Backus Naur form and its influence on programming languages

Viktor Gsteiger
University of Basel
Matriculation Number: 18-054-700

October 10, 2020
Seminar: 58826-01 - Turing Award Winners and Their Contributions

Abstract

The contribution of Peter Naur to the programming language Algol 60 and his preliminary work in the field of programming language description was a milestone of great importance to the field of computer science and the field of programming language design in particular. The simple but powerful Backus Naur form, an extension of Naur to the already existing Backus form, has shown to be the new standard in language description from its use in the Algol 60 report on. The design and logic behind Algol 60 has proven to be the foundation on which a large subset of all programming languages still in use today have been built upon. Naur's thinking ahead and in larger meta questions rather than small technicalities has proven to be fundamental for the strength of Algol based languages. Since the importance of Naur's work, it is a worthwhile undertaking to restate the most important parts of Naur's contribution to Algol as well his influence of the programming languages of the future. This report is aimed at bachelor level students of Computer Science, however, we hope to appeal to a wider audience as well.

1 Introduction

The content of this paper will present a fairly detailed review of the development of formal notations of languages. The main focus of the first half of the report will lie in the development leading up to the notation used in the Algol 60 report and the notation used in the report itself. The subsequent parts will follow Naur's personal development and his further contributions to the topic of formalization. We will wrap up this report with the impact of Naur and the impact of Algol in particular.

2 History of Formal Descriptions for Languages

The goal of defining a language by a formal system of variables and rules to derive the language as been of interest to grammarians and mathematicians since the verge of civilization. Panini, a Sanskrit philologist and grammarian which has worked between the 6th and 4th century BCE had, according to the biography *Panini* by Bhate (4), already described a notation structure for the language of Sanskrit. This is insofar interesting as it shows that the problem of a formal description of a language has existed even long before the topic of computer languages ever was one. The interest in a formal notation of a language in the west was not of general interest until the 20th century. With the advance of an interest in new, artificial languages like Esperanto or Ido and especially the rapid advances in calculating machines and first version of computers. The advance of new technologies posed new challenges in human-machine as well as human-human interactions which first had to be resolved before the underlying technological advances could be made usable for a larger audience than research scientists.

2.1 Noam Chomsky

Noam Chomsky, an American linguist teaching at MIT combined the new interest from mathematicians and linguists in his teaching groups. Chomsky gave lecture to information technology students and combined past research in the topic of string rewriting rules and formalism to develop his own theory of formal logical systems. His main idea was to define the grammar structure of English as a finite state Markov process. To achieve this idea, Chomsky (6) introduced in his paper *Three Models for Description of Language* a notation for the rewriting of strings according to a grammar. A grammar can be viewed as a machine operating on strings and giving certain output to an input. Chomsky also defined several different distinct grammars which are used in theoretical computer science until today.

The importance of Chomsky's work can not be understated, even though he was not the original inventor of the idea. He built upon work from other mathematicians as well as computer scientists. Most notably *Axel Thue* as well as *Alan Turing*. In perspective of this paper, it is of great importance to understand the concept of phrase-structure grammars, also called context-free grammars.

Chomsky defined a language L as a (finite or infinite) set of sentences of finite length. Furthermore, a string is defined as a concatenation of symbols of an Alphabet

A. The Alphabet consists of a finite set of terminal symbols. A grammar is then defined as a device that produces all of the strings that are sentences of L. A properly formulated grammar should define unambiguously the set of grammatical sentences of the language it represents. This means that a sentence which is clearly part of the language should be handled by the grammar in a fixed and predetermined manner.

A phrase-structure grammar consists of a finite vocabulary V_p, a finite set of initial strings$E$ in V_p and a finite set of rules F of the form $X \rightarrow Y$, where X and Y are strings in V_p. The rules are interpreted as an instruction to the grammar device where X has to be rewritten by Y . So any Y_i is formed from a X_i by replacement of a single symbol of X_i by some string. A derivable string from this language is a string which can be derived by the rules F. A terminal string from this language is a string which is the last line of a derivation where there is no further string to be derived.

An example from Chomsky for a phrase-structure grammar would be the following. _ is introduced for concatenation and can be replaced by an empty space and # denominate a meta-linguistic symbol to determine the boundaries of the given string:

$$\begin{aligned} \sum &: \#Sentence\# \\ F &: \\ Sentence &\rightarrow NP\_VP \\ VP &\rightarrow Verb\_NP \\ NP &\rightarrow the\_man, the\_book \\ Verb &\rightarrow took \end{aligned} \tag{1}$$

A derivable and terminal string from this grammar would be "the man took the book".

Interesting in the scope of this report is the elaboration of Chomsky that that a language consisting of basic sentences that are derived from terminal strings of a phrase-structure grammar and a set of optional transformation rules applied on these terminal strings may create a language similar to the modern English. The distinction of generative rules of a grammar and transformation rules, which are applied after generating the language may remind some of the way functions and similar constructs in programming languages work today. The insight of Chomsky provided further researchers as well as people interested in the structure of formal notations an entry point.

2.2 Backus

John Backus, an American Computer Scientist who received the ACM Turing award in 1977 for his contribution to the design of practical high-level programming systems and for publication of formal procedures for the specification of programming languages. The latter part is interesting in the scope of this report as Backus built upon the idea of Chomsky and Backus was also one of the more influential computer scientists working on the then newly established programming language international algebraic language (IAL). Backus (1) authored the report *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference* which

was the defining paper of the programming language now known as Algol 58 and a direct precursor to Algol 60 to which Peter Naur contributed substantially.

Backus, with the knowledge of the power of phrase-structure grammars, introduced a meta-linguistic formulae to formally describe the rules of the IAL programming language. The formulae could describe the phrase-structure grammars and was a more approachable notation to the idea behind these grammars. The meta-linguistic formulae introduced by Backus was a reference language to describe the functions of the language on a meta level to allow for later hardware implementation to only differ with regards to word length, overflow conditions and the like. The idea was to allow the different stake holders in the development of a programming language to communicate with a common, formal notation. The main goal of the reference language introduced by Backus was to be as precise as possible without seeming out of touch. It may also be noted that Backus only starts using his new notation on page five out of seven. He first describes the language in mostly informal ways before he proceeds to describe it in the new notation. This detail will remain relevant later in the paper as Naur has extensively written on the topic of informal and formal notation.

The syntax of the reference language introduced by Backus was the following:

- Sequences of character symbols enclosed in "<>" are meta-linguistic variables whose values are strings of symbols,
- The marks ":≡" and "or" are meta-linguistic connectives,
- Any mark that is neither a variable nor a connective denotes itself,
- The combination of marks and/or variables signifies the combination of the strings denoted

This is best illustrated by an example. The _ is introduced for the combination of two marks and or variables and will in the derivation ultimately be replaced by an empty space:

$$\begin{aligned} < sentence > &:\equiv < NP > \_ < VP > \\ < VP > &:\equiv < Verb > \_ < NP > \\ < NP > &:\equiv \text{the man\_}or\text{\_the book} \\ < Verb > &:\equiv \text{took} \end{aligned} \tag{2}$$

As one may observe, it is the same example used in the previous section and should show the power of the notation. Furthermore, as previously, one possible derivation would be "the man took the book". One notable difference to the notation used by Chomsky is that Backus introduced the mark or, however, every string in the Backus notation consisting an or could easily be rewritten into one or more strings without using the connective. It is therefore relevant to note that the notation introduced by Backus is exactly as strong as the phrase-structure grammars. The notation used by Backus was too clumsy and did not catch on to the wider field of computer science and would have almost been forgotten, even though the need for an easily understandable and strong notation was greater than ever.

3 Main part

Was sich überhaupt sagen lässt, lässt
sich klar sagen; und wovon man nicht
reden kann, darüber muss man schweigen.

(Ludwig Wittgenstein)

3.1 The biography of Peter Naur

Peter Naur, born in 1928 to a family of artists and business minded parents, had the early interest involving astronomy. He was allowed to work at the local observatory even as a very young boy. Naur published his first scientific paper with 15 and he had attained many technical skills of mathematics early on from professional astronomers which took him on as a young prodigy. After finishing his astronomy degree in Copenhagen he was recommended to conduct research at King's college, Cambridge where he focused of astronomy and the emerging field of computer science. Due to weather constraints Naur had to divert his time from astronomy and had more time to spend programming the Electronic Delay Storage Automatic Calculator (EDSAC). Peter Naur, used to do complicated computational calculations by hand focused his energy mostly on the limitations of the EDSAC such as the limited number range. After leaving Cambridge, Naur conducted research at Harvard University and Princeton, where he learned the state of the art in computing. The now more established field of computer science would be the focus of Naur, who returned to Denmark in 1953.

He joined the computer center of Copenhagen and was asked to participate in the development of an algorithmic programming language, later called ALGOL. Naur's contribution to ALGOL lay in selecting the right forms of description to define the language. This was in line with his later research as he was more interested in the meta aspects of the language rather than specific implementation. This was also in line with his previous scientific work as he knew how to do the calculations himself he was rather interested in the larger questions regarding computing and also concerning the applicability of the developed programming languages. Naur's main concern was the often overly formal reports led to a split between the industry and academia as the two interacted ever less due to different ideas on formalities and the proofs of algorithms. It was an important question to him to make the field of computer science comprehensible to a wider audience.

Naur also contributed to the establishment of computer science as an academic field in Denmark and he had continued to advocate for an applied form of computer science not only for his students but also for the field in general hereby opposing Dijkstra and Wirth structured programming agenda.

3.2 Backus Naur Form

The Backus Naur form was introduced and used by Backus et al. (2) in the *Report on the algorithmic language ALGOL 60* published in 1960. It was introduced as the reference language of the report. A reference language as defined by the report is the

defining language that is the basic reference and guide for compiler builders, for all hardware representations, for transliterating from the language used in the report to any appropriate hardware representation and should be used by the main publications concerning the ALGOL language itself. Furthermore, the reference language should allow for all characters to be determined by ease of mutual understanding and should not be defined by any hardware limitations, code notation, or pure mathematical notation. The reference language therefore should be understandable to both the wider public as well as special interest groups as compiler builder or hardware builder.
As the name Backus Naur form (BNF) already suggests was the form an extension of the previously introduced Backus. Naur introduced the notation as Backus Normal form, however it has since been renamed. Especially after Knuth (7) pointed out in his letter to the editor *Backus Normal Form vs. Backus Naur Form* that the name normal form should be reserved for true normal forms, which refer to some sort of special representation that are not necessarily a canonical form. According to Knuth, who in the end coined the name Backus Naur Form, the Backus Naur form did have a reason for distinction from the Chomsky phrase-structure grammar as the syntax from Chomsky represent a production while Backus Naur form has a quite different form.

The syntax of the Backus Naur form is described as follows and it is always also noted how it differs from the production form of Chomsky:

- Sequences of character symbols enclosed in "<>" are meta-linguistic variables whose values are sequences of symbols, which is a difference from production form as there is a clearer distinction between terminal and non-terminal symbols.
- The marks "::=" and "|", the latter with the meaning or, are meta-linguistic connectives. The connective "|" groups together all alternatives of a production system and is introduced to make the notation shorter, while the connective "::=" separates left from right.
- Any mark that is neither a variable nor a connective denotes itself.
- The combination of marks and/or variables signifies the combination of the strings denoted.
- Full names indicating the meaning of the strings being defined are used for non-terminal symbols.

An example of the notation being used in the Algol report is how an integer is built up in Algol:

$$\begin{aligned} &< \text{digit} > ::= 0|1|2|3|4|5|6|7|8|9 \\ &< \text{unsigned integer} > ::= < \text{digit} > \mid < \text{unsigned integer} >< \text{digit} > \\ &< \text{integer} > ::= < \text{unsigned integer} > \mid + < \text{unsigned integer} > \mid \\ &\qquad - < \text{unsigned integer} > \end{aligned} \tag{3}$$

Particularly the last part was introduced by Naur to clarify the notation and make it more accessible. Generally, the Backus Naur notation was introduced to give the

report a formal reference language. It has since been adapted and changed for multiple uses and an extended version of the Backus Naur form remains used by most programming languages nowadays. The notation proved to be essential for further language and compiler design. The clear vision and simplicity has remained and has allowed further generations of computer scientists to work on the field of language design.
An interesting quality of the Backus Naur form is also that with just some slight modification it is possible for the form to describe itself. Rohl (10) has shown that in his *A note on Backus Naur form*. The goal of this was to allow compilers to accept the Backus Naur form as input for their design. The approach taken by Rohl was for the brackets to surround not the meta-linguistic variables but the basic symbols. This allows the Backus Naur form to describe the meta-linguistic formulae themselves. The brackets are then a device for over-riding any meta-linguistic significance of the symbol they enclose. Rohl also introduced the . to signify concatenation to allow for better readability. The definition of the meta-linguistic formulae goes as follows:

$$\begin{aligned} \text{meta formula} &::= \text{meta variable. } <::=> \text{ .definiens} \\ \text{definiens} &::= \text{construction. } < \mid > \text{ .definiens} \mid \text{construction} \\ \text{construction} &::= \text{component. } < . > \text{ .construction} \mid \text{component} \\ \text{component} &::= \text{meta variable} \mid <<> \text{ .symbol. } <>> \end{aligned} \tag{4}$$

This definition now allows for a more automatic compiler generation as proposed by Brooker et al. (5) in the paper *The compiler compiler* where they proposed a compiler that was itself able to generate compilers for a certain hardware architecture, then the Atlas computers, with the language specification of the compilers being a phrase-structure grammar. By defining the meta-linguistic formulae themselves it would be possible to compile a compiler for Algol with the compiler compiler.

3.3 The Algol 60 Report

The Algorithmic Language of 1960, short Algol 60, was an important step in the history of programming languages as it had introduced many new aspects and formalized many older concepts to make them more applicable. Many other important programming languages have been developed with the concepts of Algol in mind with Simula, Pascal and most importantly C being the most widely used ones.

The effort of Algol 60 started after several scientists, including Peter Naur, studied and worked on the *Algol 58* report by **(author?)** (Bauer et al.) where they defined an early version of Algol. Peter Naur, working together with Jørn Jensen, found the report to be lacking and they organized a conference in Copenhagen during the year of 1959. From this conference the most notable outcome was that Naur started the *Algol Bulletin* to concentrate the international effort on Algol and to distribute the different ideas. This led to Naur being the leading European mind working on the Algol effort.

According to Naur (9) in his talk *Successes and failures of the ALGOL effort* which took place in 1968, ten years after the first version of Algol 58 was published, a strong understanding of languages as well as compiler problems led to a need for a sound, common programming language. Together with developments in the United States on Fortran as well as the developments of the Amsterdam school which supplied powerful

and general ideas about the central features of programming languages there was solid groundwork for an improved programming language. Furthermore, the development of Backus on the reference language provided a suitable notation for a formal description of the resulting language.

The resulting *Report on the algorithmic language ALGOL 60* by Backus et al. (2) defined the language Algol 60 in a mix of formal, Backus Naur form, and informal syntax. The influence of the report was wide and we will try to discuss some important developments of the language and its syntax description. The most important aspect of a programming language is the programming of computers, in which Algol was a partial success. As a core language with its formal description and the newly introduced nested function definition as well as a strong focus on the lexical scopes of methods, the language could provide the developers of the time with new and strong concepts for their work. However, the mission input and output facilities proved to be detrimental to a wider field of programming applications.

One aspect of the language which directly coincided with its rather formal notation was the publication of algorithms. The very non ambivalent description of the language made it a favorite for the development of algorithms and allowed it to be used theoretically in many applications. A further aspect of importance was on the development of formal languages and has acted as a strong stimulus for compiler designers. However, according to Naur, this development could be seen from two sides and the over formalization would definitely not help to draw in further interest from the public.

The form of the description was of great interest to Naur and would prove to be the topic of his interest in his later work. He was interested mostly in that a language could not only be understood by a computer but also by a human reader. This was also why he as the sole editor tried to use about the same amount of formal as well as informal notation to describe the language. The mixture of both was important to Naur and he later even proposed that the report should have included more informal introduction. Naur himself criticized his own work on Algol 60 strongly. Even though he was content with the formal description of the language, he was not content on how the report was received in the larger public. The Algol 60 report, as said by Naur, was very hard on the uninitiated reader.

Some problems in the documentation by Naur (8) later pointed out in his article *Documentation Problems: Algol 60* were mostly in the reliance of language description on formal notations. However, even though the Backus Naur form helped in documentation, it provided no safety against errors. Writing a report on a programming language, according to Naur, differed not substantially from any other form of communication and the integral part was that the description must be complete and unambiguous. It is therefore also important to stress that event though there has been substantial progress in constructing mechanical and formal metalanguages, the natural languages will always remain the ultimate metalanguage.

The effort in the Algol 60 report was thus not futile, however, the over formalization proved to be a thorn in the eyes of Naur and would be of particular interest to him in his further work. Generally, it can be said that the report and the corresponding language of Algol 60 proved to be strongly influential, however, far from perfect.

3.4 Type checking in the Gier Algol Compiler

3.5 Notes on Formalization

4 Conclusions

5 Further Work of Peter Naur

References

[1] Backus, J. (1959). The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *IFIP Congress*.

[2] Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., et al. (1960). Report on the algorithmic language algol 60. *Numerische Mathematik*, 2(1):106–136.

[Bauer et al.] Bauer, F. L., Bottenbruch, H., Rutishauser, H., Samelson, K., Backus, J., Katz, C., Perlis, A., and Wegstein, J. H. Algol 58.

[4] Bhate, S. (2002). *Panini*. Sahitya Akademi.

[5] Brooker, R., MacCallum, I., Morris, D., and Rohl, J. (1963). The compiler compiler. In *International Tracts in Computer Science and Technology and Their Application*, volume 3, pages 229–275. Elsevier.

[6] Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

[7] Knuth, D. E. (1964). *Backus Normal Form vs. Backus Naur Form*, chapter 7.12. ACM.

[8] Naur, P. (1963). Documentation problems: Algol 60. *Communications of the ACM*, 6(3):77–79.

[9] Naur, P. (1968). Successes and failures of the algol effort. *algol Bulletin*, (28):58–62.

[10] Rohl, J. S. (1968). A note on backus naur form. *The Computer Journal*, 10(4):336–337.