Appendix F: Syntax Diagrams

Literature:

- 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.
- 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.
- In order to master a language like SQL, you of course need examples, but you also need 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 make your understanding of the language clearer.

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.
- The Oracle SQL Reference Manual as well as many other database books contain syntax diagrams.

Lexical Syntax (1)

- Often, the syntax of languages like SQL or Pascal is specified in two steps.
- The overall syntax is not specified in terms of characters, but in terms of words/tokens/lexical symbols.
- In case of SQL, such words are, e.g.
  - Reserved words / keywords with a specific meaning, e.g. `SELECT`.
  - Identifiers used for table and column names, e.g. `COURSE`.
  - Datatype constants/literals, e.g. `'DBMS' or 123`.
  - Comparison and datatype operators, e.g. `=`, `<=`, `+`, `||`.

Lexical Syntax (2)

- This two-step approach reduces the complexity.

  The overall syntax can assume that such tokens already have been defined and does not have to get down to the level of single characters.
- Simpler formalisms such as regular expressions can be used for defining the lexical syntax (but we use syntax diagrams).

  The overall syntax needs a context-free grammar or syntax graphs (which have the same expressiveness).
- Free-format languages like SQL or Pascal allow white space (blanks, line breaks, etc.) and comments between tokens.

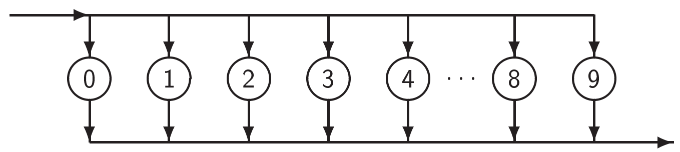
  This is also removed during the lexical analysis phase. The overall syntax analysis sees a sequence of tokens as input.

Overview

1. Lexical Syntax
2. Syntax Diagrams

Syntax Diagrams (1)

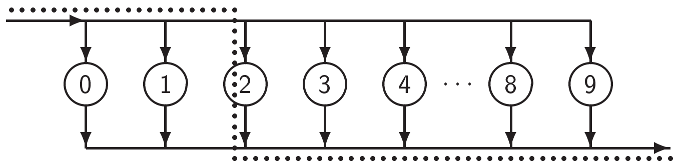
- Digit:



- A syntax diagram consists of:
  - A name, in this case "digit".
  - A start (node), depicted as an incoming arrow.
  - A finish (node), depicted as an arrow leaving the diagram.
  - Oval and rectangular boxes connected by directed edges.

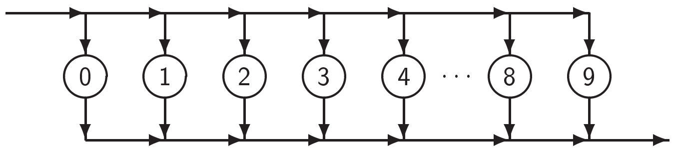
Syntax Diagrams (2)

- The formal language defined by this diagram is very simple, it is the set of digits $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Suppose you want to check whether an input, e.g. "2" really belongs to the language "digit" defined by this diagram.
- Then you must trace a path through the diagram from start to finish such that when you print every symbol in an oval box which you pass you get the given input.



Syntax Diagrams (3)

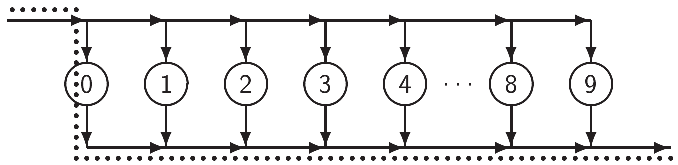
- It is important that you can follow the edges only in the given direction. Every edge has only one possible direction.
- It might be a bit difficult for the beginner to decide on the direction of an edge.
- Here is the same diagram with all directions made explicit:



- Since too many arrow heads look confusing, they are usually not repeated on every segment of an edge.

Syntax Diagrams (4)

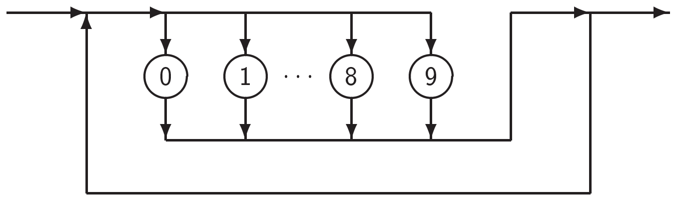
- There are nodes in the diagram where you can choose different paths.
- E.g. after you followed the incoming edge, you can choose to go down and print the digit 0 or go to the right for e.g. 2.



- When you want to analyze a given input for correctness, looking at the next input symbol will often tell you which path to choose. If you cannot find a path, the input is wrong.

Syntax Diagrams (5)

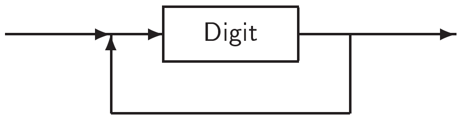
- Syntax diagrams can contain cycles in order to define languages which consist of unlimited sequences of characters/symbols.
- Digit Sequence:



- Exercise: Find a path through this diagram to show that "81" is correct. (You can pass the same edge several times.)

Syntax Diagrams (6)

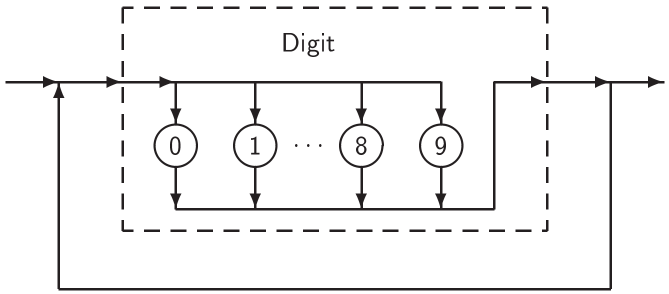
- In order to reduce the complexity, you can use previously defined syntax diagrams as modules in larger ones.
- In this case you use a rectangular box with the name of the syntax diagram in it:



- I.e. a rectangular box stands for a syntactic variable (like "verb" and "object").

Syntax Diagrams (7)

- You can substitute such a box by the diagram it stands for.
  A box has one incoming and one outgoing edge, as has the diagram. So you can simply plug in the diagram for the box.
- Digit Sequence:

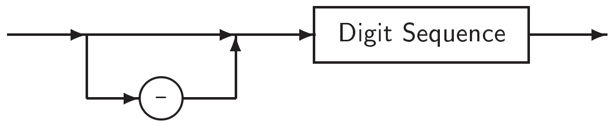


Syntax Diagrams (8)

- Another possibility to view the boxes is like procedures: When you enter a box, you go to the corresponding diagram, find a path through it, and then you go back to the original diagram where the arrow leaves the box.
- When you use syntax diagrams for analyzing input, you can understand each such procedure as reading 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.
- After some time, you know what e.g. "digit" stands for, and you don't have to explicitly look up the syntax diagram.

Syntax Diagrams (9)

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

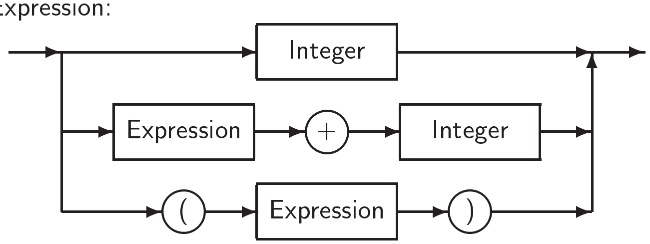


- Integer describes a language containing, e.g., 123, -45, 007. It does not contain, e.g., +89, --5, 23-42, 0.56.
- Exercise: Develop syntax diagrams for identifiers (table and column names). Each identifier must consist of a letter, followed by any sequence of letters and digits and the underscore. E.g. "`abc`", "`x_27`".

Syntax Diagrams (10)

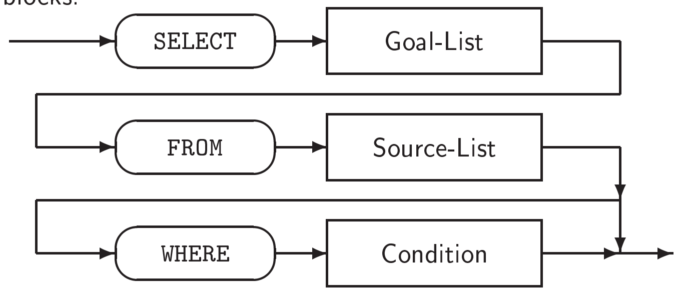
- You can define syntax diagrams recursively, e.g. the diagram for X can contain the a box labelled X.

  Mutual recursion is also allowed.
- So you cannot expand all boxes beforehand, but you can expand them “on demand” (whenever you enter a box).
- Expression:



Syntax Diagrams (11)

- In the Oracle SQL Manual, boxes and ovals are inversed (boxes for literal symbols, ovals for syntactic variables).
- Here, we used only single characters in the ovals/circles. Later, we will use words like “`SELECT`” as the basic building blocks:



Exercise

- Define syntax diagrams for the command language of a text advanture game.
- Directions north, east, south, west can be used as commands to move the player.

  Optionally, one can put a “go” in front, e.g. “go north”.
- 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”.

  You can put an article in front of an object: “take the lamp”.
- You can use a verb with multiple objects, separated by “and”: “take lamp and rope” (short for “take lamp”, “take rope”).