National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

A Complexity Measure for the Specifications of Distributed Systems

by

Xiaogang Zhu

A M.Sc. Thesis

submitted to the School of Graduate Studies and Research

in partial fulfillment of the requirements for

the Master of Computer Science Degree*

University of Ottawa

Ottawa, Ontario

Canada

*The Master of Computer Science Program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute for Computer Science

ISBN 0-315-85817-6

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEDGEMENT

I am very grateful to my supervisor, Dr. To-yat Cheung, for his valuable time, patience, guidance and advice throughout my graduate studies. His instruction for revising the draft of my thesis has greatly improved its contents and presentation.

I would like to thank the Protocol Research Group for providing an excellent research environment during the entire period of my study. In particular, I would like to thank the members of the Group, especially Xinming Ye, Yucheng Ye, Youwen Wu and Guoqiang Wang, for their helpful comments and remarks.

My wife, Xinhua Song, and my daughter, Tina, deserve special thanks. Their love, understanding and support help me complete my master study as a part-time student in such a short time.

ABSTRACT

In software engineering, complexity measurement is a quantitative approach to estimating the degree of complication, organization and coding style of a computer program or system specification. Existing complexity metrics are mostly for non-distributed systems. This thesis extends them to distributed systems. It includes three contributions. Firstly, it proposes a Petri net model, called Interacting Petri nets (I-PN), for representing the behavior of distributed systems. This model provides a graph-based representation for LOTOS, which is necessary in order to derive methods for estimating the complexity of LOTOS. Secondly, a metric, called cyclomatic complexity, is proposed for measuring the complexity of I-NPs and LOTOS specifications. Two approaches are proposed for calculating such complexity. In the first approach, a LOTOS expression is first transformed to an I-PN and then Berge's formula in graph theory is applied to determine the cyclomatic complexity of the I-PN and the corresponding LOTOS specification. In the second approach, a set of formulas for calculating the cyclomatic complexity of all the primitive LOTOS operations is derived. Based on them, the complexity of any LOTOS expression can be calculated directly from the expression itself without going through any transformation. Thirdly, as an application of our I-PN model and complexity measure, a path-testing coverage criterion, called basis set coverage, is proposed. As an illustration, the above results are applied to the Transport Protocol (Class 0).

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

INTRODUCTION

1.1 COMPLEXITY MEASURES IN SOFTWARE ENGINEERING

It is widely agreed that attaining high quality in software is an important goal of software engineering. Two of the questions are then "what is software quality ?" and "how is it measured?" Various definitions of software quality have been proposed in the literature. For example, Pressman [PRE87] defines software quality as: "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." In fact, many properties, such as complexity, correctness, reliability, reusability, efficiency, integrity, maintainability, testability, flexibility, etc., all contribute to the quality of software. Besides definitions, quantitative measurements are also important if we want to compare or improve the quality of software. For example, we cannot say "This program needs more testing than the other" unless we know how the quality of testing is defined and measured.

In this thesis, we investigate a particular software quality called "complexity". In general, it is quite impossible to give an exact definition of complexity. The complexity metrics used in the literature often do not exactly or definitely reflect the specified characteristics of a program or system. Very often, they are just a loosely-defined but practical measure used for the quantitative evaluation of some of the design-related or programming-style-related characteristics of the software. Though lacking a unified definition, however, a useful complexity metric should satisfy the following requirements:

1). It can be calculated for all programs or specifications of a specific type (e.g., basic

LOTOS).

2). Adding some elements (e.g., instructions, storage, actions, etc.) to a program or a specification should never decrease the value of its complexity.

3). In practice, it should be efficient for computation.

The first requirement ensures a usable and objective measure. The second one is a common property of most metrics. It is another way of saying that a program or specification is at least as complex as any of its parts. The third one is concerned with the applicability of a metric.

1.2 MOTIVATION AND CONTRIBUTION OF THE THESIS

In the last two decades, many complexity metrics for software measurement have been developed in the literature. Some of them have been successfully applied for software quality control and management (e.g., used as guidelines for limiting software program styles or sizes) in big software houses [MCC76]. However, these complexity metrics have been developed mainly for non-distributed software and specifications based on either sequential programming languages or conventional flowgraphs. Though some of them can be extended quite easily, in principle at least, to distributed software, their contents or meanings may have to be modified. For example, Halstead's metrics, which involve the use of operators and operands, can be used for distributed software in a straight forward manner. But, one should consider whether an operator in a sequential programming language (e.g., addition) should have the same weight (as far as complexity is concerned) as an operator in a language for distributed systems (e.g., multi-process rendezvous in LOTOS). Furthermore, in the same period of time, distributed software has become much more complicated and larger in scale because of the growing demands for more sophisticated services to be provided by networks. Therefore, from the

viewpoint of both technical and application-oriented developments, complexity control for distributed software will become an important area for software engineering research.

To start research in this direction, the first step is to look for some specification technique(s) for describing distributed systems. After some careful investigation, we have chosen LOTOS and Petri-nets as our first target, for the reasons described in the following few paragraphs.

In response to the need of more powerful and rigorous means for the investigation of distributed systems, the International Organization for Standardization (ISO) and the International Consultative Committee for Telephony and Telegraphy (CCITT) have proposed a number of formal description techniques (FDTs). Based on them many unambiguous specifications and validation methods for distributed systems have been proposed in the literature. In particular, LOTOS, the Language fOr Temporal Ordering Specification, was approved as an international standard [ISO8807] in 1989.

For a complexity metric based on the structure of a system specification (i.e., structural complexity), a graph-based model representing its behavior is required. In the case of distributed systems, one of the well-developed models is Petri net. It includes the token-transition mechanisms for explicitly describing the structure and behavior of a distributed system. Also, several Petri-net-based models for representing LOTOS specifications have been developed in the literature. A lot of experience in these representations have been gained.

In this thesis, the main objective is to develop a structural metric for measuring the complexity of the control part of a LOTOS specification. As far as we know, this problem has never been investigated in the literature. Since the existing Petri net representations for LOTOS

are not very suitable for this purpose, we have to propose a new one. Then, we derive a metric for measuring the complexity of such Petri-nets and the corresponding LOTOS specifications. Lastly, we introduce a path-testing criterion for the conformance testing of LOTOS specifications.

The main contributions of this thesis are described in greater detail below:

(1). A Petri-net-based model is proposed for representing the behaviors of distributed systems

We have developed a Petri-net model, called *Interacting Petri nets* (I-PN) for representing the behavior of distributed systems. In this model, each process of the system is represented by a Petri net. Interactions have more general meanings than just inputs and outputs. For example, the LOTOS operations disable, interleaving, etc. are considered as interactions between processes. To represent such an interaction , the Petri nets representing operands are connected by a certain method base on the interaction. In particular, we have developed a set of rules for constructing the Petri-nets representing the primitive LOTOS operations.

A full LOTOS specification has essentially two parts: the control part which consists of the operations, actions and processes and the data part which consists of data types [BOL87]. In this thesis, I-PN can represent only basic LOTOS (i.e., the control part of LOTOS).

(2). A metric is proposed for measuring the complexity of I-PNs and LOTOS specifications

Based on the I-PN model, a metric, called *cyclomatic complexity*, is proposed for

measuring the complexity of both I-PNs and LOTOS specifications. Two approaches are proposed for obtaining such complexity for LOTOS expressions. The first approach depends explicitly on their Petri-net representations. In the second approach, based on their Petri-net representations, we derive a set of formulas for calculating the cyclomatic complexity of all the primitive LOTOS operations. By means of these formulas, the complexity of any LOTOS expression can be calculated directly from the expression itself without referring to their Petri-net representations

(3). A path testing criterion is proposed for covering a LOTOS specification

Based on the I-PN model, a criterion for path-testing coverage, called *basis set coverage*, is derived. It involves the concept of independent cycle paths in an I-PN. For a LOTOS specification, the maximum number of cycle paths in a basis set of its corresponding I-PN is equal to the cyclomatic complexity of the I-PN or the LOTOS specification. If we select a set of test sequences covering a basis set of cycle paths for the I-PN of a LOTOS specification, this set of test sequences will cover all arcs and nodes (places and transitions) of the I-PN at least once.

However, note that this thesis does not deal with the problems of test sequence generation or enforcement. It only proposes a coverage criterion.

1.3 ORGANIZATION OF THE THESIS

The rest of the thesis is organized as follows. Chapter 2 contains a survey on complexity measures for software engineering. It includes five well-known complexity metrics of three types: linguistic, structural and hybrid metrics. This survey provides us with the background on

the complexity measurement of I-PNs and LOTOS specifications. In Chapter 3, as the basic knowledge for our research, a brief review on both LOTOS and Petri nets is given. In Chapter 4, a Petri-net-based model called Interacting Petri Net, is proposed as the first main contribution of this thesis. By using I-PNs to represent LOTOS specifications, a set of rules rules for constructing and simplifying I-PNs is described. Also, a formal description of the constructing rules are included in this chapter. In Chapter 5, a complexity metric for both I-PNs and LOTOS specifications is presented. A formula for computing the complexity of an I-PN and LOTOS expression is derived from graph theory. Then, a series of construction formulas for calculating the complexity of primitive LOTOS operations is deduced from the general formula. Chapter 6 includes the third contribution of this thesis: a basis set coverage criterion. In Chapter 7, as an illustration, we apply our results to the ISO Transport Protocol (Class 0). Lastly, Chapter 8 describes some conclusive remarks and possible works for future research.

Chapter 2

A SURVEY ON COMPLEXITY MEASURES FOR SOFTWARE PROGRAMS AND SPECIFICATIONS

2.1 INTRODUCTION

One of the objectives of software engineering is to produce a controllable environment for the development, understanding, testing, maintenance and extension of software. It would be even better if the environment is measurable. A complexity metric may serve such a purpose. It reflects certain characteristics of the software quantitatively. Once the complexity of a program or specification exceeds a certain limit, management of the software may become very difficult or even impossible.

In one way or another, a complexity metric is related to the following observables of a program or specification:

- types of elements
- number of elements
- relationships among elements

To develop a metric, one must decide what elements are to be involved according to the characteristics to be measured. For example, if the syntactic complexity of a program is under consideration, the entire program, subroutines, lines of code, operands, operators, etc., may be included as elements. The selection of elements affects the structural relationships among them.

If large-sized elements, such as subroutines, are selected, the relationships among the operators and operands within the subroutines become internal elements, while the interfaces among subroutines become a part of the relationships among the elements. Therefore, it is important to first clearly define the participating elements.

Roughly, complexity metrics can be classified into three categories: *linguistic metrics*, *structural metrics*, and *hybrid metrics*.

(a). Linguistic Metrics: Metrics that are directly based on the context of a program or specification regardless of its actual meaning or the relationship among the elements in the context. Examples: number of lines of code, number of statements, number of operators, etc.

(b). Structural Metrics: Metrics that are based on the structural characteristics among the objects of a program or specification. Such characteristic are usually expressed in the form of a graph-related representation, such as control flowgraphs. Examples: number of links, number of nodes, nesting depth, etc.

(c). Hybrid Metrics: Metrics that are based on some combination of the structural and linguistic properties of a program or specification.

In the following sections, five different metrics are briefly reviewed. Among them, Halstead's and McCabe's are the best known in the literature. Most of the subsequent developments in complexity measures have been based on them.

2.2 HALSTEAD'S METRICS [HAL77]

Halstead's *software science* complexity measures are probably the best-known and most

thoroughly studied linguistic metrics. Although Halstead's assumptions and analysis have been under dispute in the last two decades, his metrics are considered as effective for estimating a wide range of software characteristics. Under the assumption that the complexity of a program increases with the number of operators and operands used in the program, these metrics are based on the four primitive elements listed below:

n_1 = number of distinct operators

n_2 = number of distinct operands

N_1 = total number of occurrences of all operators

N_2 = total number of occurrences of all operands

Based on these quantities, measures for the *length*, *volume*, and *effort* of a program are defined as follows:

Length: $N = N_1+N_2$

Volume: $V = N \times \log_2(n_1+ n_2)$

Effort: $E = V^2/V^*$

where V* is the minimum volume of all possible programs that solve the same problem.

When solving a problem, different programmers may use different approaches in coding their programs, resulting in different volumes. Theoretically, there exists a minimum volume V* for any solvable problem. Since it is almost impossible to determine V*, Halstead approximates his effort measure E by the following expression:

$$E = V \times [(n_1 \times N_2) / (2 \times n_2)]$$

Example 2.1

Consider the Pascal program as shown in Figure 2.1.

```
PROGRAM2(input, output);
VAR
    a, b, d, m: integer;
BEGIN
    readln(a, b, c, d);
    IF a > b THEN
        IF b > c THEN
            m := a + b
        ELSE
            m := b + c
    ELSE
        m := b + c + d;
    writeln(m)
END.
```

Figure 2.1 An illustration of Halstead's Metrics

This program has 11 different operators: "BEGIN END", "IF THEN ELSE", "readln", "writeln", "()", ",", ":=", "+", ">", ";" and ".". They are used a total of 22 times. There are 5 distinct operands which are listed in the VAR statement. These operands are used 19 times. Thus, $n_1 = 11$, $n_2 = 5$, $N_1 = 22$, and $N_2 = 19$ Hence, the length, volume and effort for this program have the following values:

$$N = 22 + 19 = 41$$

$$V = 41 \times \log_2(11 + 5) = 164$$

$$E = 164 \times (11 \times 19) / (2 \times 5) = 3427.6$$

Though not having been proved mathematically, Halstead's metrics have been strongly supported by experimental results ([ALB83], [BAS83], [CUR79], [CUR79B], [DUN82], [FEU79A] and [FEU79B]).

2.3 McCABE'S METRIC [MCC76]

McCabe's *cyclomatic complexity* is a typical structural metric. Unlike Halstead's metrics, cyclomatic complexity does not depend on a program's operators and operands, but on its decision branches.

Based on the flowgraph of a program or a specification, the nodes, edges, and connected components are used as the basic elements in the cyclomatic complexity. It is assumed that the flowgraph has a unique entry node and a unique exit node, and that each node can be reached from the entry node and can reach the exit node. Hence, if the exit node is connected to the entry node, the flowgraph becomes strongly connected. Borrowing the concept of cyclomatic number from graph theory, McCabe's complexity measure is calculated by the following formula:

$$v(G) = e - n + 2p$$

where G is the flowgraph of a program

n = number of nodes of G

e = number of edges of G

p = number of connected components of G

Example 2.2

Consider the flowgraph of a program as shown in Figure 2.2. We have: e = 11,

n = 9 and p = 1. Thus, the cyclomatic complexity of the program is

$$v = e - n + 2 = 11 - 9 + 2 = 4.$$

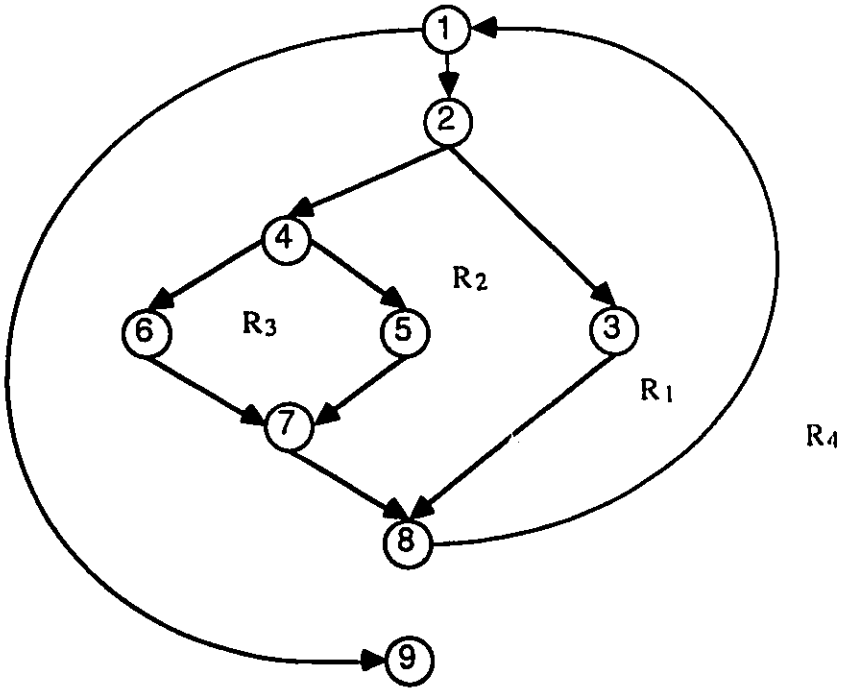


Figure 2.2 The cyclomatic complexity of a flowgraph

Besides measuring the cyclomatic complexity, McCabe's metric can also be used to control the number of testing paths. In graph theory [BER73], it is known that the cyclomatic number is equal to the maximum number of linearly independent cycles existing in a strongly connected graph. McCabe applies this fact to program flowgraphs. In order to make a flowgraph strongly connected, the exit node is connected to the entry node. Instead of independent cycles, he uses *independent paths*. By definition, a path starts at the entry node and ends at the exit node. A set of paths is independent iff no path of the set can be expressed as a simple linear combination of the other paths of the set. A basis set of a flowgraph is a set of independent paths of the flowgraph such that any path in the flowgraph can be expressed as a

simple linear combination of the paths of the set. In general, the basis set is not unique, though the number of paths of a basis set is unique. Obviously, a basis set covers all the arcs and nodes of the flowgraph. Therefore, if a basis set of independent paths is used as test sequences, the cyclomatic complexity provides an upper bound on the number of the test sequences that cover all statements and branches at least once. In software testing, this criterion is called *basis set coverage.*

Example 2.3

A basis set of the flowgraph shown in Figure 2.2 is :

p1: 1-9

p2: 1-2-3-8-1-9

p3: 1-2-4-5-7-8-1-9

p4: 1-2-4-6-7-8-1-9

The number of the paths in the basis set is 4, which is equal to the cyclomatic complexity of the flowgraph. Any other path can be linearly expressed by the basis set. For instance, the path p5: 1-2-3-8-1-2-4-5-7-8-1-9 can be expressed as:

p5 = p2 + p3 - p1

As an alternative, we can use {p2, p3, p4, p5} as a basis set. Then, path p1 can be expressed as:

p1 = p2 + p3 - p5

McCabe also provides two other methods for calculating the cyclomatic complexity of a plane flowgraph and a structured program. Thus, the cyclomatic complexity can be calculated by one of the following 3 formulas:

- Based on the number of edges, nodes and components of a flowgraph G:

$$v(G) = e - n + 2p$$

- Based on the number of regions (r) of a plane flowgraph G:

$$v(G) = r$$

- Based on the number of predicate nodes (π) of a structured program:

$$v(G) = \pi + 1$$

Example 2.4

In Figure 2.2, there are 11 nodes, 9 edges, 4 regions (R_1, R_2, R_3, and R_4) and 3 predicate nodes (nodes 1, 2 and 4). Then, $n = 11$, $e = 9$, $r = 4$, and $\pi = 3$. By the 3 formulas above:

- $v(G) = e - n + 2p = 11 - 9 + 2 = 4$
- $v(G) = r = 4$
- $v(G) = \pi + 1 = 4$

2.4 WOODWARD'S METRIC [WOO'79]

Woodward's *knot complexity* is a structural metric. It is assumed that an unstructured program has higher complexity than a structured one. The concept of "knots" is introduced to measure the complexity of an unstructured program.

Definition 2.1 Let jump(n, m) represent a jump from line n to line m. A *knot* is created iff there exists a crossing between a pair of jump statements, jump(a,b) and jump(p,q).

That is, one of following conditions is satisfied:

a) $\min(a, b) < \min(p, q) < \max(a, b)$ and $\max(p, q) > \max(a, b)$

b) $\min(a, b) < \max(p, q) < \max(a, b)$ and $\min(p, q) < \min(a, b)$

Woodward's knot complexity (C) of an unstructured program is defined as:

C = number of knots in a program.

Example 2.5

Consider the FORTRAN program in Figure 2.3.

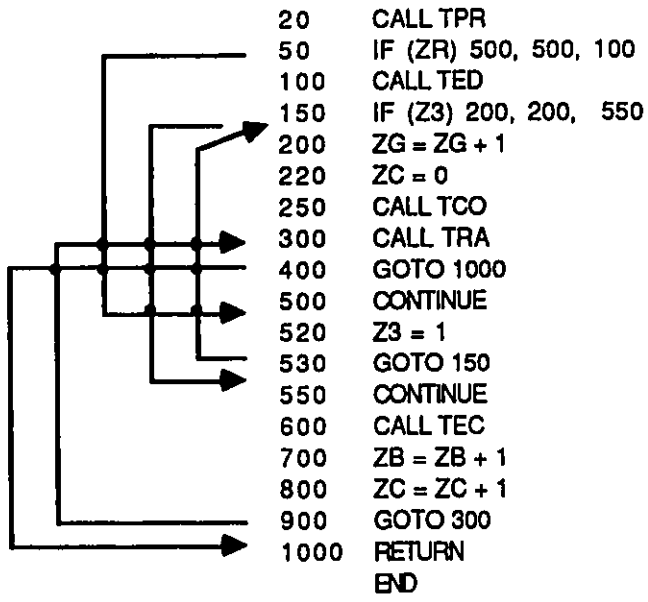


Figure 2.3 The "knots" in a FORTRAN program

This program has 9 knots represented by the 9 cross points of the jump lines. Hence, its "knots" complexity C is 9. In comparison with cyclomatic complexity, since there are 2 predicates ($\pi = 2$) in the program, $v(G) = \pi + 1 = 2 + 1 = 3$.

Obviously, the "knots" complexity of a program is related to the order of its statements. The number of "knots" in a program may be reduced by reordering the statements, as shown in the following example.

Example 2.6

Figure 2.4 (a) has 4 knots created by 4 pairs of jump statements: jump(1, 3) with jump(4, 2), jump(1, 3) with jump(2, 5), jump(3, 5) with jump(4, 2), and jump(2, 5) with jump(4, 2). After reordering, as shown in Figure 2.4 (b), there is only one knot created by jump(1, 2) with jump(3, 5). Since the number of nodes and edges in these two versions are not changed, the cyclomatic complexities of both versions are 3.

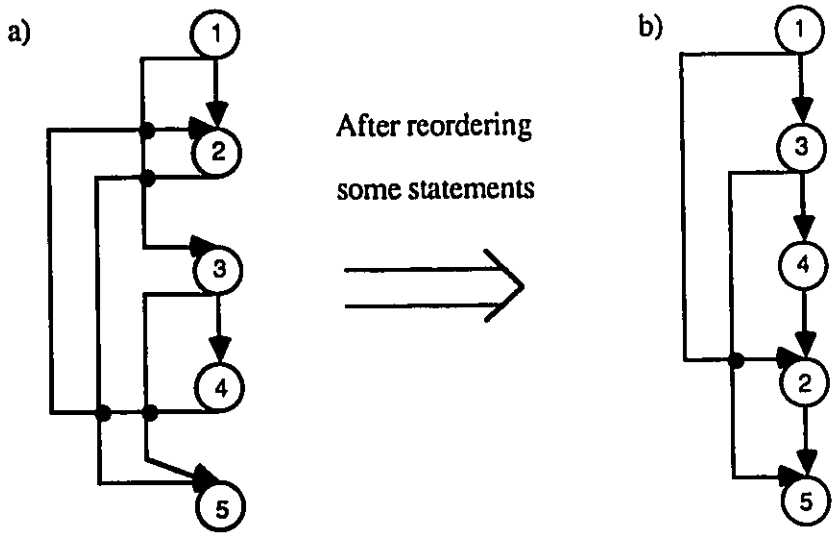


Figure 2.4 An example of reducing knot complexity by reordering the program statements

By the above example, it is obvious that McCabe's metric cannot reflect the changes in the order of program statements. Woodward's metric gives a much clearer indication of program readability, as well as a measure of unstructured programs. Because the 'while' and 'until'

looping structures can be implemented with zero knots, the knot complexity does not reflect the complexity resulting from the loops in a program. This is a weakness of this metric.

2.5 RAMAMURTHY'S METRICS [RAM88]

Ramamurthy's hybrid metrics, called *weighted measures*, are based on a combination of structural and linguistic properties of a program. Since the length, volume and effort measures in Halstead's metrics do not detect the complexity due to nonsequentiality of a structure, Ramamurthy assigns weights to some of these operators and operands so to reflect such aspects in the control flow.

Similar to Halstead's measures, Ramamurthy uses the following primitive elements (a lower case subscript w stands for a weighted measure):

n_1 = number of distinct operators

n_2 = number of distinct operands

$N_{w1} = \Sigma_{x \text{ in } T}[1+ \delta(x) \times l(x)]$

$N_{w2} = \Sigma_{x \text{ in } R}[1+ \delta(x) \times l(x)]$

where

T : the set of operators

R : the set of operands

$\delta(x)$: a function from $T \cup R$ to $\{0,1\}$. $\delta(x) = 1$, if x is within a control structure;

$\delta(x) = 0$, otherwise.

$l(x)$: the nesting level of x. $l(x) = 0$, if x is not within any control structure.

When a nonsequential structure, such as "if", "while", is added at level L, for any x within this structure $l(x) = L + 1$.

Ramamurthy's weighted measures are:

Length: $N_w = N_{w_1} + N_{w_2}$

Volume: $V_w = N_w \times \log_2(n_1 + n_2)$

Effort: $E_w = (V_w)^2 / V^*$

Ramamurthy approximates the weighted effort measure by

$$E_w = V_w \times [(n_1 \times N_{w_2}) / (2 \times n_2)]$$

Example 2.7

In Figure 2.1, the statements "readln(a, b, c, d)" and "writeln(m)" are not within any control structure. Thus, the nesting level for any operator or operand x within these two statements is $l(x) = 0$. Since the first "IF THEN ELSE" statement is added at level 0, the nesting level for any operator or operand x within this control structure is $l(x) = 1$. Since the second "IF THEN ELSE" statement is added at level 1, the nesting level for any operator or operand x within this structure is $l(x) = 2$. At nesting level 0, 8 operators "BEGIN END", "readln", "()", ",", ";", "IF THEN ELSE", "writeln" and "." are used a total of 11 times. 5 operands "a", "b", "c", "d" and "m" are used 5 times. They will be assigned a weight $[1 + \delta(x) \times l(x)] = 1$. At nesting level 1, 5 operators ">", "IF THEN ELSE", ":=", "+" and ";" are used 6 times. The 5 operands are used 6 times. They will be assigned a weight $[1 + \delta(x) \times l(x)] = 2$. At nesting level 2, 3 operators ">", ":=" and "+" are used 5 times. The 5 operands are used 8 times. They will be assigned a weight

$[1+ \delta(x) \times l(x)] = 3$. The weighted measures of the program are:

$n_1 = 11$

$n_2 = 5$

$Nw_1 = 11 + 6 \times 2 + 5 \times 3 = 38$

$Nw_2 = 5 + 6 \times 2 + 8 \times 3 = 41$

$Nw = 38 + 41 = 79$

$Vw = 79 \times \log_2(11 + 5) = 316$

$Ew = 316 \times [(11 \times 41) / (2 \times 5)] = 14251.6$

By the formulas for length, volume and effort, the weighted measures are not only related to the number of operators and operands and their number of occurrences, but also to their nesting levels. The weighted measures reflect the complexity resulting from both the code and control flow of a program. Since nesting occurs only in four structures: sequence, 'if', 'while', and 'until', and it is hard to determine the level of 'goto' statement in an unstructured program, these metrics cannot be used to measure unstructured programs.

2.6 STEPHEN'S METRICS [STE88]

All of the metrics described in the previous sections are not related to data flow. Stephen proposed some hybrid complexity measures, named *review measures*. They are related to both the control flow and the data flow of a program. The basic element used in this metric is 'chunk', as defined below:

Definition 2.2 A *chunk* is a sequence of contiguous program statements $(s_i, s_{i+1}, \cdots, s_{i+n})$, for which s_i is the only entry point (that is, no statement in the entire program transfers control to any statement in this sequence except s_i).

Stephen's metrics depend on two basic concepts: *control dependency* and *data dependency*. A control dependency exists between two chunks if there is a potential transfer of control from one chunk to the other. A data dependency exists between two chunks if a variable is defined in one chunk and potentially used in another. Stephen's review complexity of a program is defined by the following formula:

$$C = \sum_{i=0}^{\#c} \sum_{j=0}^{f(i)} (2/3)^j c(i)$$

where #c : total number of chunks in the program.

f(i) : number of dependencies on Chunk i.

c(i) : the complexity of Chunk i.

By changing the definitions of c(i) and f(i), Stephen developed the following review measures:

STDCPLA: f(i) = the number of both data and control dependencies
c(i) = the number of statements in Chunk i

STDEQ: f(i) = the number of both data and control dependencies
c(i) = 1 (consider all chunks with equal complexity)

CTLCPL: f(i) = the number of control flow dependencies
c(i) = the number of statements in Chuck i

CTLEQ: f(i) = the number of control flow dependencies
c(i) = 1

DATCPL: f(i) = the number of data dependencies

c(i) = the number of statements in Chuck i

DATEQ: f(i) = the number of data dependencies

c(i) = 1

The following measures are simply counts of dependencies or chucks:

DATFAN: Total number of data dependencies between chunks.

CTLFAN: Total number of control dependencies between chunks.

NUCHK: Total number of chunks.

Example 2.7

Consider the FORTRAN program shown in Figure 2.3. There are 6 chucks in the program, which are Chunk 1: (20 - 100), Chunk 2: (150 - 250), Chunk 3: (300 - 500), Chunk 4: (520 - 530), Chunk 5: (550 - 900) and Chunk 6: (1000). At Chunk 1, there are 3 statements, 2 control dependencies (with Chunks 2 and 4) and no data dependency. At Chunk 2, there are 4 statements, 4 control dependencies (with Chunks 1, 3, 4, and 5) and 2 data dependencies (with Chunks 4 and 5). At Chunk 3, there are 2 statements, 3 control dependencies (with Chunks 2, 5, and 6) and no data dependency. At Chunk 4, there are 3 statements, 2 control dependencies (with Chunks 1 and 2) and 1 data dependency (with Chunk 2). At Chunk 5, there are 5 statements, 2 control dependencies (with Chunks 2 and 3) and 1 data dependency. At Chunk 6, there are 1 statements, 1 control dependency (with Chunk 3) and no data dependency. Hence, the review measures of the program are:

STDCPL: $C = 3(1+2/3+(2/3)^2)+4(1+2/3+(2/3)^2+(2/3)^3+(2/3)^4+(2/3)^5+(2/3)^6)+$
$2(1+2/3+(2/3)^2+(2/3)^3)+3(1+2/3+(2/3)^2+(2/3)^3)+$
$5(1+2/3+(2/3)^2+(2/3)^3)+1(1+2/3)$

$= 43.37$

STDEQ: $C = (1+2/3+(2/3)^2)+(1+2/3+(2/3)^2+(2/3)^3+(2/3)^4+(2/3)^5+(2/3)^6)+$
$(1+2/3+(2/3)^2+(2/3)^3)+(1+2/3+(2/3)^2+(2/3)^3)+$
$(1+2/3+(2/3)^2+(2/3)^3)+(1+2/3)$
$= 13.82$

CTLCPL: $C = 3(1+2/3+(2/3)^2)+4(1+2/3+(2/3)^2+(2/3)^3+(2/3)^4)+$
$2(1+2/3+(2/3)^2+(2/3)^3)+3(1+2/3+(2/3)^2)+$
$5(1+2/3+(2/3)^2)+1(1+2/3)$
$= 40.12$

CTLEQ: $C = (1+2/3+(2/3)^2)+(1+2/3+(2/3)^2+(2/3)^3+(2/3)^4)+$
$(1+2/3+(2/3)^2+(2/3)^3)+(1+2/3+(2/3)^2)+$
$(1+2/3+(2/3)^2)+(1+2/3)$
$= 13.01$

DATCPL: $C = 3+4(1+2/3+(2/3)^2)+2+3(1+2/3)+5(1+2/3)+1$
$= 27.78$

DATEQ: $C = 1+(1+2/3+(2/3)^2)+1+(1+2/3)+(1+2/3)+1$
$= 8.44$

DATFAN: $C = 0 + 2 + 0 + 1 + 1 + 0 = 4$

CTLFAN: $C = 2 + 4 + 3 + 2 + 2 + 1 = 14$

NUCHK: $C = 6$

Chapter 3

A BRIEF INTRODUCTION TO LOTOS AND ITS PETRI-NET-BASED REPRESENTATIONS

3.1 INTRODUCTION

In this thesis, a Petri net model called *Interacting Petri net* (I-PN) is proposed for describing the behavior of a distributed system. The model get its motivation from and will be used to represent basic LOTOS. Therefore, as background information, basic LOTOS is briefly described and some of the Petri-net-based models for representing LOTOS are reviewed in this chapter.

By means of a set of operations, such as enable, disable, parallel, etc., basic LOTOS, the control part of full LOTOS, can be used to describe the control behavior of the processes of a system. Its semantics is based on a modification of Milner's Calculus of Communicating Systems [MIL80] and Hoare's Communicating Sequential Processes [HOA85].

A Petri net is a well-known model used for the specification and validation of distributed systems. In the literature, at least four Petri-net-based models have been proposed for the representation and validation of LOTOS specifications. They are the Communicating System of Petri Nets proposed by Cheung, et al. [CHE88], the Galileo net proposed by Marchena, et al. [MAR89], the Place/Transition-net proposed by Barbeau, et al. [BAR90], and the Network proposed by Garavel, et al. [GAR90].

The rest of this chapter is organized as follows. Section 3.2 gives an introduction to basic

LOTOS. The four Petri-net-based representations of LOTOS mentioned above are briefly reviewed in Section3.3.

3.2 SYNTAX OF BASIC LOTOS

In basic LOTOS, processes interact with one another without value passing. The syntax of basic LOTOS expressions and their inference rules, which are used to explain the meanings of the expressions, are listed in Figure 3.1. Refer to [BOL87] for details of the syntax and semantics of full LOTOS.

Notation used in Figure 3.1:

B, B1, B2, ... represent processes;

g, g1, g2, ... represent actions;

G represents a set of actions;

i represents an internal action;

δ represents the successful termination action.

Name	Behavior expression	Axioms or inference rule
inaction	stop	no action
successful termination	exit	exit $-\delta\rightarrow$ stop
action prefix:		
- unobservable	B = i; B1	B $-i\rightarrow$ B1
- observable	B = g; B1	B $-g\rightarrow$ B1
choice	B = B1 [] B2	B1 $-g\rightarrow$ B1' implies B $-g\rightarrow$ B1'
		B2 $-g\rightarrow$ B2' implies B $-g\rightarrow$ B2'
parallel composition:		
- general form	B = B1 |G| B2	B1 $-g\rightarrow$ B1' and g $\notin$ G implies B $-g\rightarrow$ B1'
		B2 $-g\rightarrow$ B2' and g $\notin$ G implies B $-g\rightarrow$ B2'
		B1 $-g\rightarrow$ B1' and B2 $-g\rightarrow$ B2' and g $\in$ G implies B $-g\rightarrow$ B1' |G| B2'
- full synchronization	B =B1 || B2	B1 $-g\rightarrow$ B1' and B2 $-g\rightarrow$ B2' implies B $-g\rightarrow$ B1' || B2'
- pure interleaving	B =B1 ||| B2	B1 $-g\rightarrow$ B1' implies B $-g\rightarrow$ B1' ||| B2
		B2 $-g\rightarrow$ B2' implies B $-g\rightarrow$ B1 ||| B2'
sequential composition	B = B1 >> B2	B1 $-g\rightarrow$ B1' implies B $-g\rightarrow$ B1' >> B2
		B1 $-\delta\rightarrow$ B1' implies B $-\delta\rightarrow$ B2
disabling	B = B1 [> B2	B1 $-g\rightarrow$ B1' implies B $-g\rightarrow$ B1' [> B2
		B2 $-g\rightarrow$ B2' implies B $-g\rightarrow$ B2'
		B1 $-\delta\rightarrow$ B1' implies B $-\delta\rightarrow$ B1'

Figure 3.1 Inference rules for basic LOTOS expressions

3.3 PETRI-NET-BASED REPRESENTATIONS OF LOTOS

3.3.1 A Communicating System of Petri-Nets (by Cheung and Zhu) [CHE88]

The model, a Communicating System of Petri-nets (*CSPN*), is proposed by Cheung, et al. for the representation of the processes of a distributed system specified in full LOTOS [CHE88]. Briefly, each LOTOS process is represented by a CSPN and each LOTOS operation between two processes is represented by adding transitions and/or places to connect the relevant CSPNs. This part of the model has been modified for data flow analysis of LOTOS [HUA92]. The resulting CSPN preserves the modularity of a LOTOS specification.

CSPN uses two types of tokens: control-tokens (c-tokens) and data-tokens (d-tokens). c-tokens are used to regulate the control flow of the system. d-tokens are vehicles for transferring data values. Each d-token is associated with a data list carrying three kinds of information: value-expressions, sorts and usage status (i.e., "has been defined" or "to be defined"). In order to describe special actions, CSPN introduces four types of transitions: assignment-transitions (a-transition), guarded-choice-transitions (g-transition), interactive-transitions (i-transitions) and matched-transitions (m-transitions). a-transitions represent groups of value-assignment statements. g-transitions represent guard-choice. i-transitions are primitives for interprocess communication. m-transitions represents matched synchronizations.

3.3.2 Galileo Nets (by Marchena and Leon) [MAR89]

Galileo is an environment for a designer to model, analyze and simulate concurrent systems by means of a *Galileo net*, which is a Petri net with the extension of data operations based on Pascal data types [SAN86]. Data places associated with transitions are introduced into

Galileo nets for the manipulation of data operations. The firing of transitions may depend on the predicates of the data contained in the associated data places and may modify these data.

For analysis purpose, a LOTOS specification is first transformed to a Galileo net and analysis is done through the net [MAR89]. In this approach, only well-formed recursions are considered, i.e., all recursions must be prefixed by the action prefix (;). Although Galileo supports data operations, it cannot handle abstract data types in a formal way. Furthermore, the net absorbs all the LOTOS processes and operations, i.e., the operational relationships among the LOTOS processes do not remain as part of the characteristics of their Galileo net representation.

3.3.3 Place/Transition-Nets (by Barbeau and Bochmann) [BAR90]

Barbeau and Bochmann introduce a Place/Transition net (*P/T-net*) semantics for LOTOS [BAR90]. Basically, it is a Petri-net whose transitions may be labelled with a user-defined action, a successful termination action (δ) or an internal action (i). It can represent a subset of basic LOTOS with the following constraints on recursions:

1) A process instantiation must be prefixed by an action prefix, or must be in the right sub-expression of an enabling operator (>>) or a disabling operator ([>).
2) The general parallel operator (|[ ]|) is not allowed within a recursive process definition.

Barbeau, et al. prove that every specification based on this subset of LOTOS can be transformed to an equivalent P/T-net, and vice versa. However, P/T-net is lack of data handling facilities. No validation application has been proposed.

3.3.4 Networks (by Garavel and Sifakis) [GAR90]

The *network* proposed by Garavel and Sifakis is a basic Petri-net with the extension of data operations at the transitions and a partition of the places [GAR90]. Each transition is attached with three elements: 1) a visible gate, a hidden gate "τ" or an invisible action "ε"; 2) an offer which is a list of value/variable declarations; and 3) an action which may be a null action or a composition of an assignment, a condition and/or an iteration. The partition of places is determined by a set of special components called *units*. Intuitively, each unit is a subset of places which represents a sequential behavior. A unit can also be hierarchically refined into subunits to express that the corresponding behavior is composed of several concurrent sub-behaviors. However, no details of units are given in that paper.

In Garavel's network, the inaction "stop" is represented as one place. An action is represented as two places connected by a transition. A process is represented as a set of places and transitions connected according to the operators involved. Similar to other Petri-net-based models, some constraints are imposed on recursive processes. For instance, no recursion is allowed in the following cases:

1) the operand of a general parallel operator,

2) the operand of operator "par", and

3) the left sub-expressions of the enable or disable operators.

Chapter 4

INTERACTING PETRI-NETS (I-PN) AND THEIR REPRESENTATION FOR LOTOS

4.1 INTRODUCTION

In this chapter, as the first major contribution of this thesis, we introduce a model called *Interacting Petri nets* (I-PN) and use it to represent basic LOTOS. The underlying concept and operations of this model are mainly based on LOTOS. Briefly, Petri nets representing different processes can interact with one another. Conceptually, each interaction appears in the form of an operation applied on the participating Petri nets. For example, as in LOTOS, two Petri nets may be executed in parallel (as PARALLELISM in LOTOS), or in mutual exclusion (as CHOICE in LOTOS), or one Petri net may interrupt the execution of another (as DISABLE in LOTOS).

At the present stage of our development, the meanings and types of operations adopted in this Petri-net model are exactly the same as those for LOTOS and hence will not be explicitly denoted and defined here in detail (see Chapter 3 for the semantic meanings of these operations). Instead, we shall take the following approach to describing our model:

a. For each of the three basic LOTOS elements exit, stop and action, we propose an I-PN representation.
b. For each of the primitive composite LOTOS operations, we describe how its I-PN representation can be obtained by replacing the operator with places and/or transitions

connecting the I-PN representations of its operands.

Such an approach has three objectives:

a. To show that the execution of I-PNs is, in principle at least, the same as ordinary Petri nets.

b. To develop a new Petri-net representation of LOTOS.

c. To lay the ground work for the derivation of a metric for measuring the complexity of an I-PN or a LOTOS specification.

The rest of the chapter is organized as follows. In Section 4.2, we define the notations to be used for the transformation, the basic elements of the I-PN model and the four types of I-PNs. In Section 4.3, we define a set of graphical operations and use them to describe the I-PN representations of LOTOS expressions. In Section 4.4, two rules for simplifying such representations are described. Lastly, a formal description of the rules for transforming LOTOS specifications to I-PNs is given in Section 4.5.

4.2 THE INTERACTING PETRI-NET MODEL

In this section, we describe the basic ingredients of the I-PN model and some special operations used to describe the transformations of LOTOS specifications to I-PNs.

Definition 4.1 An *Interacting Petri-net* I-PN(S) representing a LOTOS expression S is an 8-tuple $<P, T, E, L, l, p_i, p_x, p_s>$, where

P(S) : a finite set of places;

T(S) : a finite set of transitions;

E(S) : a set of arcs, i.e., $E \subset (P(S) \times T(S)) \cup (T(S) \times P(S))$;

L(S) : a set of transition labels for T(S), which is the set of event names of S;

l(S) : a labelling function l: $T(S) \rightarrow L(S)$;

p_i : the initial place;

p_x : the exit place; and

p_s : the stop place.

Besides the three basic types of elements: places, transitions and arcs, an I-PN has two special types of elements. The first one is the 'stop' place from which no arcs and hence no tokens can come out, representing the deadlock status. The second one is the 'exit' place, representing a successful termination.

The following notations and symbols will be used for the description of the transformation rules and the complexity formulas.

Definition 4.2 The following notations denote some special components of I-PN(S):

$T_x(S) = \{ t \mid t \in T(S), l(t) = \text{'exit'}\}$, i.e., the set of transitions in I-P(S) with the label "exit";

$T_t(S) = \{ t \mid t \in T(S), l(t) = \text{'exit' or } l(t) = \text{'stop'}\}$, i.e., the set of transitions in I-PN(S) with the label "stop" or "exit";

$T(G,S) = \{t \mid t \in T(S), l(t) \in G$, where G is a set of gates$\}$. i.e., the set of transitions in I-PN(S) with labels in G.

$E_x(S) = \{(p,t), (t,p) \mid (p,t), (t,p) \in E(S), t \in T_x(S)\}$, i.e., the set of arcs adjacent to one of "exit" transitions in I-PN(S).

$E(G,S) = \{(p,t), (t,q) \mid (p,t), (t,q) \in E(S), l(t) \in G$, where G is a set of gates$\}$. i.e., the set of arcs adjacent to one of transitions in T(G,S);

$N(S) = P(S) \cup T(S)$, i.e., the set of nodes of I-PN(S)

Definition 4.3 The following symbols (Figure 4.1) are used for the graphical description of an I-PN:

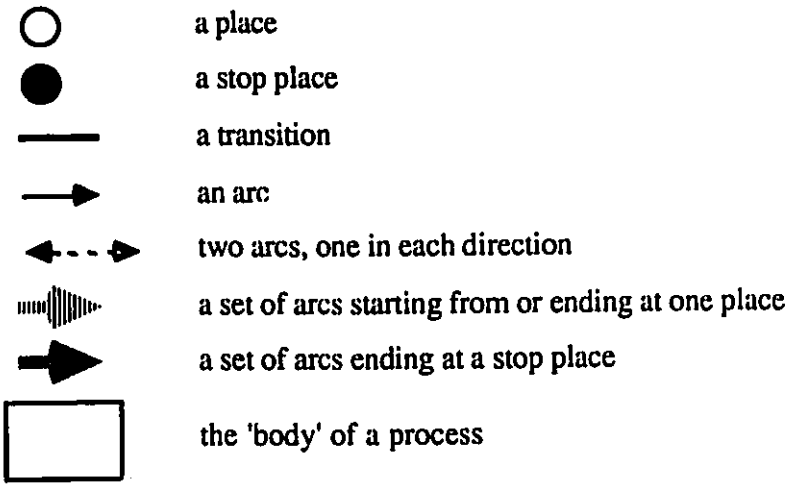


Figure 4.1 Symbols used for describing an I-PN

A LOTOS expression is composed of operators and operands. An operand is itself a LOTOS expression representing a single event or a composite behavior expression. In the I-PN model, an event is represented by a transition with a label. e.g., a transition with a special label 'exit'. The 'body' of a process is an abbreviation including all events in the process. Symbolically, the I-PN of a behavior expression S begins with an initial place p_i, follows with

a rectangle representing all actions of S, and ends possibly with an exit place p_x and/or a stop place p_s. p_x and p_s are called the *outlet(s)* of the I-PN(S). Figure 4.2 shows their graphical abbreviations.

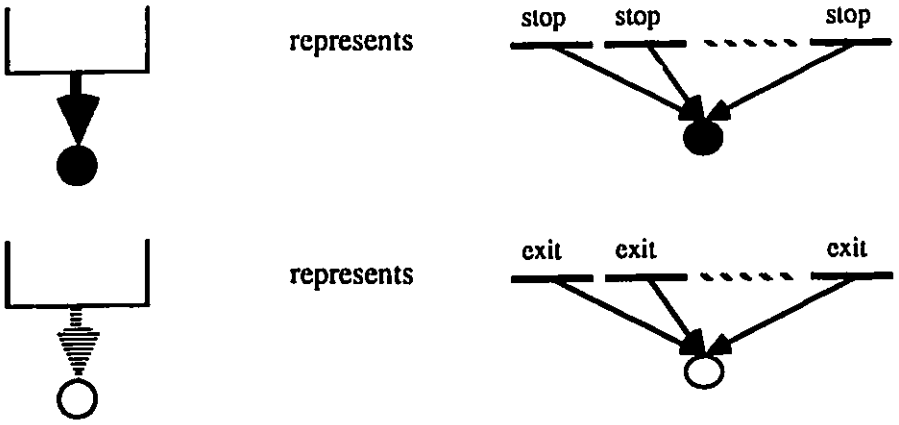


Figure 4.2 Outlet representation in an I-PN

Depending on its outlets, an I-PN(S) may be classified into four types (Figure 4.3):

Type 1: I-PN(S) has no outlet.

Type 2: I-PN(S) has only a stop outlet.

Type 3: I-PN(S) has only an exit outlet.

Type 4: I-PN(S) has both a stop outlet and an exit outlet.

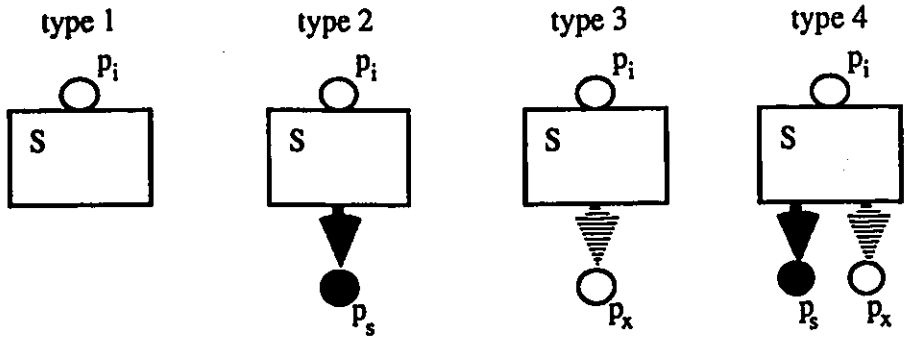


Figure 4.3 Four types of I-PNs

4.3 GRAPHICAL CONSTRUCTION OF THE I-PN FOR A LOTOS SPECIFICATION

The following five graphical operations are needed for describing the rules for transforming a LOTOS expression to its I-PN representation.

Definition 4.4: (*graphical operations* used in the construction of I-PN)

In the following, p represents a place, t represents a transition, and ϕ represents a place which actually does not exist.

Opn.1 Place or transition creation :

$$\mathbf{newp}(p_1, p_2, \ldots, p_n) \text{ or } \mathbf{newt}(t_1, t_2, \ldots, t_n)$$

This operation creates new places $p_1, p_2, .., p_n$ or new transitions $t_1, t_2, ..., t_n$.

Opn.2 Arc creation : $p \rightarrow t$ or $t \rightarrow p$ ([P])

If predicate P is satisfied, this operation adds an arc from p to t or from t to p. Otherwise, nothing is done.

Opn.3 Place merging : $p \equiv p'$

If p or p' does not exist (i.e., ϕ), nothing is done. Otherwise, place p' and its i incoming and outgoing arcs are merged into place p.

Opn.4 Transition merging : $t_{j,k} = t_j \times t_k$ [P]

In this operation, if the predicate P is satisfied, the pair of transitions t_j and t_k are

merged to form a new transition $t_{j,k}$ and all the arcs adjacent to t_j and $t_k$become adjacent arcs to $t_{j,k}$. Nothing is done if P is not satisfied.

Opn.5 Label assignment : l(t) = 'a'

This operation assigns the label 'a' to transition t.

The rules for constructing the I-PN representation of a LOTOS expression is described below.

Rules for Constructing an I-PN: Rules (a) and (b) are for constructing I-PN(stop) and I-PN(exit) respectively. For a composite LOTOS expression S representing an operation on two sub-expressions A and B, I-PN(A) and I-PN(B) should be constructed first and then I-PN(S) can be constructed by combining I-PN(A) and I-PN(B) via one of following Rules (c) - (i) :

(a). Inaction **stop** (Figure 4.4) :

I-PN(stop) is composed of an initial place p_0, a transition t_0 labelled with the special symbol "stop" and a stop place p_1 which represents a deadlock status.

Construction procedure:
newp(p_0, p_1), newt(t_0);
$l(t_0)$ = 'stop'
$p_0 \rightarrow t_0$, $t_0 \rightarrow p_1$;

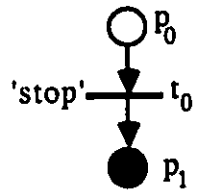


Figure 4.4 I-PN(stop)

(b). Successful Termination **exit** (Figure 4.5) :

I-PN(exit) is composed of an initial place p_0, a transition t_0 labelled with the special action "exit" and an exit place p_1 which represents a successful termination action.

Construction procedure:

newp(p_0, p_1), newt(t_0);

$l(t_0)$ = 'exit'.

$p_0 \rightarrow t_0$, $t_0 \rightarrow p_1$;

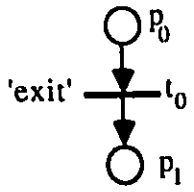


Figure 4.5 I-PN(exit)

(c). Action Prefix **a; A** (Figure 4.6) :

I-PN(a; A) is constructed by adding I-PN(A) an initial place p_0 and a transition t_0 with label "a". It means that action "a" must have been performed before the behavior expression A is executed.

Construction procedure:

newp(p_0), newt(t_0);

$l(t_0)$ = 'a'.

$p_0 \rightarrow t_0$, $t_0 \rightarrow p_{ia}$;

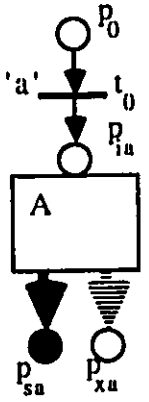


Figure 4.6 I-PN(a; A)

(c). Choice **A [] B** (Figure 4.7) :

I-PN(A [] B) is constructed by first creating an initial place p_0 and two empty transitions ε_0 and ε_1, and then connecting them to the initial places of I-PN(A) and I-PN(B). These two empty transitions represent the choice operator and correspond to no action.

Construction procedure:

newp(p_0), newt(ε_0, ε_1);

$l(\varepsilon_0) = l(\varepsilon_1) = '[]'$.

$p_0 \rightarrow \varepsilon_0$, $p_0 \rightarrow \varepsilon_1$;

$\varepsilon_0 \rightarrow p_{ia}$, $\varepsilon_1 \rightarrow p_{ib}$;

$p_{xa} = p_{xb}$, $p_{sa} = p_{sb}$.

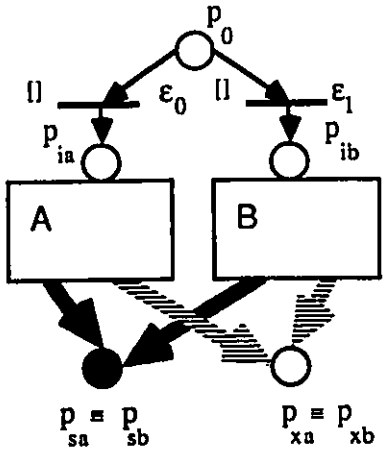


Figure 4.7 I-PN(A [] B)

Example 4.1

Consider A [] B, where A = a; stop and B = b; exit

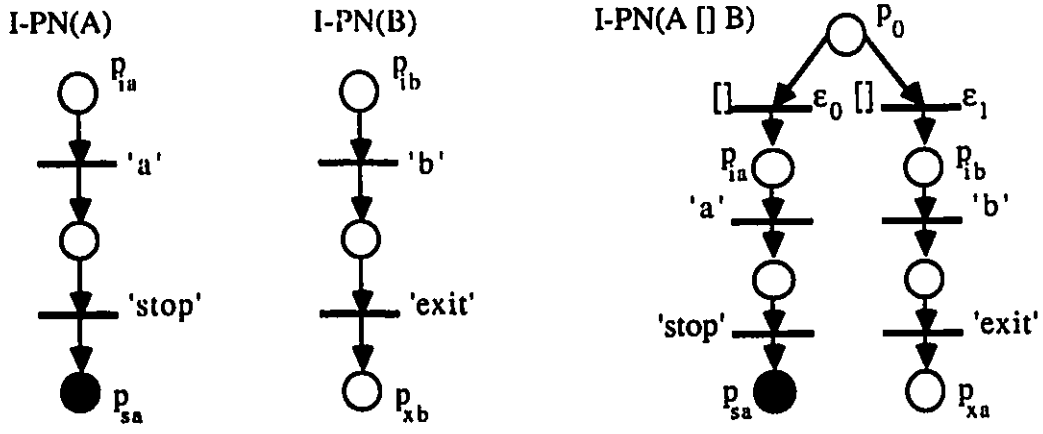


Figure 4.8 An example of I-PN(A [] B)

(e). Enabling **A >> B** (Figure 4.9) :

I-PN(A >> B) is constructed by first creating an empty transition ε, which represents the enable operator, and then connecting it to the exit place p_{xu} of I-PN(A) and the initial place p_{ib} of I-PN(B). If A has no exit (i.e., $p_{xa} = \phi$), then B can never be enabled and I-PN(A >> B) is the same as I-PN(A). By means of the empty transition ε, I-PN(A >> B) is able to express potential recursions of B.

Construction procedure:

newt(ε);

$l(\varepsilon_0) = $ '>>'.

$p_{xa} \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ib}$;

$p_{sa} \equiv p_{sb}$.

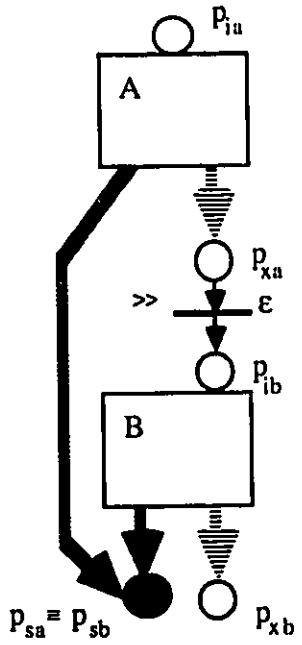


Figure 4.9 I-PN(A >> B)

Example 4.2

Consider A >> B, where A = a; stop [] b; exit and B = c; exit

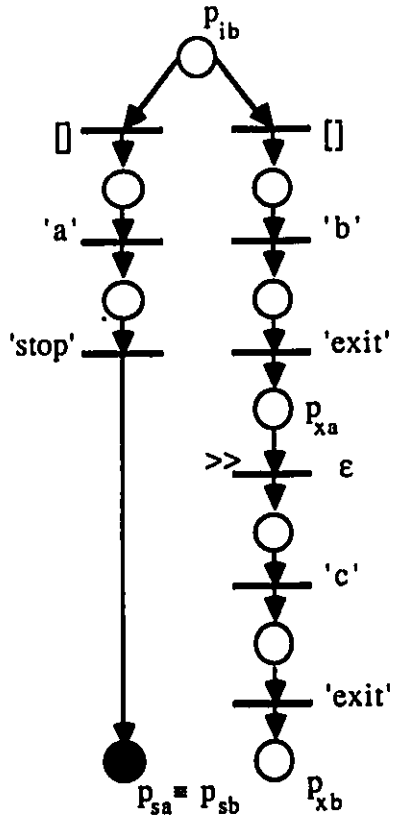


Figure 4.10 An example of I-PN(A >> B)

(f). Disabling **A [> B** (Figure 4.11) :

I-PN(A [> B) is constructed by first creating an initial place p_0 and an empty transition ε, which represents the disable operator, and connecting them to the initial places of I-PN(A) and I-PN(B). Then the initial place of I-PN(B), is connected to every terminal transition of A by an arc, and with every other non-empty transition of A by two arcs, one in each direction. The semantics of the disable operator is reflected in the following way: when the transition with label '[>' is fired, a token will be deposited into the initial places

p_{ia} and p_{ib}. Then, no transition in I-PN(A) can be fired unless a token is still in p_{ib}. After firing a transition in I-PN(A), a token will be returned to p_{ib}. Once any transition in I-PN(B) is fired, the token in p_{ib} is taken away and the transitions in I-PN(A) can no longer be fired. This disables any execution within A.

Construction procedure :

newp(p_0), newt(ε);

$p_0 \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ia}$, $\varepsilon \rightarrow p_{ib}$;

l(ε) = '[>';

$p_{ib} \rightarrow t$ [$t \in T(A), t \neq \varepsilon$], $t \rightarrow p_{ib}$ [$t \in T(A) - T_t(A), t \neq \varepsilon$];

$p_{sa} \equiv p_{sb}$, $p_{xa} \equiv p_{xb}$.

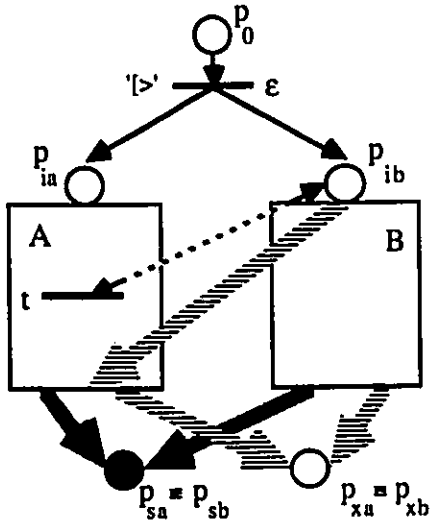


Figure 4.11 I-PN(A [> B)

Example 4.3

Consider A [> B, where A = a; stop [] b; exit and B = c; exit

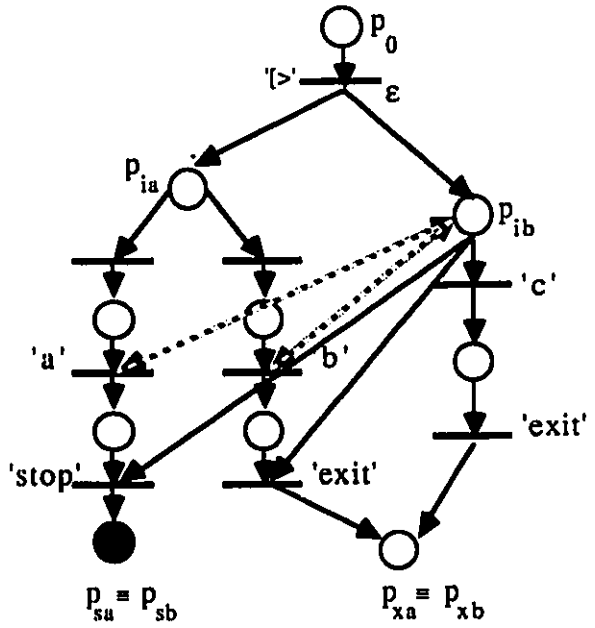


Figure 4.12 An example of I-PN(A [> B)

(g). Interleaving **A ||| B** (Figure 4.13) :

I-PN(A ||| B) is constructed by first creating an initial place p_0 and an empty transition ε, which represents the interleaving operator, and connecting it to the initial places of I-PN(A) and I-PN(B). Then, depending on the outlet types of I-PN(A) and I-PN(B), two cases should be considered. In Case 1, where both I-PN(A) and I-PN(B) have an "exit" place, each pair of their exit transitions is merged to become an exit transition of I-PN(A ||| B), and their "stop" places, if any, are merged into one stop place. In Case 2, where either I-PN(A) or I-PN(B) does not have any "exit" place, all exit transitions will become stop transitions (i.e., deadlock status), and all stop places and exit places will become a stop place.

Construction procedure:

Case 1. (both p_{xa} and p_{xb} exist)

newp(p_0), newt(ε);

$p_0 \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ia}$, $\varepsilon \rightarrow p_{ib}$;

$l(\varepsilon)$ = '|||';

$t_{j,k} = t_j \times t_k$, $l(t_{j,k})$ = 'exit' $[\forall(t_j, t_k)\ (t_j \in T_x(A)$ and $t_k \in T_x(B))]$;

$p_{sa} \equiv p_{sb}$, $p_{xa} \equiv p_{xb}$.

Case 2. (otherwise)

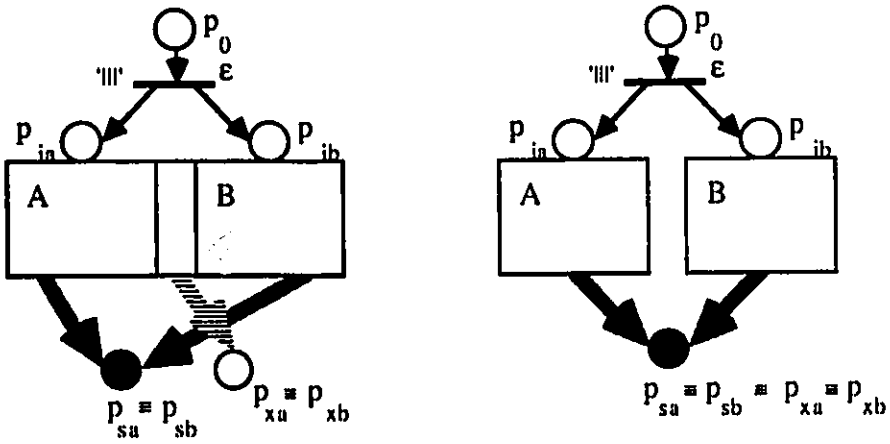
newp(p_0), newt(ε);

$p_0 \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ia}$, $\varepsilon \rightarrow p_{ib}$;

$l(\varepsilon)$ = '|||';

$l(t)$ = 'stop', $\forall\, t \in T_x(A) \cup T_x(B)$

$p_{sa} \equiv p_{sb} \equiv p_{xa} \equiv p_{xb}$.



Case 1 Case 2

Figure 4.13 I-PN(A ||| B)

Example 4.4

Consider A ||| B, where A = a; stop [] b; exit and B = c; exit

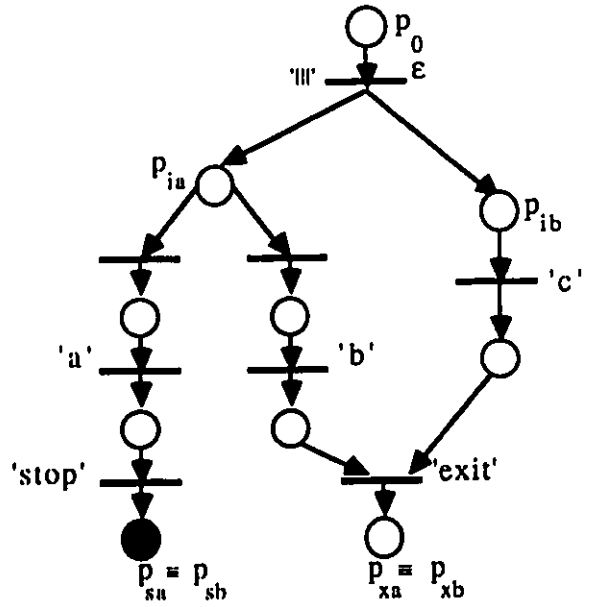


Figure 4.14 An example of Case 1 of I-PN(A ||| B)

Example 4.5

Consider A ||| B, where A = a; stop [] b; exit and B = c; stop

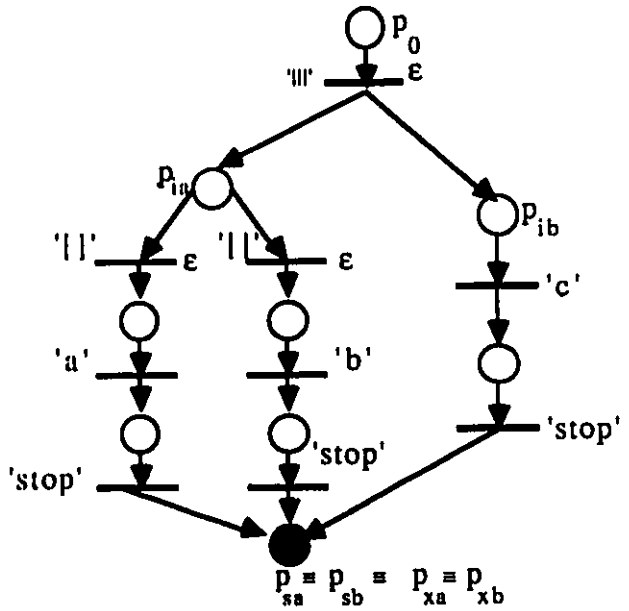


Figure 4.15 An example of Case 2 of I-PN(A ||| B)

(h). Parallelism A |G| B, where G = $[g_1, g_2, ..., g_n]$ (Figure 4.16) :

The construction of I-PN(A |G| B) is similar to that of I-PN(A ||| B). In order to represent the rendezvous between process A and process B, we have to merge the gate transitions (in G) of I-PN(A) and I-PN(B).

Construction procedure:

Case 1. (both p_{xa} and p_{xb} exist)

newp(p_0), newt(ε);

$p_0 \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ia}$, $\varepsilon \rightarrow p_{ib}$;

l(ε) = '|G|';

$t_{j,k} = t_j \times t_k$, $l(t_{j,k})$ = 'exit' $[\forall (t_j, t_k)\ (t_j \in T_x(A)$ and $t_k \in T_x(B))]$;

$t_{m,n} = t_m \times t_n$, $l(t_{m,n}) = l(t_m)$ $[\forall\ (t_m, t_n)\ t_m \in T(A)$ and $t_n \in T(B)$, $l(t_m) = l(t_n) \in G]$;

$p_{sa} \equiv p_{sb}$, $p_{xa} \equiv p_{xb}$.

Case 2. (otherwise)

newp(p_0), newt(ε);

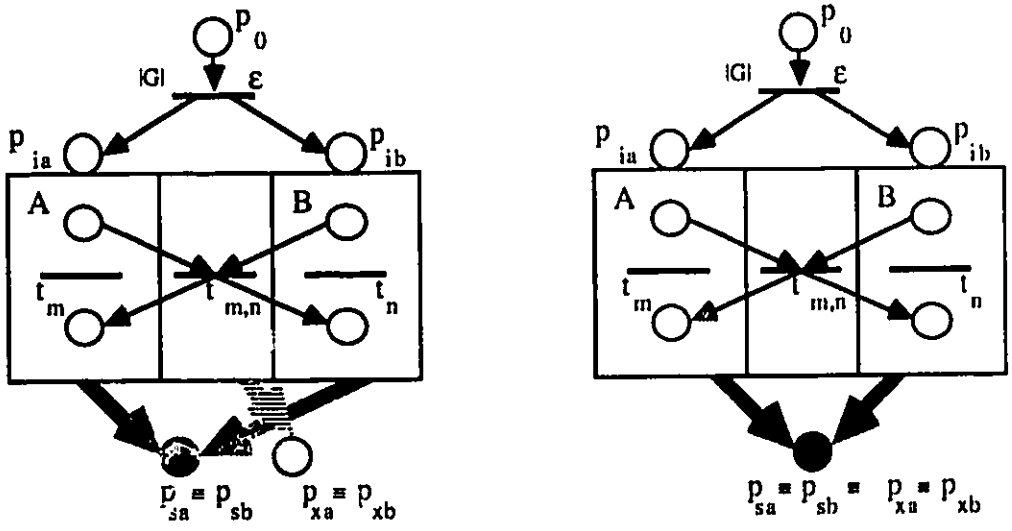


Figure 4.16 I-PN(A |G| B)

$p_0 \rightarrow \varepsilon$, $\varepsilon \rightarrow p_{ia}$, $\varepsilon \rightarrow p_{ib}$;

$l(\varepsilon) = 'iG'$;

$t_{m,n} = t_m \times t_n$, $l(t_{m,n}) = l(t_m)\ [\forall\ (t_m, t_n)\ t_m \in T(A)$ and $t_n \in T(B)$, $l(t_m) = l(t_n) \in G]$;

$l(t) = 'stop'$, $\forall\ t \in T_x(A) \cup T_x(B)$

$p_{sa} \equiv p_{sb} \equiv p_{xa} \equiv p_{xb}$.

Example 4.6

Consider A |al| B, where A = a; exit and B = (a; exit ||| b; exit)

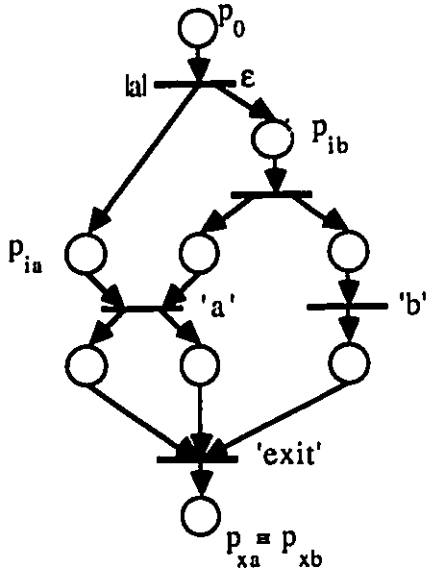


Figure 4.17 An example of Case 1 of I-PN(A |al| B)

(i). Recursive process A := ...A... (Figure 4.18)

To construct I-PN(A), a recursive occurrence of process A is first represented by a place p'_{ia} (Figure 4.18(a)), which is then merged with the initial place p_{ia} of process A (Figure 4.18(b)).

Construction procedure:

$p_{ia} \equiv p'_{ia}$.

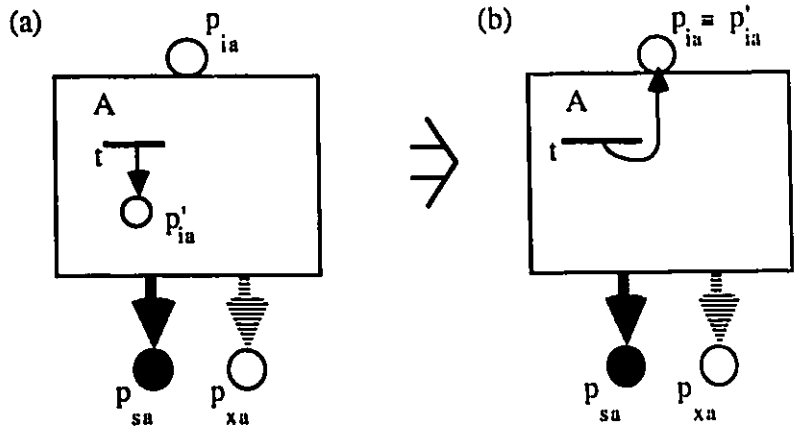


Figure 4.18 I-PN(A) with a recursive process

It should be pointed that, such an arrangement for representing recursions is just for the purpose of obtaining the complexity measurement of an I-PN. Obviously, this is not precise for representing a recursive execution. For such a purpose, each call to recursive process A should introduce a new copy of I-PN(A) to work with, so that tokens at different levels of calling will not be mixed up. However, detail of this is out of the scope of this thesis.

Example 4.7

A := a; A [] b; exit

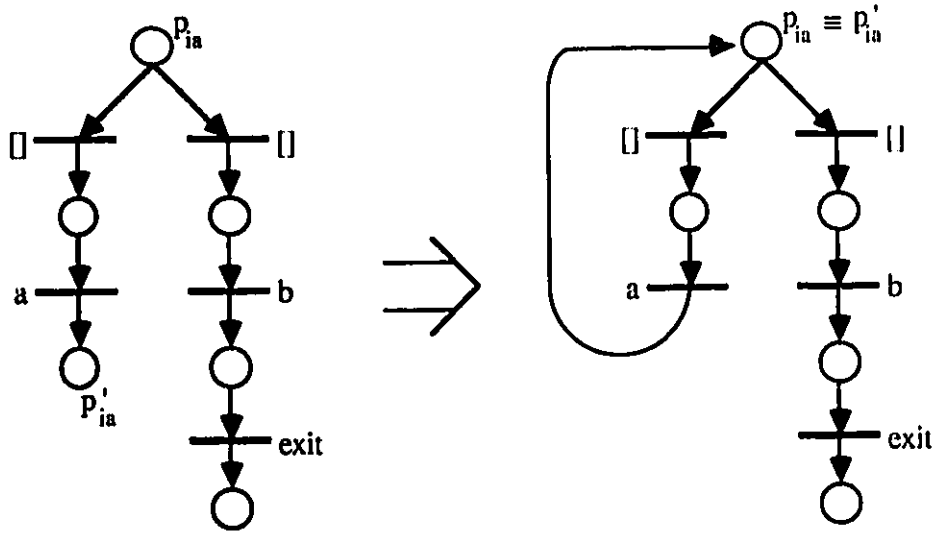


Figure 4.19 An example of I-PN(A) with a recursive process

4.4 SIMPLIFICATION OF THE I-PN REPRESENTATIONS

Some of the places and empty transitions created in the procedures for constructing an I-PN representation are useless for representing the control flow and executing the I-PNs. In this section, we introduce two rules for discarding them. Rule S1 can be used to simplify the representations for enable (>>) and choice ([]) operators, and Rule S2 for the disable ([>), interleaving (|||), and parallel (|G|) operators.

(a). Rule **S1**. If an empty transition ε that has a unique incoming place p and a unique outgoing place p' where p' is not the initial place of a recursion process, then ε and its incident arcs can be discarded (Figure 4.20) by combining the places p' and p.

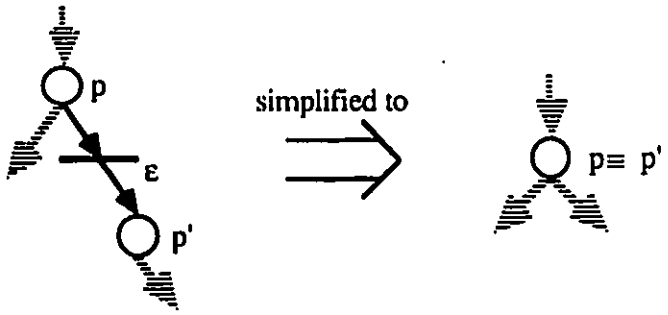


Figure 4.20 Simplification Rule S1

Example 4.9

Consider S = a; (A [] b; exit)

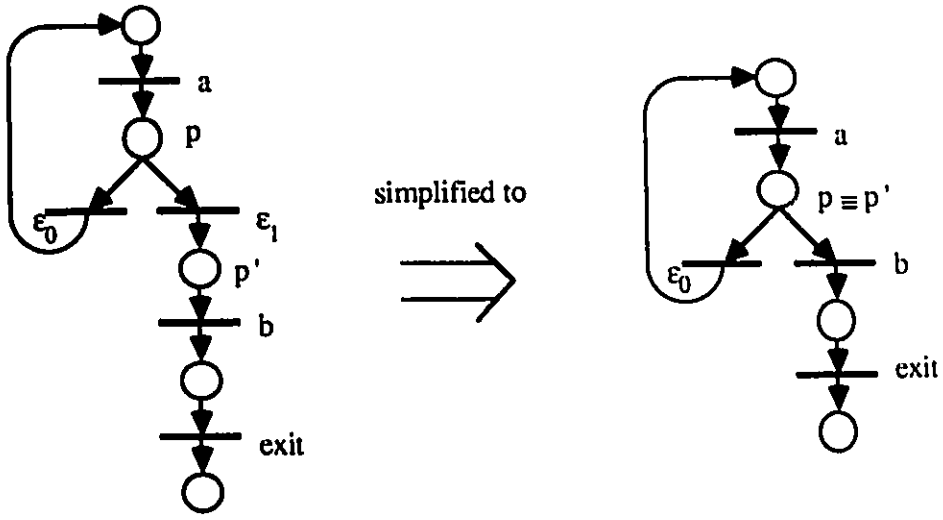


Figure 4.21 An example of simplification by Rule S1

(b). Rule **S2.** If a place p is connected to a unique incoming transition ε and a unique outgoing transition ε', where ε and ε' are empty transitions, then the place p and its incident arcs can be discarded (Figure 4.22) by combining the transitions ε and ε'.

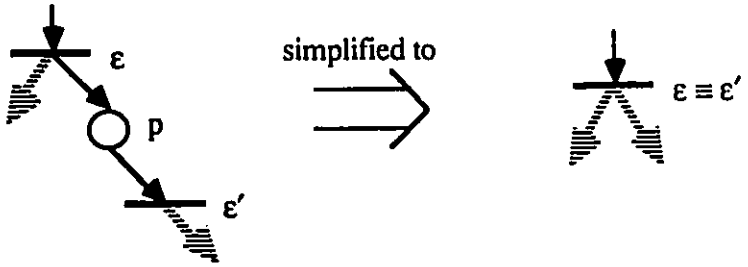


Figure 4.22 Simplification Rule S2

Example 4.10

S = (a; exit ||| b; exit) |[c; exit

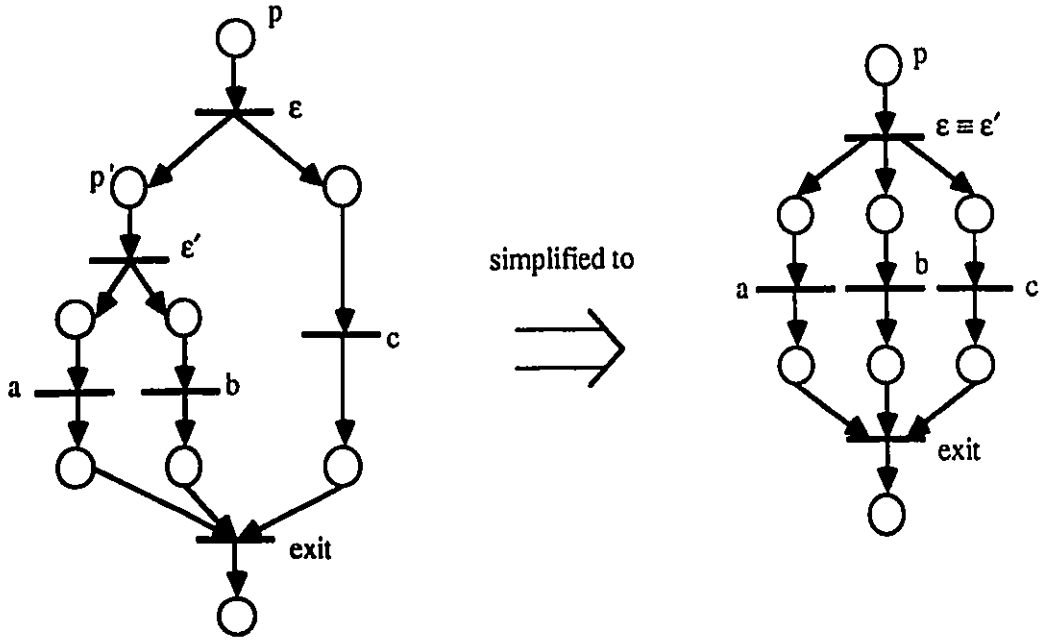


Figure 4.23 An example of simplification by Rule S2

4.5 FORMAL DEFINITION OF I-PN REPRESENTATIONS OF LOTOS EXPRESSIONS

This section includes the formal descriptions of the I-PN representations of the elementary LOTOS expressions. In these descriptions, the symbol ϕ is used to denote the non-existence of a terminal place.

(a). I-PN(stop) for Inaction **stop** :

$P(stop) = \{p_0, p_1\}$

$T(stop) = \{t_0\}$

$E(stop) = \{(p_0, t_0), (t_0, p_1)\}$

$L(stop) = \{\text{"stop"}\}$

$l(t_0) = \text{'stop'}$

$p_i = p_0$, $p_x = \phi$, $p_s = p_1$.

(b). I-PN(exit) for Successful Termination **exit** :

$P(exit) = \{p_0, p_1\}$

$T(exit) = \{t_0\}$

$E(exit) = \{(p_0, t_0), (t_0, p_1)\}$

$L(exit) = \{"exit"\}$

$l(t_0) =$ 'exit'

$p_i = p_0$, $p_x = p_1$, $p_s = \phi$.

(c). I-PN(a; A) for Action Prefix **a; A** :

$P(a; A) = P(A) \cup \{p_0\}$

$T(a; A) = T(A) \cup \{t_0\}$

$E(a; A) = E(A) \cup \{(p_0, t_0), (t_0, p_{ia})\}$

$L(a; A) = L(A) \cup \{"a"\}$

$l(t_0) =$ 'a'

$p_i = p_0$, $p_x = p_{xa}$, $p_s = p_{su}$.

(d). I-PN(A [] B) for Choice **A [] B** :

$P(A [] B) = P(A) \cup (P(B) - \{p_{xb}, p_{sb}\}) \cup \{p_0\}$

$T(A [] B) = T(A) \cup T(B) \cup \{\varepsilon_0, \varepsilon_1\}$

$E(A [] B) = E(A) \cup E(B) \cup \{(p_0, \varepsilon_0), (p_0, \varepsilon_0), (\varepsilon_0, p_{ia}), (\varepsilon_1, p_{ib})\}$

$L(A [] B) = L(A) \cup L(B) \cup \{"[]"\}$

$l(\varepsilon_0) = l(\varepsilon_1) =$ '[]'

$p_i = p_0$, $p_x = p_{xa}$, $p_s = p_{su}$.

(e). I-PN(A >> B) for Enabling **A >> B** :

Case 1. $p_{xa} = \phi$

I-PN(A >> B) is the same as I-PN(A)

Case 2. $p_{xa} \neq \phi$

$P(A >> B) = (P(A) - \{p_{sb}\}) \cup P(B)$

$T(A >> B) = T(A) \cup T(B) \cup \{\varepsilon\}$

$E(A >> B) = E(A) \cup E(B) \cup \{(p_{xa}, \varepsilon), (\varepsilon, p_{ib})\}$

$L(A >> B) = L(A) \cup L(B) \cup \{">>"\}$

$l(\varepsilon) = '>>'$

$p_i = p_{ia}$, $p_x = p_{xb}$, $p_s = p_{sa}$.

(f). I-PN(A [> B) for Disabling **A [> B** :

$P(A [> B) = P(A) \cup (P(B) - \{p_{xb}, p_{sb}\}) \cup \{p_0\}$

$T(A [> B) = T(A) \cup T(B) \cup \{\varepsilon\}$

$E(A [> B) = E(A) \cup E(B) \cup E([>)$

where $E([>) = \{(p_{ib}, t), (t, p_{ib}) \mid \forall\, t \in (T(A) - T_t(A)), l(t) \neq '\varepsilon'\} \cup$

$\{(p_{ib}, t) \mid \forall\, t \in T_t(A)\} \cup \{(p_0, \varepsilon), (\varepsilon, p_{ia}), (\varepsilon, p_{ib})\}$

$L(A [> B) = L(A) \cup L(B) \cup \{"[>"\}$

$l(\varepsilon) = '[>'$

$p_i = p_0$, $p_x = p_{xa}$, $p_s = p_{sa}$.

(g). I-PN(A|||B) for Interleaving **A ||| B** :

Case 1. ($p_{xa} \neq \phi \wedge p_{xb} \neq \phi$)

$P(A ||| B) = P(A) \cup (P(B) - \{p_{xb}, p_{sb}\}) \cup \{p_0\}$

$T(A ||| B) = (T(A) - T_x(A)) \cup (T(B) - T_x(B)) \cup T(||) \cup \{\varepsilon\}$,

where $T(||) = \{t_{j,k} \mid \forall\, (t_j, t_k),\ t_j \in T_x(A), t_k \in T_x(B)\}$

$E(A \,|||\, B) = (E(A) - E_x(A)) \cup (E(B) - E_x(B)) \cup E(||)$,

where $E(||) = \{(p, t_{j,k}) \mid t_{j,k} \in T(||), (\exists p)(p, t_j) \in E(A) \vee (p, t_k) \in E(B)\} \cup$

$\{(t_{j,k}, p) \mid t_{j,k} \in T(||), (\exists p)(t_j, p) \in E(A) \wedge (k_k, p) \in E(B)\} \cup$

$\{(p_0, \varepsilon), (\varepsilon, p_{ia}), (\varepsilon, p_{ib})\}$

$L(A \,|||\, B) = L(A) \cup L(B) \cup \{\text{"||"}\}$

$l(\varepsilon) = \text{'||'};$

$l(t_{j,k}) = \text{'exit'}$, where $t_{j,k} \in T(||)$

$p_i = p_0$; $p_x = p_{xa}$; $p_s = p_{sa}$.

Case 2. ($p_{xa} = \phi \vee p_{xb} = \phi$)

$P(A \,|||\, B) = (P(A) - \{p_{xa}\}) \cup (P(B) - \{p_{xb}\}) \cup \{p_0\}$

$T(A \,|||\, B) = T(A) \cup T(B) \cup \{\varepsilon\}$

$E(A \,|||\, B) = E(A) \cup E(B) \cup \{(p_0, \varepsilon), (\varepsilon, p_{ia}), (\varepsilon, p_{ib})\}$

$L(A|||B) = (L(A) \cup L(B) \cup \{\text{"||"}\} - \{\text{"exit"}\}$

$l(\varepsilon) = \text{'||'};$

$l(t) = \text{'stop'}$, where $t \in T_x(A) \cup T_x(B)$

$p_i = p_0$, $p_x = \phi$, $p_s = p_{su}$.

(h). I-PN(A |G| B) for Parallel A |G| B, where $G = [g_1, g_2, ..., g_n]$:

$P(A \,|G|\, B) = P(A \,|||\, B)$

$T(A \,|G|\, B) = (T(A \,|||\, B) - T_g(G, A) - T_g(G, B)) \cup T(|G|)$

where $T(|G|) = \{t_{j,k} \mid t_{j,k} = t_j \times t_k, l(t_j) = l(t_k) \in G, t_j \in T(A), t_k \in T(B)\}$

$E(A \,|G|\, B) = (E(A \,|||\, B) - \{(p_k, t), (t, p_j) \mid t \in T(A) \cup T(B), l(t) \in G\}) \cup E(|G|)$

where $E(|G|) = \{(p, t_{j,k}) \mid t_{j,k} \in T(|G|), (p, t_j) \in E(A) \vee (p, t_k) \in E(B)\} \cup$

$\{(t_{j,k}, p) \mid t_{j,k} \in T(|G|), (t_j, p) \in E(A) \vee (t_k, p) \in E(B)\}$

$L(A \,|G|\, B) = T(A \,|||\, B)$

$p_i = p_0$, $p_x = p_{xa}$, $p_s = p_{sa}$.

(i). I-PN(A) for Recursive Process A := ... A ... :

Let I-PN(A') be the I-PN of process A before merging the place p'_{ia} to place p_{ia}. Then,

$P(A) = P(A') - \{p'_{ia}\}$

$T(A) = T(A')$

$E(A) = (E(A') - \{(t, p'_{ia}) | t \in T(A'), (t, p'_{ia}) \in E(A')\}) \cup$

$\{(t, p_{ia}) | t \in T(A'), (t, p'_{ia}) \in E(A')\}$

$L(A) = L(A')$

$p_i = p'_{ia}$, $p_x = p'_{xa}$, $p_s = p'_{sa}$.

Chapter 5

CYCLOMATIC COMPLEXITY OF INTERACTING PETRI NETS AND LOTOS EXPRESSIONS

5.1 INTRODUCTION

This chapter contains the major contribution of this thesis: a new cyclomatic complexity metric for I-PNs and LOTOS expressions. Our results on this structural metric include two parts: a general formula for computing the complexity of an I-PN and a set of constructive formulas for calculating the complexity of the primitive LOTOS operations. The general formula is derived from graph theory, and the constructive formulas are obtained by applying the general formula to the individual LOTOS operations. To apply the general formula to a LOTOS expression, one first transforms the expression to an I-PN and then counts its number of places, transitions and arcs. By means of the constructive formulas, one can calculate the cyclomatic complexity of a LOTOS expression directly from the expression itself, i.e., without the necessity of transforming the expression to an I-PN.

McCabe's cyclomatic complexity is the most well-known structural metric, which is based on the control flowgraph of a non-distributed system. In order to apply it to a distributed system, we have to find a graph-based representation for the system. In Chapter 4, we propose the I-PN model for this purpose in the case of LOTOS. Similar to McCabe's approach, we use the cyclomatic number of the I-PN as the structural complexity of the corresponding LOTOS expression (Figure 5.1).

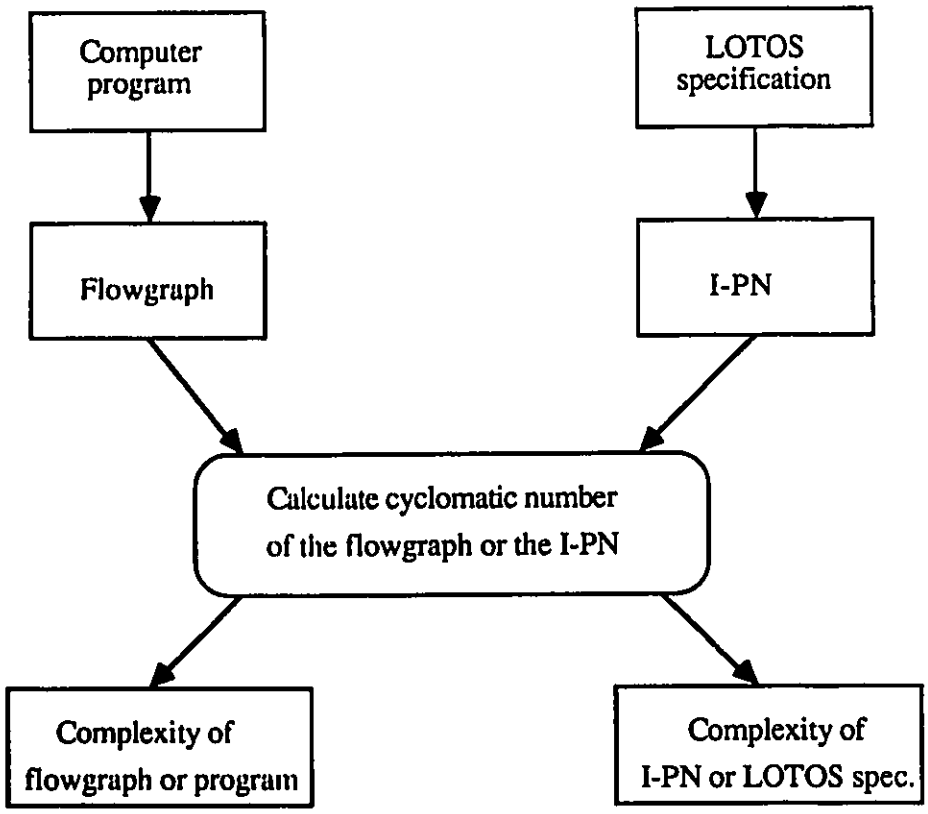


Figure 5.1 The cyclomatic complexity of computer programs and LOTOS specifications

The rest of this chapter is organized as follows. Section 5.2 briefly reviews the formulas and theorems for the cyclomatic number and its application as a complexity measure. In Section 5.3, we derive a general formula for calculating the cyclomatic complexity of an I-PN and apply it to calculate the complexity of LOTOS expressions. However, to do so, we have to generate the corresponding I-PN. For a complicated LOTOS expression, its corresponding I-PN may be huge and complicated. In order to overcome this difficulty, in Section 5.4, we introduce a series of constructive formulas, by which we can directly calculate the complexity of a LOTOS expression from the expression itself. Lastly, consistence between the constructive formulas and the general formula is proved in Section 5.5.

5.2 CYCLOMATIC COMPLEXITY OF A FLOWGRAPH

The concept of a cyclomatic number originated from graph theory [BER73]. For a graph Q with n vertices, e arcs and one connected component, Berge defines the cyclomatic number of Q as

$$v(Q) = e - n + 1$$

Generally, if Q has p connected components $Q_1, Q_2, ..., Q_p$, its cyclomatic number is

$$\begin{aligned} v(Q) &= v(Q_1) + v(Q_2) + ... + v(Q_p) \\ &= (e_1 - n_1 + 1) + (e_2 - n_2 + 1) + ... + (e_p - n_p + 1) \\ &= (e_1 + e_2 + ... + e_p) - (n_1 + n_2 + ... + n_p) + (1 + 1 + ... + 1) \\ &= e - n + p \end{aligned}$$

Definition 5.1: A *cycle* in a graph Q is a sequence of arcs $\mu = (u_1, u_2, ..., u_n)$ such that

(1) arc u_{k-1} is connected with arc u_k, where $k = 1, 2, ..., n$ and $u_0 = u_n$;

(2) $u_j \neq u_k$, if $j \neq k$.

Definition 5.2: An *elementary cycle* of a graph Q is a cycle containing no proper sub-cycles.

Definition 5.3: A set of cycles $\mu_1, \mu_2, ..., \mu_k$ in a graph Q is said to be *dependent* if there exists a linear combination of the form $r_1\mu_1 + r_2\mu_2 + ... + r_k\mu_k = 0$, where $r_1, r_2, ..., r_k$ are integers, not all zero. If the cycles are not dependent, they are said to be *independent*.

Definition 5.4: A *cycle basis* is a set of independent elementary cycles $\{\mu_1, \mu_2, \ldots, \mu_k\}$ such that any cycle μ can be written as $\mu = r_1\mu_1 + r_2\mu_2 + \ldots + r_k\mu_k$ where $r_1, r_2, \ldots, r_k$ are rational numbers. The *cyclomatic number* is the cardinality of a cycle basis. It does not depend on the choice of the basis.

Theorem 5.1 [BER73]: Let Q be a graph with n vertices, e arcs and p connected components. The cardinality of a cycle basis is $v(Q) = e - n + p$. It is the maximum number of independent cycles in Q.

McCabe developed the above theorem based on the flowgraph of a computer program. He assumed that the flowgraph has a unique entry node and a unique exit node that each node can be reached by the entry node and can reach the exit node. Then, a path in a flowgraph can be defined as follows.

Definition 5.5: Let R be a flowgraph with a unique entry node and a unique exit node. A *path* in R is a sequence of edges $\theta = (u_1, u_2, \ldots, u_n)$; such that

(1) u_1 starts at the entry node and u_n ends at the exit node,

(2) edge u_{k-1} is connected with edge u_k, where $1 < k \leq n$

In order to apply the Berge's formula to a flowgraph R, McCabe added an edge from the exit place to the entry place of a flowgraph so that the new flowgraph R' is a strongly connected and any path from the entry node to the exit node can be represented by a circuit through both the entry and the exit nodes. Suppose R has n nodes, e edges, and one

component. Then, the cyclomatic number of R' is:

$$v(R') = (e + 1) - n + 1 = e - n + 2$$

By Theorem 5.1, v(R') is equal to the maximum number of independent cycles in R'. When the added edge is removed from R', a cycle passing through both the entry and exit nodes becomes a path from the entry node to the exit node. Therefore, v(R) = v(R'), which is related to the control structure of the corresponding program. In general, if R has p components R_1, R_2, ..., R_p, the cyclomatic complexity of R is:

$$v(R) = v(R_1) + v(R_2) + ... + v(R_p) = e - n + 2\,p$$

Theorem 5.2 [MCC76]: The cyclomatic complexity of a flowgraph R is equal to the maximum number of independent paths in R.

5.3 CYCLOMATIC COMPLEXITY OF AN INTERACTING PETRI NET

In this section, we give a general formula for calculating the cyclomatic complexity of an I-PN.

Theorem 5.3: Let S be a LOTOS specification. The cyclomatic number of I-PN(S) is given by the following formula:

F $v(S) = |E(S)| - |N(S)| + q + 1$

where |E(S)| is the number of arcs in I-PN(S), |N(S)| is the number of nodes (places and transitions) in I-PN(S) and q is the number of outlets of I-PN(S).

Proof: In Section 4.2, we classify I-PNs into four types, according to its number of outlets. To derive formula F, we have to consider the folowing three cases (Figure 5.2).

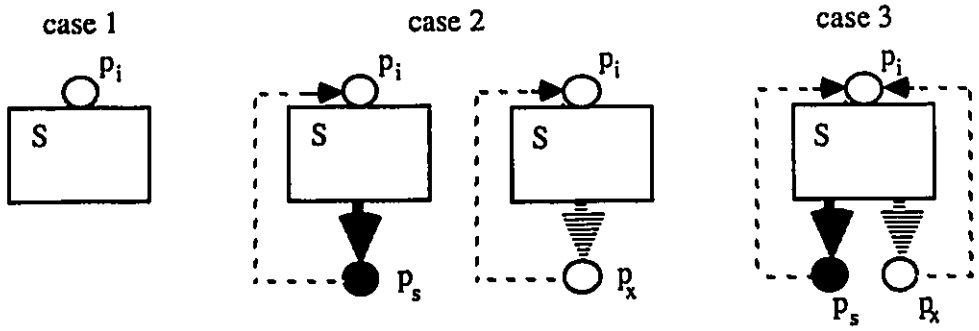


Figure 5.2 Three cases of I-PN(S)

Case 1: I-PN(S) has no outlet. In this case, no arc is to be added to the I-PN(S). By Berge's formula, we have

F1 $v(S) = |E(S)| - |N(S)| + 1$

Case 2: I-PN(S) has one outlet, p_s or p_x. In this case, an arc $p_s \rightarrow p_i$ or $p_x \rightarrow p_i$ is added to connect the outlet to the initial place. By McCabe's formula, we have

F2 $v(S) = (|E(S)| + 1) - |N(S)| + 1 = |E(S)| - |N(S)| + 2$

Case 3: I-PN(S) has two outlets, p_s and p_x. In this case, two arcs $p_s \rightarrow p_i$ and $p_x \rightarrow p_i$ are added to connect the outlets to the initial place. The cyclomatic complexity of I-PN(S) is

F3 $v(S) = (|E(S)| + 2) - |N(S)| + 1 = |E(S)| - |N(S)| + 3$

By combining formulas F1, F2, and F3, we conclude that the cyclomatic complexity v(S) of I-PN(S) is

F $v(S) = |E(S)| - |N(S)| + q + 1$

where q is the number of outlets of I-PN(S).

From Section 4.5, we notice that each simplification rule will result in discarding two nodes (one place and one transition) and two arcs from the I-PN. Hence, by the general formula F, the complexity of the simplified I-PN remains unchanged.

5.4 CYCLOMATIC COMPLEXITY OF LOTOS EXPRESSIONS

In order to apply the general formula F to a LOTOS expression S, we have to first generate the corresponding I-PN(S) and then count its number of nodes, arcs, and outlets. However, by studying the basic LOTOS operations individually, we have developed a series of constructive formulas for calculating the cyclomatic complexity directly from the primitive LOTOS expressions without generating I-PN(S) (Figure 5.3).

No.	S	v(S) (Cyclomatic Complexity of S)
CF1	stop	1
CF2	exit	1
CF3	a; A	v(A)
CF4	A [] B	v(A) + v(B)
CF5	A >> B	$v(A) + \|\{p_{xa}\}\|(v(B) - 1)$
CF6	A [> B	$v(A) + v(B) + 2\|\Gamma''(A)\| - \|\Gamma_t(A)\|$
CF7	A ||| B	$v(A)+v(B)+\|\{p_{xa}\}\|\|\{p_{xb}\}\|\|(\|E_x(A)\|-\|\Gamma_x(A)\|)(\|\Gamma_x(B)\|-1)+(\|E_x(B)\|-\|\Gamma_x(B)\|)(\|\Gamma_x(A)\|-1)\|$
CF8	A |G| B	$v(A\|\|\|B) + \|E(G,A\|G\|B)\| - \|E(G,A)\| - \|E(G,B)\| - \|\Gamma(G,A\|G\|B)\| + \|\Gamma(G,B)\| + \|\Gamma(G,A)\|$
CF9	S'	1 where: S' is a recursion of S.

Figure 5.3 Constructive formulas for the cyclomatic complexity of a LOTOS expression

The notations used in Figure 5.3 are defined as follows:

T'(S): the set of non-empty transitions in I-PN(S);

$T_t(S)$: the set of "stop" and "exit" transitions in I-PN(S);

$T_x(S) = \{ t \mid t \in T(S), l(t) = \text{'exit'}\}$, the set of "exit" transitions;

$T(G, S) = \{t \mid t \in T(S), l(t) \in G$, where G is a set of gates$\}$, the set of transitions with labels in G.

$E_x(S) = \{(p,t), (t,p) \mid (p,t), (t,p) \in E(S), t \in T_x(S)\}$, the set of all arcs adjacent to an "exit" transition.

$E(G, S) = \{(p,t), (t,q) \mid (p,t), (t,q) \in E(S), l(t) \in G$, where G is a set of gates$\}$, the set of all arcs adjacent to an transition in T(G,S);

The cardinality of the above sets can be calculated by applying the following rules repeatedly:

$$|T(G, S)| = \sum_{g \in G} |T(\{g\}, S)|;$$

$$|E(G, S)| = \sum_{g \in G} |E(\{g\}, S)|;$$

Exp.1 For S = stop (Figure 4.4) :

$|T'(S)| = 1$, $|T_t(S)| = 1$, $|T_x(S)| = 0$, $|E_x(S)| = 0$, $|T(G,S)| = 0$, and $|E(G, S)| = 0$.

Exp.2 For S = exit (Figure 4.5) :

$|T'(S)| = 1$, $|T_t(S)| = 1$, $|T_x(S)| = 1$, $|E_x(S)| = 2$, $|T(G,S)| = 0$, and $|E(G, S)| = 0$.

Exp.3 For S = a; A (Figure 4.6) :

$|T'(S)| = |T'(A)| + 1$, $|T_t(S)| = |T_t(A)|$, $|T_x(S)| = |T_x(A)|$, $|E_x(S)| = |E_x(A)|$.

$|T(\{a\},S)| = |T(\{a\}, A)| + 1$, and $|E(\{a\},S)| = |E(\{a\}, A)| + 2$.

Exp.4 For S = A || B (Figure 4.7) :

$|T'(S)| = |T'(A)| + |T'(B)|$, $|T_t(S)| = |T_t(A)| + |T_t(B)|$,

$|T_x(S)| = |T_x(A)| + |T_x(B)|$, $|E_x(S)| = |E_x(A)| + |E_x(B)|$,

$|T(G,S)| = |T(G, A)| + |T(G, B)|$, and $|E(G, S)| = |E(G, A)| + |E(G, B)|$.

Exp.5 For S = A >> B (Figure 4.9) :

Case 1. $\{p_{xa}\} = \phi$,

$|T'(S)| = |T'(A)|$, $|T_t(S)| = |T_t(A)|$, $|T_x(S)| = |T_x(A)|$, $|E_x(S)| = |E_x(A)|$,

$|T(G, S)| = |T(G, A)|$, and $|E(G, S)| = |E(G, A)|$.

Case 2. $\{p_{xa}\} \neq \phi$,

$|T'(S)| = |T'(A)| + |T'(B)|$; $|T_t(S)| = |T_t(A)| - |T_x(A)| + |T_t(B)|$,

$|T_x(S)| = |T_x(B)|$, $|E_x(S)| = |E_x(B)|$,

$|T(G, S)| = |T(G, A)| + |T(G, B)|$, and $|E(G, S)| = |E(G, A)| + |E(G, B)|$.

Exp.6 For S = A |> B (Figure 4.11) :

$|T'(S)| = |T'(A)| + |T'(B)|$, $|T_t(S)| = |T_t(A)| + |T_t(B)|$,

$|T_x(S)| = |T_x(A)| + |T_x(B)|$, $|E_x(S)| = |E_x(A)| + |E_x(B)| + |T_x(A)|$,

$|T(G, S)| = |T(G, A)| + |T(G, B)|$, and $|E(G, S)| = |E(G, A)| + |E(G, B)| + 2|T(G, A)|$.

Exp.7 For S = A ||| B (Figure 4.13) :

Case 1. If $|T_x(A)| \neq 0$ and $|T_x(B)| \neq 0$,

$|T'(S)| = |T'(A)| + |T'(B)| + |\Gamma_x(A)|\,|T_x(B)| - |T_x(A)| - |T_x(B)|$,

$|T_t(S)| = |T_t(A)| + |T_t(B)| + |\Gamma_x(A)|\,|T_x(B)| - |T_x(A)| - |T_x(B)|$,

$|T_x(S)| = |T_x(A)|\,|T_x(B)|$,

$|E_x(S)| = |E_x(A)|\,|\Gamma_x(B)| + |E_x(B)|\,|T_x(A)| - |T_x(A)|\,|T_x(B)|$,

$|T(G, S)| = |T(G, A)| + |T(G, B)|$,

$|E(G, S)| = |E(G, A)| + |E(G, B)|$.

Case 2. If $|T_x(A)| = 0$ or $|\Gamma_x(B)| = 0$,

$|T'(S)| = |T'(A)| + |T'(B)|$, $|T_t(S)| = |T_t(A)| + |T_t(B)|$, $|T_x(S)| = 0$, $|E_x(S)| = 0$,

$|T(G, S)| = |T(G, A)| + |T(G, B)|$, and $|E(G, S)| = |E(G, A)| + |E(G, B)|$.

Exp.8 For S = A |G| B (Figure 4.16) :

$|T'(S)| = |T'(A \,|||\, B)| - |T(G, A)| - |T(G, B)| + |T(G, A)|\,|T(G, B)|$,

$|T_t(S)| = |T_t(A \,|||\, B)|$, $|T_x(S)| = |T_x(A \,|||\, B)|$, $|E_x(S)| = |E_x(A \,|||\, B)|$,

$$|T(\{g\}, S)| = \begin{cases} |T(\{g\}, A)|\,|T(\{g\}, B)| & \text{where } g \in G; \\ |T(\{g\}, A)| + |T(\{g\}, B)| & \text{where } g \notin G; \end{cases}$$

$$|E(\{g\}, S)| = \begin{cases} |E(\{g\}, A)|\,|T(\{g\}, B)| + |E(\{g\}, B)|\,|T(\{g\}, A)| & \text{where } g \in G \\ |E(\{g\}, A)| + |E(\{g\}, B)| & \text{where } g \notin G \end{cases}$$

Exp.9 For S = ...S' where S' is a recursive ocrrence of S (Figure 4.18) :

$|T'(S')| = 0$, $|T_t(S')| = 0$, $|T_x(S')| = 0$, $|E_x(S')| = 0$, $|T(G, S')| = 0$, and $|E(G, S')| = 0$.

5.5 PROOF OF THE CONSTRUCTIVE FORMULAS

The constructive formulas in Table 5.1 can be derived from the general formula F (Section 5.3). Based on the general formula F, the following proofs will be discussed in terms of the four types of I-PNs (Section 4.2). The corresponding figures of I-PN(S) used in the following proofs can be found in Section 4.3.

Proof CF1 and CF2 (Figure 4.4 and Figure 4.5) :

In both I-PN(stop) and I-PN(exit), $|N(S)| = 3$, $|E(S)| = 2$, $q = 1$. Then,

$v(stop) = v(exit) = |E(S)| - |N(S)| + q + 1 = 2 - 3 + 1 + 1 = 1$

Proof of CF3 (Figure 4.6) :

Suppose that I-PN(A) has q outlets ($q = 0, 1, 2$). Then,

$v(a; A) = (|E(A)| + 2) - (|N(A)| + 2) + q + 1$

$= |E(A)| - |N(A)| + q + 1 = v(A)$

Proof of CF4 (Figure 4.7) :

Suppose that I-PN(A) has q_1 outlets, I-PN(B) has q_2 outlets, and I-PN(A[]B) has q outlets. Then,

$v(A [] B) = (|E(A)| + |E(B)| + 4) - (|N(A)| + |N(B)| + 3 - (q_1 + q_2 - q)) + q + 1$

$= (|E(A)| - |N(A)| + q_1 + 1) + (|E(B)| - |N(B)| + q_2 + 1)$

$= v(A) + v(B)$

Proof of CF5 (Figure 4.9) :

Suppose that I-PN(A) has q_1 outlets, I-PN(B) has q_2 outlets, and I-PN(A>>B) has q

outlets. Two cases should be considered:

Case 1. $p_{xa} = \phi$. i.e., $|\{p_{xa}\}| = 0$.

$v(A >> B) = v(A)$

Case 2. $p_{xa} \neq \phi$. i.e., $|\{p_{xa}\}| = 1$.

$$v(A >> B) = (|E(A)| + |E(B)| + 2) - (|N(A)| + |N(B)| + 1 - (q_1 + q_2 - q - 1)) + q + 1$$

$$= (|E(A)| - |N(A)| + q_1 + 1) + (|E(B)| - |N(B)| + q_2 + 1) - 1$$

$$= v(A) + v(B) - 1$$

Proof of CF6 (Figure 4.11) :

Suppose that I-PN(A) has q_1 outlets, I-PN(B) has q_2 outlets, and I-PN(A[>B) has q outlets. Then,

$$v(A [> B) = (|E(A)| + |E(B)| + 2|T'(A)| - |T_t(A)| + 3) - (|N(A)| + |N(B)| + 2 - (q_1 + q_2 - q)) + q + 1$$

$$= (|E(A)| - |N(A)| + q_1 + 1) + (|E(B)| - |N(B)| + q_2 + 1) + 2|T'(A)| - |T_t(A)|$$

$$= v(A) + v(B) + 2|T''(A)| - |T_t(A)|$$

Proof of CF7 (Figure 4.13) :

Suppose that I-PN(A) has q_1 outlets, I-PN(B) has q_2 outlets, and I-PN(A ||| B) has q outlets. Two cases should be considered.

Case 1. $p_{xa} = \phi$ or $p_{xb} = \phi$. Then, $|\{p_{xa}\}||\{p_{xb}\}| = 0$.

$$v(A ||| B) = (|E(A)| + |E(B)| + 3) - (|N(A)| + |N(B)| + 2 - (q_1 + q_2 - q)) + q + 1$$

$$= (|E(A)| - |N(A)| + 1) + (|E(B)| - |N(B)| + 1)$$

$$= v(A) + v(B)$$

Case 2. $p_{xa} \neq \phi$ and $p_{xb} \neq \phi$. Then, $|\{p_{xa}\}||\{p_{xb}\}| = 1$.

$$v(A \,|||\, B) = (|E(A)| + |E(B)| + |T_x(A)|\,|E_x(B)| + |T_x(B)|\,|E_x(A)| - |T_x(A)|\,|T_x(B)| -$$
$$|E_x(A)| - |E_x(B)| + 3) - (|N(A)| + |N(B)| - |T_x(A)| - |T_x(B)| +$$
$$|T_x(A)||T_x(B)| - (q_1 + q_2 - q) + 2) + q + 1$$
$$= (|E(A)| - |N(A)| + q_1 + 1) + (|E(B)| - |N(B)| + q_2 + 1) +$$
$$(|E_x(A)| - |T_x(A)|)(|T_x(B)| - 1) + (|E_x(B)| - |T_x(B)|)(|T_x(A)| - 1)$$
$$= v(A) + v(B) + (|E_x(A)| - |T_x(A)|)(|T_x(B)| - 1) + (|E_x(B)| - |T_x(B)|)(|T_x(A)| - 1)$$

Proof of CF8 (Figure 4.16) :

Suppose that I-PN(A |G| B) has q outlets. Then,

$$v(A \,|G|\, B) = (|E(A \,|||\, B)| + |E(G, A|G|B)| - |E(G,A)| - |E(G,B)|) -$$
$$(|N(A \,|||\, B)| + |T(G, A|G|B)| - |T(G,B)| - |T(G,A)|\,|) + q + 1$$
$$= (|E(A \,|||\, B)| - |N(A \,|||\, B)| + q + 1) + |E(G, A|G|B)| - |E(G,A)| - |E(G,B)| -$$
$$|T(G, A|G|B)| + |T(G,B)| + |T(G,A)|$$
$$= v(A|||B) + |E(G, A|G|B)| - |E(G, A)| - |E(G, B)| - |T(G, A|G|B)| + |T(G, B)| + |T(G, A)|$$

Proof of CF9 (Figure 4.18) :

In an I-PN, a recursive process S := ...S... is represented by merging the initial place p_{iu} and the recursive place p'_{is} (i.e. $p_{is} \equiv p'_{is}$). This merging operation will result in deleting the place p'_{is} from the corresponding I-PN(S) (i.e., v(S) will increase by one). Hence, we assign the value 1 to each recursive occurrence.

Chapter 6

A COVERAGE CRITERION FOR PATH TESTING OF LOTOS

6.1 INTRODUCTION

In this chapter, as another contribution of this thesis, a path testing coverage criterion for the selection of test sequences from a LOTOS specification is proposed. Path testing is a structural testing technique depending on a set of paths selected from a graph-based representation of the program or specification. Hence in order to derive our criterion, an I-PN is used to represent the control structure of a distributed system (i.e., functionality). The criterion covers all the elements (i.e., transitions, places and arcs) of the I-PN.

The process of testing an implementation to determine its conformance with the specification is known as *conformance testing* [ISO9646]. It involves applying test suites on the implementation. Our coverage criterion may be used to assist in the selection of test suites.

At this stage of our research, similar to McCabe's cyclomatic complexity, the cyclomatic complexity developed for I-PNs and LOTOS specifications can only be used to determine the maximum number of cycle paths in a basis set that covers all the strategic elements of an I-PN. It cannot be used to find these paths. Actual selection of these paths still depends on a human walkthrough of the I-PN.

The rest of this chapter is organized as follows. In Section 6.2, we briefly describe the basic concepts of path testing coverage criteria. In section 6.3, we introduce the concepts of cycle paths and basis sets of cycle paths for an I-PN, and derive the relationship between the

cyclomatic complexity and a basis set (Theorem 6.1). Lastly, in Section 6.4, we explain how to apply the coverage criterion for path testing.

6.2 PATH TESTING CRITERIA

A *path* in a control flowgraph of a program is a sequence of nodes and links that starts at one node (process block, decision, or junction) and ends at another (possibly the same). A path may go through some processes, decisions, or junctions one or more times. Hence, even a small program may have millions of distinct paths.

A path testing criterion must include all nodes or branches in all directions at least once. For this purpose, the following three basic coverage criteria for path testing are explored:

1). Path Coverage (C_∞): Execute all possible control flow paths through the flowgraph. Typically, this is restricted to all possible entry-to-exit paths. This is the strongest criterion in the family of path testing (In practice, it is impossible to achieve in most cases).

2). Node Coverage (C_1): Execute all nodes in the flowgraph at least once. This is the weakest criterion in the family of path testing. Testing less than this for new software is unconscionable and should be criticized.

3). Branch Coverage (C_2): Execute all branches in the flowgraph at least once. Enough tests should be used to assure that every branch alternative is exercised at least once.

Criteria C_1 have been adopted by IEEE as a minimum testing requirement for unit test. If a set of test sequences based on one criterion is a subset of that based on another criterion, the

latter criterion is said to be stronger than the former one. Obviously, C_1 is a subset of C_2. Thus, we say that criterion C_2 is stronger than criterion C_1. It must be noticed that there might be an infinite number of coverage strategies between C_1 and C_∞.

6.3 CYCLE PATHS IN AN I-PN

For a LOTOS specification of a system, its I-PN is used as a transaction flowgraph expressing the behavior of the system. A path and a cycle path in I-PN(S) are defined as follows:

Definition 6.1 Let S be a LOTOS specification. A *path* σ in I-PN(S) is a sequence of arcs:

$\sigma = (\tau_1, \tau_2, ..., \tau_{2n-1}, \tau_{2n})$ where $\tau_{2k-1} = (p_k, t_k) \in E(S)$ and $\tau_{2k} = (t_k, p_{k+1}) \in E(S)$, $k = 1, 2, ..., n$.

Definition 6.2: A *cycle path* σ in I-PN(S) is a path, such that

(a). $p_1 = p_i$;

(b). $p_n \in \{p_i, p_x, p_s\}$;

(c). $p_k \notin \{p_i, p_x, p_s\}$, $1 < k < n$;

where p_i is the initial place of I-PN(S), p_x is the exit place of I-PN(S) and p_s is the stop place of I-PN(S)

According to this definition, a cycle path in an I-PN is either a path from the initial place to a terminal place or a cycle that starts and ends at the initial place. By Condition (c), none of the

special places (p_i, p_x or p_s) appears between the first place and the last place.

Definition 6.3: A set of cycle paths $\sigma_1, \sigma_2, \ldots, \sigma_k$ in the I-PN(S) is said to be *dependent* if there exists a linear combination of the form

$$r_1\sigma_1 + r_2\sigma_2 + \ldots + r_k\sigma_k = 0$$

where $r_1, r_2, \ldots, r_k$ are integers, not all zero. If the set of cycle paths are not dependent, it is said to be *independent.*

Definition 6.4: A *basis set* of cycle paths is a set of independent cycle paths $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ such that any other cycle path σ can be expressed in the form

$$\sigma = r_1\,\sigma_1 + r_2\,\sigma_2 + \ldots + r_k\,\sigma_k$$

where $r_1, r_2, \ldots, r_k$ are rational numbers.

Next, let us concern the relationship between a basis set and the cyclomatic complexity of I-PN(S). Three cases have been considered in Theorem 5.3 for the calculation of the cyclomatic complexity of I-PN(S). In Case 1, since there is no terminal place in I-PN(S), v(S) is equal to the maximum number of independent cycles in I-PN(S). In Cases 2 and 3, v(S) is equal to the maximum number of independent cycle paths, which start from the initial place p_i and end at a terminal place p_x or p_p, in I-PN(S). These results lead to the following theorem.

Theorem 6.1: For a LOTOS expression S, the cyclomatic complexity of I-PN(S) is equal to the maximum number of linearly independent cycle paths in I-PN(S).

6.4 A PATH TESTING COVERAGE CRITERION

Definition 6.5. For a LOTOS specification S, a test sequence x of S is said to *covers* a cycle path σ in I-PN(S) iff the trace of x includes all the arcs of σ and these arcs appear in the same order as that in σ regardless of other arcs among them.

Obviously, a basis set of cycle paths in an I-PN includes all nodes and arcs of the I-PN. Therefore, we can use a basis set of cycle paths as a path testing coverage criterion (C_{cp}). It is a stronger criterion than C_1 and C_2. If a set of test sequences covers all cycle paths of a basis set, this set of test sequences is said to satisfy the criterion C_{cp}.

To apply this coverage criterion on a LOTOS specification S, we must first construct the I-PN(S), then select a basis set of cycle paths, and finally generate a set of test sequences covering the basis set. These steps are illustrated in the following example.

Example 6.1 Consider the LOTOS expression:

S : = A [> c; exit where A := a; b; exit

Step 1. Construct I-PN(S) from S and calculate its cyclomatic complexity (Figure 6.1).

In Figure 6.1, e = 15, n = 11, m = 1. By the general formula F of Section 5.3, we have

v(S) = e - n + m +1 = 15 - 11 + 1 + 1 = 6

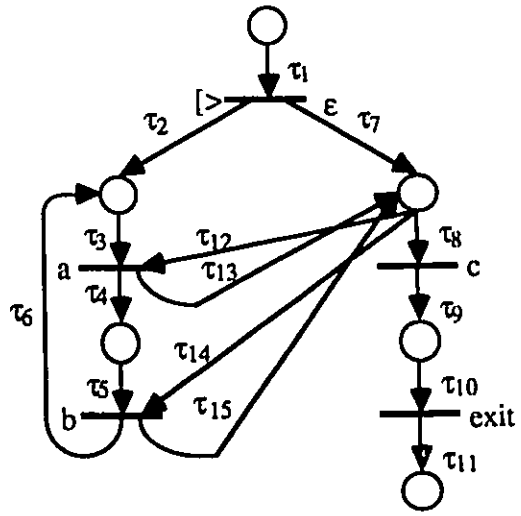


Figure 6.1 An example for generating test sequences of I-PN(S)

Step 2. Select a basis set of cycle paths from I-PN(S).

By Theorem 6.1, the maximum number of linearly independent cycle paths in I-PN(S) is 6. One basis set of cycle paths in I-PN(S) shown below:

σ_1: (τ_1, τ_7, τ_8, τ_9, τ_{10}, τ_{11})

σ_2: (τ_1, τ_2, τ_3, τ_{13}, τ_8, τ_9, τ_{10}, τ_{11})

σ_3: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_{15}, τ_8, τ_9, τ_{10}, τ_{11})

σ_4: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_3, τ_{13}, τ_8, τ_9, τ_{10}, τ_{11})

σ_5: (τ_1, τ_7, τ_{12}, τ_{13}, τ_8, τ_9, τ_{10}, τ_{11})

σ_6: (τ_1, τ_7, τ_{14}, τ_{15}, τ_8, τ_9, τ_{10}, τ_{11})

By Definition 6.4, any other cycle path can be expressed as a linear combination of these cycle paths. For example,

For σ_7: (τ_1, τ_7, τ_{12}, τ_4, τ_5, τ_{15}, τ_8, τ_9, τ_{10}, τ_{11}), we have

$$\sigma_7 = \sigma_3 + \sigma_5 - \sigma_2$$

For σ_8: (τ_1, τ_2, τ_3, τ_{13}, τ_{14}, τ_6, τ_3, τ_{13}, τ_8, τ_9, τ_{10}, τ_{11}), we have

$$\sigma_8 = \sigma_2 + \sigma_4 + \sigma_6 - \sigma_1 - \sigma_3$$

Step 3. Generate test sequences from the basis set.

T_1: (c, exit) covers σ_1,

T_2: (a, c, exit) covers σ_1, σ_2 and σ_5,

T_3: (a, b, c, exit) covers σ_1, σ_2, σ_3, σ_5 and σ_6,

T_4: (a, b, a, c, exit) covers σ_1, σ_2, σ_3, σ_4, σ_5 and σ_6.

In the test sequence generation, we try to use different sequences to cover different cycle paths. The strategy of test sequence selection is left for our future research.

Chapter 7

COMPLEXITY OF THE LOTOS SPECIFICATION OF THE TRANSPORT PROTOCOL (CLASS 0)

7.1 INTRODUCTION

The applicability of a complexity metric is one of its important features. In this chapter, by applying it to the ISO Transport Protocol (Class 0), we show that the complexity metric proposed in this thesis and the related results can be easily adopted in practice. For comparison and illustration purposes, the cyclomatic complexity of its LOTOS specification is calculated by first by applying the general formula F on the I-PN representation and then by applying the constructive formulas on the LOTOS specification. Also, a basis set of cycle paths is obtained from the simplified I-PN.

7.2 THE CYCLOMATIC COMPLEXITY OF HANDLER

As shown in Figure 7.1, the LOTOS specification HANDLER of the Transport Protocol (Class 0) [BOL87] has five processes: CONNECTION, CALLING, CALLED, DATATRANS and TERMINATION. They include eight observable interactions: conreq, conind, conres, concnf, datreq, datind, disreq and disind.

As illustration, we apply the two methods developed in Section 5.3 and Section 5.4 to calculate the cyclomatic complexity of the LOTOS specification of HANDLER.

```
specification HANDLER:noexit
behavior
    (CONNECTION >> (DATATRANS [> TERMINATION)) >> HANDLER
where
    process CONNECTION := (i; CALLING) [] CALLED
    where
        process CALLING := conreq; (concnf; exit [] disind; CONNECTION)
        endproc
        process CALLED := conind; (i; conres; exit [] i; disreq; CONNECTION)
        endproc
    endproc
    process DATATRANS:noexit := (i; datreq; DATATRANS) [] (datind; DATATRANS)
    endproc
    process TERMINATION := (i; disreq; exit) [] (disind; exit)
    endproc
endproc
```

Figure 7.1 LOTOS specification HANDLER of the Transport Protocol

Method 1 (Based on an I-PN representation of the LOTOS specification, Section 5.3)

In order to apply the general formula F, we have to construct I-PN(HANDLER) by using the transformation rules of Section 4.4. The result is shown in Figure 7.2. By formula F, the cyclomatic complexity of the LOTOS specification of HANDLER is

$$v(\text{HANDLER}) = |E(\text{HANDLER})| - |N(\text{HANDLER})| + q + 1$$
$$= 73 - 61 + 0 + 1$$
$$= 13$$

Note that the number of outlet places in I-PN(HANDLER) is zero (i.e., $q = 0$).

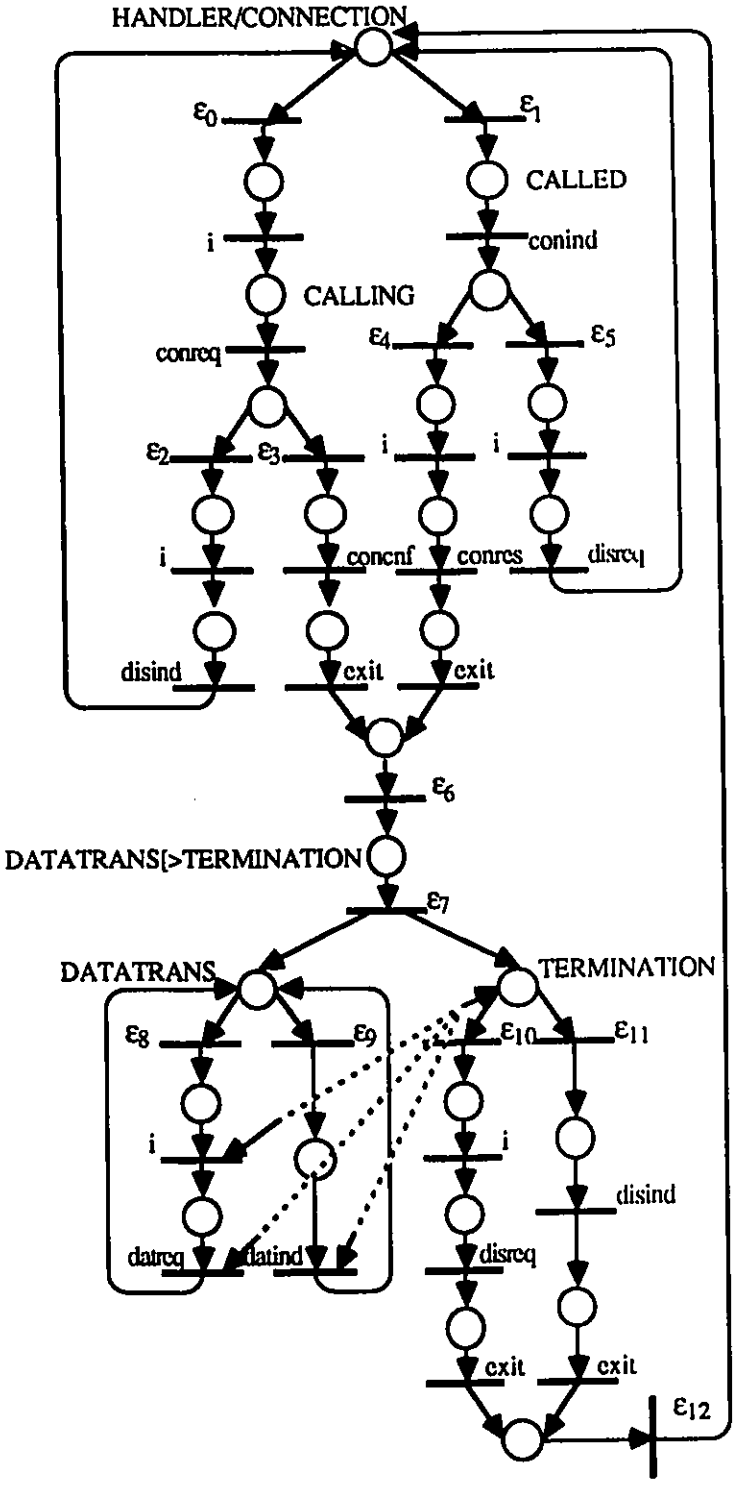


Figure 7.2 I-PN(HANDLER)

Method 2. (Based on the LOTOS specification, Section 5.4)

we apply the constructive formulas (Figure 5.3) constructing I-PN(HANDLER). To calculate the cyclomatic complexity of the LOTOS specification HANDLER,

v(CALLING)

= v(conreq; (concnf; exit [] disind; CONNECTION))

= v(concnf; exit [] disind; CONNECTION) (CF3)

= v(concnf; exit) + v(disind; CONNECTION) (CF4)

= v(exit) + v(CONNECTION) (CF3)

= 1 + v(CONNECTION) (CF1)

Similar to v(CALLING)

v (CALLED) = 1 + v(CONNECTION)

v(CONNECTION)

= v((i; CALLING) [] CALLED)

= v(i; CALLING) + v(CALLED) (CF4)

= v(CALLING) + v(CALLED) (CF3)

= 1 + v(CONNECTION') + 1 + v(CONNECTION')

= 1 + 1 + 1 + 1 (CF9)

= 4

v(DATATRANS)

= v((i; datreq; DATATRANS) [] (datind; DATATRANS))

= v(i; datreq; DATATRANS) + v(datind; DATATRANS) (CF4)

$= v(DATATRANS')+v(DATATRANS')$ (CF3)

$= 2$ (CF9)

v(TERMINATION)

$= v((i; disreq; exit) [] (disind; exit))$

$= v(i; disreq; exit) + v(disind; exit)$ (CF4)

$= v(exit) + v(exit)$ (CF3)

$= 2$

From the above results, we obtain

v(HANDLER)

$= v((CONNECTION >> (DATATRANS [> TERMINATION)) >> HANDLER)$

$= v(CONNECTION >> (DATATRANS [> TERMINATION)) + v(HANDLER') - 1$ (CF5)

$= v(CONNECTION >> (DATATRANS [> TERMINATION)) + 1 - 1$ (CF9)

$= v(CONNECTION) + v(DATATRANS [> TERMINATION) - 1$ (CF5)

$= v(CONNECTION) + v(DATATRANS) + v(TERMINATION) + 2 \times 3 - 0 - 1$ (CF6)

$= v(CONNECTION) + v(DATATRANS) + v(TERMINATION) + 5$ (CF6)

$= 4 + 2 + 2 + 5 = 13$

Note: $|T(DATATRANS)| = 3$; $|T_t(DATATRANS)| = 0$

7.3 A BASIS OF CYCLE PATHS OF I-PN(HANDLER)

In this section, we obtain a basis set of cycle paths for I-PN(HANDLER). Together, these paths cover all transitions and places of I-PN(HANDLER). Though unnecessary, we first simplify

I-PN(HANDLER) by applying Rule S1 on the empty transitions: ε_0, ε_1, ε_2, ε_3, ε_4, ε_5, ε_6, ε_8, ε_9, ε_{10}, and ε_{11}. The simplified I-PN(HANDLER) is shown in Figure 7.3.

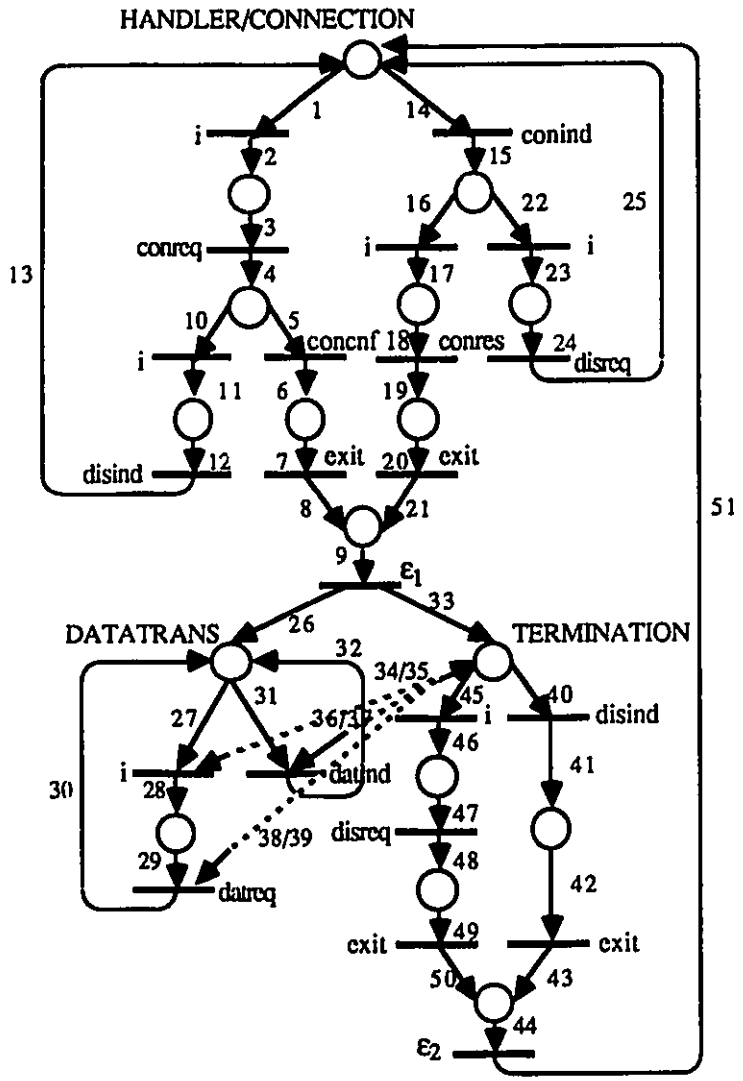


Figure 7.3 Simplified I-PN(HANDLER)

Then, we use an integer n to represent the arc τ_n in the simplified I-PN(HANDLER). By Theorem 6.1, the maximum number of independent cycle paths in I-PN(HANDLER) is 13. A

possible basis of I-PN(HANDLER) includes the following cycle paths :

P_1: (τ_1, τ_2, τ_3, τ_4, τ_{10}, τ_{11}, τ_{12}, τ_{13})

P_2: (τ_{14}, τ_{15}, τ_{22}, τ_{23}, τ_{24}, τ_{25})

P_3: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{33}, τ_{45}, τ_{46}, τ_{47}, τ_{48}, τ_{49}, τ_{50}, τ_{44}, τ_{51})

P_4: (τ_{14}, τ_{15}, τ_{16}, τ_{17}, τ_{18}, τ_{19}, τ_{20}, τ_{21}, τ_9, τ_{33}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{50})

P_5: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{33}, τ_{40}, τ_{41}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_6: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{26}, τ_{27}, τ_{39}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_7: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{26}, τ_{31}, τ_{37}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_8: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{26}, τ_{27}, τ_{28}, τ_{29}, τ_{39}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_9: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{26}, τ_{27}, τ_{28}, τ_{29}, τ_{30}, τ_{31}, τ_{37}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_{10}: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{26}, τ_{31}, τ_{32}, τ_{31}, τ_{37}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_{11}: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{33}, τ_{34}, τ_{35}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_{12}: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{33}, τ_{36}, τ_{37}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

P_{13}: (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7, τ_8, τ_9, τ_{33}, τ_{38}, τ_{39}, τ_{40}, τ_{41}, τ_{42}, τ_{43}, τ_{44}, τ_{51})

Since the above cycle paths form a basis set, any other cycle path is a linear combination of them. For example,

P: (τ_{34}, τ_{28}, τ_{29}, τ_{39}) = P_8 + P_{11} - P_6

P: (τ_{14}, τ_{15}, τ_{16}, τ_{17}, τ_{18}, τ_{19}, τ_{20}, τ_{21}, τ_9, τ_{33}, τ_{45}, τ_{46}, τ_{47}, τ_{48}, τ_{47}, τ_{48}, τ_{44}, τ_{50})
= P_3 + P_4 - P_5

Chapter 8

CONCLUSION AND FURTHER RESEARCH

In the literature, complexity measures are mainly for non-distributed systems specified in ordinary programming languages or flowgraphs. In this thesis, we embark on a new research problem concerning complexity measures for distributed systems specified in either LOTOS or Petri-nets. Our research effort results in three contributions: (1) A new Petri net model, called Interacting Petri nets (I-PNs), is proposed for representing the behavior of basic LOTOS; (2) a metric for measuring the complexity of I-PNs and LOTOS specifications; and (3) a criterion for path testing coverage.

(1) A new Petri net model for representing basic LOTOS.

In order to develop a structural metric for measuring LOTOS specifications, we first propose a Petri-net representation (I-PN) of LOTOS. The reason is not only because Petri-nets have a close relationship with CCS [MIL80], the underlying semantics of LOTOS, but also because, in our model, composite I-PNs can be composed from simple ones, exactly in the same way as composite LOTOS expressions can be constructed from simple ones.

(2) A metric for measuring the complexity of I-PNs and LOTOS specifications

Based on the I-PN representation of a LOTOS specification and the cyclomatic numbers in graph theory, we have been able to develop two approaches for calculating the cyclomatic complexity of a LOTOS expression:

a. Indirect approach: In this approach, one first transforms the LOTOS expression

to an I-PN and then counts its number of places, transitions and arcs. Berge's formula "v = e - n + p" is then applied to determine the cyclomatic complexity of the I-PN and the corresponding LOTOS specification. Obviously, the shortcoming of this approach is in the requirement of going through the transformation, which is sometimes quite tedious, especially for large specifications.

b. Direct approach: In this approach, by means of the I-PN representation of LOTOS, a set of formulas has been derived for calculating the cyclomatic complexity of the primitive LOTOS operations. Based on them, the complexity of a LOTOS expression can be calculated directly from the expression itself without going through any transformation. Obviously, this approach is much more efficient than the indirect approach.

(3) A criterion for path-testing coverage.

Based on the I-PN model and the cyclomatic complexity metric, we have derived a path testing coverage criterion, called basis set coverage. A basis set is a set of linearly independent cycle paths that cover all the places, transitions and arcs of an I-PN. This means that if a set of test sequences is derived from a basis set, it will cover all the operators and operands appearing in a LOTOS specification. The cyclomatic complexity of an I-PN is the maximum possible number of such paths in a basis set.

This is probably the first research work concerning the complexity measure of formal techniques and distributed systems, and certainly a lot more can be done in this area. In particular, we plan to continue our research on the following problems:

(1). To apply other complexity measures to the specifications of distributed systems

In this thesis, we consider basic LOTOS only (i.e., LOTOS without tl data part).

If data is taken into consideration, several other complexity measures, which include both operators and data (as operands), can be applied.

(2) To implement the cyclomatic complexity measure

We think constructive formulas for this task is quite straight forward. At present, there exist several interpreters which can translate LOTOS to other formats, such as graphical LOTOS [CHE88]. These interpreters can be used to scan through a LOTOS specification. Hence, by making use of the constructive formulas obtained in this thesis, one can quite easily implement a system for computing the cyclomatic complexity of any LOTOS expression.

(3) To develop complexity measures for other formal description techniques

In the literature, application of complexity measures is mainly on sequential programming languages or ordinary flowgraphs. In this thesis, we have extended it into the area of Petri-nets and the formal description language LOTOS. If similar complexity measures can be developed for other formal description techniques (SDL, ESTELLE) and semi-formal techniques (e.g., TTCN), we would have a better reservoir of software engineering techniques for comparing the different approaches for specifying distributed systems.

(4) To find methods for determining the independent cycle paths of a basis set.

At present, for both our approach of using Petri-nets and McCabe's approach of using flowgraphs, a cyclomatic number can only tell the maximum number of independent cycle paths in the basis set. It cannot be used to determine these paths. Hence, in order to find these sequences, other methods still have to be found. Note also that a cycle path in an I-PN may not be feasible. We foresee that this is a difficult

research problem.

(5) To compare the cyclomatic complexities of different LOTOS specification styles

A distributed system may be specified in LOTOS in different styles[VIS90]. While these styles have different advantages with respect to different objectives, it is interesting to compare their complexities.

(6) To improve some of the transformation rules

A possible improvement for the I-PN model is to refine the transformation rule for parallelism so that some useless combinations will be discarded.

REFERENCES

[ALB83] Albrecht, A.J., and Gaffey, E., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Trans. on Software Engineering, vol. SE-9, no. 6, Nov., 1983, pp. 639-648.

[BAK80] Baker, A.,"A Comparison of Measures of Control Flow Complexity", IEEE Trans. Software Eng., vol. SE-6, no. 6, pp. 506-512, Nov., 1980.

[BAR89] Barbeau M.,and Bochmnn, G.V., "Deriving Analyzable Petri-nets from LOTOS Specifications", Tech. Report, Universite de Montreal, 1989.

[BAS83] Basili, V.R., Selby, R.W., and Phillips, T.Y, "Metric Analysis and Data Validation Across Fortran Projects", IEEE Trans. on Software Engineering, vol. SE-9, no. 6, Nov., 1983, pp. 652-663.

[BEI91] Beizer, B., Software Testing Techniques, New York, Van Nostrand Reinhold Company, 1991.

[BER73] Berge, C., Graphs and Hypergraphs. Amsterdam, The Netherlands: North-Holland, 1973.

[BOL87] Bolognesi, T., and Brinksma,E., "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems 14, pp. 25-59, 1987.

[CAV78] Cavano, J.P., and J. A. McCall, "A Framework for the Measurement of Software Quality." Proc. ACM Software Quality Assurance Workshop, November, 1978, pp. 133-139.

[CHE78] Chen, E. T., "Program Complexity and Programer Productivity", IEEE Trans. Software Eng., vol. SE-4, no. 3, pp. 187-194, May, 1978.

[CHE88] Cheung, T. Y., and Zhu, Y., "A Petri-net-based Method for Specifying Distributed Systems and Deriving Executable Graphical LOTOS and ESTELLE", Tech. Report

TR-8804, Dept. of Computer Science, University of Ottawa, Feb, 1988.

[CHE89] Cheung, T.Y., Ye, X., Ye, Y.C. and G.Q. Wang, "UO-GLOTOS: A Syntax/System for Representing, Editing and Translating Graphical LOTOS", 2nd International Conf. on Formal Description Technique, pp. 33-49, 1989.

[CUR79A] Curtis, B., Sheppard, S.B., and Milliman, P., "Third Charm: Stronger Predictions of Programmer Performance by Software Complexity Metrics", IEEE, Proceedings of the 4th International Conf. on Software Engineering, Munich, March 1979.

[CUR79B] Curtis, W., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", IEEE Trans. on Software Engineering, vol. 5, March 1979, pp. 96-104.

[DAV88] Davis, J. S., "A Study of the Applicability of Complexity Measures", IEEE Trans. Software Eng., vol. 14, no. 9, pp. 1366-1372, Sept., 1988.

[DEM82] DeMarco, Controlling Software Projects: Management, Measurement, and Estimation, New York: Yourdon, 1982.

[DUN82] Dunn, R., and R. Ullman, Quality Assurance for Computer Software, McGraw-Hill, 1982.

[ELS78] Elshoff, J. L., and M. Marcotty, "On the Use of the Cyclomatic Number to Measure Program Complexity", SIGPLAN Notices, Dec. 1977.

[FEU79A] Feuer, A. R., and Fowlkes, E.G.,"Some Results from an Empirical Study of Computer Software", IEEE, Proceeding of the Fourth International Conf. on Software Engineering, Munich, March 1979.

[FEU79B] Feuer, A. R., and Fowlkes, E.G., "Relating Computer Program Maintainability to Software Measures", Proceeding of the 1979 National Computer Conference, Montvale NJ: AFIPS Press, 1979.

[GAR90] Garavel, H., "Compilation and Verification of LOTOS Specifications", IFIP Protocol Specification, Testing and Verification X, pp. 359-376, June, 1990.

[GEN84] Genrich, H. J, et al, Elements of General Net Theory, Springer LNCS 84, 1984.

[GRE76] Green, T. F., et al., "Program Structure, Complexity and Error Characteristics", Computer Software Engineering, Polytechnic Press, New York, 1976, pp. 139-154.

[GUE89] D. Gueraichi and L. Logrippo, "Derivation of Test Cases for LAPB from a LOTOS Specification", 2nd International Conf. on Formal Description Technique, pp. 33-49, 1989.

[HAL77] Halstead, M.H., Elements of Software Science. New York, Elsevier: North-Holland, 1977.

[HAN78] Hansen, W. J., "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", SIGPLAN Notices, vol. 13, no. 3, pp. 29-33, March 1978.

[HAR81] Harrison, W and Kafura, D., "A Complexity Measure Based on Nesting Level", SIGPLAN Notices, pp. 63-74, Mar. 1981.

[HOA85] Hoare, C. A. R., Communicating Sequential Processes, Prentice Hall, 1985.

[ISO7498] ISO IS 7498, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", 1983.

[ISO8807] ISO IS 8807, "Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior",1989.

[ISO9646] ISO DP9646, OSI Conformance Testing Methodology and Framework, 1988.

[LEW88] Lew, K. S., "Software Complexity and Its Impact on Software Reliability", IEEE Trans. Software Eng., vol. 14, no. 8, pp. 1645-1655, Nov., 1988.

[LIN79] Linger, R. C., Mills, H. D., and Witt, B. I., Structured Programming, Addision-Wesley Publishing Company, Inc., 1979.

[MAR89] Marchena, S., "Transformation from LOTOS Specs to Galileo Nets", Formal Description Techniques, pp. 217-230,1989.

[MCC76] McCabe, T.J., "A Complexity Measure", IEEE Trans. Software Eng., vol. SE-2, no. 4, pp. 308-320, Dec., 1976.

[MIL80] Milner. R., A Calculus of Communicating Systems, Springer LNCS92, 1980.

[MYE77] Myers, G. J., " An Extension to the Cyclomatic measure of Program complexity", SIGPLAN Notices, vol. 12, no. 10, pp. 61-64, Oct. 1977.

[PET81] Peterson, J., Petri Net Theory and the Modeling of Systems, 1981.

[PRE87] Pressman, R.S, Software Engineering, McGraw-Hill Inc., 1987.

[RAM88] Ramamurthy, B., "A Synthesis of Software Science Measures and the Cyclomatic Number", IEEE Trans. Software Eng., vol.14, no. 8, pp. 1116-1121, Aug., 1988.

[SAN86] Sanchez, C., "Galileo Model Language and Tools", Electrical Communications, Vol. 60, no. 3-4, 1986.

[VIS90] Vissers, C. A., et al, "On the use of Specification Styles in the Design of Distributed Systems", Tech. Report, University of Twente, 1990.

[WAL79] Walsh, T. J., "A Software Reliability Study Using a Complexity Measure", AFIPS National Computer Conference, pp. 761-768, 1979.

[WOO79] Woodward, M.R., "A Measure of Control Flow Complexity in Program Text", IEEE Trans. Software Eng., vol. SE-5, no. 1, pp. 45-50, Jan., 1979.