SWE 212 Analysis of Programming Languages

Week 8 Summary of '*A Programming Language'*, Chapter 5 Metaprogramming by V. Iverson

Samyak Jhaveri

APL, named after the book A Programming Language, is an interactive array-oriented language and integrated development environment based on a mathematical notation developed by Kenneth E. Iverson.

It has a combination of unique and relatively uncommon features that appeal to programmers and make it a productive programming language:

- It uses a concise set of symbols and applies functions to entire arrays without using explicit loops.
- It is can express algorithms independently of the machine architecture
- It has just one simple, consistent, and recursive precedence rule: the right argument of a function is the result of the entire expression to its right.
- It facilitates problem solving at a high level of abstraction

This summary of the book 'A Programming Language' presents the information from Chapter 5 of this book - Metaprogramming.

Explaining metaprograms, this chapter elaborates on metaprograms and metaprogramming. The chapter starts with defining the basic glossary terms related to metaprograms

1. A **metaprogram** is defined as a program whose domain is a set of argument programs.
2. **Translator** - metaprogram whose range is also a set of programs.
3. function program - an element of the range of a translator
4. **Analyzer** - metaprogram whose range is not a set of programs. An analyzer produces data that is used to analyze an application program.
5. **Director / supervisor** - A meta program that schedules and directs execution of other programs and metaprograms.

Further, the author dives into the four main types of translators: Compilers, Assemblers, Generators and interpreters.

Compiler - Converts programs given to it in one language (argument language) into the corresponding expression of another language(function language)

Assembler - A special case of compiler, an assembler as the statements of the argument program are considered to be virtually independent and therefore be treated one a t a time; also, they don't need to be analyzed into component statements as they are simple.

Generator - Generator, frequently incorporated into compilers, evaluates a special program that is typically considered to be split into two main parts - skeleton program, that determines the family of potential function programs and the specification, that determines the particular member of the family produced.

Interpreter - An interpreter executes the segment of function program corresponding to a statement of the argument program as and when it is produced, and also selects the statements of the argument program as per the sequence of execution of the function program. So essentially, an interpreter first interprets argument language into function language and then executes it.

The author continues to focus on the analysis of compound statements and their translation between common parentheses notation and Lukasiewicz notation.

The statements of programs are considered to be operators that work on the operands to give a result. An elementary statement is one that has/is an operator that belongs to a set of operators p. A finite program that has such statements is called a program in p. A compound statement in p is one whose operators are not elementary in p but can be expressed as a finite program in p. A metaprogram can be made capable of translating all elementary statements that are compound in p by adding a metaprogram that analyzes compound statements.

The **Lukasiewicz Notation** is essentially the left-list notation for a compound statement. It is also called Polish notation or the parenthesis-free notation.

Equivalent formulas are ones that have the same value for all the possible variable specifications. For instance, a symmetric formula z will be equivalent to all the different singular formulas of z in any reordered situation. Therefore a minimax form of a formula is one in which the maximum suffix dispersion is minimal.r.t the set of all equivalent formulas.

A formula written in complete parenthesis form can be translated to Lukasiewicz Notation. A complete parenthesis notation is one that includes all the possible parenthesis that could be used to express a formula in ordinary parenthesis notation. Then, once the ordinary parenthesis notation is translated into the complete parenthesis notation, it can be translated into Lukasiewicz Notation with the help of a single auxiliary file or stack vector. In a typical program to convert a formula expressed in complete parenthesis notation into its expression in L-notation, the left parenthesis are ignored. However, a similar program can also be written that only considers the left parenthesis and ignores the right.

Each component can be tested for compatibility with its predecessor to partially test for singularity. An ordered pair is considered compatible iff it can occur in some singular formula. In most programs, the formula obtained in L-notation is in reverse order which is the easiest way to evaluate it and best performed. The analysis of the statements may either be done beforehand or may be done in parallel with the synthesis of the statements. Conversely, converting an expression from L-notation to Complete Parenthesis notation is best done by a forward scan. The suffix dispersion of the whole statement is considered to be 1 and as the conversion progresses, the dispersions of successively shorter suffixes are obtained by subtracting (1 - theta(x)) for each component x until the suffix dispersion reaches 0 / null.