1988

Comparison of the Petri nets model and the Hoare processes model

Babak Shahpar
The University of Montana

A COMPARISON OF THE PETRI NETS MODEL AND THE HOARE PROCESSES MODEL

By

Babak Shahpar

B. S., Karaj College of Mathematics and
Economical Management, 1976

Presented in partial fulfillment of the requirements

for the degree of

Master of Science

University of Montana

1988

Approved by

Chairman, Board of Examiners

Dean, Graduate School

March 8, 1988

Date

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Shahpar, Babak, M.S., January 1988 Computer Science

A Comparison of the Petri Nets Model and
the Hoare Processes Model (69 pp.)

Director: Dr. Alden H. Wright

Petri nets and Hoare processes are two models for simulating and analyzing a system with concurrent and sequential events. The events of a system are represented as transitions in the Petri nets model and as alphabet of a process in the Hoare processes model.

In this thesis a trivial proof is presented to show that for every Petri net there exists a Hoare process which generates the same *prefix* language as the corresponding Petri net. This is accomplished by an obvious mapping from Petri nets to Hoare processes. Thus, the Petri Net Model has power less than or equal to that of the Hoare Processes Model. In addition, another systematically defined mapping from Petri nets to Hoare processes is given and proved to produce for each Petri net a Hoare process whose language is the prefix language of that net. This mapping has a certain advantage: it results in a Hoare process defined by an infinite set of mutually recursive equations. Thus, the (possibly infinite) language of the process is defined by a finite set of equations in terms of basic operations. This facilitates analysis of the modeled system in terms of the Hoare process and remapping the modeled system back in terms of the Petri net.

Finally, an example is given to demonstrate that there exists a Hoare process which cannot be modeled by any Petri net. Thus, the Hoare processes model is a more powerful model of computation than the Petri nets model.

Informal definitions for Petri nets and Hoare processes are presented first and later both models are defined formally. Also, formal definitions for languages of these models are defined.

Table of Contents

Abstract ii
Table of Contents iii
List of Figures v
Acknowledgements vi
1. Introduction 1
1.1. Background 1
1.2. The Petri Nets Model 1
1.3. The Hoare Processes Model 9
1.3.1. Traces 9
1.3.2. The Prefix Notation 9
1.3.3. The Choice Notation 11
1.4. The Proposed Research 12
2. Formal Definitions 15
2.1. The Petri Nets Model 15
2.1.1. Definition of a Bag 15
2.1.2. Definition of a Petri net 16
2.1.3. Multiplicity of a Place 18
2.1.4. Enabled Transitions 19
2.1.5. Firing a Transition 20
2.1.6. The Next-State Function 22
2.1.7. A Petri Net Language 22
2.2. The Hoare Processes Model 26
2.2.1. Concatenation of Traces 26
2.2.2. Definition of a Hoare Process 26
2.2.3. The General Choice Notation 27
2.2.4. Recursion 28
2.2.5. Guarded Processes 29
2.2.6. The Mutual Recursion 30
2.2.7. Concurrency 31
2.2.7.1. The Concurrency Rules 33
2.2.7.2. The General Concurrency 35
2.2.7.3. Theorem 37
3. Main Theorems 42
3.1. Theorem 42
3.2. Basic Definitions 43
3.3. The function Ψ 43
3.3.1. Example 45
3.4. Theorem 47

3.4.1. (=>) 47
3.4.2. (<=) 51
3.5. Theorem 53
3.6. Theorem 54
3.7. Definition 55
3.8. Theorem 56
3.9. A Hoare Process that Cannot be Modeled by a Petri net ..56
3.9.1. Theorem 56
3.10. Other Classes of Languages 58
3.10.1. Concealment 60
3.11. Modifying the Function Ψ 61
3.11.1. Lemma 62
3.11.2. Lemma 63
3.12. Theorem 64
4. Summary and Conclusions 66
4.1. Summary 66
4.2. Conclusions ans Suggestions for Future Research 67
4.2.1. Petri net Languages and
Other Classes of Languages 67
REFERENCES 69

List of Figures

Figure 1.1 2
Figure 1.2 4
Figure 1.3 6
Figure 1.4 7
Figure 1.5 8
Figure 2.1 18
Figure 2.2 21
Figure 2.3 24

Acknowledgments

This research paper is solely dedicated to Dr. Alden Wright without whose inspirational and technical support it would not have been possible. His expertise was the technical backbone of this paper and his personality the motivating force behind it. I am proud to be his student.

I would also like to thank Dr. William Ballard and Dr. Ronald Wilson for their invaluable guidance.

Special thanks are also due to my good friend Mohammad Paryavi for his inspirational support of my graduate study and for being so helpful in my "MacTastrophies".

I would also like to thank my good friend and colleague Wanda Smith for fixing my articles in this paper.

Last, I would like to thank my Creator for giving me a chance to exist, to think, and to choose.

Chapter One

Introduction

1.1. Background

Theoretical computer science is regarded as the foundation for computer studies. Analysis of many real problems in computer science depends on mapping the actual problem into abstractions which are based on the use of formal models. In order to find an appropriate model for a given system, a thorough understanding of the behavior and the power of the model is required. For the study of different aspects of a given system, different types of models have been proposed. There are several models for analyzing a system with concurrent and sequential events. Two of these models are Petri Nets and Hoare processes. In this chapter informal definitions for both models are presented.

1.2. The Petri Nets Model

A Petri net can be presented by a graph, where a set of nodes is associated to places and another set of nodes to transitions. Every node presents exclusively either a place or a transition. Perhaps an example would help in understanding the process of modeling a system using a Petri net. Figure 1.1 presents an example of a Petri net.

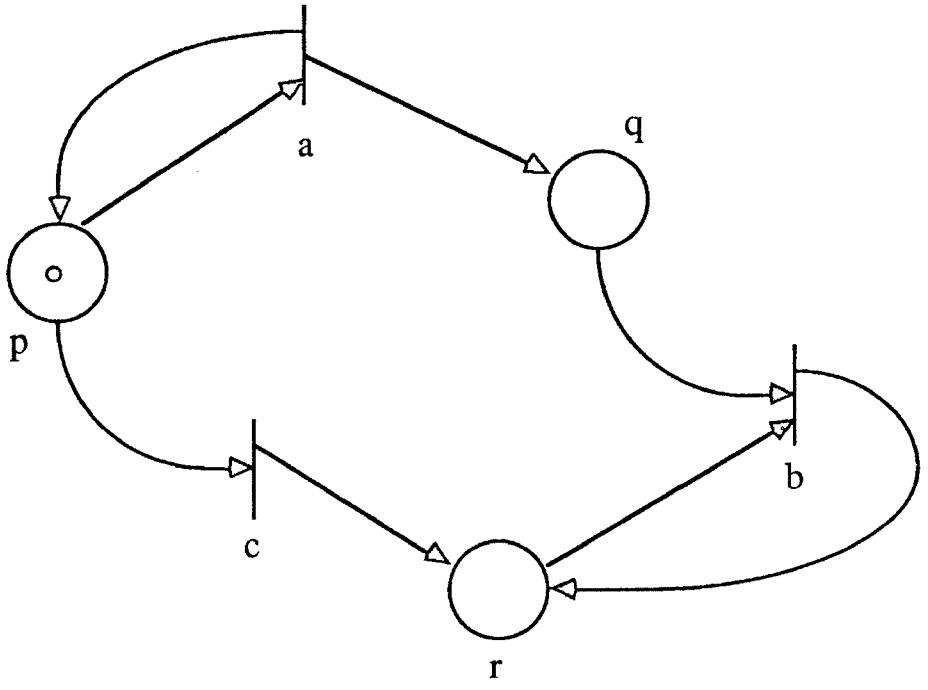

Figure 1.1
A marked Petri net.

Each transition represents an event which can occur in the system and is denoted by a bar. In Figure 1.1, the symbols a, b, and c are transitions. Places are represented by circles. In the Petri net of Figure 1.1, the symbols p, q, and r are places.

"Tokens" are used in Petri nets in order to simulate the flow of

information. In the graph of a Petri net a dot in a place represents a token. A place can hold zero or more tokens. In Figure 1.1, there is one token in p. A "marking" of a Petri net is an assignment of tokens to the set of places. A marked Petri net is a Petri net with an initial marking.

The direction of an arc determines if a place is an input place or an output place for a given transition. For example, q is an input place for transition b, and an output place for transition a. The place p is both an output place and an input place for transition a. There may be more than one arc between a transition and a place.

"Firing" a transition in the model simulates the occurrence of an event in the actual system. A transition can fire if every input place to that transition contains at least one token (if there is only one arc between each of the places and the given transition). In Figure 1.1, the transitions a and c are the only ones which can fire in the initial state. A transition that can fire in a given state is called an *enabled* transition. Among *enabled* transitions only one can fire at any given state.

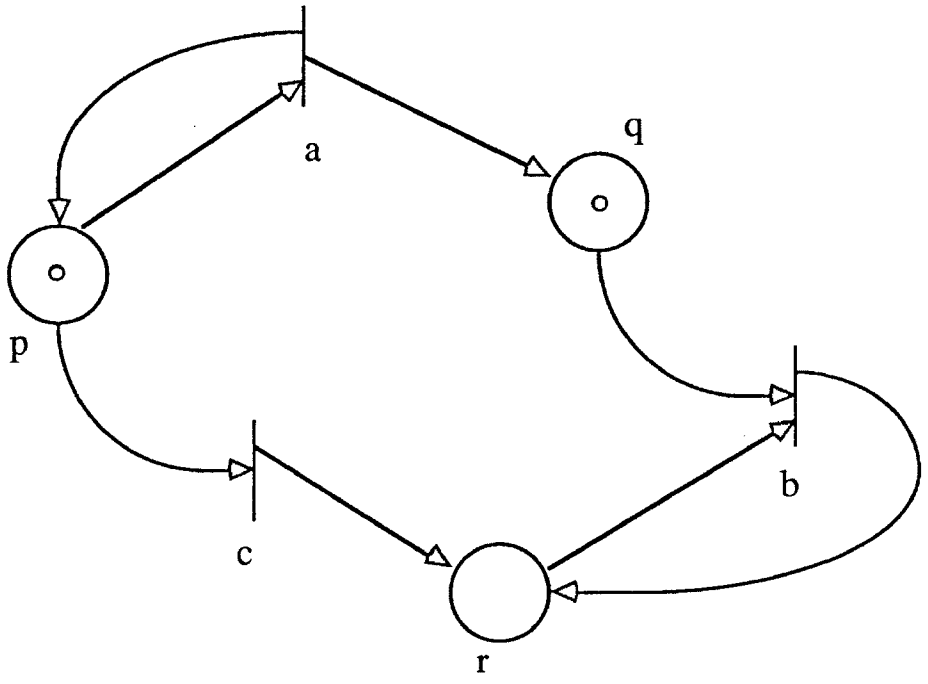

Figure 1.2

The marking resulting from firing transition a in Figure 1.1.

The simulation of dynamic behavior of the system starts with firing an *enabled* transition in a Petri net with an initial marking. Firing a transition results in a new state of the net which is defined by a new marking.

The new marking is obtained from the old one by reducing the

number of tokens in each of the input places of the fired transition by the number of input arcs and increasing the number of tokens in each of the output places of the fired transition by the number of output arcs.

Figure 1.2 illustrates the new marking for the Petri net of Figure 1.1 after transition "a" has fired.

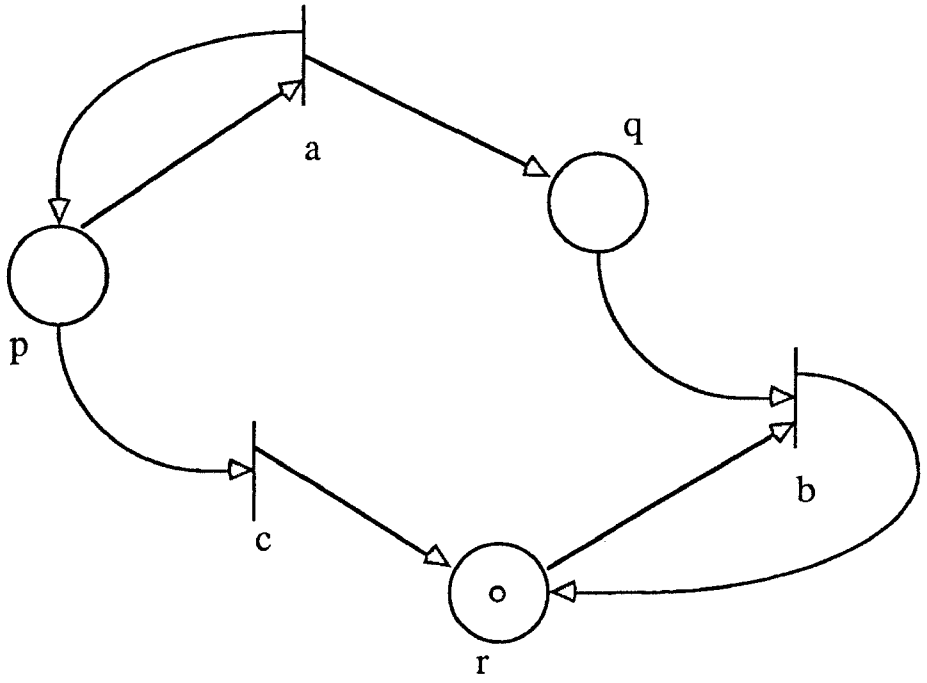

Figure 1.3

The marking resulting from firing transition c in Figure 1.1.

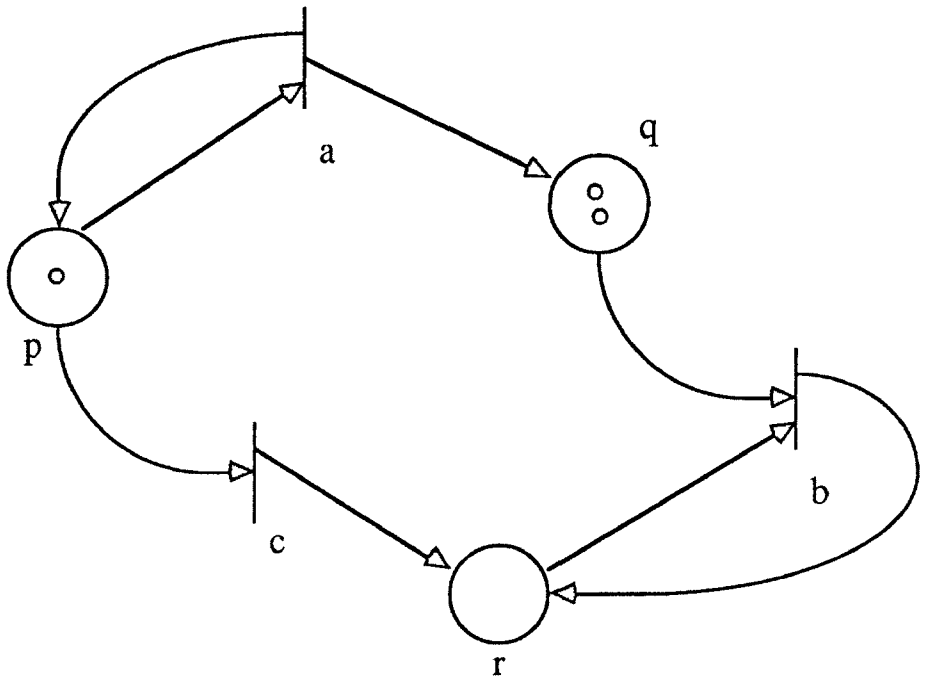

Figure 1.4
The marking resulting from firing transition a in Figure 1.2.

Figure 1.3 shows the marking for the Petri net of Figure 1.1 after transition c has fired.

If there is no *enabled* transition in a given state, then execution of the Petri net is halted. The Petri net of Figure 1.3 is in a halt state since there is no *enabled* transition.

Figure 1.4 shows the new marking for the Petri net of Figure 1.1

after two consecutive firings of transition a.

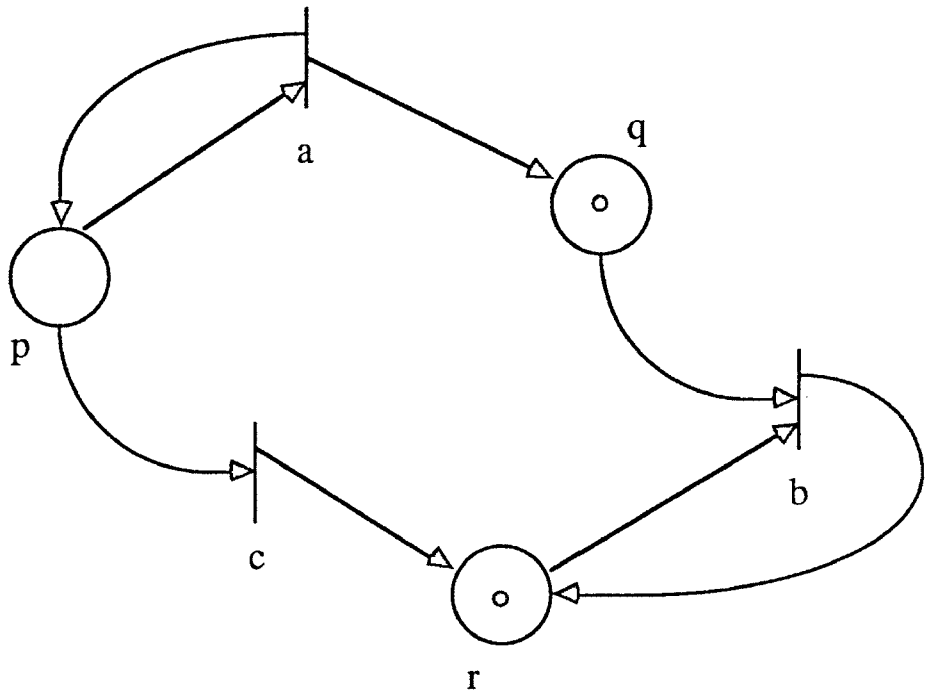

Figure 1.5

The marking resulting from firing transition c in Figure 1.2.

Figure 1.5 illustrates the marking of the Petri net of Figure 1.1 after transition a, and then transition c have fired.

1.3. The Hoare Processes Model

Hoare processes describe the behavior of an object in terms of a finite set of events called an alphabet. For example, the process of making a single phone call may have the following events in its alphabet:

pick-up	picking up the receiver
insert	the insertion of one coin
dial	dialing a number
talk	conversation on the phone
hang-up	hang-up the phone

1.3.1. Traces

A trace of a process is a finite sequence of events which has been engaged in, by the process, up to some moment in time. Two events in a Hoare process cannot occur simultaneously. A trace will be denoted by the sequence of events, separated by commas, in angular brackets. An example of a trace is:

<pick-up, dial, talk>

Upper-case letters are used to denote arbitrary processes and lower-case letters to denote events. Words in upper-case letters denote specific defined processes. The alphabet of a process P is denoted by αP.

1.3.2. The Prefix Notation

Let P be a process and x be an event. Then:

$(x \rightarrow P)$ (read "x then P")

denotes a process which first engages in the event x then behaves as P. The event x must be in the alphabet of $(x \rightarrow P)$. For example, if a pay phone consumes one coin and then breaks, it cannot engage in any event of its alphabet, and can be defined as:

$(coin \rightarrow STOP)$

The above process has only two possible traces shown as: < > and <coin>. Before the event "coin" occurs the trace of the process is < > and after the process has engaged in the event "coin" the trace of the process will be <coin>. Therefore,

$traces(coin \rightarrow STOP) = \{< >, <coin>\}$

The process of making a single phone call can be described by:

$PHCALL = (pick\text{-}up \rightarrow insert \rightarrow dial \rightarrow talk \rightarrow hang\text{-}up \rightarrow STOP)$

The set of all possible traces for PHCALL is:

traces(PHCALL) = { < >,
<pick-up>,
<pick-up, insert>,
<pick-up, insert, dial>,
<pick-up, insert, dial , talk>,
<pick-up, insert, dial , talk, hang-up>}

The process of a broken pay phone which consumes coins forever can be described recursively as:

BROKENPHONE = (coin → BROKENPHONE)

traces(BROKENPHONE) = {< >, <coin>, <coin, coin>, ...}

$$= \{coin\}^*$$

The process of an operator that can make phone calls forever can be described recursively as:

OPERATOR = (pick-up → dial → talk → hang-up → OPERATOR)

A possible trace of OPERATOR is:

<pick-up, dial , talk, hang-up, pick-up, dial>

1.3.3. The Choice Notation

If x and y are two distinct events, then:

$$(x \rightarrow P \mid y \rightarrow Q)$$

describes a process which first engages in x or y and then, depending on that choice, behaves as P or Q respectively.

For example, a broken vending machine which either consumes one coin and stops or gets a kick from a customer and then stops can be defined as the process BRVM:

BRVM = (coin → STOP | kick → STOP)

traces(BRVM) = {< >, <coin>, <kick>}

As another example, consider a vending machine which works fine until it gets a kick from a customer:

VM = (coin → pop → VM | kick → STOP)

A possible trace for VM is:

<coin, pop, coin, pop, kick>

A lazy operator who sometimes makes a phone call and at other times just picks up the receiver then hangs up can be described as:

LAZYOP = (pick-up → (dial → talk → hang-up → LAZYOP
| hang-up → LAZYOP))

A possible trace of LAZYOP is:

<pick-up, hang-up, pick-up, dial, talk>

1.4. The Proposed Research

Each model of computation has some power and limitations for simulating a system. Many studies have been done concerning relationships between models. Milner [1980] proposed a framework for comparing different models at different levels of abstraction. Peterson [1974] and Bredt [1974] suggested using the sets of languages of the models to compare them.

For example, consider finite automata and regular expressions. For modeling purposes, finite automata can be used as language generators. Regular expressions are also considered as language generators, since there exist algorithms for generating regular expressions. It has been proven that a language is regular (i.e. is defined by a regular expression) if and only if it is accepted by a finite automaton. Therefore, the finite automaton model has the same power as the regular language model. In this paper Hoare and Petri Nets Models are viewed as language generators.

The purpose of this paper is to compare the Petri Net Model and the Hoare Process Model. The chosen method of comparison is that model A has less than or equal modeling power to model B if, given an instance *a* of model A, there is an algorithm to create an instance *b* of

model B such that the language generated by a is equal to the language generated by b .

However, there are several ways of defining a class of languages for models of computation. Peterson [1974] has listed twelve different types of languages for Petri Nets.

The specific class of languages (*prefix*) chosen to represent the languages generated by Petri nets will be formally defined in Chapter two. One of the characteristics of this language is that only distinct transitions in a Petri net are allowed. Another characteristic of the language is that every possible marking of a Petri net can be assumed as a final state. Furthermore, other classes of languages for Petri nets will be considered in Chapter three.

One class of languages (prefix) for the Hoare processes can be defined as a set of all possible traces of the process. A different class can be defined as a set of successfully terminated sequences of events.

In this thesis a trivial proof is presented to show that for every Petri net there exists a Hoare process which generates the same *prefix* language as the corresponding Petri net. This is accomplished by an obvious mapping from Petri nets to Hoare processes. Thus, the Petri Net Model has power less than or equal to that of the Hoare Processes Model. In addition, another systematically defined mapping from Petri nets to Hoare processes is given and proved to produce for each Petri net a Hoare process whose language is the prefix language of that net. This mapping has a certain advantage: it results in a Hoare process defined by a finite set of mutually recursive equations. Thus,

the (possibly infinite) language of the process is defined by a finite set of equations in terms of basic operations. This facilitates analysis of the modeled system in terms of the Hoare process and remapping the modeled system back in terms of the Petri net.

Furthermore, an example will be presented to show that there exists a Hoare process which cannot be modeled by any Petri net. It can, therefore, be concluded that the Hoare Processes Model is a strictly more powerful model of computation than the Petri Nets Model.

Chapter Two

Formal Definitions

In Chapter one, informal definitions for the two models under study were presented. In this chapter, formal definitions and related concepts for the Petri nets and the Hoare processes are presented.

2.1. The Petri Nets Model

Petri net theory was first introduced by Carl Adam Petri [1962a] in his doctoral dissertation. The work of Petri was extended by several other researchers. At the present time, some of the definitions in Petri net theory are different from those introduced by Petri. For example, in the original Petri net theory, multiple arcs were not allowed. The definition of Petri nets in this paper has been taken from a book by Peterson [Peterson 81].

2.1.1. Definition of a Bag

A *bag* is similar to a set except that multiple occurrences of an element is permitted. As with set theory, the order of the elements in a bag is not important. A bag can be defined formally as:

a finite bag B over a set S is a function B: $S \rightarrow N$, such that B(s) = 0 for all but finitely many $s \in S$.

If B is a finite bag over S and $x \in S$, then:

$$\#(x, B) = B(x)$$

If S is a non empty set, S^{∞} denotes the infinite set of bags whose elements are taken from the set S. The basic concept of set theory is the membership relation. In bag theory, the basic concept is the number of occurrences of an element in the bag.

Let B be a bag over S and x an element of S. The notation:

$$\#(x, B)$$

denotes the number of occurrences of x in B. However, the notation of membership ($\in$) is used in this paper as follows:

Let B be a bag and x an element of S. Then,

$x \in B$ is true if $\#(x, B) \geq 1$

is false if $\#(x, B) = 0$

Bags are used in Petri nets to allow multiple connections between a place and a transition (multiple occurrences of arcs in a Petri net graph between a place and a transition).

2.1.2. Definition of a Petri net

A Petri net structure, M, is a quintuple, $M = (P, T, I, O, \mu)$ where:

P is a finite set of places

T is a finite set of transitions

I is a function from T to P^{∞}

O is a function from T to P^{∞}

μ is a function from P to N

The set of places and the set of transitions are disjoint. The function I is called the *input* function and the function O is called the *output* function. The function μ is called the *marking* function and is a mapping from P to N, where N is the set of non-negative integers. For each place p, $\mu(p)$ is the number of tokens in p. The marking function can be determined from the number of tokens in each place in the graphical representation of a Petri net. This function indicates the state of execution of a Petri net.

Figure 2.1 illustrates the Petri net $M = (P, T, I, O, \mu)$ where:

$P = \{ p_1, p_2, p_3, p_4, p_5\}$

$T = \{t_1, t_2, t_3\}$

$I(t_1) = \{p_1\}$ $\quad O(t_1) = \{p_2, p_3, p_4, p_4, p_4\}$

$I(t_2) = \{p_2, p_4\}$ $\quad O(t_2) = \{p_2\}$

$I(t_3) = \{p_3, p_4, p_4\}$ $\quad O(t_3) = \{p_5\}$

$\mu(p_1) = 1$

$\mu(p_i) = 0$ $\quad$ for i = 2, 3, 4, 5

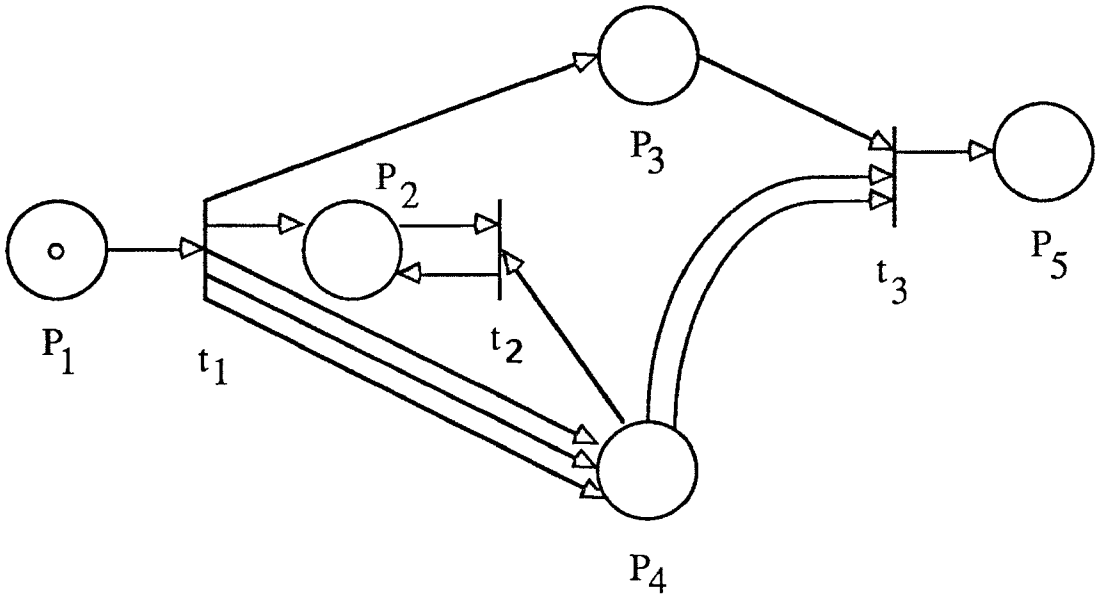

Figure 2.1
A marked Petri net.

A place p is an *input* place for a transition t if $p \in I(t)$; p is an *output* place for a transition t if $p \in O(t)$. For example, in the marked Petri net of Figure 2.1, the place p_2 is an input place and an output place of the transition t_2.

2.1.3. Multiplicity of a Place

By the definition of a Petri net, each transition has a bag of inputs

and a bag of outputs. The use of these bags over P allows a place to be a multiple input or a multiple output of a transition.

The multiplicity of an input place p for a transition t is the number of occurrences of p in the input bag of the transition t. This is

$$\#(p, I(t))$$

and corresponds to the number of arcs from the input place p to the transition t in the graph representation of a Petri net. Similarly,

$$\#(p, O(t))$$

denotes the multiplicity of an output place p for a transition t. This multiplicity is equal to the number of arcs from the transition t to the output place p. For example, in the Figure 2.1,

$$\#(p_4, O(t_1)) = 3$$
$$\#(p_4, I(t_3)) = 2$$
$$\#(p_5, I(t_3)) = 0$$

2.1.4. Enabled Transitions

A transition $t \in T$ in a marked Petri net $M = (P, T, I, O, \mu)$ is *enabled* if

$$\mu(p_i) \geq \#(p_i, I(t)) \qquad \forall\, p_i \in P$$

Thus, a transition is enabled if each input place of the transition has at least as many tokens as the number of arcs from that place to the transition. For example, in Figure 2.1, the transition t_1 is the only transition that is enabled in the initial state. Transition t_3 with $I(t_3) = \{p_3, p_4, p_4\}$ and $O(t_3) = \{p_5\}$ is enabled if there are at least one token in the place p_3 and two tokens in the place p_4. However,

$$\mu(p_4) = 0 \; < \; \#(p_4, I(t_3)) \; = 2$$

Therefore, transition t_3 is not an enabled transition with regard to the initial marking function μ.

2.1.5. Firing A Transition

A transition t in a marked Petri net $M = (P, T, I, O, \mu)$ may fire if it is enabled. At any given state, only one enabled transition fires. Firing an enabled transition results in a new marking function μ' which is defined as:

$$\mu'(p_i) = \mu(p_i) + \lambda(p_i, t) \qquad \forall\, p_i \in P$$

where

$$\lambda(p_i, t) = \#(p_i, O(t)) \; - \; \#(p_i, I(t))$$

In other words, a transition fires by removing one token from each of its input places for each arc from the place to the transition and depositing one token into each of its output places for each arc from the transition to the place.

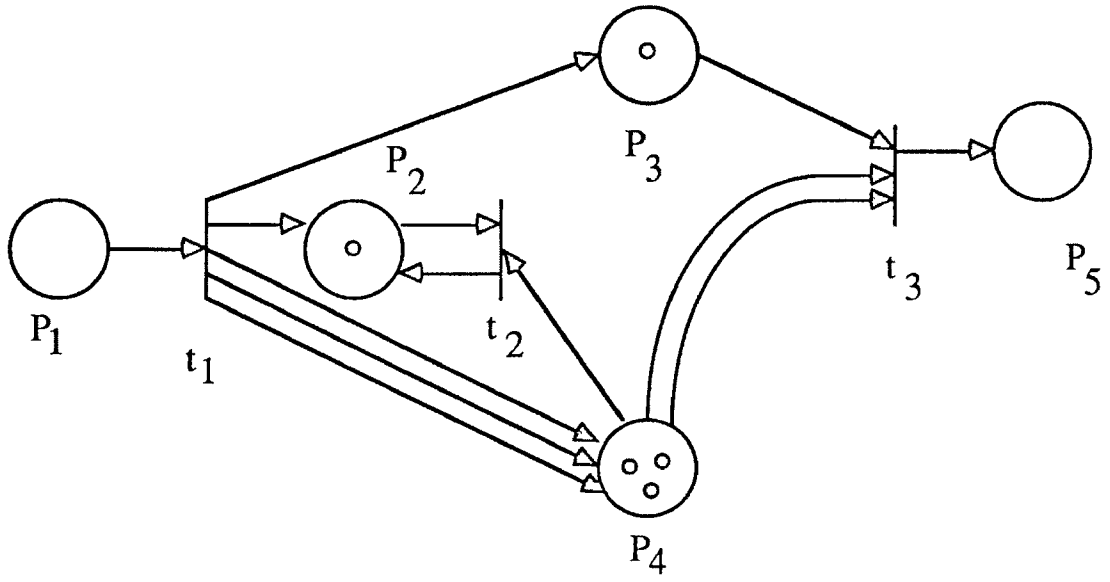

Figure 2.2

The marking resulting from firing transition t1 in Figure 2.1.

For example, firing the transition t_1 in the Petri net of Figure 2.1 results in a new marking μ' which is defined as:

$$\mu'(p_1) = 0$$
$$\mu'(p_2) = 1$$
$$\mu'(p_3) = 1$$
$$\mu'(p_4) = 3$$
$$\mu'(p_5) = 0$$

Figure 2.2 illustrates the Petri net of Figure 2.1 after the transition t_1 is fired.

2.1.6. The Next-State Function

Let $M = (P, T, I, O, \mu)$ be a marked Petri net and $t \in T$. Then, δ is the *next-state* function and is defined only if:

$$\mu(p_i) \geq \#(p_i, I(t)) \qquad \forall\, p_i \in P$$

If $\delta(\mu, t)$ is defined, then:

$$\delta(\mu, t) = \mu'$$

Thus, the notation $\delta(\mu, t)$ represents the marking of the Petri net M after the transition t is fired. The marking μ' is said to be immediately reachable from μ.

Let $t \in T$ and $\sigma \in T^*$. The notation δ is also used to denote the *extended next-state* function defined for sequences of transitions, elements of T^*, as:

$$\delta(\mu, \varepsilon) = \mu$$

$$\delta(\mu, t\sigma) = \delta(\delta(\mu, t), \sigma) \qquad \forall\, \sigma \in T^*$$

The main difference in the next-state function and the extended next-state function is that the latter function accepts a sequence of transitions (possibly an empty sequence) as its second argument. Since no confusion need result, in this paper the same notation (δ) is used to represent both functions.

2.1.7. A Petri net Language

As mentioned in Chapter one, Petri nets are to be considered as

language generators. Given a marked Petri net $M = (P, T, I, O, \mu^0)$, firing an enabled transition t_j results in a new marking $\mu^1 = \delta(\mu^0, t_j)$. Firing an enabled transition, say t_k, in marking μ^1 results in another marking $\mu^2 = \delta(\mu^1, t_k)$. Two sequences result from the execution of a Petri net: the sequence of fired transitions,

$$< t_{j_0}, t_{j_1}, t_{j_2}, \ ... >$$

and the sequence of markings,

$$< \mu^0, \mu^1, \mu^2, \ ... >$$

The relationship between these two sequences can be described as:

$$\delta(\mu^k, t_{j_k}) = \mu^{k+1} \quad \text{for } k = 0, 1, 2, ...$$

Given a sequence of transitions, a sequence of markings can be easily derived. However, given a sequence of markings, it is not always possible to derive the sequence of transitions that actually fired. Consider the following example:

Let $M = (P, T, I, O, \mu)$ be a marked Petri net, where:

$P = \{p_1, p_2\}$ $\qquad$ $T = \{t_1, t_2\}$

$I(t_1) = \{p_1\}$ $\qquad$ $I(t_2) = \{p_1\}$

$O(t_1) = \{p_2\}$ $\qquad$ $O(t_2) = \{p_2\}$

$\mu(p_1) = 1$ $\qquad$ $\mu(p_2) = 0$

Figure 2.3 illustrates the Petri net M.

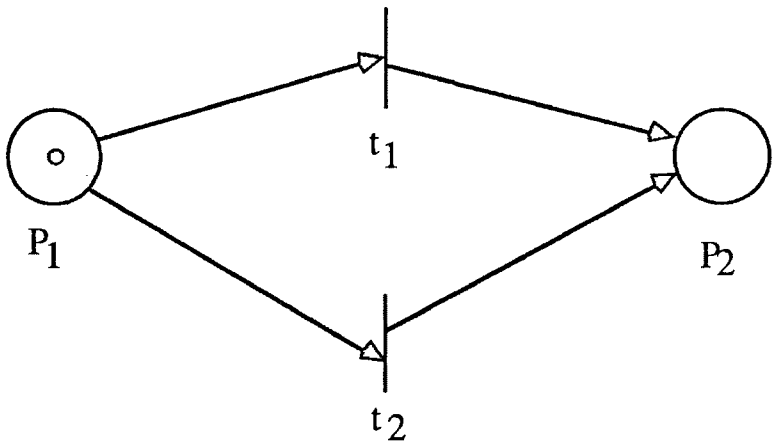

Figure 2.3
A marked Petri net.

If either the transition t_1 or the transition t_2 is fired, the resulting new marking is μ' given by $\mu'(p_1) = 0$ and $\mu'(p_2) = 1$. Thus, from the sequence:

$$< \mu, \mu' >$$

it is impossible to determine whether the transition t_1 or the transition t_2 fired. Therefore, a sequence of fired transitions of a Petri net can give a better description of the execution of the Petri net than a sequence of markings.

Each transition in a Petri net corresponds to an event in the system which is modeled. Firing a transition in the Petri net simulates

the occurrence of a corresponding event in the system. Hence, the set of transitions of a Petri net is to be considered as the alphabet of the language which is generated by execution of the Petri net. A sequence of fired transitions is a string and a set of strings constitutes a language for a Petri net.

Let $M = (P, T, I, O, \mu)$ be a marked Petri net; then,

$$L(M) = \{\sigma \in T^*: \delta(\mu, \sigma) \text{ is defined}\}$$

is called the language generated by the Petri net M. In this paper, the class of languages generated by all Petri nets is formally defined as:

a language L is in the class of *prefix* Petri net languages if there exists a Petri net $M = (P, T, I, O, \mu)$ such that:

$$L = \{\sigma \in T^*: \delta(\mu, \sigma) \text{ is defined}\}$$

For example, the language generated by the Petri net of Figure 2.1 is:

$$L = \{\varepsilon, t_1, t_1t_2, t_1t_3, t_1t_2t_2, t_1t_3t_2, t_1t_2t_3, t_1t_2t_2t_2\}$$

This class of languages is also known as the class of prefix languages for Petri nets. There are other definitions for Petri net languages. For example, a language L is in the class of *termination* Petri net languages if there exists a Petri net $M = (P, T, I, O, \mu)$ such that:

$$L = \{\sigma \in T^*: \delta(\mu, \sigma) \text{ is defined but } \forall t_j \in T\ \delta(\delta(\mu, \sigma), t_j) \text{ is undefined}\}$$

The strings in this language are the sequences of transitions which can be reached from the initial marking such that after the last transition in the sequence is fired, the execution of the Petri net will halt. For example, the language of the Petri net of Figure 2.1 with respect to the latter definition is:

$$L = \{t_1t_3t_2, t_1t_2t_3, t_1t_2t_2t_2\}$$

In this paper the prefix definition of Petri net languages is used because every occurrence of the events in the system can be

2.2. The Hoare Processes Model

An informal description of Hoare processes was given in Chapter one. A formal definition of Hoare processes will be presented in this section. The formal definition is used to prove the correctness of some rules which apply to notions such as prefix, choice, and parallel. Some of the rules which are related to this paper will be mentioned in the following sections; however, correctness or consistency of these rules will not be examined in this paper.

2.2.1. Catenation of Traces

The traces of a process were defined in Chapter one. One of the most important operations on traces is catenation. This operator constructs a trace from an ordered pair (s, t) of traces by putting them together in the given order. The result is denoted by s^t. For example,

$$<t_2, t_5, t_7>\hat{}<t_1, t_3> = <t_2, t_5, t_7, t_1, t_3>$$

The most important properties of catenation are that it is associative and has the empty trace (<>) as its unit.

2.2.2. Definition of a Hoare Process

A process is a pair:

$$(A, S)$$

where A is any set of symbols and S is any subset of A* which satisfies the following two conditions:

1) $\langle\rangle \in S$

2) $\forall s, t, \quad s\hat{}t \in S \Rightarrow s \in S$

The condition (1) simply means that the empty trace, <>, is a trace of any process. This corresponds to an intuitive notion that for any process there is a time at which the process has not engaged in any events of its alphabet. The condition (2) means that any prefix of a trace of a process is also a trace of the process.

Consider the following examples. The process which never engages in any of the events in its alphabet may be defined as:

$$STOP_A = (A, \{\langle\rangle\})$$

A process that can engage in any event of its alphabet at any time can be described as:

$$RUN_A = (A, A^*)$$

As another example, consider the process BRVM of Section 1.3.3. which can be defined as:

$$BRVM = (\{coin, kick\}, \{\langle\rangle, \langle coin\rangle, \langle kick\rangle\})$$

2.2.3. The General Choice Notation

The choice and prefix notations were defined in Chapter one. The following is a more general notation that includes both the prefix and the choice.

Let A be an alphabet. Suppose the set B is any set of events

from A, and x is a local variable, and P(x) is a function defining a process for each different x in B. Then,

$$(x : B \rightarrow P(x))$$

denotes a process over the alphabet A which first offers a choice of any event y in B and then behaves like P(y). For example, the process $STOP_A$ can be defined as:

$$(x : \{\} \rightarrow P(x))$$

The binary choice operator, | , can also be defined using the general choice notation:

$$(a \rightarrow P \mid b \rightarrow Q) = (x : B \rightarrow R(x))$$

where

$$B = \{a, b\}$$

and

$$R(x) = \textbf{If } x = a \textbf{ then } P \textbf{ else } Q$$

Thus, the binary choice, prefix, and STOP notations are defined as special cases of the general choice notation.

General choice notation can be formally defined as:

$$(x : B \rightarrow (A, S(x))) = (A, \{<>\} \cup \{<x>^s : x \in B \wedge s \in S(x)\})$$

provided $A \supseteq B$.

2.2.4. Recursion

In order to describe the entire behavior of a process that eventually stops the prefix notation can be used. If a process is designed to run forever, then a rigorous description of the entire

behavior of the process, using prefix notation, is impossible. Therefore, a shorter notation for describing repetitive patterns of a process is preferred. One such notation is recursive equations.

For example, the process BROKENPHONE of Chapter one was defined recursively as:

$$\text{BROKENPHONE} = (\text{coin} \rightarrow \text{BROKENPHONE})$$

This technique of recursive definition of processes will work correctly only if the right hand side of the equation defining a process starts with at least one event prefixed to all the other possible events of the process. Therefore, the equation:

$$Y = Y$$

does not successfully define anything since everything can be a solution to this recursive equation.

2.2.5. Guarded Processes

A process expression which is expressed as a general choice is called a *guarded* expression. If F(X) is a guarded expression containing the process name X, then the equation:

$$X = F(X)$$

has a unique solution with respect to the alphabet of X. This claim (guarded equations have a unique solution) is the fundamental theorem of recursion which has been proved by Hoare [Hoare 85].

The above claim can be informally justified by the method of substitution. Any finite amount of behavior of a process can be determined by repeatedly substituting the right-hand side of the

equation for every occurrence of the process name. Furthermore, two processes which behave the same up to any moment in time describe the same process.

In this notation X is a local name. Therefore, a solution for X in the equation X = F(X) is also a solution for Y in the equation Y = F(Y). Let A be the alphabet of X. It follows from the proof of the fundamental theorem of recursion that the solution to the equation, X = F(X), is:

$$(A, \bigcup_{n \geq 0} \text{traces}(F^n(STOP_A)))$$

provided F is a guarded expression.

The following is the definition of a function which takes a guarded process as its argument and returns a set which is called the initial menu of the process.

Let $P = (x : C \rightarrow Q)$ be a guarded process. Then, the function ξ is defined as:

$$\xi(P) = C$$

2.2.6. The Mutual Recursion

The recursive definition of a process permits a single process to be defined as the solution of a single equation. Similarly, the solution of sets of simultaneous equations with more than one unknown can be defined. For this to work properly the following two conditions must

1) all the right-hand sides must be guarded.

2) each unknown process must appear exactly once on the left-

side of one of the equations.

For example, consider the infinite set of mutually recursive equations:

$$CT_0 = (up \rightarrow CT_1 \mid around \rightarrow CT_0)$$
$$CT_{n+1} = (up \rightarrow CT_{n+2} \mid down \rightarrow CT_n)$$

where n ranges over the set of N (natural numbers). The process CT_0 defines an object that starts on the ground and may move up or around. If it moves up it can move up or down thereafter, except that when on the ground it cannot move any further down. As long as it is on the ground it can move around. The indexed name CT_n describes the behavior of the object when it is n moves off the ground.

2.2.7. Concurrency

When two processes are brought together to run concurrently, they will interact with each other through the events in which simultaneous participation of both the processes is required. Therefore, if the alphabets of the two processes that are running concurrently are different, only the events that are in both their alphabets require the simultaneous participation of both processes. However, the events that are in the alphabet of only one of the two processes are of no concern to the other process and may occur independently.

Let P and Q be two processes. Then,

$$P \| Q$$

denotes the process which behaves like P and Q running concurrently

as described above.

For example, consider a phone that no one answers. The phone can either ring and no one will pick it up or be picked up, dialed, and then hung up. The process of the phone can be described as:

$$\text{PHONE} = (\text{pick-up} \rightarrow \text{dial} \rightarrow \text{hang-up} \rightarrow \text{PHONE} \mid \text{rings} \rightarrow \text{PHONE})$$

where

$$\alpha\text{PHONE} = \{\text{pick-up, dial, hang-up, rings}\}$$

The process of a person who does nothing but make phone calls forever can be described as:

$$\text{PHONEADDICT} = (\text{pick-up} \rightarrow \text{dial} \rightarrow (\text{busy} \rightarrow \text{hang-up} \rightarrow \text{PHONEADDICT} \mid \text{talk} \rightarrow \text{hang-up} \rightarrow \text{PHONEADDICT}))$$

where

$$\alpha\text{PHONEADDICT} = \{\text{pick-up, dial, busy, talk, hang-up}\}$$

If there is a busy signal then PHONEADDICT will hang up, otherwise she will hang up after the event talk occurs. The behavior of PHONEADDICT and PHONE running concurrently can be described as follows:

$$\text{PHONEADDICT} \parallel \text{PHONE} = (\text{pick-up} \rightarrow \text{dial} \rightarrow (\text{busy} \rightarrow \text{hang-up} \rightarrow (\text{PHONEADDICT} \parallel \text{PHONE}) \mid \text{talk} \rightarrow \text{hang-up} \rightarrow (\text{PHONEADDICT} \parallel \text{PHONE})) \mid \text{rings} \rightarrow (\text{PHONEADDICT} \parallel \text{PHONE}))$$

The concurrency of two processes can formally be defined as:

$$(A, S) \,||\, (B, T) = (A \cup B, \{s : s \in (A \cup B)^* \wedge (s \uparrow A) \in S \wedge (s \uparrow B) \in T\})$$

where the notation $(s \uparrow A)$ denotes a trace which is built from the trace "s" by removing every event that is not in the set A. The order of events in $(s \uparrow A)$ is the same as in the trace "s". It is interesting to note

$$\alpha(P \,||\, Q) = \alpha P \cup \alpha Q$$

Therefore, the concurrencey operator (||) takes operands with different alphabets and generates a result with yet another alphabet.

In a case where the alphabets of operands are the same, the result also has the same alphabet; therefore, every event needs the participation of all the processes running in parallel. If the alphabet of every process is disjoint from the alphabets of the other processes, then the action of the processes running concurrently is an arbitrary *interleaving* of the actions of the component processes.

2.2.7.1. The Concurrency Rules

Let P, Q, and R be three guarded processes. Then, the following law states that the operator || is associative:

L0 $\qquad P \,||\, (Q \,||\, R) = (P \,||\, Q) \,||\, R$

The following laws elaborate further the way in which the operator || performs.

Let

$$a \in (\alpha P - \alpha Q)$$

$$b \in (\alpha Q - \alpha P)$$

$$c \in (\alpha P \cap \alpha Q)$$

$$d \in (\alpha P \cap \alpha Q)$$

Then,

L1 $(c \rightarrow P) \parallel (c \rightarrow Q) = c \rightarrow (P \parallel Q)$

L2 $(c \rightarrow P) \parallel (d \rightarrow Q) = STOP$ if $c \neq d$

L3 $(a \rightarrow P) \parallel (c \rightarrow Q) = a \rightarrow (P \parallel (c \rightarrow Q))$

L4 $(c \rightarrow P) \parallel (b \rightarrow Q) = b \rightarrow ((c \rightarrow P) \parallel Q)$

L5 $(a \rightarrow P) \parallel (b \rightarrow Q) =$

$$(a \rightarrow (P \parallel (b \rightarrow Q)) \mid b \rightarrow ((a \rightarrow P) \parallel Q))$$

The above laws can be generalized using the general choice operator.

Let

$$P = (x : A \rightarrow P(x))$$

$$Q = (y : B \rightarrow Q(y))$$

Then,

L6 $(P \parallel Q) = (z : C \rightarrow P' \parallel Q')$

where

$$C = (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P)$$

and

$$P' = P(z) \quad \text{if } z \in A$$

$$P' = P \quad \text{otherwise}$$

and $Q' = Q(z)$ if $z \in B$

$$Q' = Q \quad \text{otherwise}$$

The above laws permit a process which is defined using a concurrency operator to be redefined without that operator, as the following example shows.

Let

$$\alpha P = \{a, b\}$$

$$\alpha Q = \{b, c\}$$

$$P = (a \rightarrow b \rightarrow P)$$

$$Q = (b \rightarrow c \rightarrow Q)$$

Then,

$$P \,||\, Q = (a \rightarrow b \rightarrow P) \,||\, (b \rightarrow c \rightarrow Q) \qquad \text{by definition}$$

$$= a \rightarrow ((b \rightarrow P) \,||\, (b \rightarrow c \rightarrow Q)) \qquad \text{by L3}$$

$$= a \rightarrow b \rightarrow (P \,||\, (c \rightarrow Q)) \qquad \text{by L1}$$

Let $R = (P \,||\, (c \rightarrow Q))$, then

$$R = (a \rightarrow b \rightarrow P) \,||\, (c \rightarrow Q) \qquad \text{by definition}$$

$$= (a \rightarrow (b \rightarrow P) \,||\, (c \rightarrow Q)$$

$$\quad | \, c \rightarrow (P \,||\, Q)) \qquad \text{by L3, L4}$$

$$= (a \rightarrow c \rightarrow (b \rightarrow P) \,||\, Q)$$

$$\quad | \, c \rightarrow (P \,||\, Q)) \qquad \text{by L4}$$

$$= (a \rightarrow c \rightarrow b \rightarrow (P \,||\, (c \rightarrow Q))$$

$$\quad | \, c \rightarrow a \rightarrow b \rightarrow (P \,||\, (c \rightarrow Q))) \qquad \text{byL1, definition}$$

$$= (a \rightarrow c \rightarrow b \rightarrow R$$

$$\quad | \, c \rightarrow a \rightarrow b \rightarrow R)$$

Therefore,

$$P \,||\, Q = (a \rightarrow b \rightarrow R)$$

2.2.7.2. The General Concurrency

The concurrency operator can be defined for more than two processes as follows:

Let

$$P_i = (x : A_i \rightarrow P_i(x)) \qquad \text{for } i = 1, 2, ..., n$$

describe n guarded processes.

Let $S_n = 2^{\{1, 2, ..., n\}}$

S_n is the set of all subsets of $\{1, 2, ..., n\}$.

Let

$$B_{n,\,s} = (\bigcap_{i \in s} A_i) - (\bigcup_{i \notin s} \alpha P_i) \qquad \text{for } i = 1, 2, ..., n$$

$$B_{n,\,\{\}} = \varnothing$$

and

$$C_n = \bigcup_{s \in S_n} B_{n,\,s}$$

Then, the following defines n processes running concurrently:

$$\|_{i=1}^{n} P_i = (z : C_n \rightarrow \|_{i=1}^{n} P'_i)$$

where

$$P'_i = P_i(z) \quad \text{if } z \in A_i$$
$$P'_i = P_i \quad \text{otherwise}$$

The main difference between the general definition and the definition for two processes is the way in which the set C_n is defined.

The proof for consistency between the two definition is as follows:

2.2.7.3. Theorem

The general concurrency definition of Section 2.2.7.2. is consistent with the rule **L6** in Section 2.2.7.1. In other words:

$$\left|\left|_{i=1}^{n} P_i = ((\ldots(P_1 \,||\, P_2) \,||\, P_3) \ldots \,||\, P_{n-1}) \,||\, P_n\right.\right.$$

where the parallel operation on the left hand side comes from the definition in Section 2.2.7.2 and the parallel operations on the right hand side come from definition **L6** in Section 2.2.7.1.

Proof

The proof is by induction on "n". The theorem is true for two processes (n = 2) as shown in the following:

Let

$$P_1 = (x_1 : A_1 \rightarrow P_1(x_1))$$
$$P_2 = (x_2 : A_2 \rightarrow P_2(x_2))$$

Then,

$$\left|\left|_{i=1}^{2} P_i = (z : C_2 \rightarrow \left|\left|_{i=1}^{2} P'_i\right.\right.\right.\right)$$

where

$$P'_i = P_i(z) \quad \text{if } z \in A_i$$
$$P'_i = P_i \quad \text{otherwise}$$

$$S_2 = 2^{\{1, 2\}} = \{\{\}, \{1\}, \{2\}, \{1, 2\}\}$$

$$C_2 = \bigcup_{s \in S_2} B_{2, s} =$$

$$B_{2, \{\}} \cup B_{2, \{1\}} \cup B_{2, \{2\}} \cup B_{2, \{1, 2\}}$$

$$B_{2, \{\}} = \varnothing$$
$$B_{2, \{1\}} = A_1 - \alpha P_2$$
$$B_{2, \{2\}} = A_2 - \alpha P_1$$
$$B_{2, \{1, 2\}} = A_1 \cap A_2$$

Then,

$$C_2 = \varnothing \cup A_1 - \alpha P_2 \cup A_2 - \alpha P_1 \cup (A_1 \cap A_2)$$

Thus, for two processes, the former definition is consistent with the latter. The induction base is established.

The induction hypothesis is that the two definitions are consistent for n = k. The induction step will be to show that:

$$\left|\left|_{i=1}^{k+1} P_i = (z : C_{k+1} \rightarrow \left|\left|_{i=1}^{k+1} P'_i\right.\right.\right.$$

where C_{k+1} and P'_i are defined as in Section 2.2.7.2.

Let

$$Q = \left|\left|_{i=1}^{k} P_i\right.\right.$$

Thus,

$$((\ldots(P_1 \,||\, P_2) \,||\, P_3)\ldots \,||\, P_k) \,||\, P_{k+1} = Q \,||\, P_{k+1}$$

It remains to show that:

$$Q \,||\, P_{k+1} = \mathop{||}_{i=1}^{k+1} P_i$$

By the induction hypothesis,

$$Q = (b : C_k \rightarrow \mathop{||}_{i=1}^{k} P'_i)$$

where

$$C_k = \bigcup_{s \in S_k} B_{k,s}$$

Then, by the induction base:

$$Q \,||\, P_{k+1} = (z : C' \rightarrow Q' \,||\, P'_{k+1})$$

Note that:

$$Q' = Q(z) \qquad \text{if } z \in C_k$$

$$Q' = Q \qquad \text{otherwise}$$

and

$$P'_n = P_n(z) \qquad \text{if } z \in A_n$$

$$P'_n = P_n \qquad \text{otherwise}$$

Now, it remains to show that:

$$C' = C_{k+1}$$

$$C' = (C_k - \alpha P_{k+1}) \cup (A_{k+1} - \alpha Q) \cup (C_k \cap A_{k+1})$$

$$= (\bigcup_{s \in S_k} B_{k,s} - \alpha P_{k+1}) \cup (A_{k+1} - (\bigcup_{i=1}^{k} \alpha P_i)) \cup (\bigcup_{s \in S_k} B_{k,s} \cap A_{k+1})$$

Let $s \in S_{k+1}$

Case 1: $k+1 \notin s$

Then:

$$B_{k+1,s} = B_{k,s} - \alpha P_{k+1}$$

Case 2: $s = \{k+1\}$

Then:

$$B_{k+1,s} = (A_{k+1} - (\bigcup_{i=1}^{k} \alpha P_i))$$

Case 3: $(k+1 \in s) \wedge (s - \{k+1\} \neq \varnothing)$

Then:

$$B_{k+1,s} = B_{k,s} \cap A_{k+1}$$

Thus, for $s \in S_{k+1}$

$$B_{k+1,s} =$$

$$(\bigcup_{s \in S_k} B_{k,s} - \alpha P_{k+1}) \cup (A_{k+1} - (\bigcup_{i=1}^{k} \alpha P_i)) \cup (\bigcup_{s \in S_k} B_{k,s} \cap A_{k+1})$$

Therefore, $C_{k+1} = C'$, thus,

$$\left\|_{i=1}^{k+1} P_i = \left(\left\|_{i=1}^{k} P_i\right) \right\| P_{k+1}$$

This completes the proof.

Consider the following example:

Let

$P = (a \rightarrow b \rightarrow STOP)$ where $\alpha P = \{a, b\}$

$Q = (b \rightarrow c \rightarrow STOP)$ where $\alpha Q = \{b, c\}$

$R = (d \rightarrow a \rightarrow b \rightarrow STOP)$ where $\alpha R = \{a, b, d\}$

Then,

$$P \,||\, Q \,||\, R = (d \rightarrow a \rightarrow b \rightarrow c \rightarrow STOP)$$

The process P is ready to engage in the event "a" which is also in the alphabet of process R. But process R is not ready to engage in the event "a". Therefore, the event "a" which needs the participation of both P and R cannot occur. Furthermore, the process Q is ready to engage in the event "b" which is also in the alphabet of both P and R. The event "b" can not occur since P and R are not ready to engage in "b". Process R is ready to engage in the event "d" which is not in the alphabet of any other processes. Therefore, the event "d" is the only event that can occur first when P, Q, and R start running concurrently.

After the event "d" occurs then P and R are both ready to engage in the event "a" which is not in the alphabet of Q. Therefore the next possible event to occur is the event "a". After the event "a" occurred, then all three processes are ready to engage in the event "b". The last event to occur is the event "c" which is in only the

Chapter Three

Main Theorems

This chapter includes a trivial proof to show that for every Petri net there exists a Hoare process which generates the same *prefix* language as the corresponding Petri net. This is accomplished by an obvious mapping from Petri nets to Hoare processes. In addition, another systematically defined mapping from Petri nets to Hoare processes is given and proved to produce for each Petri net a Hoare process whose language is the prefix language of that net.

Furthermore, an example will be presented to show that there exists a Hoare process which cannot be modeled by any Petri net.

3.1. Theorem

Let $M = (P, T, I, O, \mu)$ be a marked Petri net, and let L(M) be the prefix language of M. Then, (T*, L(M)) is the equivalent Hoare process whose prefix language is L(M).

Proof

The Language L(M) is a subset of T* which satisfies the following two conditions:

$$1)\ \langle\rangle \in L(M)$$

$$2)\ \forall\, s, t,\quad s\hat{}t \in L(M) \Rightarrow s \in L(M)$$

(by the definition of prefix Petri net languages). Thus, the Hoare process (T*, L(M)) has the same prefix language as the marked Petri net M. (This easy argument was suggested by Dr. William R.

3.2. Basic Definitions

Let M = (P, T, I, O, μ) be a marked Petri net, and let t $\in$ T. Then, define:

$$IN(t) = \{p \in P : p \text{ is an input place for } t\}$$

$$OUT(t) = \{p \in P : p \text{ is an output place for } t\}$$

Thus, IN and OUT are sets (rather than bags) of input places and output places, respectively, for a given transition. Note that I(t) and O(t) were defined to be the corresponding bags.

Let

$$P = \{p_1, p_2, ..., p_n\} \qquad |P| = n$$

Recall that:

$$\lambda(p, t) = \#(p, O(t)) - \#(p, I(t))$$

Thus, $\lambda(p, t)$ is the change in number of tokens in place p if transition t fires.

3.3. The Function Ψ

Define Ψ to be a mapping from the set of all Petri nets into the set of Hoare processes as follows:

For each i = 1, ..., n and nonnegative integer k define the Hoare process $Q_{i, k}$ by:

$$Q_{i,k} = (t : C_{i,k} \rightarrow \quad Q_{i,k+\lambda(p_i,t)})$$

where

$$C_{i,k} = \{t \in T : \quad p_i \in (O(t) \cup I(t)) \wedge (k \geq \#(p_i, I(t))) \quad \}$$

(Note that, $Q_{i,k} = STOP_{p_i}$ if and only if, $C_{i,k} = \varnothing$.)

The alphabet of $Q_{i,k}$ can be defined as:

$$\alpha Q_{i,k} = \{t \in T : p_i \in (O(t) \cup I(t)) \}$$

Note that the alphabet of $Q_{i,k}$ does not depend on k.

Define:

$$\Psi(M) \quad = \quad \|_{i=1}^{n} Q_{i,\mu(p_i)}$$

Note that the set of simultaneous equations for $Q_{i,\mu(p_i)}$ has a unique solution (refer to Section 2.2.6.).

The mapping $\Psi(M)$ is defined to take a marked Petri net with n places as its only argument and to return a Hoare process. The Hoare process is defined as n guarded processes running in parallel, where each process simulates a specific place of the Petri net.

The set $C_{i,\mu(p_i)}$ is defined to be the set of all transitions in the alphabet of the process $Q_{i,\mu(p_i)}$ for which either p_i is an output place for the transition (since the condition ($\mu(p_i) \geq \#(p_i, I(t))$) is true for all output places), or p_i is an input place for the transition and the number

of tokens in p_i is greater than or equal to the number of arcs between the transition and p_i. Having the constraint,

$$\mu(p_i) \geq \#(p_i, I(t))$$

(in $C_{i, \mu(p_i)}$), guarantees that if p_i is an input place for the transition t, then it can contribute in firing the transition t when p_i is needed to participate in parallel with other processes.

3.3.1 Example

Consider the Petri net of Figure 2.1. Then,

$$\Psi(M) = \prod_{i=1}^{5} Q_{i, \mu(p_i)} =$$

$$Q_{1, \mu(p_1)} \,||\, Q_{2, \mu(p_2)} \,||\, Q_{3, \mu(p_3)} \,||\, Q_{4, \mu(p_4)} \,||\, Q_{5, \mu(p_5)} =$$

$$Q_{1, 1} \,||\, Q_{2, 0} \,||\, Q_{3, 0} \,||\, Q_{4, 0} \,||\, Q_{5, 0}$$

where

$$Q_{1, 1} = (t : C_{1, 1} \rightarrow Q_{1, 1 + \lambda(p_1, t)})$$

$$C_{1, 1} = \{t \in T : p_1 \in (O(t) \cup I(t)) \wedge (1 \geq \#(p_1, I(t)))\}$$
$$= \{t_1\}$$

Therefore,

$$Q_{1, 1} = (t_1 \rightarrow Q_{1, 0}) \quad \text{with} \quad \alpha Q_{1, k} = \{t_1\}$$

Similarly,

$$Q_{2, 0} = (t_1 \rightarrow Q_{2, 1}) \quad \text{with} \quad \alpha Q_{2, k} = \{t_1, t_2\}$$
$$Q_{3, 0} = (t_1 \rightarrow Q_{3, 1}) \quad \text{with} \quad \alpha Q_{3, k} = \{t_1, t_3\}$$

$$Q_{4,0} = (t_1 \rightarrow Q_{4,3}) \qquad \text{with} \quad \alpha Q_{4,k} = \{t_1, t_2, t_3\}$$
$$Q_{5,0} = (t_3 \rightarrow Q_{5,1}) \qquad \text{with} \quad \alpha Q_{5,k} = \{t_3\}$$

Thus, the process $\Psi(M)$ can engage only in the event t_1.

Firing the transition t_1 in the Petri net M results in a Petri net M' with a new marking μ' (Figure 2.2.).

$$\Psi(M) = \prod_{i=1}^{n} Q_{i,\mu(p_i)}$$

$$Q_{1,\mu'(p_1)} \,||\, Q_{2,\mu'(p_2)} \,||\, Q_{3,\mu'(p_3)} \,||\, Q_{4,\mu'(p_4)} \,||\, Q_{5,\mu'(p_5)} =$$

$$Q_{1,0} \,||\, Q_{2,1} \,||\, Q_{3,1} \,||\, Q_{4,3} \,||\, Q_{5,0}$$

where

$$Q_{1,0} = (t : C_{1,0} \rightarrow Q_{1,0} + \lambda(p_1, t))$$
$$C_{1,0} = \{t \in T : p_1 \in (O(t) \cup I(t)) \wedge (0 \geq \#(p_1, I(t)))\}$$
$$= \varnothing$$

Therefore,

$$Q_{1,0} = STOP_{p_1}$$

Similarly,

$$Q_{2,1} = (t_j : \{t_1, t_2\} \rightarrow Q_{2,1} + \lambda(p_2, t_j))$$
$$Q_{3,1} = (t_j : \{t_1, t_3\} \rightarrow Q_{3,1} + \lambda(p_3, t_j))$$
$$Q_{4,3} = (t_j : \{t_1, t_2, t_3\} \rightarrow Q_{4,3} + \lambda(p_4, t_j))$$

$$Q_{5,0} = (t_3 \rightarrow \quad Q_{5,1})$$

Thus, the process $\Psi(M')$ can engage in either the event t_2 or t_3.

3.4. Theorem

Let $M = (P, T, I, O, \mu)$ be a marked Petri net. Then transition t_j can fire in the Petri net M if and only if the process $\Psi(M)$ can engage in the event t_j.

Proof

Let

$$P = \{p_1, p_2, ..., p_n\} \qquad |P| = n$$
$$IN(t_j) = \{p_{i_1}, p_{i_2}, ..., p_{i_r}\}$$
$$OUT(t_j) - IN(t_j) = \{p_{o_1}, p_{o_2}, ..., p_{o_s}\}$$

3.4.1. ($\Rightarrow$)

Assume the transition t_j can fire in M. Then, by the definition of the function Ψ from Section 3.3.,

$$\Psi(M) \quad = \quad \prod_{i=1}^{n} Q_{i, \mu(p_i)}$$

where

$$Q_{i, \mu(p_i)} = (t_k : C_{i, \mu(p_i)} \quad \rightarrow \quad Q_{i, \mu(p_i)} + \lambda(p_i, t_j))$$

Recall that:

$$\xi(Q_{i,\,\mu(p_i)}) = C_{i,\,\mu(p_i)}$$

is the set of choices for the process $Q_{i,\,\mu(p_i)}$.

In order to show that the process $\Psi(M)$ can engage in the event t_j, it is enough to show that

$$t_j \in \xi(M)$$

where

$$\xi(M) = \xi\Big(\,\Big|\Big|_{i=1}^{n} Q_{i,\,\mu(p_i)}\Big)$$

$$= \bigcup_{s \in S_n} \Big(\Big(\bigcap_{i \in s} C_{i,\,\mu(p_i)} \Big) - \Big(\bigcup_{i \notin s} \alpha Q_{i,\,\mu(p_i)} \Big) \Big)$$

Therefore, it is enough to show

$$t_j \in \Big(\bigcap_{i \in s} C_{i,\,\mu(p_i)} \Big) - \Big(\bigcup_{i \notin s} \alpha Q_{i,\,\mu(p_i)} \Big) \qquad \text{for some } s \in S_n$$

Claim 1:

$$\text{for } s = \{i_1, i_2, \ldots, i_r\} \cup \{o_1, o_2, \ldots, o_s\}$$

$$t_j \in \Big(\bigcap_{i \in s} C_{i,\,\mu(p_i)} \Big)$$

Proof of Claim 1:

First, let $i \in \{i_1, i_2, ..., i_r\}$. Since the transition t_j can fire in the marked Petri net M, then by the definition of enabled transition in Section 2.1.3.,

$$\mu(p_i) \geq \#(p_i, I(t_j)) \qquad \forall\, p_i \in P$$

Then,

$$t_j \in \bigcap_{i \in \{i_1, i_2, ..., i_r\}} C_{i, \mu(p_i)}$$

Now let $i \in \{o_1, o_2, ..., o_s\}$, by the definition of $C_{i, \mu(p_i)}$; then:

$$t_j \in \bigcap_{i \in \{o_1, o_2, ..., o_r\}} C_{i, \mu(p_i)}$$

Thus,

$$t_j \in \bigcap_{i \in s} C_{i, \mu(p_i)}$$

This proves the Claim 1.

<u>Claim 2:</u>

Let

$i \notin \{i_1, i_2, ..., i_r\} \cup \{o_1, o_2, ..., o_s\}$; then,

$$t_j \notin \alpha Q_{i, \mu(p_i)}$$

<u>Proof of Claim 2</u>:

Suppose:

$$t_j \in \alpha Q_{i, \mu(p_i)}$$

Observe that:

$$\alpha Q_{i,\,\mu(p_i)} = \{t \in T : \quad p_i \in (OUT(t) \;-\; IN(t)) \text{ or } p_i \in IN(t)\}$$

The sets (OUT(t) - IN(t)), and IN(t) are disjoint.

If $t_j \in \alpha(Q_{i,\,\mu(p_i)})$ then,

Case 1:

$$p_i \in (OUT(t_j) \;-\; IN(t_j))$$

Then,

$$i \in \{o_1, o_2, ..., o_s\}$$

which is a contradiction.

Case 2:

$$p_i \in IN(t_j)$$

Then,

$$\mu(p_i) \geq \#(p_i, I(t_j))$$

and thus,

$$i \in \{i_1, i_2, ..., i_r\}$$

which is a contradiction.

Therefore, if

$$i \notin \{i_1, i_2, ..., i_r\} \cup \{o_1, o_2, ..., o_s\}$$

then,

$$t_j \notin \alpha(Q_{i,\,\mu(p_i)})$$

This proves the Claim 2.

Thus, by the definition of the general concurrency in Chapter

$$t_j \in \xi(\Psi(M))$$

Therefore, the process $\Psi(M)$ can engage in the event t_j.

3.4.2. ($\Leftarrow$)

Let $M = (P, T, I, O, \mu)$ be a marked Petri net and assume the process $\Psi(M)$ is able to engage in the event t_j. Then, the transition t_j is an enabled transition in the Petri net M.

By the definition of the function Ψ from Section 3.3.,

$$\Psi(M) = \prod_{i=1}^{n} Q_{i, \mu(p_i)}$$

where

$$Q_{i, \mu(p_i)} = (t_k : C_{i, \mu(p_i)} \rightarrow Q_{i, \mu(p_i)} + \lambda(p_i, t_j))$$

Since the process $\Psi(M)$ can engage in the event t_j, then:

$$t_j \in \xi\left(\prod_{i=1}^{n} Q_{i, \mu(p_i)} \right)$$

$$= \bigcup_{s \in S_n} \left(\left(\bigcap_{i \in s} C_{i, \mu(p_i)} \right) - \left(\bigcup_{i \notin s} \alpha Q_{i, \mu(p_i)} \right) \right)$$

Therefore, for some $s \in S_n$

$$t_j \in \left(\bigcap_{i \in s} C_{i, \mu(p_i)} \right) - \left(\bigcup_{i \notin s} \alpha Q_{i, \mu(p_i)} \right)$$

Thus,

$$t_j \in C_{i,\mu(p_i)} \quad \forall\ i \in s$$

and

$$t_j \notin \alpha Q_{i,\mu(p_i)} \quad \forall\ i \notin s$$

In order to show that the transition t_j can fire in the marked Petri net M, by the definition of enabled transition in Section 2.1.3., it will be enough to show:

$$\mu(p_i) \geq \#(p_i, I(t_j)) \qquad \forall\ p_i \in P$$

Claim

The transition t_j is an enabled transition in M.

Suppose that:

$$\mu(p_i) < \#(p_i, I(t_j)) \qquad \text{for some } p_i \in IN(t_j)$$

Case 1: $i \in s$

Recall that:

$$C_{i,k} = \{t \in T : \ p_i \in (O(t) \cup I(t)) \wedge (k \geq \#(p_i, I(t)))\ \}$$

Thus,

$$t_j \notin C_{i,\mu(p_i)}$$

which is a contradiction.

Case 2: $i \notin s$

Since:

$$p_i \in IN(t_j)$$

Then, by the definition of alphabet,

$$t_j \in \alpha Q_{i,\, \mu(p_i)}$$

which is a contradiction. This ends the proof of the claim. Thus, the transition t_j is an enabled transition in the Petri net M.

This ends the proof of the Theorem 3.4.

3.5. Theorem

Let M = (P, T, I, O, μ) be a marked Petri net and let t_j be an enabled transition. Suppose that firing the transition t_j results in the Petri net M′, and engaging the process Ψ(M) in the event t_j results in the process Q. Then:

$$\Psi(M') = Q$$

Proof

If the transition t_j fires in the marked Petri net M then, M′= (P, T, I, O, μ') denotes the Petri net with marking μ',

where

$$\mu'(p_i) = \mu(p_i) + \lambda(p_i, t_j) \qquad \forall\, p_i \in P$$

Then:

$$\Psi(M') = \prod_{i=1}^{n} Q_{i,\, \mu'(p_i)}$$

$$= \prod_{i=1}^{n} Q_{i,\, \mu(p_i) + \lambda(p_i, t_j)}$$

By the definition of Ψ, then:

$$\Psi(M') = \Psi(M)/t_j$$
$$= Q$$

(The notation " PROC/eve " describes the behavior of the process PROC after it engaged in the event eve.) This ends the proof of the

3.6. Theorem

Let M = (P, T, I, O, μ) be a marked Petri net and let $\sigma \in T^*$. Then, the string σ is acceptable by M if and only if it is acceptable by

Proof

Let

$$\sigma = t_{f_1} t_{f_2} \ldots t_{f_s}$$

The proof is by induction on s. The theorem is true for strings of length one, by the Theorem 3.4.

Assume exactly the same strings of length (s-1) are acceptable by both models; then the induction step is to prove that exactly the same strings of length s are also acceptable by both models. If the string σ is acceptable by the Petri net M, then:

$$\alpha = t_{f_1} t_{f_2} \ldots t_{f_{s-1}}$$

is acceptable by the Petri net M. By induction hypothesis the string α is acceptable by the process $\Psi(M)$. Similarly, if the string α is acceptable

by the process $\Psi(M)$, then the string α is also acceptable by the Petri net M.

Let the Petri net:

$$M^{s-1} = (P, T, I, O, \mu^{s-1})$$

describe the state of M after firing (s - 1) transitions:

$$t_{f_1}, t_{f_2}, ..., t_{f_{s-1}}$$

Furthermore, let process Q describe the behavior of the process $\Psi(M)$ after the string α occurs. Then, by the Theorem 3.5. and the induction hypothesis:

$$\Psi(M^{s-1}) = Q$$

By the Theorem 3.4., the transition t_{f_s} can fire in the Petri net M^{s-1} if and only if the process $\Psi(M^{s-1})$ can engage in the event t_{f_s}.

Therefore, the string σ is acceptable by the Petri net M if and only if it is acceptable by the process $\Psi(M)$.

3.7. Definition

Two Petri nets M and M' are equivalent when σ is acceptable by M if and only if it is acceptable by M'. The notation " $\approx$ " is used for equivalence. The same definition is used for equivalence of two

3.8. Theorem

Let M and M' be two Petri nets. Then:

$$M \approx M' \text{ if and only if } \Psi(M) \approx \Psi(M')$$

Proof

By the theorem 3.6., the string σ is acceptable by $\Psi(M)$ if and only if it is acceptable by M. Furthermore, by the definition of equivalence, the string σ is acceptable by M if and only if it is acceptable by M'. Then, by the theorem 3.6., the string σ is acceptable by M' if and only if it is acceptable by $\Psi(M')$.

This theorem proves that the function Ψ is a one-to-one function up to the equivalence relation.

3.9. A Hoare Process that Cannot be Modeled by a Petri net

In this section an example of a Hoare process that cannot be modeled by any Petri net will be presented. Recall the process CT_0 from Section 2.2.6.,

$$CT_0 = (up \rightarrow CT_1 \mid around \rightarrow CT_0)$$
$$CT_{n+1} = (up \rightarrow CT_{n+2} \mid down \rightarrow CT_n)$$

3.9.1. Theorem

There exist no Petri net that can model CT_0.

Proof

The proof of the theorem is by contradiction. Let M = (P, T, I, O, μ) be a marked Petri net that models CT_0, where,

$$T = \{up, down, around\}$$

Since the transition "around" can fire in CT_0 and does not fire in CT_n (for all $n > 0$) then, I(around) must be nonempty.
(Note that if the transition "around" did not have any input place then it could have fire in any state.) By the definition of enabled transition,

$$\mu(p_i) \geq \#(p_i, I(around)) \qquad \forall\, p_i \in P$$

Let μ' be the new marking for the Petri net M after the transition "up" fires. Then, the process CT_0 becomes the process CT_1. In CT_1 the transition "around" cannot fire. Thus, there exists at least one place $p_k \in I(around)$ such that:

$$\mu'(p_k) < \#(p_k, I(around))$$

From the two above inequalities, it can be concluded that:

$$\mu'(p_k) < \mu(p_k)$$

The above inequality shows that when the transition "up" fires, the number of tokens in the place p_k is reduced.

By the definition of function λ,

$$\lambda(p_k, up) = \#(p_k, O(up)) - \#(p_k, I(up))$$

but by a remark in Section 3.1.,

$$\lambda(p_k, up) = \mu'(p_k) - \mu(p_k)$$

Thus, above argument shows that:

$$\lambda(p_k, up) < 0$$

After firing the transition "up",

$$\lfloor \mu(p_k) \ / \ \lambda(p_k, up) \rfloor$$

times, there would not be enough tokens in the place p to fire the transition "up". Thus, the Petri net cannot model the process:

$$CT \lfloor \mu(p_k) \ / \ \lambda(p_k, up) \rfloor + 1$$

which is a contradiction. Therefore, there exist no Petri net that can model the process CT_0.

3.10. Other Classes of Languages

Petri nets and the Hoare processes have been compared with regard to prefix languages. In the remaining of this chapter, the two models will be compared with regard to another class of languages.

If M = (P, T, I, O, μ) is marked Petri net and F is a finite set of final markings for M then, yet another class of languages called *G-type* (Peterson [81]) can be defined as:

$$L_g(M, F) = \{\sigma \in T^* : \text{there exists } \mu_f \in F \text{ such that } \delta(\mu, \sigma) \geq \mu_f\}$$

In other words, a sequence of transitions σ is in $L_g(M, F)$ if, after the transitions of σ have fired, the Petri net M has the marking $\delta(\mu, \sigma)$, and $\delta(\mu, \sigma)$ has at least as many tokens in each place as some final marking in F.

For example, Let $F = \{\mu_f\}$

where

$$\mu_f(p_1) = 0$$
$$\mu_f(p_2) = 1$$
$$\mu_f(p_3) = 0$$
$$\mu_f(p_4) = 0$$
$$\mu_f(p_5) = 1$$

then, the G-type language of the Petri net relative to F of Figure 2.1 is:

$$L_g = \{t_1t_3, t_1t_3t_2, t_1t_2t_3\}$$

The class of prefix languages is a subset of the class of G-type languages. Every prefix language for a given Petri net is a G-type language with a final marking of zero in every place (F = { (0, 0, . . ., 0) }). Clearly, the union of all G-type languages for a given Petri net is the prefix language of that Petri net.

It is possible to define a class of languages for Hoare processes with features analogous to those of the G-type languages for Petri nets. In particular, a class of languages can be defined to contain an special symbol for a successful termination of a Hoare process (thus, a successful termination will be represented differently from a deadlock which is presented by the process $STOP_T$). Hoare defines the process $SKIP_T$ as a process which does nothing but terminate successfully. Thus, the traces of the process $SKIP_T$ can be defined as:

$$traces(SKIP_T) = \{<>, <\surd>\}$$

The alphabet of the process $SKIP_T$ can be defined as:

$$\alpha SKIP_T = T \cup \{ \surd \}$$

(Note that successful termination is regarded as a special event, denoted by the symbol $\surd$ (read "success").)

The following rules are given by Hoare [85]:

$$((x: B \rightarrow P(x)) \;||\; SKIP_A) = (x: (B - A) \rightarrow (P(x) \;||\; SKIP_A))$$

$$STOP_A \;||\; SKIP_B = SKIP_B \qquad \text{if } \surd \notin A \text{ and } B \supseteq A$$

The *ST-type* (successfully terminated) language of a Hoare process P (denoted by $L_{st}(P)$) can be defined as:

$L_{st}(P) = \{ \sigma \in (\alpha P)^* : \sigma^\wedge\langle\surd\rangle$ is in the prefix language of P$\}$

The success event $\surd$ is recorded in the traces of the language but is not part of the string (i.e. it works as a termination symbol).

3.10.1. Concealment

The alphabet of a process contains those events which are considered to be relevant to a given system. However, in describing the internal behavior of a system, some events represent the internal transitions which may denote the communication between the concurrently acting components of that system. These events can occur in a system without being observed or recorded. A sequence of events can be modified by the removal of all such internal events.

If C is a finite set of events to be concealed and σ is a string then

$$\sigma \setminus C$$

is the string which all the occurrences of any event in C is removed

from it.

If P is a process then

$$P \setminus C$$

is a process which behaves like P, except that each occurrence of any event in C is concealed.

If L(M) is the language of the Petri net M then

$$L(M) \setminus C$$

is the language which all the occurrences of any event in C is removed from the strings in L(M).

3.11. Modifying the Function Ψ

If a G-type language is given for a Petri net then a modified function Ψ may be defined in such a way as with ST-type language equal to the given G-type language to generate a Hoare process. In particular, Ψ must be modified so that it can recognize a successful termination of the process.

Let M = (P, T, I, O, μ) be a marked Petri net and let F = $\{\mu_1, \mu_2, \ldots \mu_m\}$ be a set of final markings for M. A new Petri net C = (P, T $\cup$ E, I', O', μ) is constructed from M and F as follows. The Petri net C has the same places as M but it has m added ε-labeled transitions (i.e. elements of E):

$$E = \{\varepsilon_1, \varepsilon_2, \ldots \varepsilon_m\}$$

For each μ_i in F the input bag of the corresponding ε-labeled transition can be defined by $\#(p, I'(\varepsilon_i)) = \mu_i(p)$ (i = 1, 2, ..., m). The output bags of all the ε-labeled transitions are empty. Furthermore, if $t \neq \varepsilon_i$

$$I'(t) = I(t)$$

$$O'(t) = O(t)$$

The Petri net C is constructed such that only one of the ε-labeled transitions can become enabled when the Petri net C is in the corresponding final state (i.e. final marking).

3.11.1. Lemma

Let $M = (P, T, I, O, \mu)$ be a marked Petri net, let F be the set of final markings for M, and let the Petri net C be constructed from M as in section 3.11. If $\sigma \in L_g(C, F)$, then $\sigma' = \sigma \setminus E$ is in the prefix language of C.

Proof

Assume the string σ' is not in the prefix language of C. Let $\sigma' = \tau' t \gamma'$ where τ' is the longest substring of σ' such that $\delta_C(\mu, \tau')$ is defined. Thus, after firing the sequence of transitions in τ' then the transition t is not an enabled transition.

Then the string σ can be decomposed into $\sigma = \tau t \gamma$ where

$$\tau \setminus E = \tau'$$

$$\gamma \setminus E = \gamma'$$

Since the string σ is in the prefix language of C, then τt, which is a prefix of σ, is also in the prefix language of C. Thus, after firing the transitions in τ the Petri net C is in the marking

$$\delta_C(\mu, \tau)$$

and the transition t is an enabled transition.

Every ε-labeled transition has a non-empty input bag and an empty output bag. Thus, firing an ε-labeled transition always consumes some tokens. Therefore,

$$\delta_C(\mu, \tau') \geq \delta_C(\mu, \tau)$$

This inequality and the fact that the transition t is enabled in C with the marking $\delta_C(\mu, \tau)$ indicate that t must be also enabled in the marking $\delta_C(\mu, \tau')$ which is a contradiction.

This ends the proof of the lemma 3.11.1.

3.11.2. Lemma

Let M = (P, T, I, O, μ) be a marked Petri net, let F be the set of final markings for M, and let the Petri net C be constructed from M as in section 3.11. The concealed G-type language of the Petri net C with respect to E is the same as the G-type language of the Petri net M. In other words,

$$L_g(C, F) \setminus E = L_g(M, F)$$

Proof

Let $\sigma \in L_g(M, F)$. Since every transition in M is also a transition in C with the same input bag and the same output bag then $\sigma \in L_g(C, F)$.

$$\delta_C(\mu, \sigma) \geq \mu_f \qquad \text{for some } \mu_f \in F$$

Let $\sigma' = \sigma \setminus E$ be the sequence of transitions obtained from σ by

removing the ε-labeled transitions.

Since the string σ' is in the prefix language of C (by lemma 3.11.1.) the marking $\delta_C(\mu, \sigma')$ is defined. Firing an ε-labeled transition always consumes some tokens. Therefore,

$$\delta_C(\mu, \sigma') \geq \delta_C(\mu, \sigma)$$

so that

$$\delta_C(\mu, \sigma') \geq \mu_f$$

But since σ' does not have any ε-labeled transition, then

$$\delta_C(\mu, \sigma') = \delta_M(\mu, \sigma')$$

and

$$\delta_M(\mu, \sigma') \geq \mu_f$$

Thus,

$$\sigma' \in L_g(M, F)$$

This proves the lemma 3.11.2.

3.12. Theorem

Let M = (P, T, I, O, μ) be a marked Petri net, let F be a set of final markings for M, and let the Petri net C be constructed from M as in section 3.11. Then, there is a Hoare process $\Psi(C) \;||\; (E \rightarrow SKIP_T)$ such that

$$L_{st}(\Psi(C) \;||\; (x : E \rightarrow SKIP_T)) \setminus E = L_g(M, F)$$

Proof

$\sigma \in L_g(M, F)$

$\Leftrightarrow$ $\sigma \in L_g(C, F) \setminus E$ (by lemma 3.11.2.)

$\Leftrightarrow$ $\delta_C(\mu, \sigma) \geq \mu_f$ for some $\mu_f \in F$ (by definition of L_g)

$\Leftrightarrow$ $\Psi(C)/\sigma$ can engage in an event in E (by construction of C)

$\Leftrightarrow$ $\sigma\surd \in L_{st}(\Psi(C) \,||\, (x: E \rightarrow SKIP_T)) \setminus E$ (by definition of L_{st})

This ends the proof of 3.12.

Chapter Four

Summary and Conclusions

4.1. Summary

The goal of this thesis has been the comparison of the Petri nets Model and the Hoare processes Model. The method of comparison was to show that for every Petri net there is a Hoare process that can generate the same prefix language as the Petri net, and there is a Hoare process which cannot be simulated by any Petri net.

The function Ψ was defined to map a marked Petri net to an equivalent Hoare process. This function, for each place in a marked Petri net, defines a corresponding Hoare process as an infinite set of mutually recursive equations. The processes run in parallel so that, in any given state, the set of the next possible choices for the Hoare processes is the same as the set of enabled transitions for the Petri net. The number of tokens in each place is used as the index in the equations which define the Hoare processes.

The Hoare processes start running in parallel with the indexes which are the same as the initial marking of the Petri net. Change in number of tokens in each place, during the execution of a Petri net, will result in the same change in the index of the corresponding Hoare process. The parallel operator, together with the function Ψ,

guarantees that only the enabled transitions of a Petri net, in any given state, appear as possible next events for the equivalent Hoare process.

In case of a deadlock in the Petri net, the equivalent Hoare process is the process STOP which cannot engage in any events of its

4.2. Conclusions and Suggestions for Future Research

As it was mentioned early in this chapter, the main goal of this research was to make a comparison between the Petri nets model and The Hoare process model. This goal has been achieved to a great extent. A particular type of language for each model was selected, and it was proved that Hoare processes are more powerful than Petri nets, with regard to those languages.

Future research can be done by choosing a different type of language for each model. For example, the class of *L-type* languages can be chosen for the Petri nets model. If the set F is a finite set of final markings for a Petri net then, yet another class of languages called *L-type* can be defined as:

a language L is in the class of *L-type* Petri net languages if there exists a marked Petri net $M = (P, T, I, O, \mu)$, and a finite set of final markings F such that:

$$L(M) = \{\sigma \in T^* : \delta(\mu, \sigma) \in F\}$$

Peterson [81] proves that the class of prefix languages is a subset of the class of L-type languages.

4.2.1. Petri net languages and Other Classes of Languages

Consider the class of L-type languages which does not require distinct labels for transitions and allows ε-labeled transitions, then Peterson [81] shows that the following statements are true about this larger class of Petri net languages and other formal languages. Every regular language is a Petri net language. This can be proved by showing that every finite state machine can be mapped to a Petri net which generates the same language as the finite state machine. Some context-free languages are Petri net languages (e.g. $L = \{a^n c b^n : n > 1\}$). However, there exist context-free languages which are not Petri net languages (e.g. $L = \{\sigma\sigma^R : \sigma \in T^*\}$). Furthermore, there exist context-sensitive but not context-free languages which are Petri net languages (e.g. $L = \{a^n b c^n d e^n : n > 0\}$). Finally, it is possible to show that all Petri net languages are context-sensitive languages.

REFERENCES

Azema, P., Diaz, M. *Multilevel Description Using Petri Nets.* IEEE, (September 1975), 188-190.

Dijkstra, E. *Cooperating Sequential Processes.* Academic Press, New York, NY, 1968.

Han, Y., Kinney, L. *Petri Net Reduction and Verification.* Honeywell, Minneapolis, MN, 1977.

Harrison, A. H. *Introduction to Formal Language Theory.* Addison-Wesley Pub. Com., Reading, Mass., 1978.

Hoare, C. A. R. *Communicating Sequential Processes.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

Hoare, C. A. R. *Communicating Sequential Processes.* Comm. ACM 21 (8), (1978), 667-677.

Lewis, H. R., Papadimitriou, C. H. *Elements of the Theory of Computation.* Printice-Hall, Inc., Englewood Cliffs, NJ, 1981.

Mandrioli, D., Ghezzi, C. *Theoretical Foundations of Computer Science.* John Wiley & Sons, Inc., New York, NY, 1987.

Peterson, J. L. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

Peterson, J. L. *Petri Nets.* Computing surveys 9, 3 (September 1977), 223-252.