(Anmerkungen zur) Geschichte der Programmiersprachen

Prof. Dr. Hans J. Schneider
Department Informatik (Informatik 2)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Wintersemester 2011/12

Overview

- The first languages were defined by English prose.
  - Advantage: Easy to read
  - Disadvantage: Prone to ambiguities and misunderstanding
- First approach to formal syntax definition: Algol 60
  - Backus developed a notation based on Chomsky's contextfree grammars (that allow efficient parsing algorithms).
  - Each definition consisted of three sections: Syntax - Examples - Semantics
- "Static semantics":
  - Syntactic properties that are not contextfree were described under the heading "Semantics".
  - In the literature, this decision led to the misunderstandable notion of "static semantics", since these "semantic" properties could be checked at compile-time.
- Van Wijngaarden and his group extended the definition to include non-contextfree properties.

Definition of Fortran: Expressions

By repeated use of the following rules, all permissible expressions may be derived.[1]

1. **Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 3 and 1 are fixed point expressions, and ALPHA and A(I,J,K) are floating point expressions.**
2. **. . .**
3. **If E is an expression, and if its first character is not + or -, then +E and -E are expressions of the same mode as E. Thus -A is an expression, but +-A is not.**
4. **If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc. are expressions.**
5. **If E and F are expressions of the same mode, and if the first character of F is not + or -, then E+F, E-F, E*F, and E/F are expressions of the same mode. Thus A-+B and A/+B are not expressions.**

[1] Aus dem ersten FORTRAN Manual (1956), zitiert nach:
J.W. Backus: *The history of Fortran I, II, and III*, ACM SIGPLAN Not. 13, 8 (1978), pp. 165-180

Definition of Fortran: Subscripts

- **Subscripts**

  General form: Let v represent any fixed point variable and c (or c') any unsigned fixed point constant. Then a subscript is an expression of one of the forms:

  ```
  v    v+c  c*v+c'
  c    v-c  c*v-c'
  c*v
  ```

  The symbol * denotes multiplication. The variable v must not itself be subscripted.

- **Subscripted variables**

  General form: A fixed or floating point variable followed by parentheses enclosing 1, 2, or 3 subscripts separated by commas.

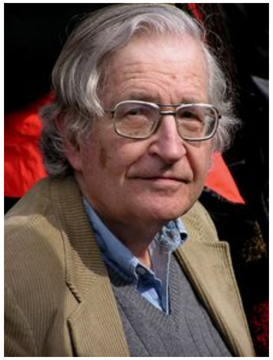
  Examples:

  ```
  A(I)    K(3)    BETA(5*J-2,K+2,L)
  ```

[1] loc. cit.

Chomsky-Grammatiken

- **Noam Chomsky führt 1956 im Zusammenhang mit linguistischen Fragestellungen den Begriff der „Phrase structure grammar“ ein.**[1]
- **1959 formalisiert er diese auf der Basis der Semi-Thue-Systeme und definiert eine Hierarchie von Grammatiken und Sprachen.**[2]
- **A phrase structure grammar consists of a finite set of ”rewriting rules” of the form $\phi \to \psi$, where ϕ and ψ are strings of symbols. It contains a special ”initial” symbol S (standing for ”sentence”) and a boundary symbol # indicating the beginning and end of sentences. Some of the symbols of the grammar stand for words and morphemes (grammatically significant parts of words). These constitute the ”terminal vocabulary.” Other symbols stand for phrases, and constitute the ”nonterminal vocabulary” (S is one of these, standing for the ”longest” phrase).**


N. Chomsky (*1928)

[1] N. Chomsky: *Three models for the description of language*, IRE Transactions on Information Theory 2(1956), pp. 113-124
[2] N. Chomsky: *On certain formal properties of grammars*, Information and Control 2 (1959), pp. 137-167

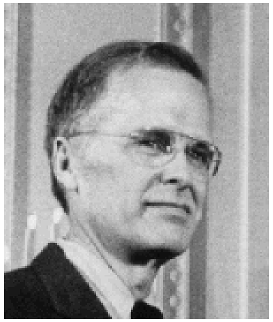
Chomsky-Hierarchie

- **Die Chomsky-Hierarchie schränkt schrittweise die Form der zulässigen Ersetzungsregeln (Produktionen) ein:**[1]
  - **Restriction 1: If $\varphi \to \psi$, then there are A, φ_1, φ_2, ω such that $\varphi = \varphi_1 A \varphi_2$, $\psi = \varphi_1 \omega \varphi_2$, and $\omega \neq I$.**
  - **Restriction 2: If $\varphi \to \psi$, then there are A, φ_1, φ_2, ω such that $\varphi = \varphi_1 A \varphi_2$, $\psi = \varphi_1 \omega \varphi_2$, $\omega \neq I$, but $A \to \omega$.**
  - **Restriction 3: If $\varphi \to \psi$, then there are A, φ_1, φ_2, ω, a, B such that $\varphi = \varphi_1 A \varphi_2$, $\psi = \varphi_1 \omega \varphi_2$, $\omega \neq I$, $A \to \omega$, but $\omega = aB$ or $\omega = a$.**
- **Die unbeschränkten Grammatiken entsprechen den Semi-Thue-Systemen; das Wortproblem ist unentscheidbar.**
  - **Einschränkung 1 macht das Wortproblem entscheidbar, aber mit exponentiellem Aufwand.**
  - **Einschränkung 2 (kontextfreie Grammatiken) erlaubt Algorithmen mit polynomialem Zeitbedarf.**
  - **Einschränkung 3 entspricht den regulären Ausdrücken (endlichen Automaten).**

[1]N. Chomsky: *On certain formal properties of grammars*, Information and Control 2 (1959), pp. 137-167

Das Entstehen der BNF

- Ende 1958 erscheint eine erste Vorversion von Algol (International Algebraic Language).[1]
- Im November 1959 stellt J.W. Backus auf der ersten internationalen Konferenz über „Information Processing“ in Paris ein Konzept zur formalen Beschreibung der Sprache vor.[2]
- Backus:
  - There must exist a precise description of those sequences of symbols which constitute legal IAL programs.
  - For every legal program there must be a precise description of its "meaning", the process or transformation which it describes, if any.
  - Only the description of legal programs has been completed however.


J.W. Backus (1924-2007)

[1] A.J. Perlis/K. Samelson: Preliminary Report – International Algebraic Language, Comm. Assoc. Comput. Mach. 1, 12 (1958), pp. 8-22

[2] J.W. Backus: The syntax and semantics of the proposed international algebraic language of the Zurich ACM-Gamm conference, Proc. Int. Conf. Information Processing, Oldenbourg, München, 1960, pp. 125-132

Backus-Naur-Form: Konzept

- **Die Backus-Naur-Form (anfangs auch Backus Normal Form[1]) entspricht den kontextfreien Grammatiken. Die Produktionen werden durch metalinguistische Formeln codiert.**
- **Bausteine:[2]**
  - **Sequences of characters enclosed in the brackets <> represent metalinguistic variables whose values are sequences of symbols.**
  - **The marks ::= and | are metalinguistic connectives.**
  - **Any mark in a formula, which is not a variable or a connective, denotes itself.**
  - **Juxtaposition of marks and/or variables in a formula signifies juxtaposition of the sequences denoted.**
- **Bedeutung der expliziten Verknüpfungen:**
  - **::= bedeutet *„is defined as“*.**
  - **| bedeutet *„or“*.**

[1] D.E. Knuth: *Backus Normal Form vs. Backus Naur Form*, Communicat. Assoc. Comp. Mach. 7, 12 (1964), pp. 735-736
[2] P. Naur (Ed.): *Report on nthe algorithmic language ALGOL 60*, Numerische Mathematik 2 (1960), pp. 106-136

Backus-Naur-Form: Beispiel

- **Definition arithmetischer Ausdrücke:**[1]

```
<arithmetic expression> ::= <term>
      | <adding operator> <term>
      | <arithmetic expression> <adding operator> <term>
<term> ::= <factor>
      | <term> <multiplying operator> <factor>
<factor> ::= <primary>
      <factor> ↑ <primary>
<primary> ::= <unsigned integer> | <variable>
      | (<arithmetic expression>)
<multiplying operator> ::= × | / | ÷
<adding operator> ::= + | −
```

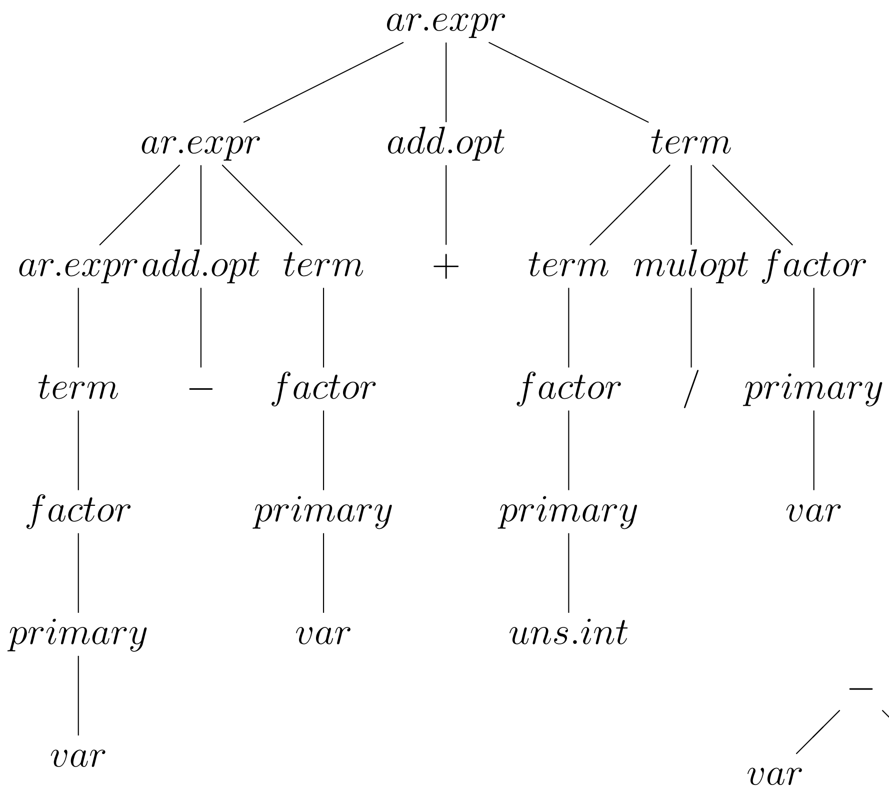
P. Naur (*1928)

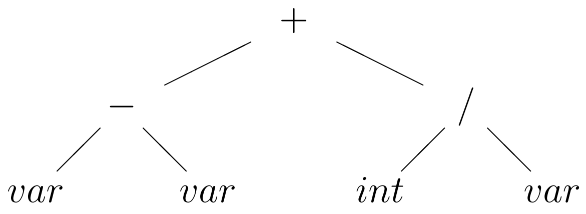
[1] P. Naur (Ed.): Revised Report on the Algorithmic Language ALGOL 60, Communicat. Associat. Comput. Mach. 6, 1 (1963), pp.1-17; Numer. Math. 4 (1963), pp. 420-453

Graphische Darstellungen

- Kontextfreie Grammatiken erlauben eine graphische Darstellung der Satzstruktur (Programmstruktur) als Ableitungsbaum ("Parse tree‘“):
  - Die Blätter sind mit terminalen Symbolen, die übrigen Knoten mit nichtterminalen Symbolen markiert.
  - Der Baum gibt die Struktur gemäß der Grammatik wieder, z.B. den Operatorenvorrang.
  - Unwesentlich verschiedene Ableitungen führen zum gleichen Baum.
  - Gibt es zu einer Zeichenfolge zwei unterschiedliche Ableitungsbäume, so ist die Grammatik mehrdeutig.
- Nicht alle nichtterminalen Symbole, die in einer Ableitung auftreten, sind zum Verständnis der abgeleiteten Zeichenkette erforderlich. Der abstrakte Syntaxbaum konzentriert sich auf das Wesentliche.
- Anmerkung: Die Übersetzung zwischen beiden Darstellungen kann durch Graphersetzungsregeln formal beschrieben werden.

Beispiel zur graphischen Darstellung





Lexikalische Struktur

- The lexical structure of a programming language, i.e., the structure of its words or *tokens*, can be considered separately from syntactic structure, although in some cases, it is an inextractable part of syntax.[1]
- Typische Bausteinkategorien: Schlüsselwörter, Konstanten (Literale), Sonderzeichen (Operatoren, Trennzeichen, usw.), Bezeichner („Identifier“)
- Probleme und Lösungen:
  - FORTRAN: Keine Unterscheidung zwischen Schlüsselwörtern und Bezeichnern
  - ALGOL 60: Schlüsselwörter fett gedruckt bzw. in Apostroph eingeschlossen
  - Später: Schlüsselwörter sind reservierte Zeichenfolgen.
  - Anfangs waren Bezeichner auf eine maximale Länge beschränkt, oder nur die ersten sechs bzw. acht Zeichen dienten der Unterscheidung.
- Die saubere Trennung erlaubt die separate Konstruktion von „Scanner“ und „Parser“.

[1] K.C. Louden: Programming Languages - Principles and Practice PWS-KENT Publ., Boston, 1993

Lexikalische Struktur in Fortran und C

- **Das ursprüngliche Fortran ist ein Beispiel, wo „Scanner“ und „Parser“ nicht getrennt werden können:**

```
DO 99 I = 1.10          DO 99 I = 1,10
```

entsprechen den Pascal-Formulierungen:

```
DO99I := 1.10           for I := 1 to 10 do
```

- **Die Definition in C zeigt das Problem auf:**[1]
Blanks, horizontal and vertical tabs, newlines, formfeeds, and comments (collectively, "white space") ... are ignored except as they separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants. If the input stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

[1]B.W. Kernighan/D.M. Ritchie: The C programming language (ANSI Standard C), Prentice-Hall, Englewood Cliffs, N.J., 1988

Cobol: Syntaxbeschreibung

- **For some languages, such as Cobol and PL/I, much greater conciseness of specification can be achieved by using a two-dimensional layout whereby alternatives may be listed vertically within a large pair of braces. Square brackets indicate optionality, and "…" denotes arbitrary repetition of the preceding unit.**[1]

- **Beispiele (COBOL-ADD, PL/I-Loop):**

$$\texttt{ADD}\left\{\begin{matrix} \textit{constant} \\ \textit{variable} \end{matrix}\right\}\left[,\left\{\begin{matrix} \textit{constant} \\ \textit{variable} \end{matrix}\right\}\right]\ldots\left\{\begin{matrix} \texttt{TO} \\ \texttt{GIVING} \end{matrix}\right\}\textit{variable}$$

$$\textit{expression}\left[\begin{matrix} \texttt{TO}\ \textit{expression}\ [\texttt{BY}\ \textit{expression}] \\ \texttt{BY}\ \textit{expression}\ [\texttt{TO}\ \textit{expression}] \end{matrix}\right][\texttt{while}\ (\ \textit{expression}\ )]$$

- **I think it is fair to say that the COBOL syntactic philosophy (e.g. few verbs with many options per verb) simply could not have been carried through, if we had used BNF from the beginning.**[2]

[1] F.G. Pagan: Formal Specification of Programming Languages – A Panorama Primer, Prentice-Hall, Englewood Cliffs, N.J., 1981

[2] J.E. Sammet: The early history of COBOL, ACM SIGPLAN Not. 13, 8 (1978), pp. 121-161

Extended Backus-Naur Form

- **Some drawbacks of the BNF:**
  - **Recursive BNF rules are often used to define repetition of substructures.**
  - **Alternatives are often used to define optional substructures.**
- **Defining Pascal, N. Wirth developed a variant of the BNF, called Extended BNF (EBNF), to avoid these drawbacks.**[1]

N. Wirth (*1934)

- **Changes for more convenient notation:**
  - **Nonterminals no longer enclosed in $< \ldots >$.**
  - **Terminals enclosed in “ ... ”.**
- **Adopted as a standard by the International Organization for Standardization (ISO/IEE 14977).**

[1] K. Jensen/N. Wirth: *Pascal – User Manual and Report*, Springer, Berlin, 1974

Extended Backus-Naur Form: Details

- **Recursive BNF rules are often used to define repetition of substructures:**
  - <arithmetic expression> ::= <term>
    | <arithmetic expression> <adding operator> <term>
  - **replaced by:**
    <arithmetic expression> ::=
    <term> { <adding operator> <term> }*
- **Disadvantage: obscures left-associativity of operators.**
- **Alternatives are often used to define optional substructures:**
  - <if statement> ::= if <condition> then <statement>
    | if <condition> then <statement> else <statement>
  - **replaced by:**
    <if statement> ::=
    if <condition> then <statement> [else <statement>]

W-Grammars

- A W-grammar consists of two finite sets of rules, the *metaproductions* and the *hyperrules*. These combine to permit the formation of a potentially infinite set of productions, which are used to define the syntax and the context-sensitive requirements.
- Metaproductions:
  - Metaproductions are context-free productions.
  - The nonterminals of metaproductions are called metanotions.
  - Their terminal strings are called protonotions.
- Hyperrules:
  - A hyperrule is a blueprint from which context-free productions can be obtained by replacing each metanotion by a protonotion derived from the metaproductions.
  - In making this substitution, all occurrences of the same metanotion in the hyperrule must be replaced by the same protonotion.

[1]A. van Wijngaarden et al.: Report on the Algorithmic Language Algol 68, Numer. Math. 14 (1969), pp. 79-218
[2]M. Marcotty et al.: A sampler of formal definitions, ACM Comput. Surveys 8, 2 (1976), pp. 191-276

W-Grammars: Notations

- Metaproductions:
  - The nonterminals of metaproductions, called metanotions, are written in uppercase letters.
  - Their terminal strings, the so-called protonotions, consist of lower case characters (with blanks added to improve readability).

  -
    ```
    PLAIN  ::  int; real; bool; ...
    MODE   ::  PLAIN; ref MODE.
    ```
- Hyperrules:
  - A hyperrule is a blueprint to create context-free productions.

  -
    ```
    ref MODE assignation :
        ref MODE destination, becomes symbol, MODE source.
    ```
- Lists of identifiers, lists of parameters, etc:

```
ALPHA    ::  a; b; c; ...; z.
NOTION   ::  ALPHA; NOTION ALPHA.

NOTION list : NOTION;   NOTION list, comma symbol, NOTION.
```

W-Grammar Defining Expressions

- **Hyperrules:**

```
MODE PRIORITY formula:
      LMODE PRIORITY operand,
            LMODE and RMODE giving MODE PRIORITY operator,
            RMODE PRIORITY plus one operand.
MODE PRIORITY operand:
      MODE PRIORITY formula;
      MODE PRIORITY plus one operand.
MODE expression:
      MODE PRIORITY operand.
```

- **Metaproductions:**

```
LMODE      ::  MODE.
RMODE      ::  MODE.
PRIORITY   ::  priority NUMBER.
```

[1] Report Sect. 8.4 (simplified)

W-Grammar Defining Expressions (Cont'd)

- **Metaproductions:**

```
PRIORITY   ::  priority NUMBER.
NUMBER     ::  one; TWO; THREE; FOUR; FIVE;
               SIX; SEVEN; EIGHT; NINE.
TWO        ::  one plus one.
THREE      ::  TWO plus one.
...
```

- **Hyperrules:**

```
MODE PRIORITY NINE plus one operand:
     MODE primary.
MODE primary:
     MODE constant;
     ref MODE variable;
     open symbol, MODE expression, close symbol.
```

Generative Power of W-Grammars

- A finite set of attributes can be treated by increasing both the set of nonterminal symbols and the set of productions.
  - Construct new symbols that consist of a symbol together with special values of the attributes.
  - Copy the productions for each valid combination of the new symbols.
- Sintzoff could show that the generative power of W-grammars is the same as that of unrestricted Chomsky grammars.[1]
- The word problem is undecidable.
- Koster proposed a restricted syntax.[2]

[1] M. Sintzoff: Existence of a van Wijngaarden syntax for every recursively enumerable set, Annales Soc. Sci. Bruxelles II (1967), pp. 115-118

[2] C.H.A. Koster, Affix grammars, in: J.E.L. Peck (Ed.): Algol 68 implementation, North-Holland, Amsterdam, 1971, pp. 95-109

Extensible Languages

- **Galler/Perlis proposed an extension to ALGOL 60 for adding new data types and operators to the language.**[1]
- **Irons:**[2] **New syntactic constructs can be introduced in a program by defining**
  - **the syntax similar to an attributed version of BNF, and**
  - **the semantics similar to a procedure body using the base language including the extensions up to this one.**
- **Such an extension is similar to a procedure declaration, but it allows the programmer to use a more appropriate syntax instead of the fixed syntax of a procedure call, e.g.,**
  `find X in LIST` **instead of** `find(X, LIST)`
- **Irons's system treated the extensions similar to precompiled procedures and implemented the extended syntax by bootstrapping.**

[1] B.A. Galler/A.J. Perlis: A proposal for definitions in ALGOL, Communicat. Associat. Comput. Mach. 10,4 (1967), pp. 204-219

[2] E.T. Irons: Experience with an extensible language, Communicat. Associat. Comput. Mach. 13, 1 (1970), pp. 31-40

Example of an Extension

- **Example (slightly modified): Right-hand assignment:** $e \rightarrow v$
  `<rhd-ass> :: <e, expression> --> <v, identifier> :: 'v := e'`
  - `<rhd-ass>`: **Nonterminal symbol to be defined.**
  - `<e, expression>`: `e` **must satisfy the definition of an expression.**
  - `'v := e'`: **The semantics is the same as that of the usual assignment.**
- **Many people felt that a programming language should be extensible such that it may be used in another application area, but only very few implementations are known.**

Sammet: The Use of English as a Programming Language

- **When I say that I wish to use English as a programming language, I very definitely include mathematics or any other scientific notation.**
- **Reasons why to use English:**
  - **There are too few programmers for the machine capacity we have. Therefore, it is desirable to eliminate the need for professional programmers for handling applications.**
  - **To put every person in communication with the computer is to speed up and make easier his ability to get his problem solved. It was for just such reasons that systems such as** FORTRAN **were developed. I simply want to extend this concept to its furthest limits.**
  - **Every application area has its own jargon, and the less a user has to jump out of this the better off he is. Only by encompassing all jargon areas – which would mean all of English – can we achieve this.**
- **Using English definitely involves the requirement for the computer to query the user if there is any doubt.**

[1] J.E. Sammet: The use of English as a programming language, Communicat. Associat. Comput. Mach. 9 (1966), pp. 228-230