SLAC (Pub)-198
June 1966

LINGO, A DYNAMIC SYNTAX DIRECTED COMPILER*

by

Edward A. Burfine
Stanford Linear Accelerator Center
Stanford, California

ABSTRACT

LINGO is a table driven syntax directed compiler which translates sentences in a simple precedence phrase structure programing language to symbolic code. The tables which direct translation from language text to symbolic code can be modified during translation. Table modification is directed by meta-language text. Many blocks of language text, where each block is composed of sentences of a different simple precedence phrase structure programing language, can be translated at one time. This is done by changing tables between the translation of blocks of language text.

(Submitted to Communications of the ACM)

*Work supported by the U. S. Atomic Energy Commission.

1. Introduction

Syntax directed compilers translate language text in a programing language to symbolic code or machine code. The programing language may change for each translation, but it is fixed during the translation.

LINGO allows the programing language to change during the translation. This is done by changing the tables which control language text to symbolic code translation. The number of changes and the position in the language text where the changes occur are left to the programer. With a fixed environment, i.e., the memory of the digital computer, these tables define a simple precedence phrase structure programing language. A change in these tables changes the programing language.

Changes in the tables may 1) extend the programing language to include new data spaces or grammatical constructs, 2) remove the definition of certain data spaces or grammatical constructs from the programing language, or 3) reinstate a previously defined programing language.

A programer using LINGO defines the language he wants to use by meta-language text. He then writes sentences in that programing language until a modification of the programing language is necessary for efficient programing. He then modifies the definition of the programing language using meta-language text, and proceeds to write sentences in the new programing language. This modification process can be repeated as often as necessary.

In some ways this work parallels the work of Wijngaarden and deBakker.[1,2,3] It differs in three major respects, however, which are: 1) A more general method of defining languages by syntactic rules and semantic interpretation rules is presented. 2) The meta-language in LINGO is a simple precedence phrase structure programing language which is parsed in the same way as the programing

languages. The semantic interpretation rules for the meta-language are fixed and represent operations on the tables which drive the compiler. 3) The compiler is as efficient as the programer makes it. There is no pyramid of definitions, as in the method of recursive definitions of Wijngaarden[1] and deBakker.[3]

2. Phrase Structure Programing Languages

Using the notation of Wirth and Weber[4], a phrase structure programing language L_p (V, Φ, B, A, Ψ, E) is completely specified by

1. V the vocabulary,
2. Φ the finite set of syntactic rules,
3. B the set of terminal symbols,
4. A the initial non-terminal symbol,
5. Ψ the set of interpretation rules, and
6. E the environment for elements of

The environment E for language text in LINGO is fixed and is the memory of the digital computer. The initial non-terminal symbol A for the language is fixed at <PROGRAM>.

In a context free grammar, the vocabulary V and the set of terminal symbols B may be obtained from Φ. Thus, with E and A fixed, L_p is defined by Φ and Ψ.

Φ may be thought of as a set of syntactic rules in Backus Normal Form and Ψ as a set of partial programs in an assembly language. Let T be a translation which adds to, removes, or changes rules in Φ and adds to, removes or changes the corresponding partial programs in Ψ.

If $T\Phi_o = \Phi_1$, and $T\Psi_o = \Psi_1$ then

$$L_p(V_1, \Phi_1, B_1, A, \Psi_1, E) = L_p\left(V(T\Phi_o), T\Phi_o, B(T\Phi_o), A, T\Psi_o, E\right),$$

where $V(T\Phi_o) = V_1$ and $B(T\Phi_o) = B_1$.

Thus, a new phrase structure programing language may be formed from the old one by performing a transformation on Φ and Ψ. To maintain a precedence phrase structure programing language, the transformation T must be restricted such that the resulting language L_p is a precedence phrase structure programing language. If this restriction is violated, LINGO will terminate translation and give an error message.

The transformation T is represented in LINGO by meta-language text.

3. Translators

LINGO is composed of two translators.

The first translator is used for the translation of language text in a particular programing language to symbolic code. It is driven by a set of tables which can be modified. This translator is written as an ALGOL procedure called COMPILER.

The second translator accepts text which directs the modification of the tables in COMPILER. It is also driven by tables. In this case the tables are fixed and define the meta-language when the environment is the tables in COMPILER. This translator is written as an ALGOL procedure called METACOMPILER.

Some of the tables in COMPILER are modified directly by METACOMPILER, while others are recalculated using an ALGOL procedure SYNTAXPROCESSOR and the algorithm of Floyd.[5] Parsing is directed by the precedence functions of Floyd and is accomplished by the algorithm of Wirth and Weber.[4]

4. Tables

Every table in METACOMPILER is fixed. In COMPILER, every table is treated as data and can be changed during translation. There are six types of tables in LINGO. They are:

1. A symbol table containing the vocabulary V. This table is subdivided into B and V - B.

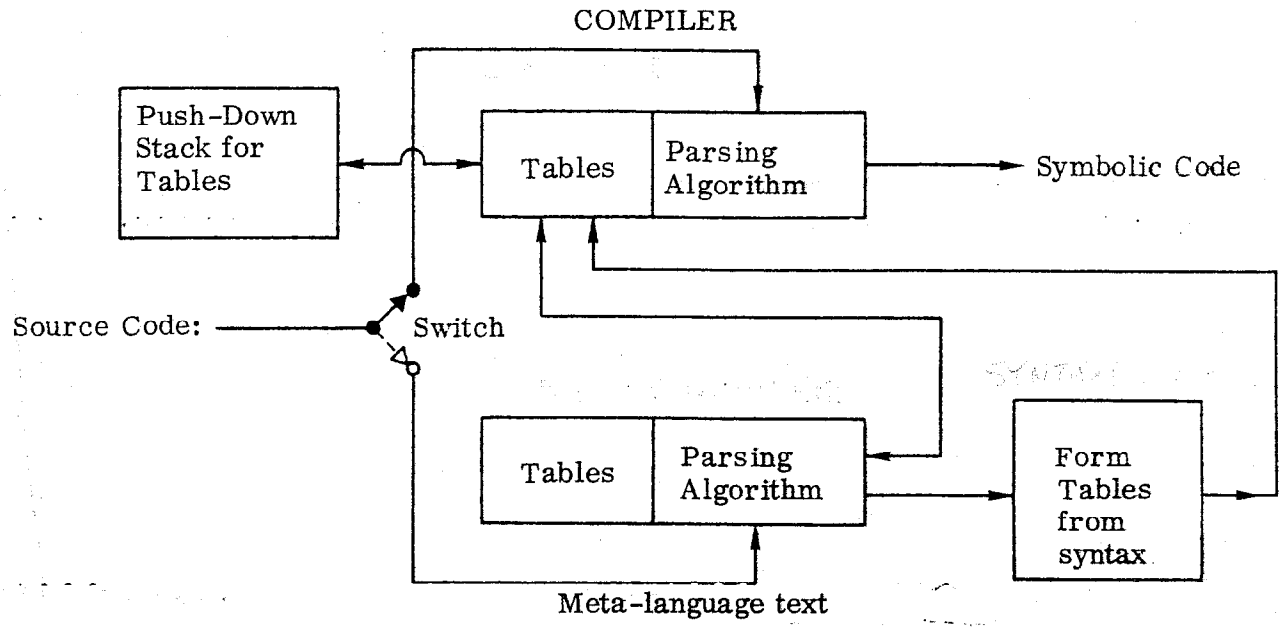

Fig. 1--Translators and Data Flow in LINGO

2. A table of syntactic rules (in numeric form).
3. A master table which relates symbols in the symbol table to syntactic rules.
4. The two precedence functions F and G of Floyd.[5]
5. A table containing syntactic rules of [illegible] Form.
6. A table containing the corresponding semantic interpretation rules of Ψ such that there is a one-to-one correspondence between semantic interpretation rules and syntactic rules.

Tables of type 1 through 4 are found in both COMPILER and METACOMPILER. Tables of type 5 and 6 are only found in COMPILER.

5. Table Modification

Meta-language text translated by METACOMPILER directs modifications in tables of types 5 and 6 by adding, changing, defining, or deleting syntactic rules and their associated semantic interpretation rules. Thus the meta-language

is itself a simple precedence phrase structure programing language where the semantic interpretation rule for each meta-linguistic equation is an operation on tables of type 5 and 6 or a signal to SYNTAXPROCESSOR to form tables of types 1 through 4.

6. Control of the Source Code

The Source Code is composed of both blocks of language text and blocks of meta-language text. At the beginning of the translation, the text is assumed to be language text and is processed by COMPILER.

When the word delimiter SAVE is encountered the tables in COMPILER and the local pushdown stack (state of COMPILER) are saved in a pushdown store and control is passed to METACOMPILER. In Fig. 1 this may be thought of as placing the switch in the down position. This is implimented as a procedure call of METACOMPILER by COMPILER.

METACOMPILER processes one block of meta-language text. This block must be syntactically correct. When parsing of the block is complete, a procedure call of COMPILER by METACOMPILER is executed. In Fig. 1 this is represented by placing the switch in the up position. This recursion allows LINGO to move up and down a tree of languages in a recursive manner. When COMPILER is called, language text in the new language is translated.

A return through the recursion is performed when the word delimiter RETURN is encountered in COMPILER. At this time the language text in the present language is parsed to completion. If no syntax error occurs, the tables and local pushdown stack which define the previous language and the previous state of COMPILER are restored. A procedure return from COMPILER through META-COMPILER to COMPILER is then executed.

7. The Meta-language

The meta-language is based on a structure composed of a single block. A block is composed of sentences of the form

< operator > < syntax > IS < semantic interpretation rule > ;

The block may contain as many sentences as necessary to define the new language. The initial language in LINGO is quite simple and is defined only to give the programer a language to operate on with meta-language text.

It is

< program > : : = < block >

< block > : : = < block body >

< block body > : : = < identifier >

In a production compiler, the initial language would be an ALGOL-like language.

There are four meta-language operators. They are

1. DEFINE — define a new metalinguistic variable with a metalinguistic equation and a semantic interpretation rule,
2. APPEND — add to the definition of a previously defined metalinguistic variable by appending the right-hand side of the given metalinguistic equation to a previous equation and appending the given semantic interpretation rule,
3. CHANGE — delete all previous definitions of the metalinguistic variable and define it as in 1 ,
4. DELETE — delete the definition of the metalinguistic variable from the grammar.

Syntax is defined in a modified Backus Normal Form. Word delimiters and delimiters are enclosed in quotations to differentiate them from metalinguistic delimiters.

Semantic interpretation rules are represented by machine language code. The symbol * , used as an address in the symbolic code, represents the identifier which was last seen by the parsing procedure. Identifiers are stored by the parsing algorithm in a pushdown stack and removed as used.

Several examples of sentences in the meta-language will help to clarify this discussion. Consider the sentences

DEFINE < statement > ← "IDENT". " ← ". < leftpart >. ";" IS STO*;

DEFINE < leftpart > ← < expression > IS EMPTY;

The equivalent Backus Normal Form for the syntactic rules is

< statement > : : = < identifier > ← < leftpart > ;

< leftpart > : : = < expression >

The semantic interpretation rules are

STO * : Store the value of the stack in the location specified by the last identifier seen in the parsing algorithm,

EMPTY : Null operation.

8. An Example of Language Translation in LINGO

In the following example, numbers are used to represent blocks of text to be discussed. In some cases the block is a single word delimiter.

Initially, language text is translated by COMPILER in the basic language of LINGO.

1. The first identifier encountered by COMPILER is SAVE. The tables defining the basic language and the state of COMPILER are stored in the pushdown stack and METACOMPILER is called.
2. The block of meta-language text from START to FINISH is translated by METACOMPILER and a new language is defined. Sentences of this language are ALGOL-like statements with the right part a simple arithmetic expression using only addition.

use the marker to get a better print.

```
AVE
  START
       CHANGE <BLOCK>←<BLOCKBODY> IS EMPTY;
       CHANGE <BLOCKBODY>←<STATEBLOCK> IS EMPTY;
       DEFINE <STATEBLOCK>←<STATEMENT>.<STATEBLOCK> IS EMPTY;
       APPEND <STATEBLOCK>←<STATEMENT> IS EMPTY;
       DEFINE <STATEMENT>←"IDENT"."←".<LEFTPART> ";" IS STK *;
       DEFINE <LEFTPART>←<EXPRESSION> IS EMPTY;
       DEFINE <EXPRESSION>←<EXPRESSION>."+"."IDENT" IS STK *, ADD;
       APPEND <EXPRESSION>←"IDENT" IS STK *
  FINISH
       X  ← Y + Z + W;
        Y        ;
        Z        ;

        W        ;

  SAVE
    (          X          );
       START
            CHANGE <EXPRESSION>←<EXPRESSION>."-".<TERMPART> IS SUB;
            APPEND <EXPRESSION>←<TERMPART> IS EMPTY;
            DEFINE <TERMPART>←<TERM> IS EMPTY;
            DEFINE <TERM>←<TERM>."x".<FACTOR> IS MUL;
            APPEND <TERM>←<FACTOR> IS EMPTY;
            DEFINE <FACTOR>←"IDENT" IS STK *;
            APPEND <FACTOR>←"(".<LEFTPART>.")" IS EMPTY
       FINISH
       X ← X - Y x (Q - R);
        X        ;
        Y        ;
        Q        ;
        R        ;



    RETURN
    (          X          );
       X  ← Y + Z + W;
        Y        ;
        Z        ;

        W        ;

  ETURN
    (  -       X          );
```

Fig. 2--~~Computer Run of LINGO~~

An Example of Dynamic Language Definition and Translation using LINGO

11. Bibliography

1. WIJNGAARDEN, A. VAN. "Generalized Algol," Annual Review in Automatic Programing, Vol. 3, pp. 17-26.
2. WIJNGAARDEN, A. VAN. "Recursive Definition of Syntax and Semantics," IFIP Working Conference on "Formal Language Description Languages," Vienna, 1964.
3. deBAKKER, J. W. "Formal Definition of Algorithmic Languages," Mathematisch Centrum, MR-74, Amsterdam, 1964.
4. WIRTH, NIKLAUS, and WEBER, HELMUT. "Euler: A Generalization of ALGOL and its Formal Definition," Part I and Part II, C.A.C.M., pp. 11-23, January, 1966, and pp. 89-99, February, 1966.
5. FLOYD, ROBERT W. "Syntax Analysis and Operator Precedence," J.A.C.M. Vol. 10, pp. 316-333, July, 1963.