'91 韓國自動制御學術會議論文集 1991. 10.22~24

PLC용 DFLSP의 모델링 및 분석에 관한 연구

노 갑선, 박 재현, 권 욱현

서울대학교 공과대학 제어계측공학과

A Study on the Modeling and Analysis of DFLSP of PLC

Gab Seon Rho, Jaehyun Park, and Wook Hyun Kwon

Seoul National University

ABSTRACT

The mathematical modeling and analysis results of a dataflow logic solving processor(DFLSP) for programmable logic controller(PLC) are proposed in this paper. The logic program language is formalized using a dataflow graph model. From this dataflow graph, the instruction precedence relationship, and deadlock problems, which are major properties of a logic program, are described.

1. Introduction

The programmable logic controller(PLC) is widely used in factories to replace electro-mechanical relays and its application can be found in many areas, such as chemical industries, manufacturing, mining, etc. Nowadays, factory automation is very complex and requires many control and sensor points. Thus larger PLC's which can handle more control and sensor points are needed. To achieve a fast computing performance, a special architecture logic solving processor(LSP) is used in many large PLC's. In the recent decade, efforts to make a fast LSP were performed by several major companies and universities, and some of them provide a good architecture of LSP for large PLC's[1-3].

Our previous works are concentrated on the development of a new architecture of LSP based on dataflow computing. This data flow logic solving processor(DFLSP) shows not only good computing capability in solving control programs but also with data synchronization and dynamic load balancing capabilities that are very difficult to perform in other multiprocessor systems[4]. Like all of the previous researches focused on the development of new hardware architecture of LSP, the software of LSP is treated briefly.

The *ladder diagram language*, a widely used PLC language, is modeled using *dataflow graph*, which is the basic language of DFLSP. Using the modeled dataflow graph, properties which are sometimes able to occur are analyzed.

2. Logic Programming based on Dataflow

A logic language used for programming PLC is described briefly and its modeling is proposed in this section. Since a programmable controller was originally developed as a sequential control device to replace electro-mechanical relays in factories, the language for PLC should be well fitted for a sequential control problem. Currently, several kinds of programming language for PLC are used. Among them, *ladder language* is one of the most widely used. Ladder language is represented by a diagram of matrix type symbolic arrays. The PLC instruction is able

to be divided into two large groups, one is the logical instruction and the other is box instruction. The PLC program consists of a mixture of these two type of instructions, which occupy one node in the ladder language matrix. The instructions in the ladder language is connected by a line called a branch. The typical ladder language is shown in Fig. 1.

Logic instructions include AND, NOT-AND, OR, and STORE instruction and box instructions include more complex instructions such as COUNTER, TIMER, SHIFT REGISTER, DRUM. In a real application program, most of the instructions are logic instructions. This means that most of logic program solving time is spent to solve logic instructions and this time can be reduced significantly by LSP.

The ladder program is a matrix type, graphical language and each instructions are solved sequentially from left to right. The instructions for ladder program such as in Fig.2-(a), are solved in (X1)-(X2)-(X3)-(Y1)-(Y2) order. The ladder program is able to be converted into a *2 input directed graph*. The 2 input directed graph consists of two nodes connected by a directed branch and each node has two inputs. With this representation, the ladder program in Fig. 2-a is transformed into the directed graph in Fig. 2-b. This directed graph the is the basic language used for a dataflow computer and is called *dataflow graph*.

As a basic language of the dataflow computer, dataflow graph has two types of nodes called *links* and *actor*. Actors describe operations, while links receive a token from a single actor and transmits values to one or more actors by way of arcs. Basically, in dataflow computer, nodes(actors and links) are enabled for execution when all input arcs contain tokens and no output arcs contain tokens.

3. Architecture of DFLSP

Before describing logic programs of dataflow LSP(DFLSP), the architecture of DFLSP should be described briefly. DFLSP consists of five main parts: token queue, distribution network, LSU, ASU, and re-matching unit. In a dataflow machine, peak performance can be obtained by using all possible sequences concurrently. However it is impossible to make so many LSU's, hence, only some tokens are enabled at the same time and the rest of them are stored in the token queue. The distribution network distributes tokens from the token queue to the LSU or ASU in idle state. LSU's solve all logic instructions concurrently and ASU solves block type instructions. If LSU meets an output reference instruction of current scan cycle or a store instruction, LSU sends that token to the re-matching unit. The re-matching unit matches the store instruction with the output reference instruction. When re-matching occurs, it sends tokens to the token queue in order to be solved.

The performance of DFLSP evaluated by computer simulation with odeling and simulation done by SIMAN[8], and simulated with varying portions of store instruction and number of LSU's can be calculated. From a simulation results, the DFLSP solves 1K of instructions in 200 μsec with one LSU, and 8 LSU's in 80 μsec with. The program solving time is hardly influenced by the percentage of store instruction.

4. Modeling of DFLSP Language

4.1 Basic Definitions

A dataflow graph is directed graph composed of nodes(actors and links) and edges as in Dennis' model[5,7].

$G = < A \cup L, E >$

where

$A = \{a_1, a_2, \cdots, a_n\}$ *is set of operator*

$L = \{l_1, l_2, \cdots, l_m\}$ *is set of links*

$E \subseteq (A \times L) \cup (L \times A)$ *is set of edges.*

Actors represent logical operators and links are treated as place holders of data values(tokens) as they flow from

actors to actors. Edges are the channels of communication. Starting node set S, initially enabled nodes are defined as

$$S = \{l \in L \mid (a, l) \notin E, \forall a \in A\} \quad (2)$$

And similarly, the terminating set T is a subset of links which have tokens after all instructions are solved.

$$T = \{l \in L \mid (l, a) \notin E, \forall a \in A\} \quad (3)$$

The set of input links to an actor a and the output links from an actor a are denoted by I(a) and O(a).

$$I(a) = \{l \in L \mid (l, a) \in E\} \quad (4)$$

$$O(a) = \{l \in L \mid (a, l) \in E\} \quad (5)$$

A marking is defined as a mapping

$$M : L \to \{0, 1\}. \quad (6)$$

A link is said to contain a token in a marking M if M(l) = 1. The input firing semantic set F_i and the output firing semantic set F_o are defined as

$$F_i(a, M) \subseteq I(a)$$
$$F_o(a, M) \subseteq O(a). \quad (7)$$

The input firing semantic set refers to the subset of input links that must contain tokens to enable the actor, and the output firing semantic set refers to the subset of links that receive tokens when the actor is fired.

A firing is a partial mapping from markings to markings. The firing rule in a dataflow graph is that an actor a is firable at a marking M if the following conditions hold.

$$\begin{cases} M(l) = 1 & \text{for all } l \in F_i(a, M) \\ M(l) = 0 & \text{for all } l \in F_o(a, M). \end{cases} \quad (8)$$

Thus, a new marking M' resulting from the firing of an actor a at marking M can be derived as follows.

$$M'(l) = \begin{cases} 0 & \text{if } l \in F_i(a, M) \wedge \\ & \quad l \notin F_o(a, M) \\ 1 & \text{if } l \in F_o(a, M) \\ M(l) & \text{otherwise,} \end{cases} \quad (9)$$

A solving sequence σ is a sequence of operators in the order in which the operators are solved. When operators can be solved concurrently, the order is arbitrary. An operator a is said to belong to σ if a is fired in the firing sequence σ. We can define $I(\sigma)$ like I(a) and $O(\sigma)$ like O(a) as follows.

$$I(\sigma) = \{ l \in L \mid l \in I(a), \text{ where } a \in \sigma \}$$
$$O(\sigma) = \{ l \in L \mid l \in O(a), \text{ where } a \in \sigma \} \quad (10)$$

We say M leads to M' via σ if M' is the new marking that is derived from the marking M when the actors in the firing sequence σ are fired. This is denoted by $M \to M'$. The set of markings generated by a firing sequence σ will be known as marking set M^σ.

A forward marking class $M^{\sigma+}$ of a marking M is the set of

$$M^\sigma = \{M' \mid M \xrightarrow{\Sigma} M' \text{ for any subsequence } \Sigma \text{ that is a prefix of } \sigma\}. \quad (11)$$

markings which can be derived from M via some firing sequence.

$$M^{\sigma+} = \{ M' \mid M \xrightarrow{\sigma} M' \text{ for firing sequence } \sigma \} \quad (12)$$

Finally, we can define input semantic set and output semantic set of forward marking class $M^{\sigma+}$ like $I(\sigma)$ as follows.

$$I(M^{\sigma+}) = \{l \in L \mid l \in F_i(a, M^+) \text{ where } a \in \sigma\}$$
$$O(M^{\sigma+}) = \{l \in L \mid l \in F_o(a, M^+) \text{ where } a \in \sigma\} \quad (13)$$

4.2 Language formalization of DFLSP

Before the LSP starts solving a logic program, IOP scans all input sensor points and creates tokens for corresponding links and power nodes. As a result, the starting set S consists of power nodes P and input sensor nodes X

$$S = P \cup X. \quad (14)$$

The logic program is solved from power nodes and sequences from all of power nodes produce possible sequence set Σ,

$$\Sigma = \{ \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n \} \quad where\ n = |P|. \quad (15)$$

At the starting time of DFLSP, all links in P are stored in the token queue and only m(number of LSP) tokens are distributed. Among the sequence in Σ, only m sequences are enabled at the same time.

The logic operator in ladder language can be interpreted as an actor and branch between two actors that can be interpreted as an link. Using this interpretation, we can get all of the sets defined in the previous section. For the example in Fig. 2, the operator set A = $\{a_1, a_2, a_3, a_4\}$, link set L = $\{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9, l_{10}\}$, starting node set S = $\{l_1, l_2, l_3, l_4\}$, and terminal node set T = $\{l_{10}\}$ are acquired. There can be two possible firing sequences σ_1, σ_2, $\sigma_1 = \{a_1, a_2, a_4, l_5\}$ and $\sigma_2 = \{a_3\}$.

5. Properties of DFLSP Language

5.1 Dependence and precedence

We can define reference set R as

$$R(\sigma) = I(\sigma) - O(\sigma). \quad (16)$$

<Theorem 1> σ_2 depends on σ_1 if and only if

$$O(\sigma_1) \cap R(\sigma_2) \neq \varnothing. \quad (17)$$

(Proof)

(1) If σ_2 is dependent on σ_1, then at least one of the actors of σ_2 refers one ore more outputs of actors in σ_1. this means

$$\begin{aligned} & O(\sigma_1) \cap I(\sigma_2) \neq \varnothing \\ & O(\sigma_1) \cap I(\sigma_2) - O(\sigma_1) \cap O(\sigma_2) \neq \varnothing \\ & O(\sigma_1) \cap \{I(\sigma_2) - O(\sigma_2)\} \neq \varnothing \\ & O(\sigma_1) \cap R(\sigma_2) \neq \varnothing \end{aligned} \quad (18)$$

(2)

$$\begin{aligned} & if \quad O(\sigma_1) \cap R(\sigma_2) \neq \varnothing \\ & then \quad O(\sigma_1) \cap \{I(\sigma_2) - O(\sigma_2)\} \neq \varnothing \\ & \qquad O(\sigma_1) \cap I(\sigma_2) \neq \varnothing \end{aligned} \quad (19)$$

This means no output of actors in σ_1 is referred by actors in σ_2.

Q.E.D.

<Remark> If σ_2 is dependent on σ_1 then σ_1 should be solved prior to σ_2, (i.e, σ_1 should precede σ_2).

5.2 Deadlock

If σ_1 is dependent on σ_2 and σ_2 is dependent on σ_1, two sequences σ_1 and σ_2 are in a deadlock state and cannot be solved. So The deadlock state in the PLC program is described as follows.

$$O(\sigma_1) \cap R(\sigma_2) \neq \varnothing \ \wedge \ O(\sigma_2) \cap R(\sigma_1) \neq \varnothing. \ \blacksquare$$

6. Examples of Dataflow graph for DFLSP

Examples of dataflow graph is illustrated in this section. By these example, we can find an algorithm to check the deadlock problem and the off-line sequence scheduling policy. The ladder language in Fig. 3-a can be translated into a dataflow graph as shown in Fig 3-b. In this example, we can find the following sets easily.

$$\begin{aligned} & Power\ node\ set \quad P = \{l_0, l_8, l_{14}\} \\ & X\ ref.\ set \quad X = \{l_5, l_6, l_7, l_{12}, l_{20}\} \\ & Y\ ref.\ set \quad Y = \{l_4, l_{18}, l_{19}, l_{23}\} \end{aligned}$$

Operation sequences are,

$$\begin{cases} \sigma_1 = \{a_1, a_2, a_3, a_4\} \\ \sigma_2 = \{a_5, a_6, a_7\} \\ \sigma_3 = \{a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}\} \end{cases}$$

Input link set and output link set are,

$$\begin{cases} I(\sigma_1) = \{l_0, l_1, l_2, l_3, l_5, l_6, l_7\} \\ I(\sigma_2) = \{l_4, l_8, l_9, l_{10}, l_{12}\} \\ I(\sigma_3) = \{l_4, l_{11}, l_{14}, l_{15}, l_{16}, l_{17}, l_{18}, l_{20}\} \end{cases}$$

$$\begin{cases} O(\sigma_1) = \{l_1, l_2, l_3, l_4\} \\ O(\sigma_2) = \{l_9, l_{10}, l_{11}\} \\ O(\sigma_3) = \{l_{15}, l_{16}, l_{17}, l_{18}, l_{19}, l_{23}, l_{24}\} \end{cases}$$

We find reference set as,

From these,

Another example in Fig. 4 shows a deadlock situation.

$$\begin{cases} R(\sigma_1) = I(\sigma_1) - O(\sigma_1) = \{l_0, l_5, l_6, l_7\} \\ R(\sigma_2) = I(\sigma_2) - O(\sigma_2) = \{l_4, l_8, l_{12}\} \\ R(\sigma_3) = I(\sigma_3) - O(\sigma_3) = \{l_4, l_{11}, l_{14}, l_{20}\} \end{cases}$$

$$O(\sigma_1) \cap R(\sigma_2) = \{l_4\} \neq \varnothing$$

$$O(\sigma_2) \cap R(\sigma_3) = \{l_{11}\} \neq \varnothing$$

$$\therefore \sigma_1 \succ \sigma_2 \succ \sigma_3$$

In this example, the two possible sequences are

$$\sigma_1 = \{a_1, a_2, a_3\}$$

$$\sigma_2 = \{a_4, a_5, a_6\}$$

the I, O, R sets are given as,

$$I(\sigma_1) = \{l_0, l_1, l_2, l_4, l_7\}$$

$$I(\sigma_2) = \{l_3, l_5, l_6, l_7, l_8\}$$

$$O(\sigma_1) = \{l_1, l_2, l_3\}$$

$$O(\sigma_2) = \{l_6, l_7, l_9\}$$

$$R(\sigma_1) = \{l_0, l_4, l_7\}$$

$$R(\sigma_3) = \{l_3, l_5, l_8\}$$

From these, we get

$$O(\sigma_1) \cap R(\sigma_2) = \{l_3\} \neq \varnothing$$

$$O(\sigma_2) \cap R(\sigma_1) = \{l_7\} \neq \varnothing$$

$$\therefore \sigma_1 \succ \sigma_2 \ \wedge \ \sigma_2 \succ \sigma_1.$$

This is a deadlock state.

7. Conclusion

The architecture of a dataflow logic solving processor(DFPLC) for programmable logic controller(PLC) and its language are proposed in this paper. As the proposed DFLSP is designed based on the dataflow architecture, it has an inherently concurrent processing and data synchronization capability. It also has a dynamic load balancing capability which increases the utilization factor of the whole system. Since DFLSP is a dataflow computing device, the ladder language is translated and formalized by a dataflow graph. With this formalized dataflow graph, algorithms to determine precedence relationship and deadlock problems are proposed.

Computer simulation by SIMAN shows that the store instruction hardly influences the overall performance of the DFLSP. The simulated logic solving time is 200 μsec with one LSU and 80 μsec with eight LSU's.

REFERENCES

[1] Jongil Kim,"The Architecture of Logic Solving Processor(LSP)," *Proceedings of 88 ISL Winter Workshop*

[2] Jongil Kim,"An Architecture of a Multiprocessor Based Programmable Controller with Emphasis on its System Bus," *Proceedings of 89 ISL Winter Workshop*

[3] Wook Hyun Kwon, Jongil Kim, and Jaehyun Park, " Architecture of a Ladder Solving Processor(LSP) for Programmable Controllers," *Submitted to I.E*

[4] Jaehyun Park, "Architecture of Data Flow Logic Solving Processor for Programmable controllers," *Proceedings of 90 ISL Winter Workshop*

[5] J.B. Dennis, "First version of data flow procedural language," Lecture Notes in computer Science, Vol. 19. Berlin: Springer Verlag, 1974.

[6] Krishna M. Kavi, Bill P. Buckles, and U. Narayan Bhat, "A Formal Definition of Data Flow Graph

Models," *IEEE Trans. on Computers Vol. C-35, No. 11*, November 1986.

[7] Chang Naehyuck, "A Study on the Dataflow PLC Architecture with Arithmetic and Logic Solving Unit Interface," *Proceeding of '91 ISL Winter Workshop*

[8] C. Dennis Pegden, *Introduction to SIMAN with Version 3.0 Enhancements*, System Modeling Corporation, December 1986.

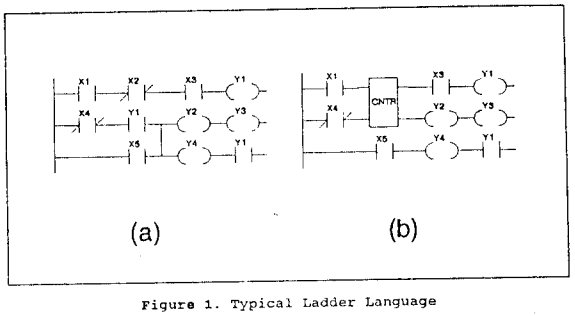

Figure 1. Typical Ladder Language

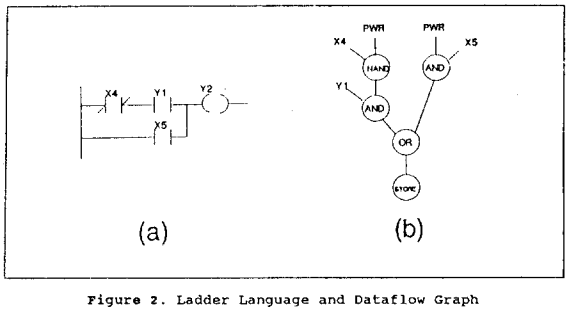

Figure 2. Ladder Language and Dataflow Graph

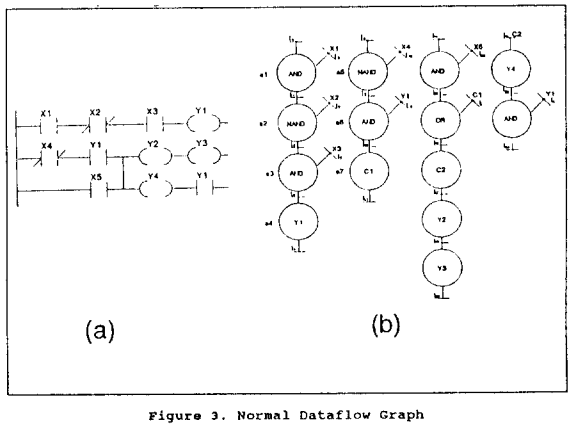

Figure 3. Normal Dataflow Graph

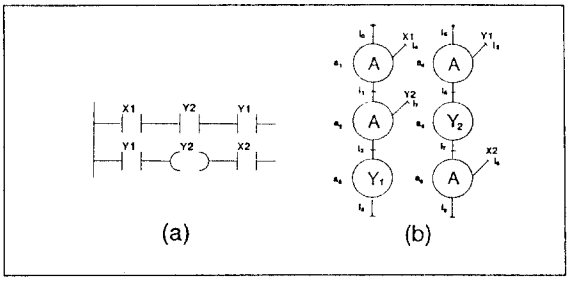

Figure 4. Deadlock Situation in Dataflow Graph