Peter Naur's contribution to formal notations and beyond

Viktor Gsteiger

University of Basel

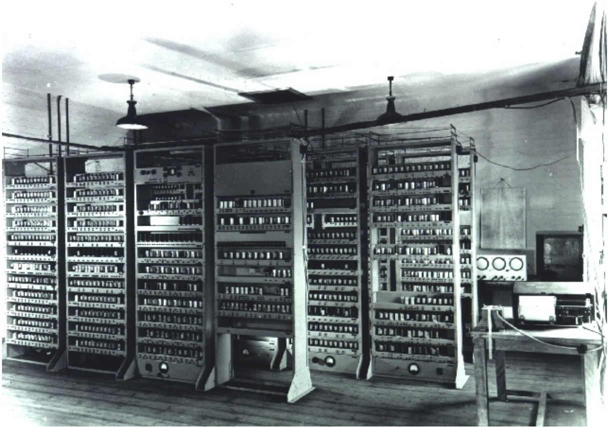
Seminar Turing Award Winners and Their Contributions

v.gsteiger@unibas.ch

November 26, 2020

Formal notations in history

New need for formal notations



Peter Naur

Peter Naur's contribution

Citation

For fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming.

Phrase structure grammar

Definition

Quadruple $G = (\Sigma, V, P, S)$, where

- Σ *terminal symbols*
- V *variables* with $V \cap \Sigma = \emptyset$
- $P \subseteq (V \cup \Sigma)^{+} \times (V \cup \Sigma)^{*}$ *production rules*
- $S \in V$ *start symbol*

Phrase structure

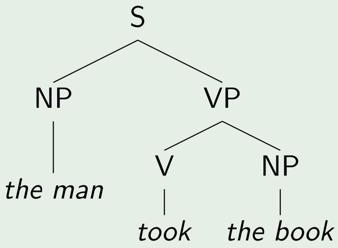
Example

$$G = (\{\text{the man}, \text{the book}, \text{took}\}, \\ \{S, NP, VP, V\}, P, S)$$

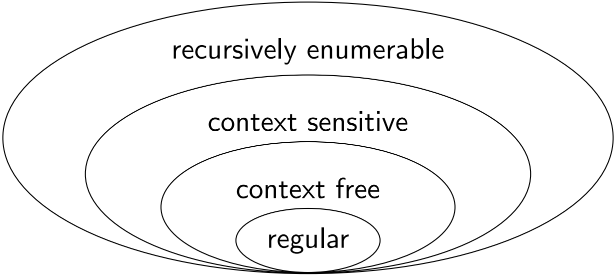
$$\begin{aligned} P &: \\ S &\rightarrow NP\ \ VP \\ VP &\rightarrow V\ \ NP \\ NP &\rightarrow \text{the man} \\ NP &\rightarrow \text{the book} \\ V &\rightarrow \text{took} \end{aligned}$$

Phrase structure in tree form

Example



Chomsky Hierarchy



Backus Naur form

Example

$$
\begin{aligned}
< S > &::=< NP > \quad < VP > \\
< VP > &::=< V > \quad < NP > \\
< NP > &::= \text{the man} \mid \text{the book} \\
< V > &::= \text{took}
\end{aligned}
$$

Power of Backus Naur Form

- BNF generates exactly the context free languages
- EBNF has Exceptions (Rule - Exception)
  $\rightarrow$ not context free

Theorem

Production rules of the form Rule - Exception have no BNF equivalent

Theorem

Production rules of the form Rule - Exception have no BNF equivalent

Proof.

Let $L_1 = \{a^n b^n a^m | n, m \geq 0\}$ be generated by
$\langle L_1 \rangle ::= \langle X \rangle \langle A \rangle$
$\langle X \rangle ::= a \langle X \rangle b | \varepsilon$
$\langle A \rangle ::= \langle A \rangle a | \varepsilon$

Let $L_2 = \{a^n b^m a^m | n, m \geq 0\}$ be generated by
$\langle L_2 \rangle ::= \langle A \rangle \langle X \rangle$
$\langle X \rangle ::= a \langle X \rangle b | \varepsilon$
$\langle A \rangle ::= \langle A \rangle a | \varepsilon$

Theorem

Production rules of the form Rule - Exception have no BNF equivalent

Proof (Cont.)

Now show $L_3 = L_1 \cap L_2 = \{a^n b^n a^n | n \geq 0\}$ not context free by contradiction.

For given p $\geq$ 1 we can choose p such that s $= a^p b^p a^p \in L_3$. PL tells us $s = uvwxy$ with substrings u, v, w, x, and y, such that $|vx| \geq 1$, $|vwx| \leq p$, and $uv^i wx^i y \in L_3$. Now uvx can not contain more than two distinct symbols. Thus $uv^2 wx^2 y \notin L_3$.

Theorem

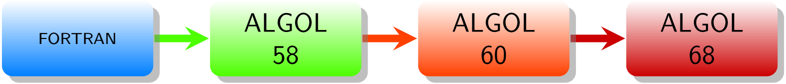
Production rules of the form Rule - Exception have no BNF equivalent

Proof (Cont.)

Thus $L_3 = \{a^n b^n a^n | n \geq 0\} = L_1 - (L_1 - L_2)$ is not context free. Thus Rule - Exception can not be modelled in BNF, since BNF generates only context free languages. □

- EBNF is more powerful than BNF
- BNF is exactly as powerful as context free grammars
- EBNF suffers from Russel-like paradoxes, e.g. xx = "A" - xx;

Historical context



Fortran

GENERAL FORM	EXAMPLES
1 to 5 decimal digits. A preceding + or − sign is optional. The magnitude of the constant must be less than 32768.	3 +1 −28987

Algol 58

3.31 Integers and numbers

$$\langle\text{digit}\rangle :\equiv 0 \textit{ or } 1 \textit{ or } 2 \textit{ or } 3 \textit{ or } 4 \textit{ or } 5 \textit{ or } 6 \textit{ or } 7 \textit{ or } 8 \textit{ or } 9$$

$$\langle\text{integer}\rangle :\equiv \langle\text{digit}\rangle \textit{ or } \langle\text{integer}\rangle\ \langle\text{digit}\rangle$$

$$\langle\text{dn}\rangle :\equiv \langle\text{integer}\rangle \textit{ or } \langle\text{integer}\rangle \textit{ or } \langle\text{integer}\rangle \textit{ or } \langle\text{dn}\rangle\ \langle\text{integer}\rangle$$

$$\langle\text{si}\rangle :\equiv +\langle\text{integer}\rangle \textit{ or } -\langle\text{integer}\rangle \text{ or } \langle\text{integer}\rangle$$

$$\langle\text{en}\rangle :\equiv \langle\text{dn}\rangle\ {}_{10}\langle\text{si}\rangle \textit{ or } {}_{10}\langle\text{si}\rangle$$

$$\langle\text{number}\rangle :\equiv \langle\text{integer}\rangle \textit{ or } \langle\text{dn}\rangle \textit{ or } \langle\text{en}\rangle$$

Algol 60

```
<digit> ::= 0|1|2|3|4|5|6|7|8|9
     Digits are used for forming numbers, identifiers, and strings.
<unsigned integer> ::= <digit>|<unsigned integer><digit>
<integer> ::= <unsigned integer>|+<unsigned integer>| -<unsigned integer>
```

+ Additional examples, semantics and types

Algol 68

b) An arithmetic value has a "length number", i.e., a positive integer characterizing the degree of discrimination with which the value is kept in the computer. The number of integers (real numbers) of given length number that can be distinguished increases with the length number up to a certain length number, the number of different lengths of integers (real numbers) {10.1.a,c}, after which it is constant.

c) For each pair of integers (real numbers) of the same length number, the relationship "to be smaller than" is defined {10.2.3.a, 10.2.4.a}. For each pair of integers of the same length number, a third integer of that length number may exist, the first integer "minus" the other one {10.2.3.g}. Finally, for each pair of real numbers of the same length number, three real numbers of that length number may exist, the first real number "minus" ("times", "divided by") the other one {10.2.4.g,l,m}; these real numbers are obtained "in the sense of numerical analysis", i.e., by performing the operations known in mathematics by these terms on real numbers which may deviate slightly from the given ones {; this deviation is left undefined in this Report}.

Properties of Algol 60

- Block scope
- Call-by-value and call-by-name parameter passing
- Formal specification
- No I/O facilities

Block scope

Example

```
begin
    integer X;
    X := 5;
    begin
        integer X, Y;
        X := 4;
        Y := 8;
    end
    print(X);
    Y := 12;
end;
```

Call-by-value

- Callee parameter Value of arguments given
- No modification of caller argument

Call-by-name

- Pass thunk
- Similar to call-by-reference
- Parameter is passed without evaluation

Call-by-name

Example

```
procedure swap (a, b);
integer a, b, temp;
begin
    temp := a;
    a := b;
    b := temp
end;
```

Call-by-name

Example

swap(x, y):

```
temp := x;
x := y;
y := temp
```

swap(i, x[i]):

```
temp := i;
i := x[i];
x[i] := temp
```

I/O facilities

Example

```
begin
    print('Hello world!');
    comment error!
end;
```

Project

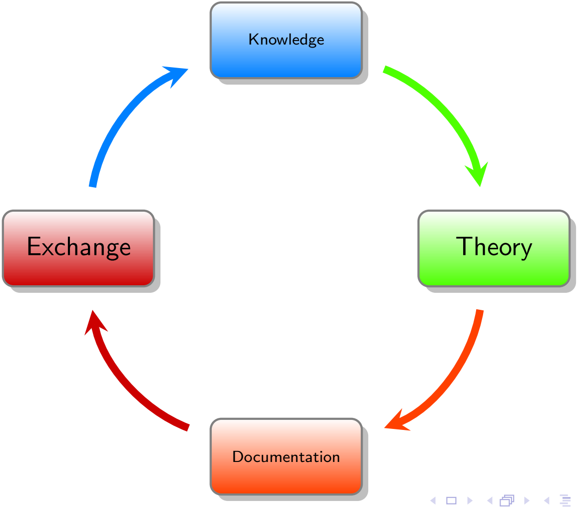
ALGOL 60 Tutorial

We shall not forget the roots of our tools

Programming as theory building

- Source code contains theory
- Transfer theory to next programmer
- Theory should be observable

Role of formal descriptions



Problems of over-formalization

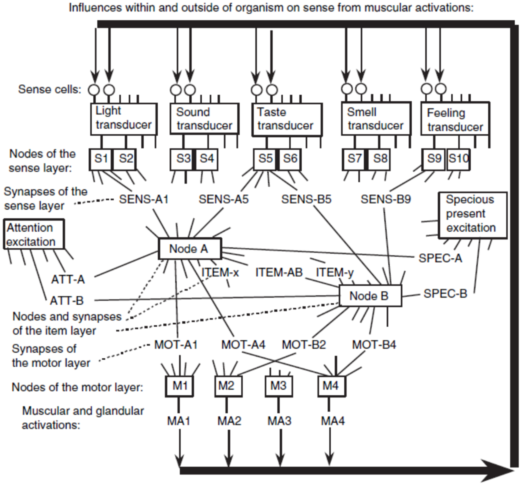
- Hard to exchange
- Prone to errors
- Goal in itself
- Disconnected from environment

$\rightarrow$ For an example see Proof Versus Formalization by Peter Naur

Computing Versus Human Thinking

- Description of computers
- Description of human thinking

Synapse state description



Importance of formal notation

"For achieving clarity any formal mode of expression should be used, not as a goal in itself, but wherever it appears to be helpful to authors and readers alike."

(Peter Naur, Formalization in Program Development)

Discussion

Do you think that human thinking will be formally describable?