A Language for Structured Programming:LSP(PL/1)

Kyota Aoki,* Koichiro Ochimizu,** Junichi Toyoda* and
Kokichi Tanaka*

1. Introduction

Structured Programming proposed by E.W.Dijikstra(1)(2) is of great advantage to recognize the logical structure of a program. Well structured programs are easily verified to be correct and are easy to be understood. On the other hand, it is also undeniable fact that programmers search for an auxiliary tool effective to implement structured programming.

In this paper, the authors describe a programming language LSP(L) which enables us to make well-structured L programs in a systematic way, where L is the existing programming language such as ALGOL, PL/1 or assembly languages. Thus the form of LSP(L) depends on the programming language L. Here let us mainly discuss LSP(PL/1) and its translator. Programming by LSP(PL/1) is divided into several stages as follows: starting with a given problem, an action† is decomposed into several subactions and each subaction is also decomposed into several subactions. Throughout the decomposition , only three refinement rules are applied in a repeated way. Each of them has one entry and one exit, which gives the frame of the syntax of LSP(PL/1). All of the subactions thus obtained are decomposed into several statements of PL/1††. Finishing this last stage, programming is said to be completed. To develop a program by this method is nothing else but to replace each action by a sequence of subactions in a repeated way. This reminds us that there is a close relation between replacing-actions and rewriting rules of the generative grammar. By formalizing a decomposing process as a rewriting rule, the syntax of LSP(PL/1) is obtained. Thus a LSP(PL/1)

---

† "action" is the similar notion to "processor" described in reference (3), (Chapter 2, Fundamental Notions).

††A PL/1 statement is a subaction which is no more decomposed.

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 15, No. 12 (1974), pp. 955～961.

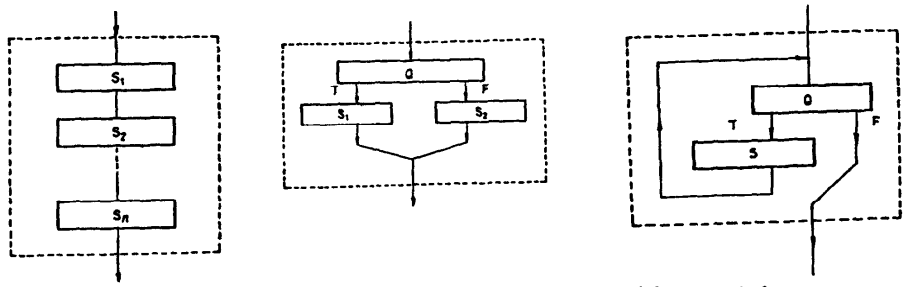
* Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University
** Department of Information Science, Faculty of Engineering, Shizuoka University

source program is assumed to be an ordered sequence of rewriting rules.

A LSP(PL/1) translator does the source program to the well-structured PL/1 program. Since the translator looks the "history of programming" through the sequence of rewrinting rules. The translator automatically arranges the printing format of a program and inserts the comment for each functional block in the object PL/1 program. If there was not the translator, these processes would be overheads of programming to many of programmers. The authors' system undertakes these important but troublesome works.

2. LSP(L)

In this section, the syntax of LSP(L) is given. Three refinement rules construct the core of the syntax of LSP(L), which are shown in fig. 1.



(a) concatenation (b) selection (c) repetition

fig. 1 Three refinement rules

LSP(L) is generated by a grammar for LSP(L), that is, GSP(L) defined as follows: GSP(L) = (V_N, V_T, P, S), where V_N is a set of nonterminal symbols, V_N is a set of terminal symbols, P is a set of rewriting rules and S is a start symbol. They are defined as follows: V_N = {<action>, <action name>, <action 1>, <action 2>, <action 3>}, V_T = {SEL, WHITE, ,, ;, :=, @, [,]} U {<permissible statement in L>} U {<string over the data set of L>}, S = <program>. P is a set of rewriting rules such as: (1) <program> → <action>, (2) <action> → <action> <action>, (3) <action> → <action name> : = [<action 1>]<action>, (4) <action> → <action name> : = SEL[<action 3>, <action 2>, <action 2>]<action>, (5) <action> → <action name> : = WHILE[<action 3>, <action 2>]<action>, (6) <action> → ε (ε is a empty string), (7) <action 1> → <action 1>;<action 1>, (8) <action 1> → <action 2>, (9) <action 2> → <action name>, (10) <action 2> → <permissible statement in L>, (11) <action 3> → <logical expression in L>, (12) <action name> → <string over the data set of L>.

Obviously, rule(3) corresponds to concatenation, (4) does to selection and (5)

does to repetition.

3. LSP(PL/1)

Let us now define LSP(PL/1). The syntax diagrams of LSP(PL/1) based on the definition are shown in fig. 2. The syntax and the semantics of LSP(PL/1) are explained according to fig. 2 in the following.

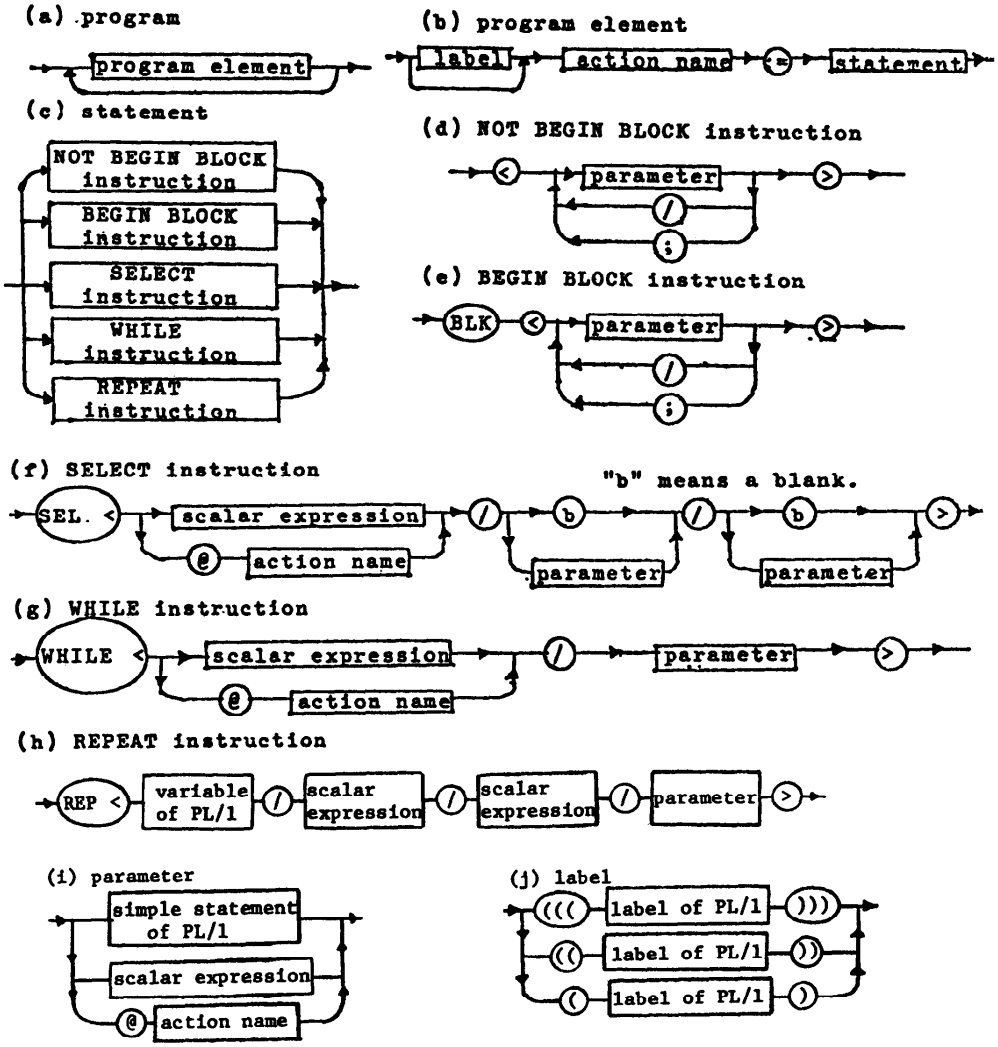


fig. 2 Syntax diagrams of LSP(PL/1)

A LSP(PL/1) program is defined as a sequence of program elements(fig. 2-a). Each program element means that some action designated by "action name" is decomposed into a sequence of subactions represented by the statement(fig. 2-b). A statement is one of five instructions(fig. 2-c), and the reason why five instructions are

provided for LSP(PL/1) is as follows. LSP(PL/1) requires at least three kinds of instructions corresponding to three refinement rules described in fig. 1. There are two types to realize concatenation: one is "NOT BEGIN BLOCK instruction"(fig. 2-d) and the other is "BEGIN BLOCK instruction"(fig. 2-e). The latter is represented as a BEGIN block structure in the object PL/1 program, but the former is merely represented as a DO group of a sequence of PL/1 statements in the object PL/1 program. In fig.2-d and 2-e, a sequence of parameters enclosed by "<" and ">" corresponds to a sequence of subactions decomposed. A parameter is a PL/1 statement or an action name following "@" which means that more decomposition required (fig. 2-i). A delimeter between parameters is ";" or "/", where "/" means CRLF. There is one type to realize selection. It is called "SELECT instruction" (fig. 2-f) and, in the object PL/1 program, represented as an IF compound statement. In fig. 2-f, the first "parameter" denotes a THEN clause and the second denotes an ELSE clause. There are two types to realize a repetition. One is "WHILE instruction" (fig. 2-g) and the other is "REPEAT instruction" (fig. 2-h). In fig. 2-g, "parameter" denotes a DO body. A WHILE instruction is represented as a DO WHILE statement in an object PL/1 program. A REPEAT instruction is represented as a DO loop in an object PL/1 program. Each program element is able to have a label. Labels enclosed by "(((" and ")))" are interpreted to be external procedure names. Labels enclosed by "((" and "))" are interpreted to be internal procedure names. Labels enclosed by "(" and ")" are interpreted to be other labels. More detailed explanation about the syntax of LSP(PL/1) is abbreviated due to the space limitation. The authors' LSP(PL/1) translator is written by PL/1 (program size is about 2000 steps) and it is implemented on the FACOM 230-45S. Detailed explanations about the translator are also abbreviated.

4. An example of programming

A simple example of LSP(PL/1) programming is shown. A programming example is to make "a magic square of odd order". A LSP(PL/1) program is shown in fig. 3. An output of the translator is shown in fig. 4. The authors design the program as shown in fig. 5. In fig. 4, action names in LSP(PL/1) program are utilized as the comments in an object PL/1 program. Proper indentations are achieved according to the logical structure of the program.

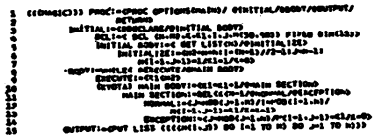


fig. 3 A LSP(PL/I) source program

5. Conclusion

The LISP(PL/1) proposed here encourages programmers to make programs by the use of structured programming technique. Moreover, a programming system on LSP(PL/1) is able to make programmers free from the tedious work in structured programming. As a result, programmers can devote themselves to the problem solving.

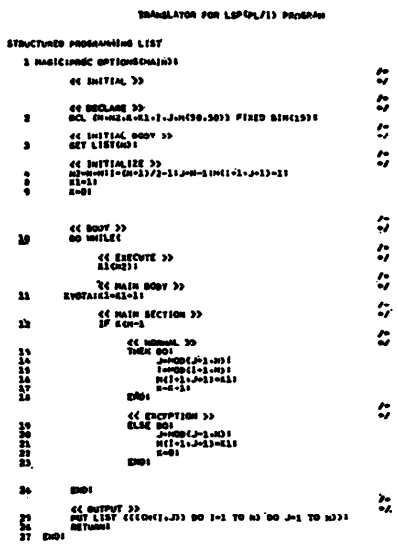


fig. 4 An object PL/I program for fig. 3

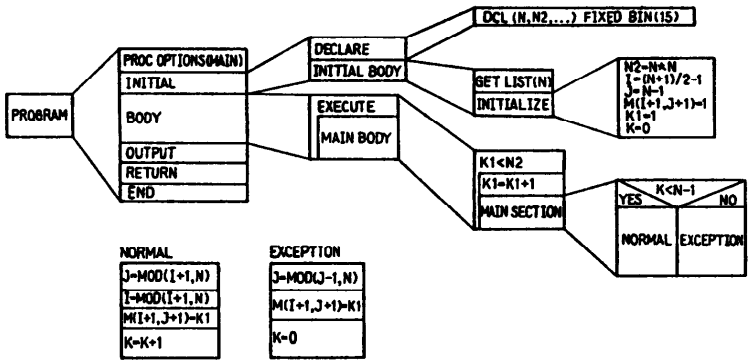


fig. 5 The refinement process

References

(1) O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare: Structured Programming, Academic press, London, 1972.

(2) E. W. Dijkstra: GO TO Statement Considered Harmful (Letters to the Editor), Comm. ACM, Vol. 11-3, pp. 147 - 148, 1968.

(3) Niklaus Wirth: Systematic Programming: An Introduction, Pretice-Hall, Inc., Englewood Cliffs, N. J., 1973