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 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 thinking in larger meta questions rather than small technicalities has proven to be fundamental for the strength of the 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 as his influence of the programming languages of the future. This paper 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 history of the Backus-Naur form. From giving a background to the development of the form, we will cover much of the forms technicalities and a proof to properly settle it into the context free grammars. Furthermore, we will discuss the importance and impact of Naur's contribution to Algol 60 and its impact on the description of programming languages.

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 civilisation. Pānini, a Sanskrit philologist and grammarian which has worked between the 6th and 4th century BCE had, according to the biography by **?**, already described a notation structure for the language of Sanskrit which resembled the structure of the Backus Notation. However, the interest in a formal grammar 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 introduction of the computer there was a new interest in language description.

2.1 Noam Chomsky

Noam Chomsky, an American linguist teaching at MIT combined the new interest from mathematicians and linguists and defined the grammar structure of English as finite state Markov processes. **?** introduced a notation for the rewriting of strings according to a grammar as well as he established the distinction between several classes of grammars.

The idea of Chomsky was to define a language L as a (finite or infinte) set of sentences of finite length. Furthermore a string is defined as a concatenation of symbols of an Alphabet A which consists of a finite set of terminal symbols. A grammar is now defined as a device that produces all of the strings that are sentences of L.

The interesting idea of Chomsky was now to define that any grammar is based on a finite number of sentences and that it projects this set to an infinite set of grammatical sentences by establishing general transformation laws. A properly formulated grammar should furthermore 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.

? further developed three different types of languages which were based on two grammars he defined. The more important grammar for this report is the phrase-structure grammar, also called context free grammar, which we will now describe in detail.

A phrase-structure grammar consists of a finite vocabulary V_p, a finite set of initial strings $\sum$ 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 X_i by replacement of a single symbol of X_i by some string.

An example from **?** for a phrase-structure grammar would be the following. $^\frown$ is introduced for concatenation and can be replaced by an empty space and # define the boundaries of the grammar:

$$\begin{aligned}\sum : \#Sentence\# \\ F : Sentence \rightarrow NP^\frown VP \\ VP \rightarrow Verb^\frown NP \\ NP \rightarrow the^\frown man, the^\frown book \\ Verb \rightarrow took\end{aligned}$$

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

Chomsky elaborated 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.

This is the most important idea of Chomsky which had a large impact on the formal notation of programming languages later on. The distinction of generative rules of a grammar and transformational rules, which are applied after generating the language may remind some of the way functions and similar constructs in programming languages work today.

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 continued the preliminary work of Chomsky and was also one of the more influential computer scientists working on the then newly established programming language international algebraic language (IAL). IAL was a predecessor to the programming language known as ALGOL 60 which will play a role later on in this report and especially in the contribution of Peter Naur.

Backus introduced a metalinguistic formulae to formally describe the rules of the IAL programming language. The metalinguistic 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. A reference language always if only referring to the syntax. This goal of a reference language to describe the rules called for an exact and precise description of those rules.

? introduced the syntax of the metalinguistic formulae in the influential preliminary report on the syntax and semantics of the IAL. The syntax of the reference language proposed by Backus was the following:

- Sequences of character symbols enclosed in "<>" are metalinguistic variables whose values are strings of symbols,
- The marks ":$\equiv$" and "or" are metalinguistic 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:

$$< ab >:\equiv (or[or < ab > (or < ab >< d >$$

Where <d> denotes the decimal digits. As one may observe, the variable <ab> is used recursively. Some values for <ab> are:

$$((($$
$$[86$$

This simple metalinguistic formulae, which should remind oneself to the phrase-structure grammar established by Chomsky, allowed Backus to describe the IAL in a formal way which would allow hardware and software engineers to create compilers and programs for the new language without needing further explanation. However the formulae devised by Backus did not yet catch on with the general field of language design. For this, first a Danish Computer Scientist named Peter Naur would have to introduce some changes and simplifications.

3 Main part

3.1 The need for a formulae

Before we talk about the contribution of Peter Naur to the field of programming language description as well as his contribution to compiler designs, I would like to talk about where the need for a formal notation to describe a language comes from. We have seen in the background that the need for a formal description of a language has been established long ago, however, this does not explain the real need for this in the modern times of programming language description.

A programming language, especially the programming language ALGOL 60, has according to **?** the design to be a vehicle for expressing the process of scientific and engineering calculations and numerical analysis. The actual use of a programming language may be far wider, however, in the core most languages are still equipped for this very basic task. The important part of this is that the programming language has to understandable for both the human as well as the computer. This implies two different approaches to the description of a programming language.

On one hand, a formal definition is important to simplify the process of compiler creation, while at the same time being understandable to the human reader. The philosophies on how formal a programming language should be described differ largely. **?** Himself was of the opinion that an overly formal description of a programming language would hurt rather than improve the language. According to **?**, a formal language is only an extension of the informal language rather than a replacement. This signifies an approach in Naur's personal philosophy that desriptions should be easily understandable and approachable for human readers by also including informal language.

This still meant that there had to be some sort of formulae to describe the programming language in a formal way enough to prevent issues of clarity or even worse mistakes. The formal notation to describe a programming languages would also be powerful enough to create all strings allowed in the language while not being overly powerful or complicated. The result of this was that the notation would have to be a version of Chomsky's phrase-structure grammar.

3.2 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. Peter 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 Combridge, 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. In the beginning he mostly focused on the highly influental Zurich report of 1958 which lay the ground work for ALGOL. Naur, a very pragmatic computer scientist, was not content with the Zurich report and quickly organized a conference in Copenhagen to discuss the difficulties Naur saw in the Zurich report. After the conference, which did not satisfy Naur entirely, he initialed the ALGOL Bulleting. By editing this discussion journal, Naur became unintentionally the leading European behind the effort to create ALGOL.

Naurs 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. One important piece of this description is the Backus-Naur form named by **?** (originally called the Backus normal form). The Backus-Naur form is a metalanguage of metalinguistic formulas for chomsky context-free grammars. Naur applied the notation proposed by **?** and modified it slightly and used it for the important ALGOL 60 report by **?**.

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

3.3 Backus-Naur form