ソフトウェア工学 101−14
（1994．11．18）

ペトリネットに基づくOSのための テストプログラムジェネレータの試作

渡辺 晴美　　工藤 知宏
東京工科大学

本報告では、OSの入力をシステムコールに限定した場合のテストプログラムのジェネレータの試作について述べる。テストケースの生成には、テスト・スイートが比較的短いことと、実装が持つ可能性のある誤りを可能な限りカバーすることが要求される。有限状態オートマトンに基づく方法は、トランスファーエラーすなわち状態遷移に伴ったエラーを検出できるという点でその他の方法に比べ後者の要求をより満たしている。しかし、タスク間通信などの並列性を持つソフトウエアについては、タスク毎に非決定性有限状態オートマトンを記述することになるため、オペレーティング・システムの検査には適さない。そこで本システムでは、個々のタスクをトークンに割り付けることができるカラーペトリネットをテストケースの生成に用いた。

A Testing Program Generator for Operating Systems based on Petri Nets

Harumi Watanabe　　Tomohiro Kudoh
Tokyo Engineering University

In this report, an experimental testing program generator for operating systems is presented. In this generator, inputs are restricted to system calls. In general, a test suite should have the following properties: relatively short, and cover possible faults as much as possible. While test selection methods based on Finite State Machines satisfy the second property better than other methods, they are not suitable for testing concurrent systems since every task should be represented with an Nondeterministic Finite-State Machine. To cope with this problem we adopt coloured Petri nets.

1 はじめに

OSの開発過程におけるテストの工程が占める割合は非常に大きいため、テストケースが自動的に生成されることは重要である。テストを行うためには、具体的な入力を準備する手段と、結果を判定する手段が必要である[3]。この有効な入力を準備する手段と結果を判定する手段が本報告に示すテストプログラム・ジェネレータ(TPジェネレータ)である。ここで、テストケースではなくテストプログラム(TP)である理由は、OSのテストの入力形式に関係する。OSの入力は割込とシステムコールの2つに大きく分類できるが、システムコールが入力となる場合、入力の形式はOS上で動くシステムコールのシーケンス、すなわちシステムコールを発行するプログラムになる。本報告では入がシステムコールである場合のテストケースの生成システムについて述べる。

ところで、より理想的なテストを行うためには、評価基準が必要である。理想的なテストを行うためのテストケースの選定基準は、VALIDかつRELIABLEであることと定義されている[7]が、完全なテスト・ケースを生成することは不可能である場合が多い。そこで、テストケースの生成には、テストケースの集合であるテストスイート(test suite)が比較的小さいことと、実装が持つ可能性のある誤りを可能な限りカバーすることが要求される。

本報告では、まず上記の性質を満たしているかどうかを評価する方法と、従来のテストケースの生成方法について述べる。次に、本システムの基礎となるカラーペトリネットとその可達木の同値マーキングによる縮約方法について定義し、TPジェネレータによるTPの生成方法について述べる。最後に従来の方法との比較を行う。

2 テスト

本節では、テストケースの評価方法と従来の生成方法について示す。

2.1 テストケースの評価

テストスイートの信頼性を評価する方法には、定性的な方法として、エラーの制御構造を分析する方法がある。エラーの制御構造は、以下の3つに分けることができる。[10]

AとA'は、同じ入力アルファベットを持った最小の有限状態オートマトン[12]であるとする。Aは仕様を有限状態オートマトンによりモデル化したものであり、A'は正しいオートマトンである。

【オペレーション・エラー(Operation Errors)】

AがA'と等しくなく、かつ(Aに状態を加えたり削除することなしに)Aの外部機能のみを変えることにより、A'と等しくなるようにAを変更できるならば、Aはオペレーション・エラーを持つという。

【トランスファー・エラー(Transfer Errors)】

AがA'と等しくなく、かつ(Aに状態を加えたり削除することなしに)Aの状態遷移関数のみを変えることにより、A'と等しくなるようにAを変更できるならば、Aはトランスファー・エラーを持つという。

【エキストラ・ステート(Extra States)】

AがA'と等しくするために、Aの状態の数を減らさなければならないならば、Aはエキストラ・エラーを持つという。AとA'は最小であるため、状態の数が等しくないということは、AとA'が等しくないということを暗に示している。

2.2 従来のテスト・ケースの抽出方法

テスト・ケースの抽出方法は、大きく2つに分類できる。有限状態オートマトンによりテストの対象をモデル化し、その状態遷移の系列をテスト・ケースとする方法と、それ以外の方法である。前者は、トランスファー・エラーを検出できる点で後者よりRELIABLEである。また、OSのテストのためには、並列性が記述できるモデルでなければならい。有限状態オートマトンに基づく方法では、この問題を通信非決定性有限状態オートマトンのWp法への適用により解決している。このWp法はW法とDS法をもとに作られた。そこでこの4つの方法について以下にまとめる。

【W 法 (W-method)】 [10]

有限状態オートマトンに基づきテストケースを生成する方法である。W 法は、集合 P と Z によりテストケースを生成する。P は仕様 (正しい有限状態オートマトン)S の状態遷移の系列であり、Z は仕様 S の特徴集合 (characterization set)W と入力集合 X からなり、$Z = (\{\epsilon\} \cup X \cup X^2 \cup \cdots \cup X^{m-n}).W = X[m-n].W$ である。ここで m は仕様 S の状態の最大数であり、n は最小形である実装 I の状態数である。P と Z により正しい出力 (仕様 S の出力) と入力の組合せを得ることで、トランスファー・エラーを検出可能なテストケースを生成する。

【DS 法 (Distinguish Sequence method)】 [11]

ハードウエアのテストのための方法である DS 法の特徴は、以下の 2 段階アプローチによりテストスイートを短くできることである。

《フェーズ 1》仕様に定義された各々の状態が、実装に存在するかどうかをチェックする。

《フェーズ 2》実装の正しい出力と遷移のために、仕様に定義されたフェーズ 1 の残りのトランジション全てをチェックする。

【Wp 法 (Partial W-Method)】 [9]

Wp 法の特徴は、W 法のテストスイートの長さを DS 法の 2 段階アプローチにより短くできることである。

【通信非決定性有限状態オートマトン (CNFSM : Communicating Nondeterministic Finite-State Machines) の Wp 法への適用】 [8]

並列性を持つソフトウエアのテストをするために、CNFSM へ Wp 法を適用する方法である。しかし、CNFSM のシステムは、一般に、有限個の状態だけを持ち、同値な言語を生成するオートマトンによってモデル化することはできない。そこで、可達解析を行なうことにより、NFSM(Nondeterministic Finite-State Machines) に変換し、Wp 法を適用する。本報告では、この方法を CWp 法と呼ぶことにする。

3 カラーペトリネット

TP ジェネレータは、カラーペトリネットに基づき可達木を作成することにより TP を生成する。テスト・スイートをできる限り短くするために、可達木は、同値マーキング (equivalent marking) による木の縮約を行なう。そこで、本節では、カラーペトリネットと同値マーキングによる可達木構築のアルゴリズムを定義する。

3.1 カラーペトリネット

【定義 1】 カラーペトリネットを以下のように定義する。

$CPN = (P, T, C, A, M_0, I)$

$P = \{p_1, p_2, \cdots, p_m\}$ はプレースの有限集合。

$T = \{t_1, t_2, \cdots, t_n\}$ はトランジションの有限集合。

$P \cap T = \phi$でかつ $P \cup T = \phi$

Cはカラー関数である。

A はアークの集合である。

M_0は初期マーキングを表す関数である。

I は抑止アークの集合である。

ここでは、P, T, C(p), C(t) は、全ての$p \in P$と$t \in T$に対し有限であるとする。

3.2 カラーペトリネットの可達木

カラーペトリネットの可達木 (CP ツリー) について、「5 人の哲学者」(図 1) を例に解説する。

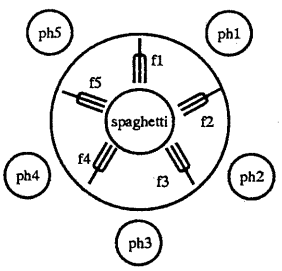

図 1: 5 人の哲学者

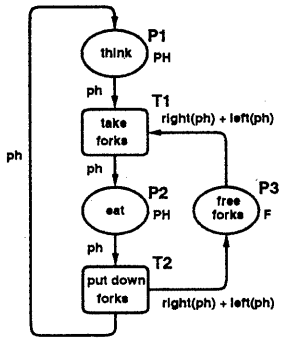

図 2: 5 人の哲学者のカラーペトリネットグラフ

PHILOSOPHER SYSTEM		T1 PH	T2 PH	M_0
THINK	PH	-ID	ID	ΣPH
EAT	PH	ID	-ID	
FREE FORKS	F F	-RIGHT -LEFT	RIGHT +LEFT	ΣF F

図 3: 5 人の哲学者のカラーペトリネット行列

上記のマーキングは以下のようになる。

$M_1 = (ph_2+ph_3+ph_4+ph_5, ph_1, f_3+f_4+f_5)$

$M_2 = (ph_1+ph_3+ph_4+ph_5, ph_2, f_1+f_4+f_5)$

$M_3 = (ph_2 + ph_4 + ph_5, ph_1 + ph_3, f_5)$

ここで、直観的に M_1 と M_2 は同値である。この考え方は、テトス・ケースを手動で生成する場合によく用いられる同値分割法の本質に類似している。テストの同値分割法は、同値なクラスのなかの 1 つのテストケースがエラーを発見すれば、同値なクラスの他のすべてのテストケースも、それと同じエラーを発見するであろうという考えに基づいた方法である [2]。よって、本システムでは、可達木の生成方法としてこの方法を使用する。そこで、以下に同値関係と同値マーキングによる CP ツリーを構築する方法 [13] を定義する。

【定義 2】 同値について定義する。

$M_1 = \varphi(M_2)$ であるような対称 $\varphi \in \Phi$[13] が存在するならば、カラーペトリネットの 2 つの ω マーキング [1] M_1 と M_2 は同値 (equivalent) であり、$M_1 \approx M_2$ と表される。

【定義 3】 CP ツリーの生成アルゴリズムを定義する。

1. もし、ノード y が、以前にあったノード z の一つを厳密に被覆するならば、$M_y(p)(c) > M_z(p)(c)$ を満たす全ての $p \in P$ かつ全ての $c \in C(p)$ に対し、$M_y(p)(c) := \omega$ である。

2. 各々の (可達な) 同値クラスの中の一つのノードのみが、木の先端に展開される。同値なノードの集合の中の一つノードのみが、木に含まれている。他のノードは削除されるが、木が持つ同値なノードへのアークは他の同値なノードの存在の情報を持つ。

3. 各々のノードは ω マーキングとノードラベルに関係付けられる。ノードラベルは、(空かもしれない) 状態情報のシーケンスであり、状態情報は、そのノード以前に処理されたノードのマーキングと同値か、または、前のノードを被覆するか、または、dead である。

4. ノード n_1 からノード n_2 への各々のアークは、アークラベルと関係付けられ、アークラベルは事象情報のリストである。$t \in T$ かつ $c \in C(t)$ である場合に、各々のアークラベルの要素は (t, c) の組である。リストの各々の事象の組は、n_1 のマーキングで利用可能になる。リストの最初のペアの事象が n_2 のマーキングに帰着する。ゆえに、他のペアの事象も、n_2 のマーキングに同値であるマーキングに帰着する。

以上より、図 2の可達木は図 4のようになる。

4 テストプログラムの自動生成

本システムでは、まず仕様を CPN によりモデル化する。この際、各トランジションには発火を生起するプログラムへの入力データを、プレースにはその状態を示すデータを関連付ける。これが、TP ジェネレータへの入力データとなる。次にこの CPN から CP ツリーを作成する。この際、CP ツリーは、前述した同値まーきんぐによる縮約を行なう。CP ツリーの有効枝はそれぞれトランジションに対応するため、根から葉に至るそれぞれの経路上の各枝 (トランジション) に関連付けられた入力データのシーケンスの集合がテストスイートとなる。また、CP ツリーの根は、発火による OS の状態をトークンのカラーと、マーキングの情報により表して

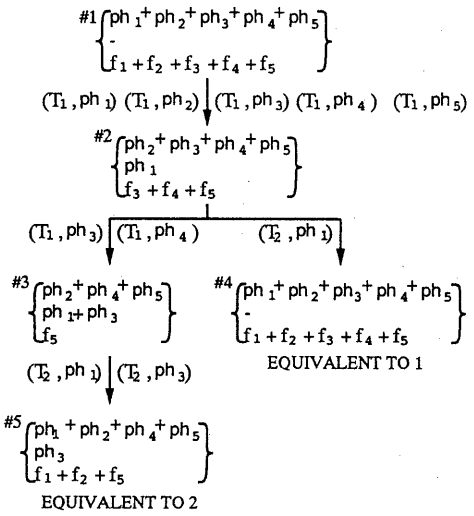

図 4: 5 人の哲学者の CP ツリー

いるため、根の系列から入力したときの正しい出力のシーケンスを得る。

4.1 定義

【定義 4】 CP ツリーを以下のように定義する。

$CPT = (N, B)$

Nは、CP ツリーの根の集合、Bは、CP ツリーの有効枝の集合であり、各々$N = \{n_1, n_2, \cdots, n_{m_N}\}$, $B = \{b_1, b_2, \cdots, b_{m_B}\}$ である。m_Nは根の最大個数であり、m_Bは有効枝の最大個数である。

【定義 5】 テストスイート TP を以下のように定義する。TPは、テストケースtp_iの集合であり、$TP = \{tp_1, tp_2, \cdots, tp_{m_{TP}}\}$ である。

また、テストケースすなわちテストプログラム tp_iを $tp_i = (E, R)$ と定義し、Eはテストの入力集合すなわちシステムコールの集合であり集合 Bの要素の連接で表す。Rはテストの Eに対する正しい出力の集合であり、集合 Nの要素の連接で表し、各々、

$E = \bigcup_{b \in B} b$, $R = \bigcup_{n \in N} n$

である。

4.2 テストプログラム生成アルゴリズム

テストプログラムの生成アルゴリズムを以下に示す。

(1) 入力されたデータを基に、TP ジェネレータ内に CPN を構築する。TP ジェネレータへの入力例を図 5に示す。

(2) 入力されたデータを基に、TP ジェネレータ内の CPN に初期マーキングを行なう。

(3) 構築した CPN から CP ツリーを作成する。

この際、入力時に定義したトークンのカラーをもとに、等しいものを同値とし、同値マーキングによる縮約を行なう。

(4) CP ツリーをもとに、TP を作成する。

CP ツリーの有効枝は、一つのシステムコールとなる。また、根は入力したシステムの状態、すなわち、例えば task1 が今 RUN 状態であり、資源を使っているなどという情報を持つ。したがって、CP ツリーの根が示している RUN 状態のタスクが、その根から発している有効枝が示すトランジションすなわちシステムコールを発行することになる。

```
/* GetRsc システムコールを発行するトランジション */
transition[1] = {
    /* トランジションのラベル */
    label = "t1:GetRsc";
    /* 発火時の処理 */
    action = {%GetRsc();%};
    /* 入力アーク集合 */
    input = {arc[1]};
    /* 出力アーク集合 */
    output = {place[2]};
};
/* task1 を示すトークン */
token[1] = {
    /* トークンのラベル */
    label = "token1:task1";
    /* トークンのカラー */
    color = TASK;
};
```

図 5: TP ジェネレータの入力例

4.3 適用例

例として、セマフォ・システムコールのテストプログラムを生成する。セマフォ・システムコールの仕様は以下のとおりである。

【P 命令】システムコール名：GetRsc();

セマフォが管理している資源のカウンター値が 0 より大きいかどうかを確認し、大きければその値を 1 減らし、そのタスクに資源の使用が許可される。値が 0 ならばタスクは WAIT 状態になる。

【V 命令】システムコール名：RelRsc();

資源のカウンター値を 1 増加する。その際、P 命令により資源を獲得できずに WAIT 状態にあるタスクがあれば、その内一つを READY 状態にする。もし、P 命令が一度も発行されていないならば (資源のカウンターが初期値である場合)、カウンターの値はそのままである。

また、タスク、資源は静的に生成され、タスクの数は 2 個、資源の数は 2 個とする。

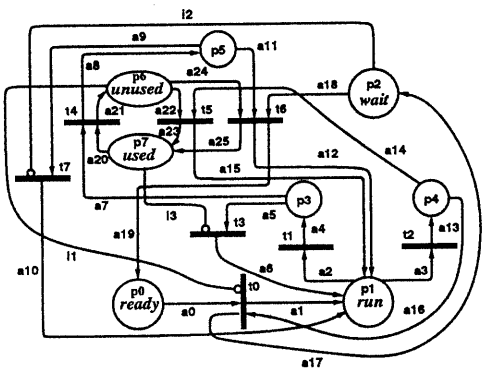

図 6: セマフォのカラーペトリネット

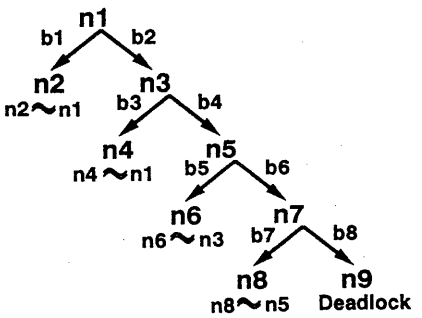

図 7: セマフォの CP ツリー

	n1	n2	n3
run	task0	task0	task0
ready	task1	task1	task1
wait	-	-	-
unused	rsc0+rsc1	rsc0+rsc1	rsc1
used	-	-	rsc0

	n4	n5	n6
run	task0	task0	task0
ready	task1	task1	task1
wait	-	-	-
unused	rsc0+rsc1	-	rsc0
used	-	rsc0+rsc1	rsc1

	n7	n8	n9
run	task1	task0	-
ready	-	task1	-
wait	task0	-	task0+task1
unused	-	-	rsc1
used	rsc0+rsc1	rsc0+rsc1	rsc0

表 1: セマフォの CP ツリーの根

以上をカラーペトリネットによりモデル化すると図 6 になる。図 6 の情報は、TP ジェネレータには、図 5 の形で入力する。ここで、task0 と task1 は同値であり、また rsc0 と rsc1 は同値であるので同値マーキングを行なう。また、「6 つの簡単な縮約則」[4][5][6] を適用すると、図 7 の CP ツリーを得る。この CP ツリーの有効枝と根を、ネットデータを入力した際の各々の action に置き換えることにより、テスト・プログラムを生成する (図 8)。

図 7 に示したセマフォの CP ツリーの各々の根を表 1 にまとめる。また、b1, b3, b5, b7 は、各々、RelRsc を示すトランジションの系列を縮約したものであり、b2, b4, b6, b8 は、各々、GetRsc を示すトランジションの系列を縮約した系列である。

以上により 5 つの tp を得る。各々のプログラムのシーケンスは、

$E_1 = b_1$,
$E_2 = b_2 \cdot b_3$,
$E_3 = b_2 \cdot b_4 \cdot b_5$,
$E_4 = b_2 \cdot b_4 \cdot b_6 \cdot b_7$,
$E_5 = b_2 \cdot b_4 \cdot b_6 \cdot b_8$

である。

```
task0()              task1()
{                    {
    GetRsc();            RelRsc();
    GetRsc();        }
    GetRsc();
}
```

図 8: TP ジェネレータの出力例

5 評価

本節では、TP ジェネレータと従来の生成方法との比較を行なう。まず、2.2に示した CWp 法によるテストスイートの生成方法を以下に示す。

CWp 法は以下の手順でテストスイートを生成する。[8]

1. システムを CNFSM によりモデル化する (図 9参照)。
2. CNFSM の可達解析を行い NFSM へ変換する (図 10参照)。
3. NFSM へ Wp 法を適用する。

ここで、CNFSM の構造を示す (図 9参照)。

1. 各々の CNFSM は、NFSM と入力 FIFO キューを持つ。
2. 一組のマシン間は、2 つの FIFO チャネルを持つことことができ、各々のチャネルは、通信のために設計されている。
3. ある一組のマシンが、各々、片方のマシンとその間にあるチャネルを通して通信できるならば、あるマシンからの信号は FIFO キューを通して渡され、そして片方のマシンの入力キューに入る。

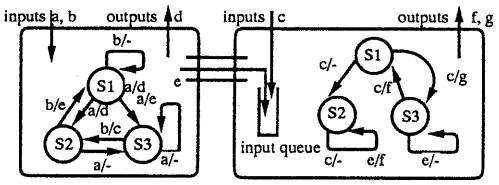

図 9: CNFSM のシステム

CNFSM のシステムは、一般に、有限個の状態だけを持ち、同値な言語を生成するオートマトンによってモデル化することはできない。そこで、可達解析を行なうことにより、NFSM を生成し、Wp 法を適用する。図 10は、図 9から NFSM を生成した例である。

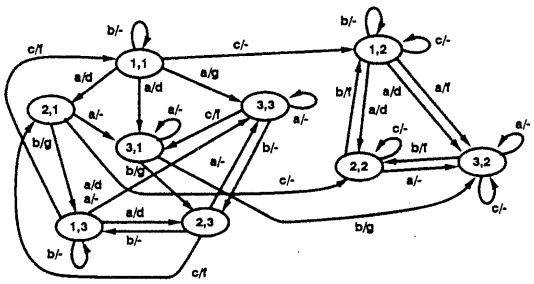

図 10: 可達解析による NFSM への変換

次に Wp 法のテストスイートの生成方法を示す。[9] Wp 法は、以下の 2 段階アプローチを用いることでテストシーケンスを得る。

フェーズ 1 仕様で定義された状態全てが実装の状態と同じであることと、初期状態から各々の状態までの遷移をチェックすること。

フェーズ 2 フェーズ 1 ではチェックしないが、仕様には定義されている遷移全てに対する実装をチェックすること。

もし、実装 I が両方のフェーズに合格するならば、その実装が仕様と同値であることが示される。

以上より、本システムでのカラーペトリネットに基づく方法と従来の方法を比較すると以下の長所がある。

1. CWp 法よりも処理方法が簡単である。

   CWp 法は、一度 CNFSM から NFSM へ変換する必要があるが、本システムによる方法では、仕様をモデル化したカラーペトリネットから、直接テストシーケンスを得る木を作成することができる。

2. 同値な状態を並列に持つ場合、CWp 法よりも短いテストスイートを得ることができる。

   例えば、前節に示したセマフォを CNFSM でモデル化すると、タスク毎に状態を持たなければならないので、ready、run、wait

を2個づつと、セマフォの状態2つの合計8個の状態は、最低持たなければならない。そこで、これをNFSMへ変換すると8*8の状態になる。これを縮小しても8個以下にはならず、さらに、このNFSMの状態遷移図は、閉路であるため、たとえ8個の状態であっても、遷移の数は8個以上になる。さらに、入力はP命令とV命令とセマフォが0かどうかの4種類であるため、本システムにおいてCPツリーに基づいて得たテストプログの数5よりもはるかに大きな数であることがわかる。

6 おわりに

本報告では、OSのテストのためのTPジェネレータの試作について論じた。TPジェネレータはCWp法と比較して、カラーペトリネットによってモデル化されるため、タスク間通信など並列性を持ったソフトウエアに対する処理が単純である。また、カラーペトリネットに基づいているため、トークンへの畳み込みにより状態数を減らすことができる。さらに、同値マーキングによる木の縮約を行なうため、従来のWp法よりも短いテストスイートを得ることができる。

参考文献

[1] James L. Peterson, "Petri Net Theory and the Modeling of Systems", Englewood Cliffs, New Jersey, Prentice Hall Inc., 1981, 市川享信, 小林重信 訳, "ペトリネット入門–情報システムのモデル化–", 共立出版社, 1984

[2] Glenford J.Myers, "The Art of Software Testing", John Wiley & Sons, Inc., 1979, 長尾 真, 松尾 正信 訳, "ソフトウエア・テストの技法", 近代科学社, 1980, 3

[3] 玉井 哲雄, 三嶋 良武, 松田 茂広. "ソフトウエアのテスト技法", 共立出版社, 1993, 10

[4] 村田 忠夫, "ペトリネットの解析と応用", 近代科学社,1992

[5] 村田 忠夫, 辻 孝吉"ペトリネットによる並行処理プログラムの解析手法", 情報処理, Vol.34, No.6, p.701-709, 1993, 6

[6] M.Notomi, T.Murata, "Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis", IEEE Transactions on Software Engineering VOL.20, No. 5, pp.325-336, May 1994

[7] J.B.Goodenough, S.L.Gerhart, "Toward a theory of test data selection", in Int. Conf. Reliable Software. Los Angeles, CA, 1975

[8] G.Luo, G,v,Bochman, A.Petrenko, "Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method", IEEE Transactions on Software Engineering VOL.20, No.2, pp.149-161, February 1994

[9] S.Fujiwara, G.v.Bochmann, F.Khendek, M.Amalou, A.Ghedamsi, "Test Selection Based on Finite State Models", IEEE Transactions on Software Engineering VOL.17, No.6, pp.591-603, June 1991

[10] T.S.Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering VOL.4, No.3, pp.178-187, March 1978

[11] G.Gonenc, "A method for the design of fault-detection experiment", IEEE Transactions on COMPUTERS VOL.C-19, pp.551-558, June 1970

[12] A.Gill,"Introduction to the Theory of Finite-State Machines", NewYoork:McGraw-Hill,1962

[13] K.Jensen, "COLOURED PETRI NETS", Lecture Notes in Computer Science, No254, Springer-Verlag, 207-247, 1987