Appendix A: Syntax Diagrams

References:

- Kathleen Jensen/Niklaus Wirth: PASCAL — User Manual and Report, 4th Edition. Springer, 1991.
- Niklaus Wirth: Compilerbau (in German). Teubner, 1986.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
  Appendix A is a short introduction to syntax diagrams.
- Don Chamberlin: A Complete Guide to DB2 Universal Database.
  Morgan Kaufmann, 1998.
  Section 1.1.2 is a very quick introduction to syntax diagrams.

Overview

1. General Remarks about Syntax Formalisms

2. Syntax Diagrams

Syntax Formalisms (1)

- An SQL query is a sequence of ASCII characters, but not every sequence of ASCII characters is a valid SQL query.

  This is not quite precise: (1) The ASCII encoding is not required. The Standard defines an SQL character set that includes letters `a-zA-Z`, digits, the space, and these special characters: `"%&'()*+,-./:;<=>?_|`. (2) Extended character sets (with national characters) can be used.

- There are various formalisms for specifying clearly and unambiguously the subset of all character sequences which are legal in a language like SQL.

  Other character sequences are said to be syntactically incorrect / to contain syntax errors.

Syntax Formalisms (2)

- The most important syntax formalisms are:
  - ◇ Regular Expressions
  - ◇ Context-Free Grammars (CFGs)
  - ◇ Syntax Diagrams
- Syntax Diagrams are equivalent in their expressive power to context-free grammars.
- Both are more powerful than regular expressions.

  I.e. every language which can be defined by a regular expression can also be defined by a CFG/syntax diagram.

Syntax Formalisms (3)

- In order to master a language like SQL, one of course need examples, but one also should to be able to read a formal specification of the language.

  This can be used as a reference for doubtful cases, but it will also improve the understanding of the language. E.g. syntactic categories introduced in the formal definition are often useful concepts. If one has only seen n examples and learnt nothing more general, all that one can really do are these n examples.

- The Oracle SQL Reference Manual as well as many other database books contain syntax diagrams.
- The standard uses a context free grammar.

Lexical Syntax (1)

- The syntax of languages like SQL or Pascal is normally specified in two steps.
- One first defines how "words" (formally called tokens or lexical symbols) can be composed from single characters.

  This part of the definition is the lexical syntax of the language.
- Then the overall syntax of the language is defined in terms of these tokens.
- This two-step approach reduces the complexity.

Lexical Syntax (2)

- In case of SQL, such words/tokens are, e.g.
  - ◇ Reserved words / keywords with a specific meaning, e.g. `SELECT`.
  - ◇ Identifiers used e.g. for table and column names, e.g. `STUDENTS`.
  - ◇ Datatype constants/literals, e.g. `'DBMS'` or `123`.
  - ◇ Comparison and datatype operators, e.g. `=`, `<=`, `+`, `||`.
  - ◇ Punctation characters, e.g. parentheses and the comma.

Lexical Syntax (3)

- Simpler formalisms such as regular expressions can be used for defining the lexical syntax.

  In this course, syntax diagrams are used for the lexical syntax, too. However, when implementing an SQL parser, one would use a tool like `lex` (based on regular expressions) for the lexical syntax, because it runs faster than a more powerful tool like `yacc` that is required for the overall syntax. Since there are much more characters than tokens, it makes sense to condensate the input first with a fast tool.

- Free-format languages like SQL or Pascal allow white space (blanks, line breaks, etc.) and comments between tokens.

  The lexical analysis phase removes white space (including coments). The overall syntax analysis sees a sequence of tokens as input.

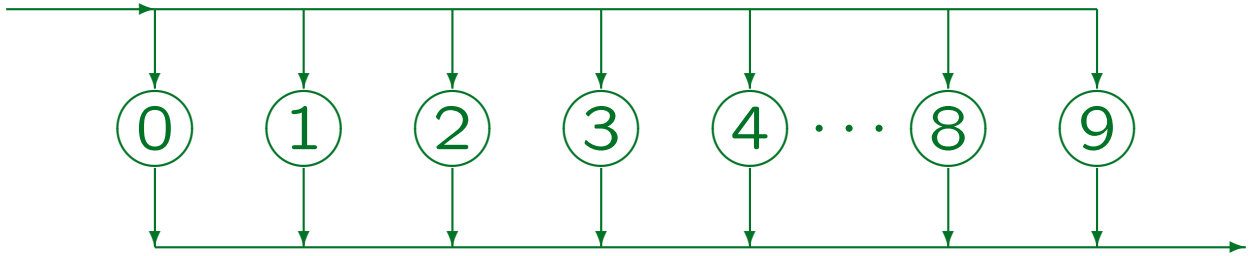
Overview

1. General Remarks about Syntax Formalisms

2. Syntax Diagrams

Syntax Diagrams (1)

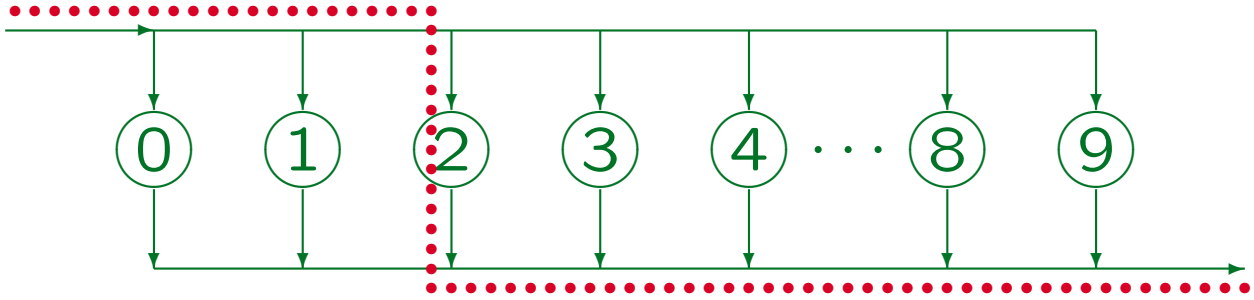
- Digit:



- A syntax diagram consists of:
  - ◇ A name, in this case “Digit”.
  - ◇ A start (node): Arrow that enters the diagram.
  - ◇ A finish (node): Arrow that leaves the diagram.
  - ◇ Oval and rectangular boxes connected by directed edges.

Syntax Diagrams (2)

- The formal language defined by this diagram is very simple, it is the set of digits $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- In order to check whether an input, e.g. "2", really belongs to the language "Digit" defined by this diagram, one must trace a path through the diagram that corresponds to the input:



Syntax Diagrams (3)

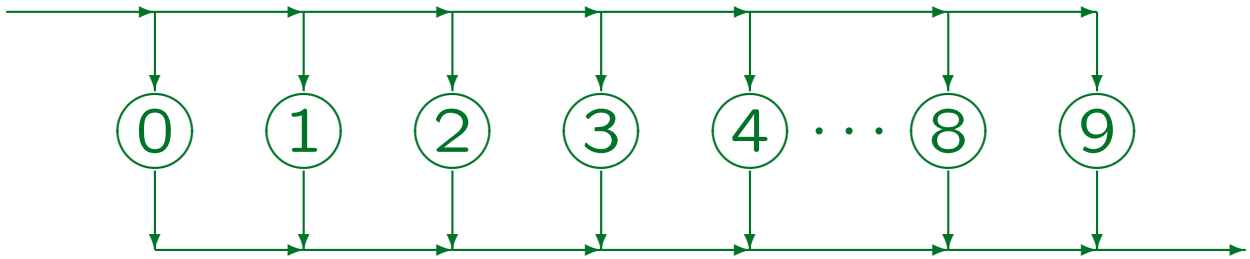
- Such diagrams can be used in two ways:
  - ◇ In order to generate legal words of the language "Digit", one follows a path through the diagram from start to finish and prints every symbol in each oval box through which one passes.
  - ◇ In order to determine that a given input is syntactically correct, one must find a path through the diagram such that whenever one reaches an oval, the character in the oval is the next input character, which is then read away.

Syntax Diagrams (4)

- Every edge has only one possible direction.

  Sometimes it might be a bit difficult for the beginner to decide on the direction of an edge.

- If all directions are made explicit, the diagram looks as follows:



- Since this looks complicated, the arrow heads are normally not repeated on every segment of an edge.

Syntax Diagrams (5)

- There are branching points in the diagram where one can choose different paths.

  E.g. after following the incoming arrow, one can choose to go down and print/read the digit 0 or go to the right for digits 1 to 9.

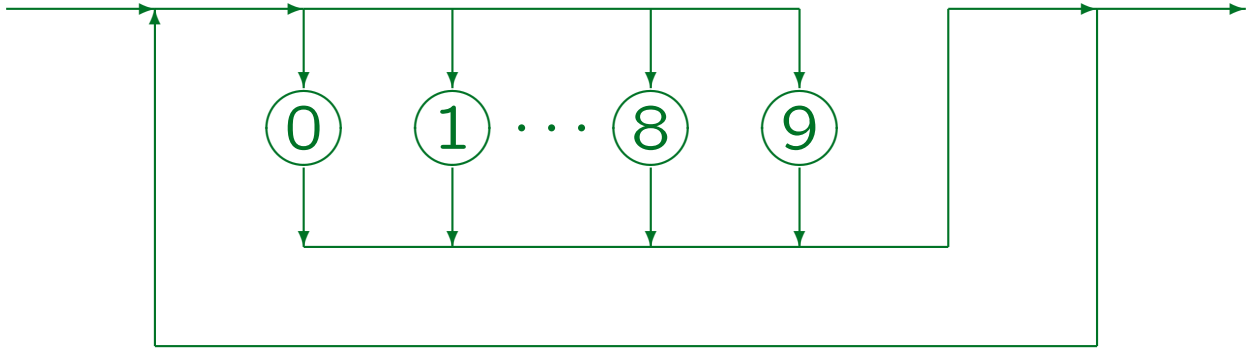
- When analyzing a given input for correctness, looking at the next input symbol will often determine a unique path that must be chosen.

  One tries to construct syntax diagrams such that at every branching, the ovals that can be reached in the two directions contain disjoint symbols.

- If one cannot find a path, the input is wrong.

Syntax Diagrams (6)

- Syntax diagrams can contain cycles (e.g. there are infinitely many different possible SQL queries).
- Digit Sequence:



- Exercise: Find a path to show that "81" is correct.

  One can pass through the same edge several times.

Syntax Diagrams (7)

- Previously defined syntax diagrams can be used as modules in larger ones.
- This is symbolized by drawing a rectangular box with the name of the syntax diagram in it.
- Digit Sequence:

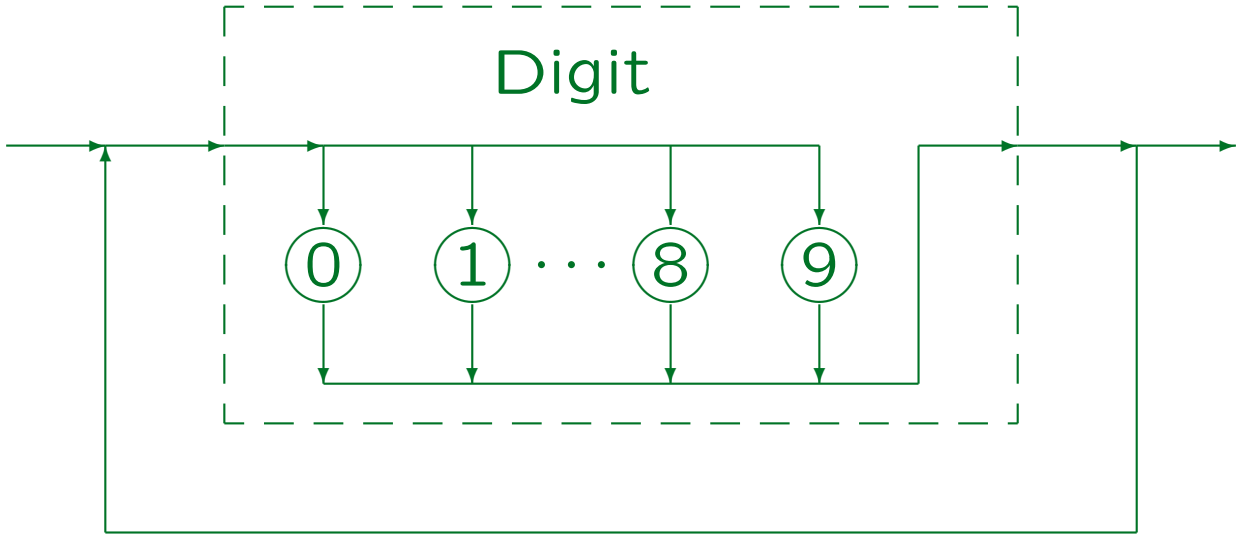
- I.e. a rectangular box stands for a syntactic variable or category (like “verb” and “object”).

Syntax Diagrams (8)

- A box can be replaced by the diagram it stands for.

  A box has one incoming and one outgoing edge, as has the diagram.
  So one can simply plug in the diagram for the box.

- Digit Sequence:



Syntax Diagrams (9)

- One can view the boxes also like procedures:
  - ◇ When a box is entered, one go to the corresponding diagram,
  - ◇ finds a path through it,
  - ◇ and then returns back to the original diagram where the arrow leaves the box.
- When syntax diagrams are used for analyzing input, each such procedure reads part of the input.

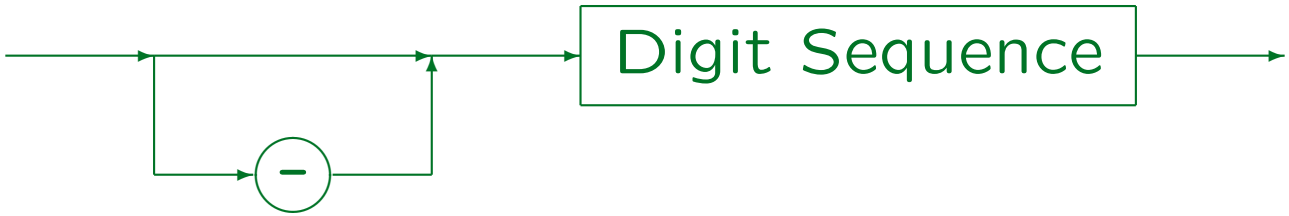
  E.g. when "digit" is called, it expects to see in the input a digit, and it reads this away. Whatever comes after it remains in the input to be analyzed by other procedures.

Syntax Diagrams (10)

- Of course, after some time, one knows what e.g. "Digit" stands for, and does not have to look up the syntax diagram explicitly.
- Each diagram defines a formal language (set of character strings).
- When passing through a box, one can read/print any element of the language defined by the corresponding syntax diagram.

Syntax Diagrams (11)

- Ovals and boxes can be freely mixed in one diagram.
- Integer:



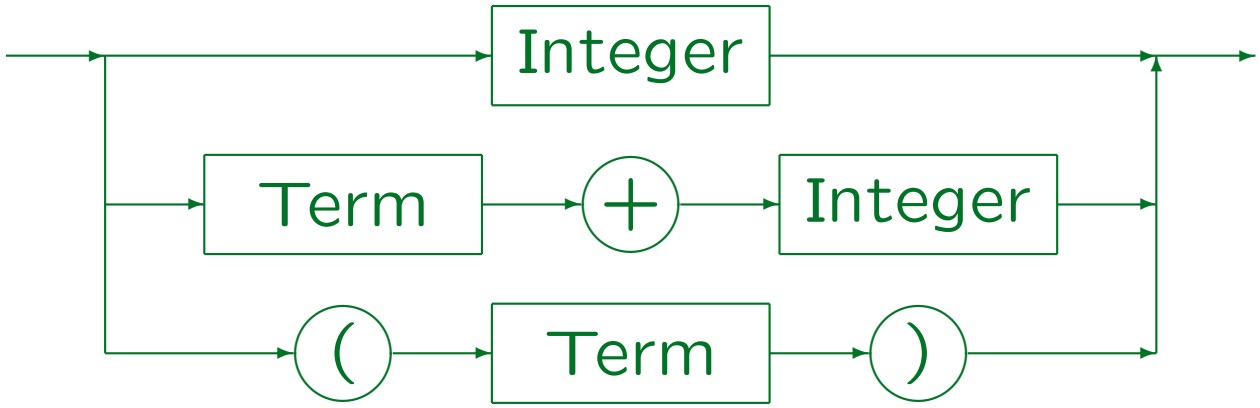
- The diagram "Integer" describes a language containing, e.g., `123`, `-45`, `007`.
- It does not contain, e.g., `+89`, `--5`, `23-42`, `0.56`.

Syntax Diagrams (12)

- Syntax diagrams can be defined recursively, e.g. the diagram for X can contain the a box labelled X.

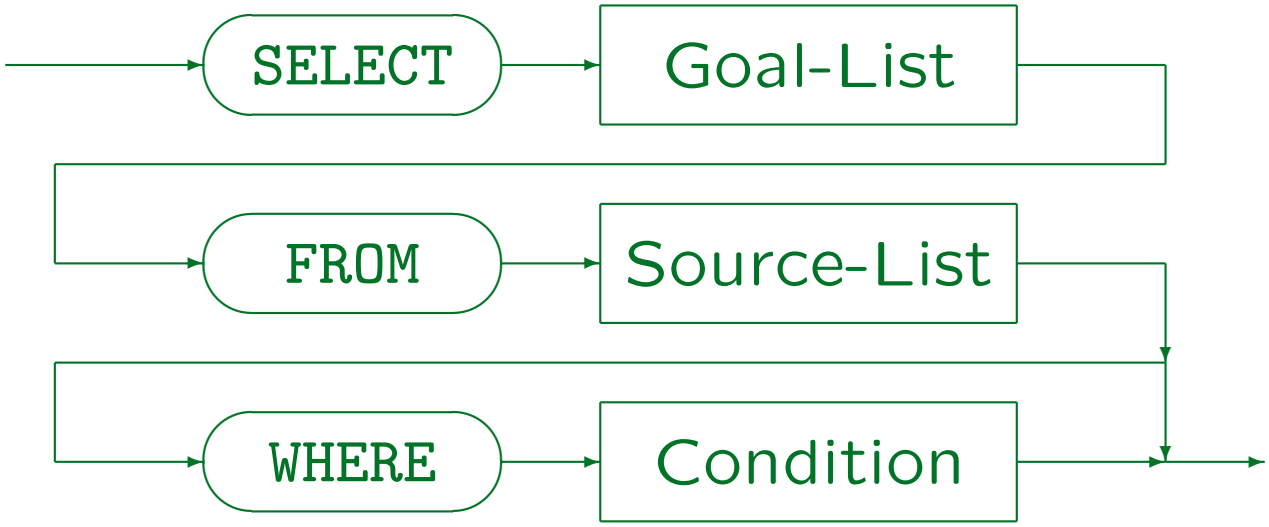
  Mutual recursion is also allowed. With recursion, one cannot expand all boxes beforehand, but one can expand them "on demand" (whenever the box is entered).

- Term:



Syntax Diagrams (13)

- When defining the lexical syntax (possible words or tokens), single characters appear in the ovals.
- Later, words/tokens like "SELECT" are used as the basic building blocks:



Syntax Diagrams (14)

- In the Oracle SQL Manual, boxes are used for literal symbols, and ovals for syntactic variables.

  I followed the original notation, as e.g. in the Pascal reference.

- In the book by Chamerlin about DB2, a more compact notation is used.

  Ovals are used for syntactic variables that are defined by a syntax diagram, uppercase (without any box) is used for key words that must appear as written, lowercase (without box) is used for token types such as "column name". Furthermore, diagrams can extend over multiple lines without explicit backward arrow (Special symbols mark the start and the end of a diagram. If an arrow simply leaves the diagram on the right side, continue in the next line). Finally, a special notation is used for options that can be specified in any order.

Exercise

- Define syntax diagrams for the command language of a text advanture game.
- Typical commands consist of a verb and an object, e.g. "`take lamp`".

  Verbs are e.g. "`take`", "`drop`", "`examine`", "`use`".
  Objects are e.g. "`lamp`", "`sword`", "`rope`".
- One can optionally use an article: "`take the lamp`".
- A verb can be used with multiple objects, separated by "and": "`take the lamp and the rope`".

  This stands for "`take lamp`", "`take rope`".