ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 2 March 2013

FORMAL SEMANTICS, SYNTAX, PRAGMATICS: AN ESSENCE OF PROGRAMMING LANGUAGE DESIGN

Kebande Rigworo Victor[1], Karani Nyachiro Nelson[2]

[1, 2] Department of Computer Science, Egerton University, Njoro, KENYA.

[1] vickkebande@yahoo.com , [2] rirubius@yahoo.com

ABSTRACT

Programming language design constructs envisages all the essential attributes that contribute to better language, development methods and various ways through which the language design and extensions can be implemented. The syntax constitutes the form, the semantics constitutes the meaning those factors in static and dynamic analysis of programs, and pragmatics shapes the language so that correctness of programs can be achieved. Developing a language with this constructs will aim at achieving systematic approaches and optimization of methodologies used, this will enhance domain specific languages. By use of syntax, semantic and pragmatics a correlation between, expressions, values, type systems will ignite compilers and virtual machines to give a better understanding of the program and performing analysis and program synthesis, the action will help in improving and checking for correctness.

Keywords: Formal semantics, Syntax, Pragmatics, Programming.

INTRODUCTION

The formal development and design of any program needs to be associated with some programming linguistics (slonneger, Barry,1995) the prominence is on the structure of the program, the pattern, the meaning and how the entire program will be implemented within its predefined , user defined structure, and its syntactic and semantic composition. In order for a particular programming language to carry out its expansive task however, design must also attempt to convey a meaning or the intended function. We try to analyze different design constructs which are concerned with semantics [1], this includes the study of meaning in human language, syntax which depicts the rules to be governed , generally mathematical descriptions of syntax use formal grammars [5] they should be precise, concise, clear(sewell,2008)and finally Pragmatics which highlights how the concept can be implemented, description and examples of how the various features of the language are intended to be used[6].Implementation of the language is facilitated by the language processors like compilers, interpreters and Auxiliary tools i.e syntax checkers, debuggers, source editors etc.).We study major issues in this field: (1) the nature of meaning, (2) the contribution of syntactic structure to the interpretation of programming statements, and (3) the possible influence of language design on thought.

Influence of Language Design Concepts and paradigms

We have examined values, type's scope and binding and how they influence language design concepts on a paradigm that contains different dependencies [10] on programming.

Copyright © 2013 SAVAP International
www.savap.org.pk

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 2 March 2013

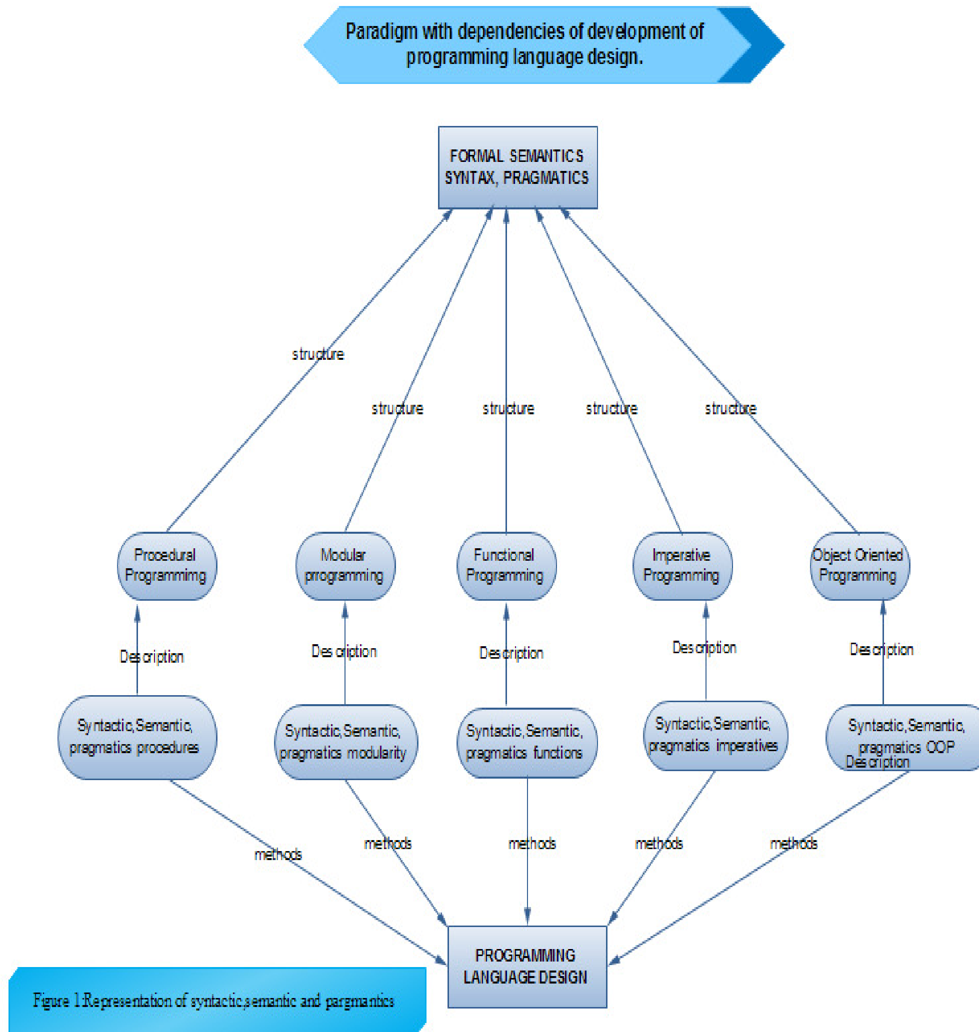


Figure 1. Representation of Synthetic,Semantic and pargmantics

Values and types for Language design

Values can help build the language structure in different aspects, they can be store, can be computed, stored [8] and at large they take part in creation of data structures. To build this design aspects of types[8] which belongs to the same type, the types have to be introduced to follow the meaning [1] of the design structure, this is composed of integers [9] range with precisions(**-2^{15} ~ 2^{15}-1 or -2^{31} ~ 2^{31}-1**) short integer with precisions (**-32768-+32767**),unsigned integers(**0 ~ 65535 or 0 ~ 2^{32}-1**),long integer (**-2^{31} ~ 2^{31}-1**), unsigned[9] long integers(**0 ~ 2^{32}-1**) floating point numbers ,character numbers with precisions **(-128-+127**),unsigned character numbers(**0-255**).Other types that spice the language [8] includes structures, unions and arrays.

(Watt, Findlay, 2004) proposed that values belongs to the same types either though primitive types or composite types which further attracts [2] cardinalities, mappings, powerset and recursive decomposition

Example 1: Composite Type for Cartesian product

A Cartesian product will contain

$A \times B = \{(m, n) \mid m \in A, n \in B\}$ [A intersection B]

Example:

A = {a, b, c} B= {1, 2}

$A \times B$= {(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)}

It depicts that values that m carries belongs to [8] type A and values that n carries belongs to type B.

Example 2: Composite Types For Disjoint Union

A + B= {left x | x ∈ A} ∪ {right x | x ∈ B} [A union B]

Example:

A = {1, 2, 3} B = {3, 4}

A + B = {left 1, left 2, left 3, right 3, right 4}

Example 2 depicts that the type A is combined with all the elements of type B

Binding and scope for Language design.

Developing a program by use of binding provides an association that is created between different [2] entities mostly two entities, this can be a memory location, a name or a function within development of a program; predominantly it occurs between [2] the name being represented and the object. Bindings in development of language needs to incorporate reference environments which factors in many issues like; Binding time which contains design time, implementation time, program writing time, link time, load time and compile time. Binding effects can be felt in the program development either as early or late binding. The effect of this is having the lifetime of the object bound to the language to be developed and invoke all the necessary parameters.

Example 3: Highlight Of C++ Language.

```
thisClass myObject=*new thisClass
{
theirClass myObject=*new thisClass
myObject………………………………..(i)
}
myObject………………………………...(ii)
{
delete myObject.
```

In creation of a lifetime binding *myObject* in (i) is bound to other object but after the new class in (ii) before it is destroyed in *delete myObject.*

SYNTAX

Tree Syntactic Structure and Interpretation.

The tree syntactic representations **[7]** (tree structures) generated are significant not only for determining the form of program logic that is to be used in developing programming structures statements, but also determining their interpretation. In this section, we will consider the Relevance of syntactic structure to three aspects of the programming logic statement interpretation and representation of structural ambiguity.

Syntax Structure Ambiguity

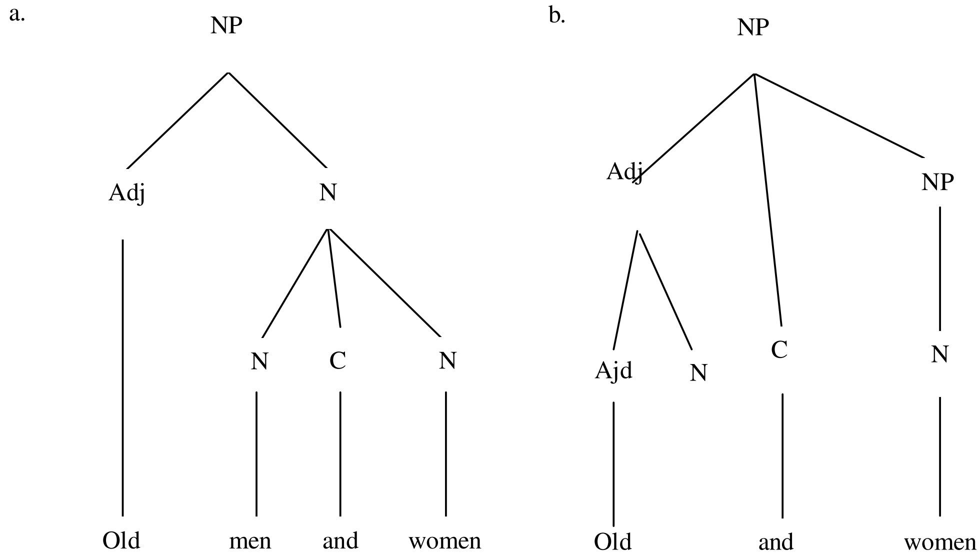
As noted in section 1 on syntax, programming statements are ambiguous because their component structure can **[4]** be arranged into phrases in more than one way; this is called structural ambiguity and is to be distinguished from lexical ambiguity. Structural ambiguity is

Copyright © 2013 SAVAP International
www.savap.org.pk

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 2 March 2013

exemplified by phrases like old men and women, where we take old to be a property of both the men and the women or the men alone. These two interpretations or readings can be linked.

These two interpretations or readings can be coupled to separate syntax structures, as Figure 2 shows. (C = connector.) Figure 2a. Corresponds to the reading in which old modifies men as well as women. This is shown by making the adjective a sister of the category that dominates both nouns. In Figure 2b.



SEMANTICS

Formal semantics is the branch [5] of semantics that studies the logical aspects of meaning**.** Semantic features an approach to meaning, it tries to associate a word's intention with an abstract concept consisting of smaller [1] components called semantic features. This componential analysis is especially effective when it comes to representing similarities and differences among words with related meanings.

Semantics analysis allows us to cluster [15] all entities into innate modules. Hence, man and boy could be grouped together as [+PERSON, +MALE], while man and woman could be put in a class defined by the features [+PERSON, +ADULT].

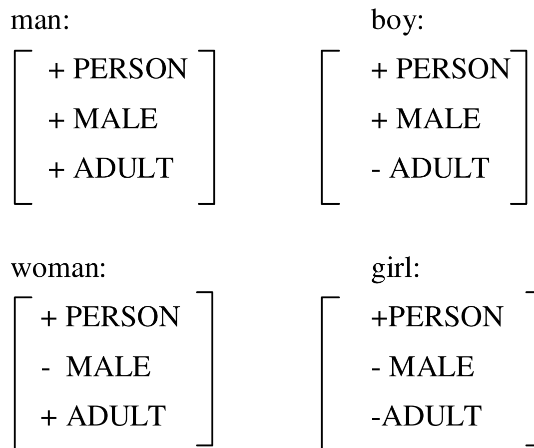


Figure 3. Semantic feature composition for man, woman, boy, girl.

Copyright © 2013 SAVAP International
www.savap.org.pk

Componential analysis gives its most impressive results [3] when applied to sets of statements referring to classes of entities with shared properties. A few simple features will allow us to distinguish among subclasses of people-men, women, boys, and girls.

COPY SEMANTICS VS REFERENCE SEMANTICS

Values and types must always undergo a transition; i.e a program will assign a composite value to a variable of the same type (watt, Findlay 2004).

(Watt,Findlay2004) describes the copy value semantics as the semantics that will copy all the existing [2] components of the composite value into the entire corresponding components of that composite variable and in reference [2] semantics the assignment makes the composite variable contain a pointer or reference to that variable.

Example 4: Case: C++ Copy Semantics

```
Struct year
{
int month,day
};……………………………(x)
Year thisyear={2013};
Year  nextyear;
nextyear=this year;
```

(x) denotes the end of struct declaration for copy semantics,year nextyear contains the year of thisyear

Example 5: Case:C++ Reference Semantics

```
Year* yearP = new year;
year* yearQ = new year;
*yearP = yearA;
yearQ = yearP;
```

In reference semantics two variables *yearP and yearQ contains reference or pointers to date variables.

PRAGMATICS

There exists a major factor in programming language statements interpretation; this involves a broad scope of knowledge that is often called pragmatics. This includes the programmers actions, the way he plans to implement the created programs using syntax and semantics and it addressee's the programmers background and attitude and what he is intended to create, their understanding of the context in which a programming language statement is created from, and their knowledge of the way of in which language is used to communicate information.

CASE: PROGRAMMING LANGUAGES

Procedural programming and Imperative programming

It is a model that is based on moving bits around and changing machine state.

Programming languages based on the imperative paradigm have the basic unit of abstraction is the procedure, whose basic structure is a sequence of statements that are executed in

Copyright © 2013 SAVAP International
www.savap.org.pk

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 2 March 2013

succession, abstracting the way that the program counter is incremented so as to proceed through a series of machine instructions residing in sequential hardware memory cells.

Example 6

```
/* A program in C-like syntax, */
inti=1;
main() {
int y = 5;
printf("%d\n",f(y)+g(y));
printf("%d\n",g(y)+f(y));
}
int f(int x) {
i = i*2;
returni*x;
}
int g(int x) {
returni*x;
}
```

Modular Programming

It basically involves breaking a program down into subcomponents called modules. Each module is composed of an independent or self contained block of instructions. Modules are also referred to as routines, subroutines, or subprograms or procedures. An approach of modular programming ensures that a programmer can valuate the potential of hiding data and procedures to protect against unauthorized access form other modules.

Example 7.Modular Programming

```
Module_1
{
//self contained block of instructions
}
Module_2
{
//self contained block of instructions
}
…
….
Module_n
{
//self contained block of instructions
}
```

Functional Programming

Functional programming t treats computation as the assessment of arithmetic functions and avoids state and inconsistent data. It emphasizes the relevance of functions.It has a first class functions and a higher order functions.

Copyright © 2013 SAVAP International

Example 8. Functional Programming

```
int sum (int i, int j)
{
if (i>j) return 0;
else return i + sum(i + 1, j);
```

Functional version with recursion. Here following the syntax the function will call itself again and again. The function Sum calls itself more than once.

Object oriented programming

Object oriented programming adopt concepts of classes and objects with their syntax. Classes display various attributes (Balagurusamy.2008).Old classes can have the power of extending to other classes.

Example 9: A Class That Wants To Inherit another Class Follows the Following Syntax

```
class old_classname: public/private/protected new_classname
{
//code for inheriting a particular task
}
```

The *old_classname uses a (:)* visibility mode to inherit properties of the new class *new_classname*

Example:Outside Class Definition Syntax

```
return_type classname : : function_name(argument)
{
//code……………...
}
```

Return type carries the type of value to be presented then a defined classname. The scope resolution operator (::) presents a link to the function name, and a choice of arguments.

Objects use dot operator to access the contents of the class via the member functions (E balagurusamy, 2008) defined in the public section.

Example: Object Syntax

```
Classname object;
Object.member function;
```

CONCLUSION

The inclusion of formal semantics to programming language is apprehensive with an extensive array of phenomena which basically includes the nature of meaning, the peak responsibility of syntactic structure in the interpretation of sentences, and the upshot of pragmatics and the programmer's attitude on the understanding of programming language statements. Even though solemn issues and obstacles remains in all this key areas of programming, in current years programmers at least begun to identify the type of relations, mechanisms, and principles involved in the understanding of programming language. These include the concept of extension,word meaning, Command Requirement in the case of syntax interpretation, and thematic role assignment in the case of programming language interpretation.

Copyright © 2013 SAVAP International
www.savap.org.pk

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 2 March 2013

REFERENCES

Slonneger, K. (1995). *Formal syntax and semantics of programming languages*: A laboratory based approach / Kenneth Slonneger, Barry L. Kurtz.

David A, W. (2004). *Programming Language Design Concepts*. Chichester: John Wiley & Sons Ltd.

John, L. (1977). *Semantics. Vols. 1 and 2*. London: Cambridge University Press.

Ravi et al., (2007). Ullman *Compilers, Principles, Techniques, and Tools* (2nd ed.).

Sewell, P. (2008) Semantics of programming languages Available on http://www.cl.cam.ac.uk/teaching/0809/Semantics/notes-mono.pdf

Hennessy, M. (1990). *The Semantics of Programming Languages*. Chichester: John Wiley & Sons Ltd.

Arnold m. Zwicky (1990). Syntactic representation and phonological shapes, simple and composite. *Yearbook of Morphology 1990*

Onur Tolga S¸ehito˘glu . (2008).Programming language values and types. Available on http://ocw.metu.edu.tr/pluginfile.php/2979/mod_resource/content/0/lectures/02-valuesandtypes.pdf

Balagurusamy, E. (2008).Programming in ANSI C 3rd ed. Tata McGraw-Hill

Wilde, N. (1990). Understanding Program dependencies. Available on; http://www.sei.cmu.edu/library/abstracts/reports/90cm026.cfm

Briscoe, T. (2011). Introduction to Formal Semantics for Natural Language. Available on http://www.cl.cam.ac.uk/teaching/1011/L107/semantics.pdf

Schroeder, Mark. (2011). Is semantics formal? Available on: http://www-bcf.usc.edu/~maschroe/research/Schroeder_Is_Semantics_Formal.pdf

Schroeder, M. (2008). *Being For: Evaluating the Semantic Program of Expressivism*. Oxford: Oxford University Press

Müller, P. (1997). Introduction to Object-Oriented Programming Using C++. Available on http://www.tiem.utk.edu/~gross/c++man/

Copyright © 2013 SAVAP International
www.savap.org.pk