, replace the metalinguistic equals by an ordinary equals, the vertical bar by a comma, and the brackets by quotes, as in Atlas Autocode. Thus:

```
meta formula = meta variable.'='.definiens
definiens    = construction.'|'.definiens , construction
construction = component.'.'.construction , component
component    = meta variable , '''.symbol.'''
```

These modifications to BNF allow us also to describe other metalinguistic languages such as, for example, Burkhardt's. One convention is needed because Burkhardt uses bold-face type for basic symbols in his definitions. We will use quotes here, though this complicates the description.

```
meta formula = integer.meta variable.definiens
definiens    = construction.'|'.definiens , construction
construction = quantified component.construction ,
    quantified component
quantified
component    = component , '$'.'''.integer.'''.
    component ,
  '↑'.'''.integer.'''.component , 'Δ'.component
component    = integer , '''.symbol.''' , '*' ,
    '{'.definiens.'}', 'el' , '#'.'''.integer.'''
```

We could, of course, expand this modified BNF to include Burkhardt's or any other facilities, but will not do so, since it is beyond the scope of this note.

References

BROOKER, R. A., MACCALLUM, I. R., MORRIS, D., and ROHL, J. S. (1963). The Compiler Compiler, *Annual Review in Automatic Programming*, Vol. 3, London: Pergamon.

BROOKER, R. A., MORRIS, D., and ROHL, J. S. (1967). Compiler Compiler facilities in Atlas Autocode, *Computer Journal*, Vol. 9, p. 350.

BURKHARDT, W. H. (1965). Meta-Language and Syntax Specification' *Comm. ACM*, Vol. 8, p. 304.

NAUR, P. (Ed.) (1963). Revised Report on the Algorithmic Language ALGOL 60, *Computer Journal*, Vol. 5, p. 349.

Samuel N. Alexander

Samuel N. Alexander, who died of cancer at the age of 57 at his home in Chevy Chase, near Washington, on 4 December 1967, was one of the true pioneers of the computer field. In 1946, he became head of the newly formed Electronic Computer Laboratory of the National Bureau of Standards and, as such, was responsible for advising the U.S. Government on the steps it should take toward the acquisition of computers; in order to obtain the background experience necessary, it was decided to construct a computer for the Bureau. To Alexander must go the credit for the fact that this machine—known as the SEAC—was the first of the American stored-program computers to be successfully completed. It was also the best engineered of all the very early machines. It served as a test bed for the Williams tube memory and various peripheral devices, besides doing much useful computing for the U.S. Atomic Energy Commission and other U.S. agencies.

Sam was a good friend to me in the early years. I always came away from a talk with him feeling stimulated and instructed. He bubbled over with talk, but in his judgement he was calm and cautious. I shall always be grateful that I was able to meet Sam again as recently as last August when we were both members of a panel of computer pioneers at the 20th Anniversary ACM meeting in Washington. He had been let out of hospital for the day, and was looking terribly ill and emaciated, but in every other way he was still the old Sam we had all known. Many will remember his remarks on that occasion, with his generous references to the work of other groups.

Sam received a number of awards for his contributions to the computer field, including two gold medals, and a medal of the Swedish Royal Academy of Engineering Sciences. The latest was the Henry Goode Memorial Award, which was presented to him in Washington just before the panel discussion at the ACM meeting. He leaves a widow, a son, and two daughters.

M. V. Wilkes