A TAXONOMY FOR THE SYNTAX AND SEMANTICS OF PROGRAMS

by

J. A. N. LEE

Language Research Center

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061

June 1975

Report No. CS75013-R

The work reported herein was supported in part by National Science Foundation Grant No. DCR74-18108

Abstract

A comparison is made between the competence and performance models of Chomsky and the two elements of syntax proposed by McCarthy. These concepts are then expanded over utterances in natural languages and programs from programming languages, to cover the syntax of a program, the underlying abstract elements of the program, the meaning of the program, the algorithm which the program represents, the program itself and the statement of the problem being solved.

Keywords:

Syntax, Semantics, Abstract Syntax, Abstract Semantics, Concrete Syntax, Concrete semantics, Abstract Text, Concrete text, natural languages, programming languages.

CR Classifications 4.2, 5.23, 3.42

Introduction

Throughout the study of natural languages, the two models proposed by Chomsky (1) play an important part in distinguishing between those aspects of the generation of sentential forms which are purely mechanical and those which require understanding and knowledge. The competence model of language requires only that the system can generate grammatically acceptable sentences whereas the performance model demands that the utterances be meaningfully acceptable. Whilst competence requires only a knowledge of the language, performance additionally requires a knowledge of the domain of discourse including the applicable relationships between words, such as the applicability of a restricted set of adjectives to a certain class of nouns.

Examining the models of competence and performance with respect to programming languages, it is found that more is assumed than just simply syntax and semantics. In an attempt to understand this issue, the concepts of concrete and abstract syntax, as propounded by McCarthy (2) were examined closely, together with the computer related notion of semantics. It was found that these three elements by themselves formed an incomplete system, the concept of an algorithm being noticeably absent. However, by subdividing the notion of semantics into concrete and abstract forms, this shortcoming was overcome.

Concepts from Natural Languages

Examining closely the knowledge necessary to generate a meaningful utterance, it was determined that four essential elements exist

(1) Knowledge of, and the ability to utilize the syntax of the language of communications, including the dictionary of available words and the rules of formation of sentences (the so-called grammar of a language)

(2) Knowledge of the environment of discourse including information regarding the attributes of elements in the environment* which are named by words in the syntax,

(3) knowledge of, and the ability to synthesize a concept (an abstract idea) which is to be communicated to a recipient, and

(4) knowledge of the acceptably meaningful forms of sentences in the language of communication.

Comparing these elements with Chomsky's models, it is found that the competent system requires knowledge of only the syntactic elements of the language, whereas the performance model requires abilities with respect to all the three other elements. In essence, the grammar of a

---

*Knowledge of the attributes of an object must also imply knowledge of the applicable attributes and consequently the actions which are pertinent.

language is the basis for a competence model and the actual message delivered is a manifestation of the performance model.

Applying McCarthy's definition of a programming language to this situation, he also advocates four elements (3)

> "1) Give the syntax in an abstract analytic form; i.e. for each type of expression name the predicates for telling how it is composed and for getting its parts.
> 2) The abstract syntax makes no commitments about how sums, products. etc., are actually represented by symbolic expressions. To define a concrete syntax one represents the abstract syntactic predicates and functions by functions of strings.
> 3) Next, one defines what information is included in describing the state of the computation; e.g., this includes the value currently assigned to the program variables.
> 4) Then one describes the semantics of the language by defining a function and giving the state that results from applying the program to the initial state."

This 4-tuple differs from our previous list in the following manner. McCarthy assumes that a program already exists and wishes to determine the semantics of the language in terms of the results of executing that program (that is, applying the program to the initial state). One may conclude therefore that McCarthy's model of a language is

deductive rather than generative, and thus the omission of the concept of the task to be fulfilled from his model is intentional. Viewing this model from another stance, if one chooses to make the term semantics synonomous with algorithmic meaning then McCarthy's model has the effect of taking a program and deducing its meaning in terms of knowledge regarding the language and the contents of "the state of computation."

This anomaly can be explained in terms of the 4-tuple proposed herein, and by introducing the concepts of abstract and concrete semantics as shown in Figure 1.

	CONCRETE	ABSTRACT
SYNTAX	A description of the spatial relationships between symbols in the media of communication.	Knowledge regarding the attributes of objects named by the language symbols.
SEMANTICS	The meaning of the actual message	The concept to be transmitted by the message.

Figure 1.

The Application of the Concepts to Programming Languages

In attempting to divide the concept of semantics into two partitions there was a propensity to confuse abstract syntax and abstract semantics as a result of the (now) generally accepted notion of abstract syntax. That is, there exists the idea that abstract syntax refers to the underlying structure of the program which has been generated through the use of the corresponding concrete syntax. This then leads to the added confusion that abstract syntax is related to the control structure of the program.

Firstly it is important to clarify the issue that the term "syntax" is applicable to objects in the domain of discourse of language, including the language itself. To refer to "structure" in this sense implies the static relationships between elements of this set of objects, rather than the dynamic control structure of the program which is evidenced only during its execution.

Hence the abstract syntax of a program is:

(i) a description of the underlying forms of that program, its essential composition being devoid of any restrictions which might be imposed on the concrete text of the program so as to render it readable to a human and processable by a syntactic analyzer,

(ii) knowledge of the domain of applicability of operators and functions to the elements of that domain.

For example, the abstract syntax of a LISP program would show its intrinsic elements without the necessary punctuation (parentheses and spaces) which aid* the readability of the LISP text. Additionally the abstract syntax contains knowledge of the data objects (lists) and the names of applicable functions. Also included must be such items as the number of elements (arguments) which are needed by each function to complete its repertoire of data. The abstract syntax may well contain (or have the ability to check) those truly syntactic properties which are difficult (if not impossible) to specify in terms of context free systems. Ledgard's (4) production systems basically contain two parts which correspond closely to the notions herein of "concrete" and "abstract" syntax. Additionally the production system contains a processor which not only verifies the correctness of the generated statements, but also develops an environment through which the validity of contextually dependent features can be tested. This can be viewed either as a post-production validation system or an intra-production prompting activity.

The important distinction between the environment which Ledgard uses and the state which McCarthy assumes is in the domain of applicability. Production systems operate over a syntactic environment which maps identifiers (language elements) onto a list of attributes (c.f. the above description of abstract syntax) whereas the McCarthy state

---

*(sic!)

is the initial environment of the semantic interpreter. Obviously there exists an overlap between these two notions in the area of abstract syntax.

Once having established the syntactic elements of a program, we can turn our attention to its semantic constituents. Given the knowledge of the language forms (that is, according to Chomsky, to be competent in the language) and cognizance of the domain of applicability of references in the program written in language, the originator of a program next conceives of the problem to be solved and expresses that solution in some algorithmic entity. This algorithm is general in form and is unrelated to the language or the data sets described above. The manifestation of the algorithm as a program need not necessarily be feasible; on the other hand, the same algorithm in the context of differing languages may result in vastly different implementations.

For example, given the basic notion of sorting a vector of values, in the environments of (say) LISP and APL*, not only would widely different texts be produced but also the actual implementations of the same algorithm would result. However, a more detailed sort algorithm, such as a bubble sort, would restrict the implementation to the point where (perhaps) neither LISP nor APL would be fulfilling its greatest potential. Such an evaluation is outside our scope

---

*Assuming the lack of a sort function using the grade up operator.

In natural languages, abstract semantics are readily described as "the concept to be transmitted..."; this ease of definition may be due to a lack of understanding of the psychological factors involved, but such a simple definition is not readily forthcoming in relation to programming languages. At this level of discussion of the domain of programming languages, there exist two entities which are distinguishable; a statement of the problem requiring solution and a statement of the solution to the problem. The latter we recognize as an algorithm, the gradation of detail providing a form of "top down" approach to the description of an algorithm, and to the manifestation of that algorithm in terms of some language. However, the transformation of the statement of the problem into the statement of a solution is not as apparent, as is obvious to most students who are confronted with a statement of the form:

"Prove XX"s n-th theorem"

and are required to develop steps (that is, an algorithm) between the axioms of the environment of discourse and the stated theorem.

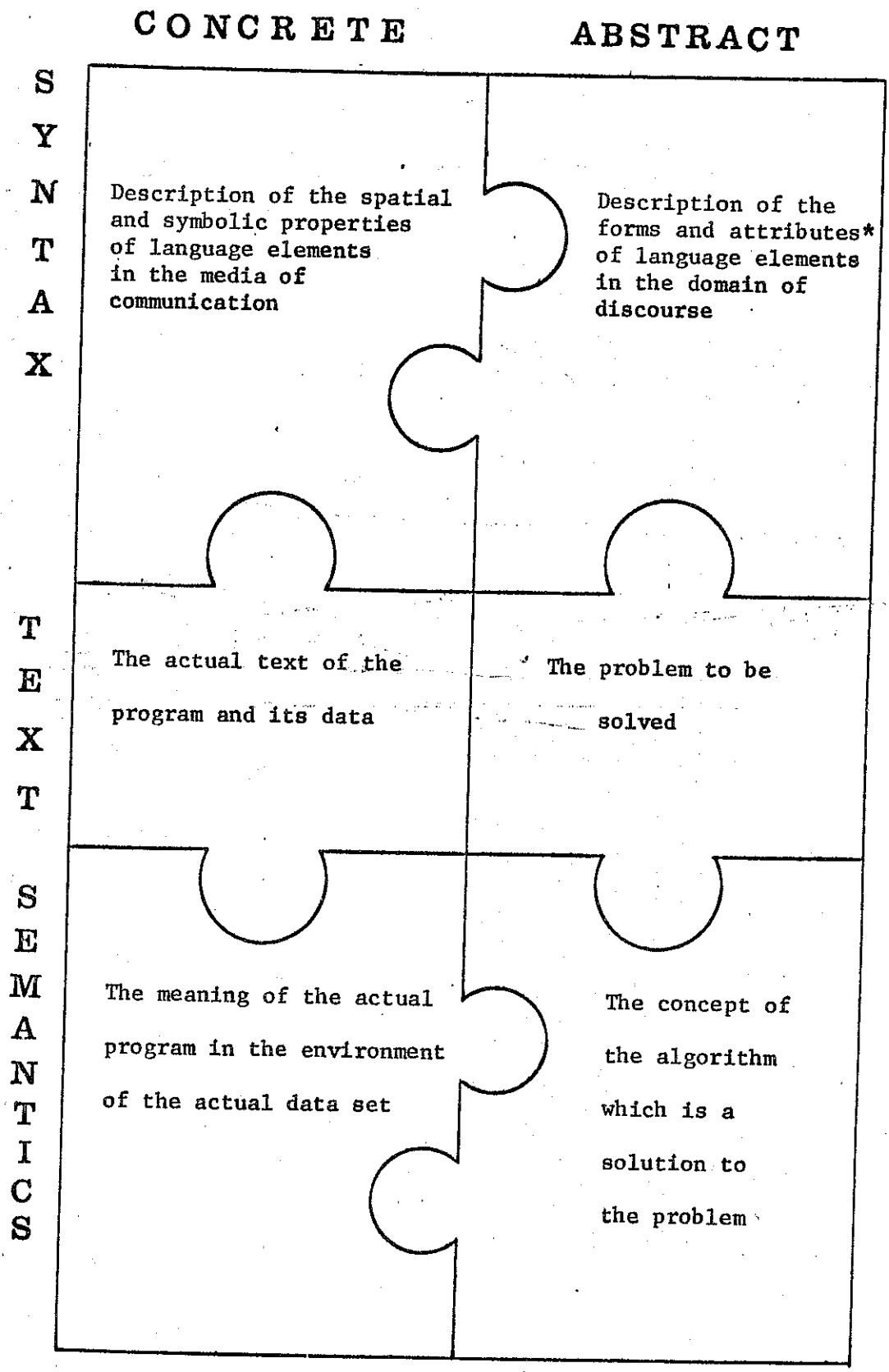

Figure 2.

*Included in the attributes are the sets of applicable functions as well as the data descriptions.

The algorithm has been defined (5) as follows: "A defined process or set of rules that leads and assures development of a desired output from a given input. A sequence of formulas and/or algebraic/logical steps to calculate or determine a given task; process rules."

From the point of view of abstract syntax and semantics, an algorithm can be considered to be merely a control structure over a finite set of the elements (that is, both data and functions) described in the abstract syntax. Whilst the language elements have forms and properties individually, the algorithm expresses the dynamic mating of these elements in order to achieve a solution to the posed problem. It is at this level that the applicability of a certain algorithm can be determined; that is, if the language elements (data objects and operators) do not exist in the abstract syntax (that is, are not described in the abstract syntax) then the algorithm cannot be implemented in that particular language directly. Figure 2 shows the relevant chart for a programming language, this diagram being an extension of figure 1 to utilize the terminology of programming languages and to show the additional concept of texts in this domain. Using these elements of a problem definition and solution in terms of a program, we can further deduce the process of program development as shown in figure 3. This diagram shows clearly the seven stages of program development:

1. The statement of the problem to be solved.
2. The development of a solution.

3. Choosing an applicable language by checking the requirements (functional and structural) of the algorithm with those provided by the language.

4. The transformation of the algorithm, as mapped onto the abstract syntax.

5. The generation of an actual program by the consolidation of the elements of the concrete syntax.

6. The deduction of the meaning of the generated program.

7. The validation of the meaning of the program as being a solution to the stated problem.

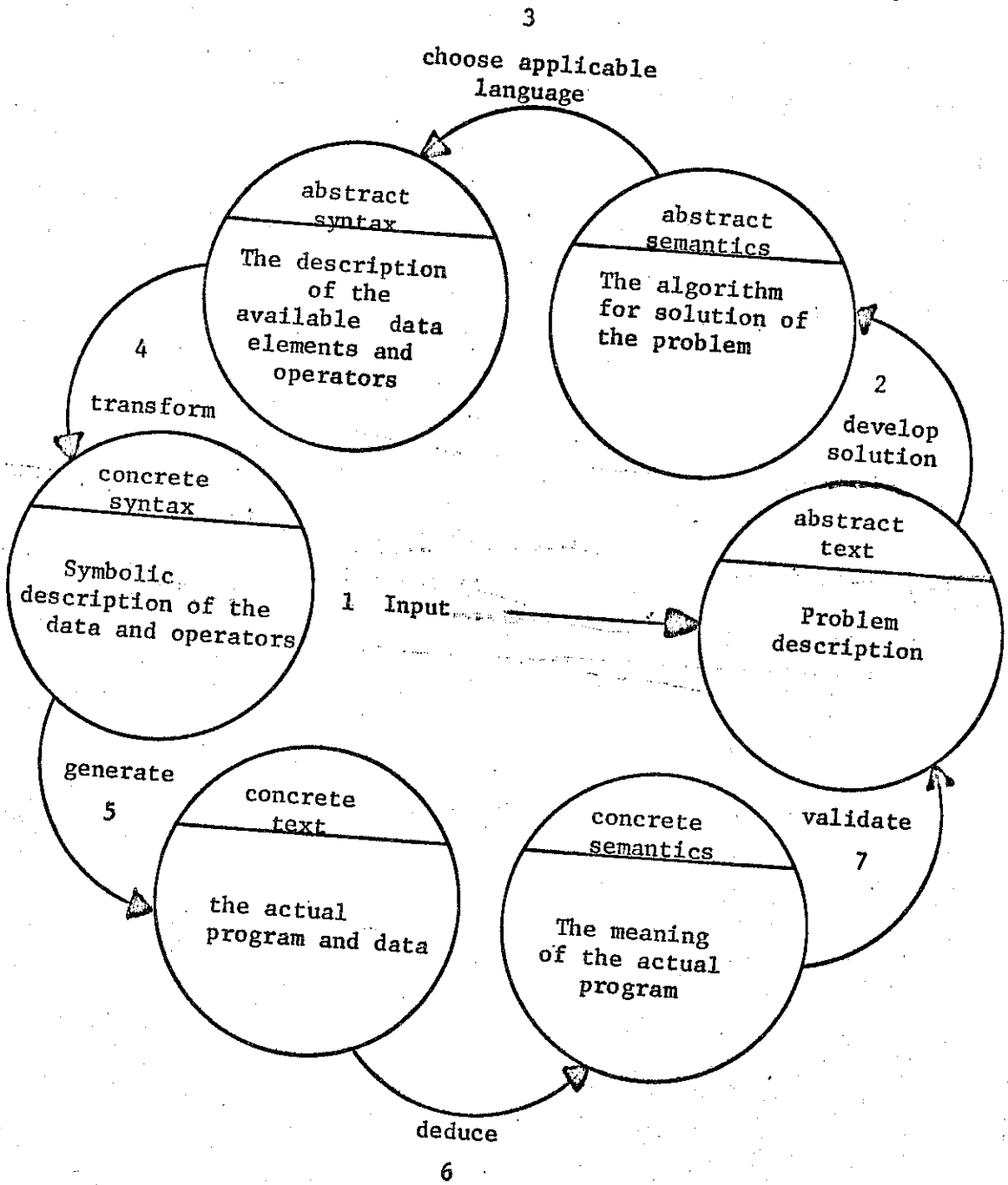
3
choose applicable language
abstract syntax
The description of the available data elements and operators
abstract semantics
The algorithm for solution of the problem
4
transform
2
develop solution
concrete syntax
Symbolic description of the data and operators
1 Input
abstract text
Problem description
generate
5
concrete text
the actual program and data
concrete semantics
The meaning of the actual program
validate
7
deduce
6

Figure 3.

Summary

This paper elucidates the difference between the meaning of a program and its associated data, and the meaning of an algorithm, by extending the concept of abstraction, as applied to syntax, to semantics. Most importantly, a distinction is made between the (so-called) syntax and semantics of a language and the syntax and semantics of a program written in that language. By identifying this distinction the abstractions of syntax and semantics over programs is much clearer, the abstract syntax being the underlying structure of the symbolic form of the program and its data, whilst the abstract semantics is related directly to the abstract meaning of a program - that is, its underlying algorithm. Similarly, the concepts of concrete and abstract text amalgamate with the taxonomy, to show how the actual program and the statement of the problem key into the concepts of concrete and abstract, syntax and semantics.

Acknowledgements

The author wishes to acknowledge the discussion involving T.C.Wesselkamper, Peter Marks and Deborah Macock which originally coined the term "abstract semantics" and for the fruitful later discussions which forged a concrete reality from the notion.

References

(1) Chomsky, N., Aspects of the Theory of Syntax, M.I.T. Press, Cambridge, MA, 1965.

(2) McCarthy, J., A Basis for a Mathematical Theory of Computation, P.Braffort and D.Hirshberg (Eds.), North-Holland Publ. Co., Amsterdam, 1963.

(3) McCarthy,J., A Formal Description of a Subset of ALGOL, in Formal Language Description Languages, IFIP Working Conference, T.B.Steel, Jr.(Ed.), North-Holland Publ. Co., Amsterdam, 1966.

(4) Ledgard, H.F., Production Systems: or Can We Do Better than BNF?, Comm. ACM, Feb. 1974, V17, N2, pp 94-102.

(5) Sippl, C.J., Computer Dictionary and Handbook, H.W.Sams & Co., Indianapolis, 1966.