Research Report

KSTS/RR-85/003
9 Feb., 1985

Novel Parallel Implementation of (condition)/(while do loop) Expressions using Petri Net Models

by

Jayantha Herath, Hideo Mizuba, Nobuo Saito, Kenji Toda, Yoshinori Yamaguchi, Toshitsugu Yuba

J. Herath, H. Mizuba, and N. Saito
Department of Mathematics, Keio University
K. Toda, Y. Yamaguchi, and T. Yuba
Electrotechnical Laboratory

Department of Mathematics
Faculty of Science and Technology
Keio University

KSTS/RR-85/003
Februrary 9, 1985

NOVEL PARALLEL IMPLEMENTATION OF (CONDITION)/(WHILE DO LOOP) EXPRESSIONS USING PETRI NET MODELS

Jayantha HERATH, Hideo MIZUBA*, Nobuo SAITO*,*
Kenji TODA, Yoshinori YAMAGUCHI, Toshitsugu YUBA

Department of Mathematics, Keio University,Yokohama, JAPAN

Electrotechnical Laboratory, 1-1-4, Umesono, Sakuramua,

Niiharigun, Ibaraki, 305, Japan.

ABSTRACT

This paper first describes Petri nets and their application in distributed and parallel processes. The modelling of computer programs using Petri nets is then discussed. The implementation of conditional operations and while-do loop operations using Petri nets and the shortcomings of the implementation are discussed in detail. A new simple and powerful parallel implementation for conditional operations and while- do loop operations is introduced, and its effectiveness is evaluated. The applications of this new interpretation are described.

1 Introduction

The Petri net, a parallel flow graph, was introduced in [1]. It is a simple, natural and powerful method of describing the information flow control of asynchronous parallel or distributed systems. This makes the Petri net an excellent working tool for such systems. A Petri net graph models the static properties of a system, and the dynamic properties result by executing the graph. It can be considered as a collection of transitions, places and arcs. Arcs connect the transitions and places, and are used as paths to transfer information between places and transitions. Places can be regarded as the temporary storages to store the tokens which are processed by transitions.

When all the input tokens of a transition are available, the transition is fired. Firable transitions are selected non-deterministically and are fired asynchronously. The Petri net execution is controlled by token distribution (called marking). A fired transition generates new output tokens and transfers them to the output places.

Petri nets are powered by handling constraints, exclusive OR transitions, switches, inhibitor arcs and time. Petri nets have been extensively used because of their capability of clearly describing concurrency, conflicts and synchronization of processes. The Petri net has been widely used to model various general properties of concurrent, distributed and asynchronous processes. Concurrent or distributed processing system specifications can be described clearly by using Petri nets. Petri net is used to model various kinds of asynchronous parallel or distributed system models such as concurrent computing systems. Original Petri nets were used to represent the logical behaviour of systems and were often used to model von Neumann computing systems. One of the major recent applications of the Petri net concept is modeling new generation computing systems such as dataflow and demandflow computers.

This paper considers modelling computer programs using Petri nets. Sequential and parallel computer programs can be modelled using Petri nets. This paper first briefly describes existing conditional and while-do loop expressions using Petri nets, and the necessity to introduce new Petri net mechanisms to represent conditional and while-do loop expressions. This paper describes a new parallel implementation of condition and while-do loop operations using Petri nets. Finally the effectiveness is evaluated.

2 Modelling computer programs using Petri nets

We can model computer programs, sequential and parallel programs with Petri nets. A statements in a computer program is an action to be performed. Actions can be represented using transitions in a Petri net. The result of a statement execution is the intermediate state between statements. This intermediate state can be represented by places in the Petri net. The results of statement execution can be stored temporarily in places. These results control the following transitions to be executed. Any computer program can be represented by a combination of

1. structured sequence of sequential statements
2. structured parallel statements
3. structured conditional statements
4. structured loop statements

A sequential program can be represented using 1, 3 and 4 above. Parallel programs use any combination of the above four types of statements to represent the parallel model of the computer program. A concurrent program is a collection of expressions constructed from assignments and quantifier free formulas by means of program connections. Composition or sequence of sequential statements, is represented by begin ... end program connective. Branching,is obtained by means of IF ... THEN ... ELSE ... conditional expressions program connective or WHILE ... DO iterative program connective. Parallel execution is obtained by means of Cobegin ... Coend connective.

Bipartite graph representation can be used to show the possible flow of control. The nodes of the graph are partitioned onto two sets. The set of places, circles in figures, and the set of transitions, boxes infigures. The edges of the graph connect nodes of two different kinds never of the same one. Each diagram has an input and an output. Boxes may contain assignment instructions or open formulas.

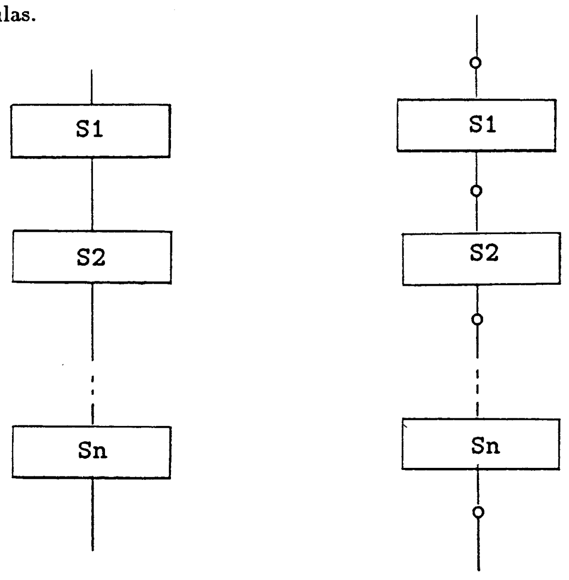


Fig. 1 (a) Flowgraph Fig. 1(b) Petrinet
Fig. 1 Sequential statements

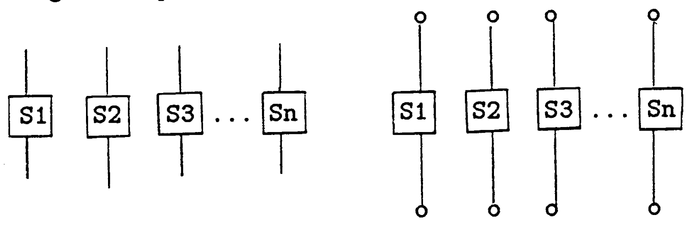


Fig. 2(a) Flowchart Fig. 2(b) Petrinet
Fig. 2 Concurrent statements

Fig. 1(a) shows the flowchart of the execution of ordered sequential statements S1, S2, ... Sn. The Petri net representation is shown in Fig. 1(b).

Fig. 2(a) and Fig. 2(b) show the flowchart and Petrinet representations of the execution of concurrent statements S1, S2, ... Sn.

2.1 Conditional Operations

General conditional expression in computer programming can be written as:

IF (condition) THEN statement'A', statement'B', ... ELSE statement'a', statement'b', ...

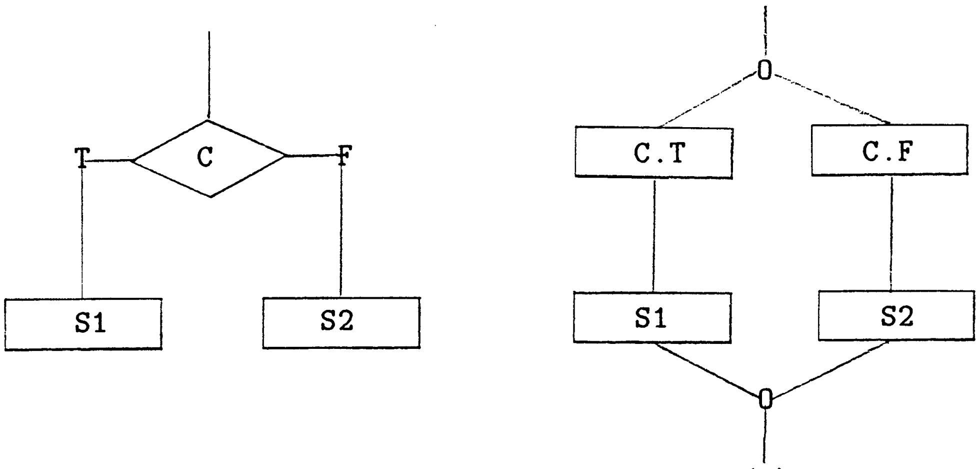


Fig. 3(a) Flowchart Fig.3(b) Petrinet
Fig. 3 Conditional operation representation

Fig. 3(a) shows conditional expression representations using flowchart. In conditional statement implementation, input consists of a single control arc with a single data token conneted to the statement. The branching of the statements to be executed starts here. There are two labelled output arcs, one going to the then statement and one going to the else statement. The evaluation of the boolean condition forces the correct set of statements to execute.

In Fig. 3(b) the Petri net configuration of conditional expression representation, a place node is used to represent a conditional expression. Basically, the conditional expression is an action to be performed, and therefore a conditional operation must be represented as a transition in a Petri net. The existing conditional expression is a sequential interpretation of two OR PARALLEL concurrent actions. Therefore, in concurrent systems we can use the concurrent interpretation. This can be expressed simply as

CONDITION = (POSITIVE CONDITION) OR (NEGATIVE CONDITION)

One statement gives the positive state of the condition expression and the other statement gives the negative state of the condition operation. This clearly distinguishes the statements to be executed after satisfying the condition. The new interpretation can be used to interpret conditional expressions in parallel programming as

IF (Condition operation) THEN statement'A', statement'B', ...

IF (NOT Condition operation) THEN statement'a', statement'b', ...

We can rewrite the flowchart representation and Petri net representation according to the new definition as shown in Fig. 4. It can be seen that no join or merge operation is performed at the end of conditional expression execution.

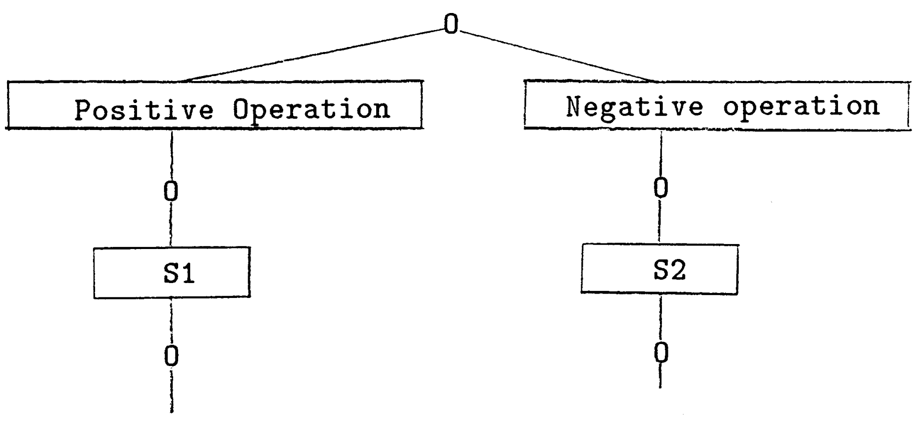


Fig. 4 Novel implementation of condition operations

Execution of a condition operation now needs two input tokens for both positive and negative operations. The input data token will only satisfy either of the operations. If the data satisfies the positive condition, the set of statements satisfied by this positive condition are executed. If the data satisfies the negative condition, the batch of statements to be executed thereafter will be executed. The conditional expression is represented as actions or transitions in the Petri net scheme. Here the execution of both OR combined operations gives TRUE boolean value output when the operation is TRUE. No False output is given. The parallel execution of two operations increase the speed of parallel programs and expresses the programs clearly. An advanced version of this combined operation is applied in dataflow computing models. Here no boolean output is given as the result of an operation execution. If the condition is satisfied the input data is given as the result of execution, and no output is given if it does not satisfy the condition.

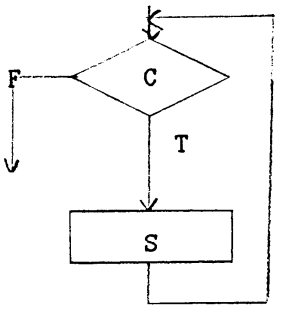
2.2 While Do Loop expressions

General WHILE DO LOOP expression in computer programming can be written as

WHILE(expression)

LOOP (DO statement'A', statement'B', ...)

In while-do loop implementation, the control arc to the while statement gives the boolean value output.



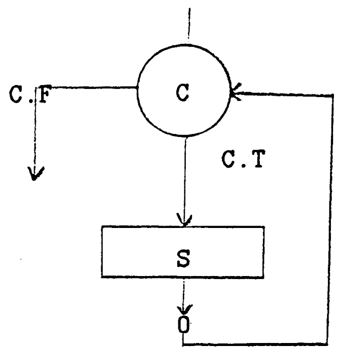


Fig. 5(a) Flowchart Fig. 5(b) Petrinet

Fig. 5 Loop While do -iteration representation

Execution of the while expression can also be divided into two OR parallel, concurrent actions as in the case of condition expressions.

WHILE (EXPRESSION) = (WHILE POSITIVE EXPRESSION) OR (WHILE NEGATIVE EXPRESSION)

Fig. 6 shows the new interpretation of flowcharts and Petri net models shown in Fig. 5. WHILE (Condition operation) LOOP (DO statement'A', statement'B', ...)

WHILE (NOT Condition operation)LOOP (DO statement'a', statement'b', ...)

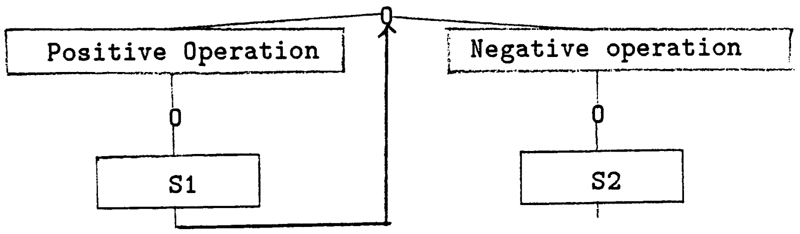
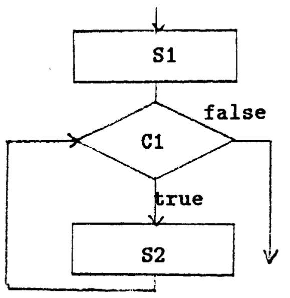


Fig. 6 Novel implementation of While Loop operations

3 Evaluation of NOT(OPERATION)



```
S1; q:= 0   r:= x;

C1; while  r>= y;

S2; loop q:= q+1; r:= r-y;
```

Fig. 7(a) Flowchart

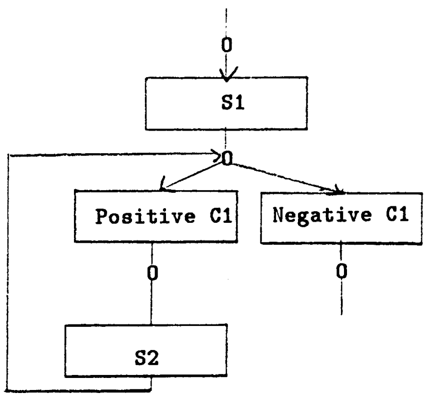


Fig. 7(b) Novel Petrinet representation
Fig. 7 Division by subtraction

The NOT(OPERATION) described above is simple. This results in a new, powerful Petri net representation for computer program modelling. The new implementation gives either of the boolean value results if the condition is satisfied, and this result is used to execute the statements in the corresponding branch of the computer program. Fig. 7 shows division by repeater subtraction.

An advanced version which gives the input value of the condition operation if the condition is satisfied as the result of an execution is used in dataflow models. The results obtained are briefly described in the following subsection.

3.1 Applications

We have implemented this control mechanism in dataflow computing system simulating models to represent condition operations with NOT(OPERATIONS). Results showed that this implementation helped to double the speed of dataflow computing and reduce garbage collected. This new interpretation resulted in a very simple, high speed unification control mechanism to implement logic programming such as Prolog in dataflow schemes.

4 Conclusions

Computer program modelling using Petri nets was discussed, and novel mechanism of representing conditional and while do operation in modelling using Petri nets was presented. The new interpretation clearly distinguishes the transitions from places in Petri net models. This increased the parallelism in concurrent systems. An advanced version of the interpretation which gives data value output as the result of a conditional operation instead of boolean value output was introduced. The results obtained by applying this new interpretation in dataflow computing environment was briefly described. This implementation increased the efficiency of dataflow parallel computing environments. The implementation increased the efficiency of parallel computing.

References

1. C. Petri; Fundamentals of a Theory of Asynchronous information flow, Information Processing 62, Proceeedings of the 1962 IFIP COngress, North Holland, 386-390, August 1962.

2. George W. Cherry; Parallel Programming in ANSI Standard Ada Reston Publishing Company, 1984

3. Jayantha Herath; Performance evaluation of a data-driven machine using a software simulator, *Masters Thesis, Univ. of Electrocommunications, Tokyo, Japan,* (March 1984).

4. Jayantha Herath, Hideyo Mizuba, Saito Nobuo, Kenji Toda, Yoshinori Yamaguchi, Toshitsugu Yuba; Not(operation): An efficient unification control mechanism to implement prolog languages in dataflow computing systems, *to be published.*

5. Jayantha Herath, Saito Nobuo, Kenji Toda, Yoshinori Yamaguchi, Toshitsugu Yuba; Not(operation): To reduce remaining packet garbage collected in dataflow computing systems, *to be published.*

6. Jayantha Herath, Saito Nobuo, Kenji Toda, Yoshinari Yamaguchi, Toshitsugi Yuba, Not(operation): A solution to matching bottleneck problem in dataflow machines, *to be published.*