The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

DERIVING PROGRAMS FROM PARALLEL ALGORITHMS OF LOGICAL CONTROL

Dmitrij . I. CHEREMISINOV

Institute of Engineering Cybernetics of National Academy of Sciences of Belarus,
Surganov str., 6, 220012, Minsk, Belarus, *cher@newman.bas-net.by*

Abstract. *A problem of program implementation of parallel algorithms of logical control is considered. Parallel algorithms are represented in PRALU as a set of linear algorithms that can be executed under the control of mechanism of Petri net type. The goal is to build the compiler from language PRALU. This compiler can be used as working tool to build programmable logic controller (PLC) or supervisory control application or Forth application.*

Key Words. *Parallel algorithms of logical control, Petri nets, program implementation, compilation*

1. INTRODUCTION

Although developed primary for logic control, PRALU [10] can be used to structure any application that involves sequences of operations and controlled flow of execution, and where it is important to be able to represent communication between the parts of system. This language has textual and graphical forms. The advantage of the graphical form is the simplicity and declarativeness. The textual form is used as formal tool or as intermediate representation.

A programmable logic controller (PLC) is digitally operating electronic system designed for the use in an industrial environment. It was originally developed to replace electro-magnetic relay circuits or solid-state logic blocks [7]. Relay Ladder Logic (RLL), the most common graphical language for PLC is symbolic representation of relay circuits. The core of PLC software is a program, which interprets RLL diagrams. This program continually scan RLL diagram. The time elapsed during a scan (proportional to the length of RLL) determines response time of the PLC. PLC with language PRALU have less response time.

Supervisory control applications such as set-point control, monitoring, fault detection and production optimisation are mainly operator support systems. There is a large industrial interest in applying AI (Artificial Intelligence) techniques to this type of the application [10]. These applications typically use the object-oriented knowledge base with the rule-based programming. A large problem with the rule-based system is their lack of structure. Language PRALU can be used to structure the set of rules in these applications.

Programming language Forth [8] was originally developed for small embedded control mini- and micro-computers. It has been used in a wide variety of applications, but his main area is

distributed real time control systems. Using PRALU as programming language of Forth systems extends their fighting chance by the logic control.

The common feature for all above-mentioned applications is the generic lack of capacity of previously used languages. Using PRALU in this case offers the capability to improve execution speed and maintenance of the application.

In this work the set of operations (Intermediate Language) is proposed, which is concise to represent any PRALU algorithm as a program for a single processor computer. To prove the sufficiency of the chosen set, all PRALU constructions are considered, and equal sequences of operations of intermediate language are shown. The implementation of this intermediate language is inexpensive. Any operation in this set can be implemented as a short sequence (average length is equal to 3 for Intel 86 family) of modern microprocessor commands. As a result the quality of target program is achieved.

The proposed method of program implementation of parallel algorithms of logical control can be used for deriving a program from any specification that can be mapped into the state based representation with arcs labeled with a symbol of events. In [5] a technique for converting behavior description into Petri net is described. Thereafter, as the graphical form of PRALU algorithm is the interpreted Petri net, it is possible to implement programmatically any state based representation.

To design and debug of PRALU algorithms there are programming environments on the most commonly used platforms – IBM PC, MS Windows [12] and MS DOS [2]. In [4] ActiveX component is described that can be used as control engine of supervisory control applications. This component constitutes its own interface through which the execution of PRALU algorithms can be animated.

2. MINIMAL SEMANTICS OF PRALU

In [10] for describing the rules of executions of PRALU algorithms the notion of parallel automaton is used. But this semantics of PRALU is oriented to hardware implementation. Many tasks (for example, optimal state encoding) are insignificant in program implementation. Parallel systems software design requires attention to detail beyond that normally required for hardware systems.

The process of implementation of PRALU can be viewed as the replacement of the PRALU semantics by another, more detail one. A language can have several semantics that distinguish of detail level. The minimal semantics is a formal system that states fundamental properties only, and other correct semantics must include these properties. The minimal semantics puts a problem of the validity of an implementation on the firm ground of the formalism.

In the minimal semantics a PRALU algorithm (see example 5.1) is viewed as formulae of logic calculus. This logic calculus combines the linear time temporal logic and the branching time temporal logic. The objects of this calculus are a time interval, *operation* and a time point, *event*. The operation can be active, *executing* or passive, *stopping*. The states of operation are given in a schedule of the process of execution of the PRALU algorithm. The time is the notion of the minimal semantics and must be understood as in temporal logic.

In the minimal semantics of PRALU following presumptions are supposed to be valid.

1. The operations are connected (there is no time gap between adjacent operations).
2. The execution sequence of operations is deterministic (the next time point is unique).
3. The execution of operation depends on the same set of operations (symmetry of time point).

4. The event can be either the result of operation or the reason of firing of operation.

If event is the reason of firing operation then this operation is called “wait” and is denoted as “-”. If event is a result of operation it is called “action” (is denoted as “->”). The operations of PRALU are orthogonal by the causal relation between events and operations. The formal description of minimal PRALU semantics is in [3].

The causal relation (partial order) between the operations is determined from control structure of PRALU algorithms (fig. 1). The structure of PRALU algorithms directly identifies the causal relations between operations. The full behavior includes the causal relation between events. Consequently the behavior of algorithms depends on an interaction of operation by events (information exchange). In minimal PRALU semantics the model of information exchange is the same as in CCS [6]. As a result PRALU is not implementable in minimal semantics.

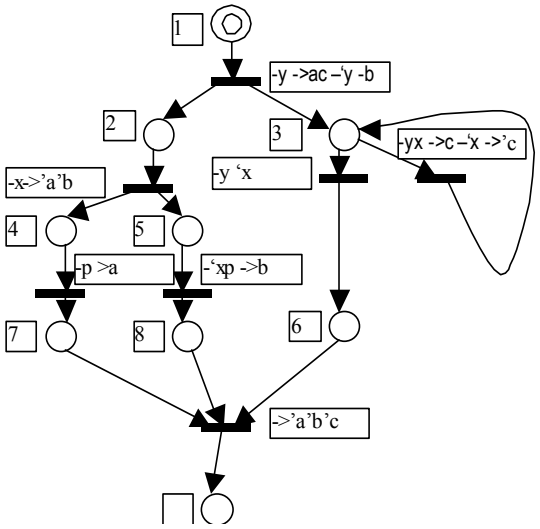

Fig. 1. Representation of PRALU algorithm as Petri net

The minimal semantics is primary a specification and it is useful because it lets us formally specify many different implementations of behaviour. Known methods of the PRALU algorithms validation [10] are true in minimal semantics, and consequently they are true in any correct elaboration of minimal semantics.

3. ELABORATIONS OF MINIMAL SEMANTICS

To be implemented semantics must predict the behaviour of PRALU algorithms unambiguously. The elaborated semantics must determine one-valued relation between the plan of operation execution and the causal structure of events. The information about the allowed orders and the times of events are captured in elaborated semantics.

Let us suppose that the wait operation starts at the same time point as event occurs that is the reason of firing this operation, and the action operation ends at the same time point as the event occurs that is the result of this operation (as it takes place in hardware implementation). In this case the relation between the plan of operation execution and the causal structure of events is determined by supposition about the duration of action in PRALU algorithms.

The hardware implementation of a PRALU algorithm has fully-specified behavior. The times of all operations are determined by the structure of a circuit. In formal terms the duration of all action operations is *constant* for every operation (it is not changed during execution of

algorithm). The hardware implementation has their communication scheduled statically in design time.

In CCS [6] the message exchange is determined with the concurrency relation. This relation fixes the set of couples of events. Each couple determines a single exchange event. But this relation is not a part of CCS model. Likewise language PRALU does not have explicit means to use this approach.

Other means is measured time, which is often used in concurrency determination. In PRALU measured time equals to the supposition about equality of times (continuance) of all action operations in the algorithm. The other form of this supposition is: the operation can not be executed in parallel to yourself.

4. INTERMEDIATE LANGUAGE

The converting Petri net into a program is considered in [9]. Our method produces a faster program. We propose the set of operations (Intermediate Language), which is concise to represent any PRALU algorithm as a program, and inexpensive implementation of this intermediate language. The proposed set of operations is the basis of algorithmic decomposition of the source algorithms. The wait and action operations are not unit actions and rather are the compositions in the proposed intermediate language.

To implement a concurrent algorithm on a single processor we must sort the plan of operation execution. This procedure is called scheduler. The scheduler of PRALU intermediate language is an object, which has properties and methods. The scheduler properties are a wait queue and a prepared queue. The scheduler methods are: thread start, thread stop, and thread interrupt. The scheduler methods are included into PRALU intermediate language.

Thread start operation includes into prepared queue the operation given in argument. Thread interrupt includes the next operation of algorithm into wait queue and takes the top operation from prepared queue and fires it. Thread stop stops current thread, takes the top operation from prepared queue and fires it. Thread stop operation has a conditional form.

The initial state of the scheduler is: prepared queue contains the first operation of the algorithm and wait queue is empty. If the scheduler has an empty prepared queue, then it copies content of wait queue into prepared queue and empties the wait queue. The realization of a queue is don't care for correct implementation of concurrency; this is the matter of the productivity of a program.

Prepared queue of a Forth system [8] is a data stack and wait queue is a call stack. In microprocessor program prepared queue is programmatically realized and wait queue is a call stack of microprocessor.

Besides this, there are operations of setting input or output buffers in PRALU intermediate language. To do information exchange, the output buffer is copied into the input buffer, when prepared queue is empty. At the same point of time the data output is executed from output buffer, and the external signals input into the input buffer. This guarantees that concurrently executing operations of the source algorithms have equal continuance.

Current marking of Petri net is represented by control vector. Each bit in this vector corresponds to the net place (the label of chain in source algorithm). The initial state of this vector is 0 in all bits. A mask vector is used to control the execution of dependent chains. The initial state of this vector is 0 in all bits. The conditional form of a thread stop operation tests this vector. If an argument is given, then this operation stops current thread, if mask vector bit is equal to 0.

Table 1.The instruction code of PRALU intermediate language.

Symbol	Function
@	Thread interrupt
%	Thread start
A	Thread stop
$	Set of output buffer
O	Test of input buffer
P	Set of control vector
R	Reset of control vector
M	Set of mask vector
C	Reset of mask vector

5. CONVERTING PRALU ALGORITHM INTO A PROGRAM

The compilation of PRALU is a substitution of PRALU operations by a sequence of intermediate language operations. The compiler PRALU uses the following patterns of substitution:
action operation (->x,…,y) => $(x,..y);
wait operation (-a,…,b) => @Aw1,..wn:O(a,…,b)Mw1,…,wn.
goto operation (->n,…,m) => Pw1,…wn:%n,…,%m
The pattern of wait operation and goto operation depends on parsing of source algorithm. This is general form.

5.1. Example.

1: -y ->ac –‘y ->b ->2.3	**1: @O(y) $(a,c) @O(‘y) $(b) %2%3 A**
2: -x ->’a’b ->4.5	**2: @O(x) $(’a,’b) %4%5 A**
3: -y’x ->6	**3: @Aw1O(y’,x)Mw1Mw2 P6 %7 A**
3: -yx ->c –‘x ->’c ->3	**4: @Aw1O(y,x) Mw1 $(c) @O(‘x) $(’c) @@ Pw1 %4%3 A**
4: -p ->a ->7	**5: @O(p) $(a) Mw2P7 %7 A**
5: -‘xp ->b ->8	**6: @O(‘x,p) $(b) Mw2P8 %7 A**
6.7.8: ->’a’b’c ->.	**7: @Aw2O(P6, P7,P8)Mw2 C(P6, P7,P8) $(a’b’c) A**

The result of compilation of algorithm on the left column is shown on the right one.

6. CONCLUSIONS

In this paper, we have discussed a systematic method to implement PRALU algorithm as a program. This method can be used for deriving a program from any specification that can be mapped into state based representation with arcs labeled with symbols of events.

In real life the proposed method has been used as a tool for designing programmable logic controllers, supervisory control applications, and Forth logic control capability.

REFERENCES

[1] Karl-Eric Arzen, “Grafcet for intelligent supervisory control application”, *Automatica*, Vol. 30, No. 10, pp. 1513 – 1525, 1994

[2] D.I.Cheremisinov, *The visualization of behavior PRALU algorithms*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, 1988 (in Russian)

[3] D.I.Cheremisinov, "The time model of PRALU algorithms", *Logic design automation of digital systems*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, pp. 46-55, 1991 (in Russian)

[4] D.I.Cheremisinov, "The engine of PRALU as ActiveX component", *Logic design*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, vol. 2, 1997 (in Russian)

[5] J.Cortadella, M.Kishinevsky, L.Lovagno, A.Yakokovlev, "Deriving Petri nets from finite transition system", *IEEE Trans. on Computers*, Vol. 47, No 8, pp. 859-882, 1998

[6] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985

[7] G. Michel, *Programmable Logic Controllers*, Wiley, New York, 1994

[8] C.H. Moore, "FORTH: A new way to program a mini-computer", *Astr. and Astrophys. Suppl.*, Vol. 5, pp. 497-511, 1974

[9] R.A. Nelson, L.M. Haibt, P.T. Sheridan, "Casting Petri nets into programs", *IEEE Trans. Software Eng*., Vol. 9, No. 5, pp. 590-602, 1983

[10] A.D.Zakrevskij, *Parallel algorithms for logical control*, Minsk, Institute of Engineering Cybernetics of NAS of Belarus, 1999 (in Russian)

[11] A. Zakrevskij, B.Steinbach, "Sequent automaton - a model for logical control", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Republic of Belarus, Minsk, Sept. 5-6, pp. 211-215, 2000

[12] A.D. Zakrevskij, Y.V. Pottosin, V.I. Romanov, I.V. Vasilkova, "Experimental system of automated design of logical control devices", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Republic of Belarus, Minsk, Sept. 5-6, pp. 216-222, 2000