Industrial Automation

(Automação de Processos Industriais)

Discrete Event Systems:

Languages, BNF, compilers

http://users.isr.ist.utl.pt/~jag/courses/api19b/api1920.html

Prof. José Gaspar, 2019/2020

Bibliography:

- **Introduction to Discrete Event Systems**, Christos Cassandras and Stephane Lafortune. Springer, 2008.

- **Compilers: Principles, Techniques, & Tools (2nd Edition)**, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Addison-Wesley Professional, 2006
[know as the "Dragon Book"]

- **Introduction to Automata Theory, Languages, and Computation (2nd ed)**, John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Addison-Wesley, 2001
[known as the "Cinderella Book"]

- **Teaching EBNF first in CS 1**, R.E. Pattis, 25th SIGCSE Technical Symposium on Computer Science Education, 1994, pp. 300-303, (see also https://www.ics.uci.edu/~pattis/misc/ebnf2.pdf)

Computer science history: BNF

1914-1940 – Description of language, incl. phrase structure. String **rewriting** rules [Axel Thue, Emil Post, Alan Turing].

1956 – **Noam Chomsky**, teaching linguistics at MIT, clear distinction of generative rules, as in **context-free grammars**, and transformation rules.

1959 – **John Backus** (IBM) proposed a metalanguage to describe the syntax of a new programming language. **BNF** = Backus-Naur form. ABNF = Augmented BNF.

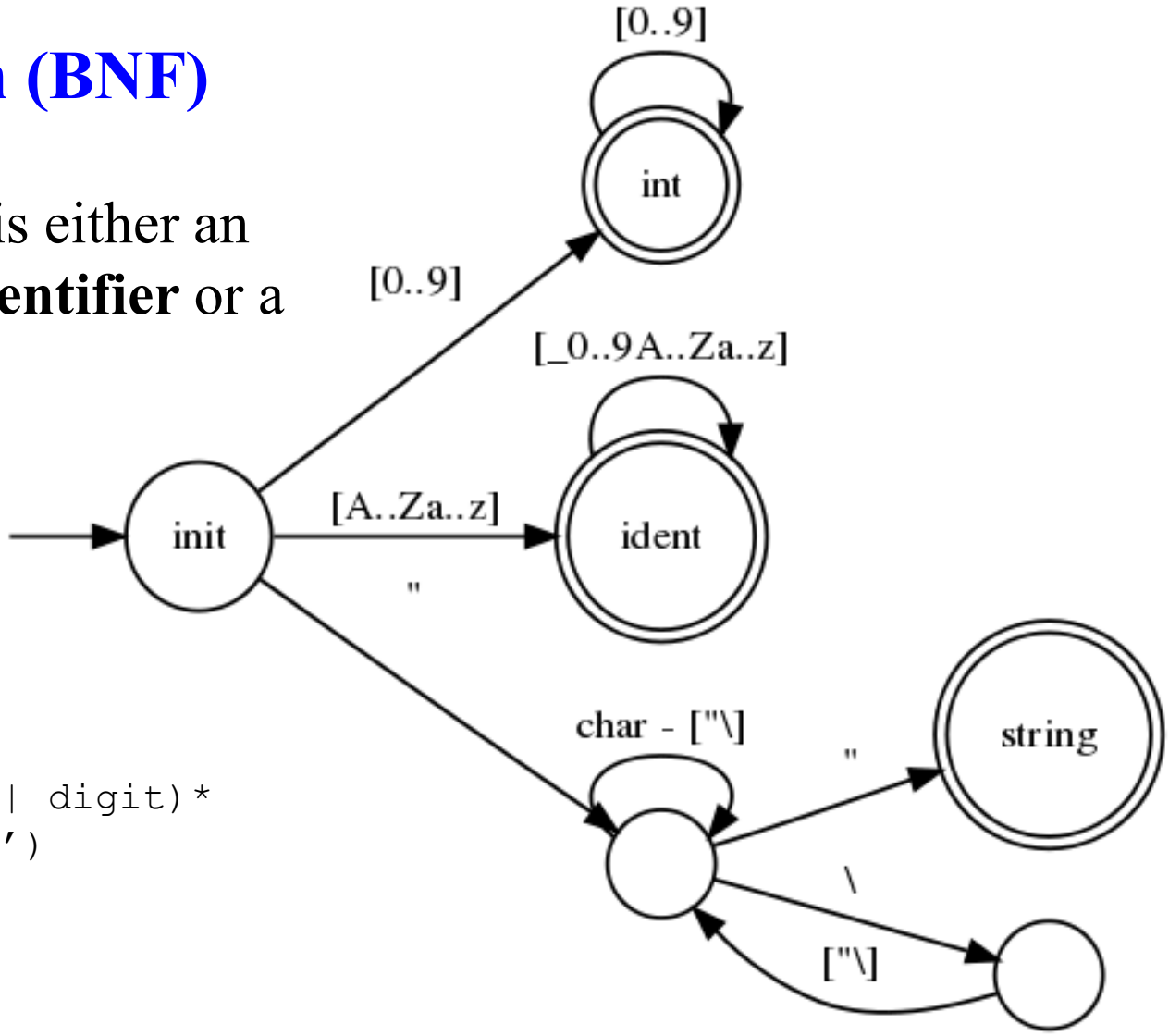
Backus-Naur form (BNF)

Example of a possible BNF for a U.S. postal address:

```
 <postal-address> ::= <name-part> <street-address> <zip-part>

      <name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL>
                    | <personal-part> <name-part>

  <personal-part> ::= <initial> "." | <first-name>

 <street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

       <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
    <opt-apt-num> ::= <apt-num> | ""
```

Backus-Naur form (BNF)

Example: a string that is either an **integer literal** or an **identifier** or a **string literal**.

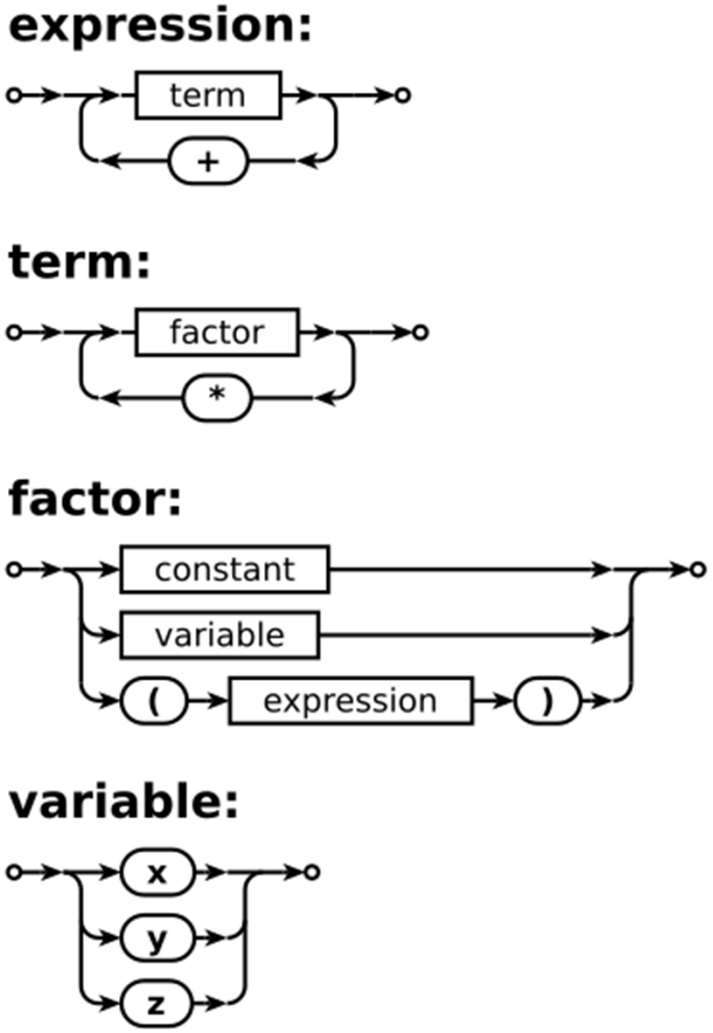


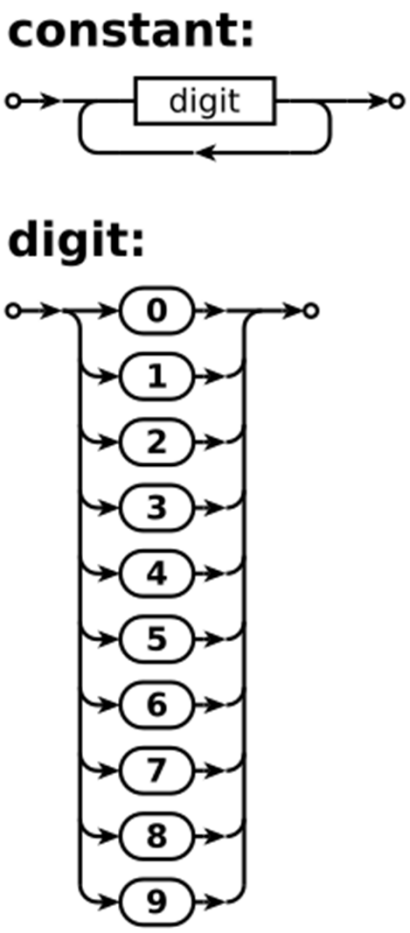
The corresponding regular expression:

```
Str ::= digit digit*
| letter (’_’ | letter | digit)*
| ’"’ (char - (’\’ | ’"’)
| ’\’ (’\’ | ’"’))* ’"’
```

[Ranta12] "Implementing Programming Languages. An Introduction to Compilers and Interpreters", Aarne Ranta, College Publications, 2012

Graphical alternative to BNF, Syntax chart / Syntax diagram





From BNF to a compiler

BNF code is processed by other tools:
- **Lex** (Alex for Haskell, JLex for Java, GNU Flex for C)
- **Yacc** (Happy for Haskell, Cup for Java, GNU Bison for C)

Lex = **Lexical analyzer** (Mike Lesk and Eric Schmidt, 1070s)
Lex makes a finite automata to find regular expressions, “tokens”

Yacc = **Yet Another Compiler Compiler** (Stephen Johnson, 1070s)
Yacc reads “tokens”/”strings” and verifies grammar

Lex and Yacc are typically used together: Lex processes the string input to **output tokens**, Yacc makes a **parser for processing the tokenized** input provided by Lex.