ソフトウェアテストシンポジウム 2004
JaSST'04: Japan Symposium on Software Testing 2004

ソースコード生成による利用モデルの作成と分析

高木 智彦† 古川 善吾‡

香川大学 〒761-0396 香川県高松市林町 2217-20

E-mail: † s02g463@stmail.eng.kagawa-u.ac.jp, ‡ zengo@eng.kagawa-u.ac.jp

あらまし 統計的テストは，大量のテストケースを用いて，ソフトウェアの実際の利用状況における信頼性を評価できるため，今後有効な手法の一つになると考えられる．ただし，ソフトウェアの利用状況を状態遷移図上の確率分布として定義した利用モデルが必要であり，作成には手間がかかることが多い．そこで本研究では，ソースコード生成法を用いた開発において，簡単に利用モデルを作成する手法を提案する．また，利用モデルの分析が設計やテストに貢献することについても言及する．

キーワード 統計的テスト，利用モデル，状態遷移図，実行履歴，ソースコード生成

Construction and Analysis of Usage Model with Code Generation

Tomohiko TAKAGI† Zengo FURUKAWA‡

Kagawa University 2217-20 Hayashi-cho, Takamatsu City, Kagawa 761-0396, JAPAN

E-mail: † s02g463@stmail.eng.kagawa-u.ac.jp, ‡ zengo@eng.kagawa-u.ac.jp

Abstract Statistical testing is an effective technique using a lot of testcases and evaluating software's reliability on actual usage. This testing needs usage model which shows software usage as Markov chain. But it often takes time and effort to develop it. So this paper proposes a simple method to construct usage model using source code generation based on state transition diagram. Model analysis contributes to software design and testing.

Keyword Statistical Testing, Usage Model, State Transition Diagram, Execution History, Code Generation

1. はじめに

近年，ソフトウェアは日常生活のあらゆる場面に浸透したため，バグが社会に及ぼす影響は大きいものとなっている．しかしながら，従来のように，ソフトウェアのすべての機能を網羅的にテストするだけでは，要求される信頼性を十分に満足できない場合がある．

統計的テスト[1][2]は，大量のテストケースを用いて，ソフトウェアの実際の利用状況における信頼性を評価できるため，今後有効な手法の一つになると考えられる．この方法では，(1) ソフトウェアの仕様である状態遷移図にユーザの利用状況を関連付けてマルコフ連鎖（すなわち利用モデル）を作成し，(2) 利用モデルの遷移確率に基づいてテストケースを生成する．(1) では，ソフトウェアのプロトタイプや過去のバージョンの実行履歴などを用いる．実行履歴が入手できない場合は，開発者が仕様に基づいてユーザの利用の仕方を推定することもある．

これまでの我々の研究において，利用モデルの作成に関するいくつかの提案を行ってきた．文献[3]では，ユーザの利用の仕方を推定しながら，状態遷移図上に直接実行系列を入力することによって利用モデルを作成した．この方法は，入力作業が単純である反面，ユーザの利用特性を真に反映した実行系列を求めるのが難しい．そこで文献[4]においては，実際のプログラムの実行履歴を状態遷移図に対応付けるための手法を考察した．すなわち，状態遷移図の状態や遷移に対応するコード行をあらかじめ指定しておくことによって，コード行の実行回数を状態遷移図にマッピングするという方法である．しかし，状態や遷移に対応するコード行を手作業で指定する必要があるため，手間がかかる．また，状態遷移図とソースコードとの間に単純な対応関係が存在するとは限らない．そこで本稿では，状態遷移図からソースコードを生成することによって，上述の問題を回避し，容易に利用モデルを作成する手法を提案する．

まず 2 節で状態遷移図，ソースコード生成法，および統計的テストについて概観する．次に 3 節で今回提案する利用モデルの作成手順を，4 節でサンプルプログラムへの適用例を説明する．そして 5 節では利用モデルの分析について言及する．最後に，6 節で試作中のテスト支援システムを紹介し，7 節で本手法に対する考察を行う．

2. 背景

利用モデルの作成方法を説明するのに先立って，本手法における重要な知識について説明する．

2.1. 状態遷移図

2.1.1. 概要

状態遷移図は，イベントに対するシステムの動的な振る舞いを記述したものである．一般に，機能仕様の記述方法としてだけでなく，機能テストの指針としても利用されることが多い.表記方法や解釈については，使用目的や開発組織によって多少の差異が存在するが，本稿では，統一モデリング言語（UML）[5]の状態図に従うものとする．

2.1.2. UML 状態図

UML は，オブジェクト指向モデリング言語であり，デファクトスタンダードであるといえる．その中で状態図は「オブジェクトやサブシステムなどの実体がイベントに応じて状態遷移する様子をグラフとして記述するもの」として定義されている．以下に基本的な表記法や解釈を示す．

なお，本稿では紙面の都合上，最低限の説明に留める．詳細については，文献[5]を参照されたい．

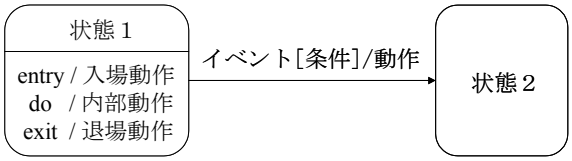

図 1. 状態図の表記法

状態は，動作の実行やイベント待ち，特定条件の満足などを意味する.状態の動作としては,入場動作（当該状態に入った直後に実行する動作），内部動作（当該状態に滞在中に実行する動作），退場動作（当該状態から出るときに実行する動作）がある．また，状態図を入れ子にすることができる．

遷移は，ユーザ入力や割込みなどのイベントの発生によって状態が変化する(発火する)ことを意味する．イベント名や付随動作，発火条件を定義できる．

2.2. ソースコード生成法

2.2.1. 概要

ソースコード生成とは，仕様書からプログラムの骨組み（スケルトンコード）を，ツールを用いて機械的に生成することである．短期開発・品質改善を目的として従来から広く利用されている．状態遷移図からスケルトンコードを生成する手法としては，状態遷移表に基づくもの[6]やオブジェクト指向方法論（OMT）に基づくもの[7]などがある．前者はよく知られた代表的な方法である．後者はシンプルなコードを生成できる点が特長である．

2.2.2. 状態遷移表に基づく方法

入力コード表(入力値をコード化した表),遷移表(現在の状態と入力コードの組み合わせから，実行すべき出力ルーチンと次の状態を決定する表）を用いる手法である．基本動作は次のようになる．

1. プログラムは現在の状態を変数の値として保持している．
2. イベントが発生したら，入力コード表から入力値に対応する入力コードを求める．
3. 遷移表から，現在の状態と入力コードに対応する出力ルーチンを呼び出し実行する．そして新しい状態を求め，現在の状態と置き換える．

2.2.3. OMT に基づく方法

状態をクラスとして定義し（状態クラス），イベントや動作をカプセル化する．状態クラスは，すべてのイベントをインタフェースとして定義した抽象化クラスを継承する．また，サブ状態は親状態の振る舞いを継承するものと見なす．基本動作は次のようになる．

1. プログラムは現在の状態を状態オブジェクト（状態クラスをインスタンス化したもの）として保持している．
2. イベントが発生したら，状態オブジェクトの対応するメソッドを実行する．
3. メソッドは，まず現在の状態の退場動作，遷移に付随する動作を呼び出す．次に，遷移先の状態クラスをインスタンス化し，現在の状態オブジェクトと置き換える．その後，入場動作や内部動作を呼び出す．

2.2.4. 共通の性質

状態遷移表や OMT に基づくコード生成法において，特筆すべき共通の性質が 2 つある．1 点目は，プログラム自身が状態遷移モデルを内部に持っている点である．つまり，プログラムは現在の状態とイベントから振る舞いを決定する．現在の状態は変数の値として表されるため,この変数の履歴は状態遷移の履歴となる．2 点目は,状態や遷移が特定のプログラム要素と 1 対 1 で対応している点である．状態遷移表に基づく方法では，状態と入力コードの組み合わせから決定される出力ルーチン呼出しが遷移に対応する．また，OMT に基づく方法では，状態がクラスに，遷移がメソッドに対応する．これは，プログラム要素の実行回数が，対応する状態や遷移の実行回数と一致することを意味する．

以上の 2 点は，本研究の核心である．これらの性質は，コード生成を行う上では自然なことであり，他の手法にも当てはめることができると考えられる．

2.3. 統計的テスト

2.3.1. 概要

統計的テストは，クリーンルーム開発手法における機能テスト法として開発された．クリーンルーム開発手法によって作成されたコードはプログラミング終了時点での品質が高いため，バグを発見するためではなく，品質を測定するためにテストを行う．従来の開発手法によって作成されたコードを統計的テストの対象とする場合は，ある程度の品質を既に達成していることが前提となる．従って，システムテストや受け入れテストとして採用するとよい．

2.3.2. テスト手順

これまでに提案されている統計的テストの一般的な手順を以下に示す．

1. 利用モデルの作成
   ソフトウェアの仕様として作成される状態遷移図,およびユーザのソフトウェアに対する使用特性を用いてマルコフ連鎖を作成する.マルコフ連鎖とは,状態遷移図において遷移先が確率的に決定されるものをいう．これが利用モデルである.利用モデルでは,ソフトウェアに対する,ユーザ,利用の仕方，環境などを定義できる.利用の仕方は，ユーザの動作によって定義される．環境は,ソフトウェアが動作するプラットフォームや外部データベースなどの考慮すべき要因である.利用の状況についても，夜間と昼間，緊急時と平常時などに分類する.
2. テストケースの作成
   利用モデルの遷移確率に従い,初期状態から終了状態に至る状態と遷移の列としてパスを作成する.作成したパスは，ソフトウェアに対する入力条件であり，かつ期待出力でもあるため，テストケースと見なすことができる.テストケースの分布は利用モデルの分布と一致する.また，ソフトウェアの全機能を網羅しているとは限らない.
3. テストの実施およびテストモデルの作成
   テストケースを実行し,得られた出力を状態遷移図に記入する.誤りが発生した時には，新たに誤り状態を追加する.このようにして作成されるマルコフ連鎖がテストモデルであり,テスト履歴として利用する.
4. 信頼性評価
   誤り状態に達しない確率を求めることによって，信頼性を評価する.また，誤り状態に達するまでの実行回数から，MTTF（Mean Time To Failure）を求めることができる.
5. テスト終了判定
   以下のような指標に基づいてテストの十分性を判定する.
   - 利用モデルとテストモデルの差が与えられた閾値より小さい.
   - 信頼性が与えられた目標値より大きい.
   - MTTF が与えられた目標値より大きい.

2.3.3. 従来の手法との比較

統計的テストはユーザの立場から行い，ソフトウェアそのものの信頼性を求める．テスト十分性は MTTF などの測定結果による．統計的テストでは，原理的に出荷した後の使用環境において発生確率の高い欠陥が検出される.

一方，従来のテストは開発者の立場から行う．すなわち，プログラムや仕様に基づいて網羅的にテストを行い，作業の品質（カバレッジ）によって対象となるソフトウェアの信頼性を予測している.

3. 利用モデルの作成

2 節では背景として，状態遷移図やコード生成法，統計的テストについて説明した．本節では，本研究で考察した利用モデルの作成手順についてまとめる.

1. 状態遷移図を作成する
   プログラムの振る舞いを,外部仕様の観点から状態遷移図にまとめる.
2. プログラムを作成する
   状態遷移表に基づく手法や OMT に基づく手法を用いて,状態遷移図からスケルトンコードを生成し，実装を行う．コード生成によって，プログラムに状態遷移モデルを組み込み，さらに，状態や遷移を特定のプログラム要素に 1 対 1 で対応付けることができる.
3. プログラムを実行し，実行履歴を得る
   プログラム(一通りの実装が完了しているリリース版やプロトタイプなど)をユーザに試用してもらい，利用モデル作成のための実行履歴を得る.実行履歴の記録に関しては,以下に示すような方法が考えられる.
   - 状態遷移図からスケルトンコードを生成する際に，状態やイベントの履歴を記録するコードを組み込む方法.
   - 一般的なコードカバレッジツールを用いる方法．プログラムに自動挿入されるモニタ関数によって，種々のプログラム要素の実行回数を記録できる.

4. 実行履歴から遷移確率を求める

3.で得られた実行履歴を状態遷移図に記入し，状態や遷移の実行回数を明らかにする．そして遷移確率を計算し，利用モデルを完成させる．プロトタイピングの対象外の機能やユーザが全く利用しなかった機能などは，実行回数を関連付けることができない．この場合，一定の遷移確率を別途割り振る必要がある．

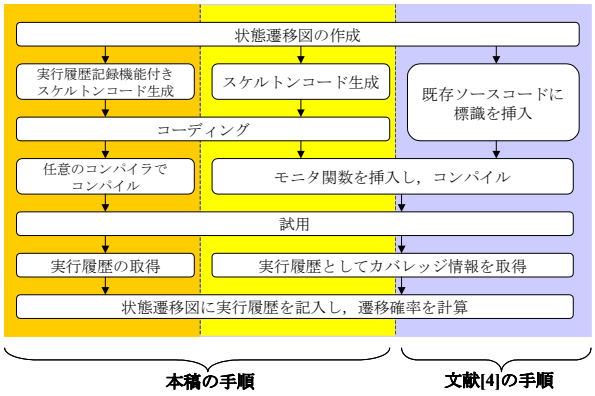

図 2. 手順概略

4. 適用例

本節では，3 節で示した手法の適用例として，単純なストップウォッチプログラムを取り上げる．

4.1. ストップウォッチの仕様

図 3 に状態遷移図を，図 4 に GUI 設計を示す．

ストップウォッチは，計測中（Measuring），停止中（Idle），一時停止中（Pause）の 3 つの状態をもっている．起動後，まず初期化を行った上で Idle 状態に遷移する．Idle 状態において Start･Stop ボタンを押下すると，Measuring 状態に遷移する．Measuring 状態では，計測値の更新を継続的に行う．そして再度 Start･Stop ボタンを押下すると，Pause 状態に遷移する．Pause 状態において，Reset ボタン押下時は計測値を初期化して Idle 状態に遷移し，また Start･Stop ボタン押下時は Measuring 状態に遷移する．

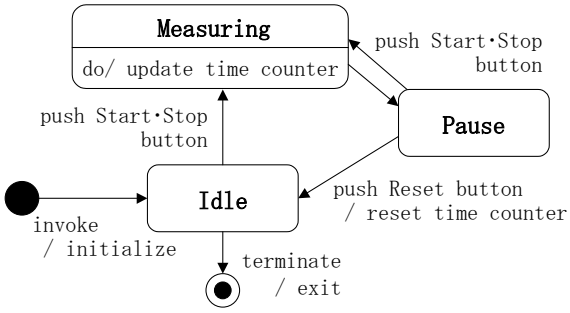

図 3. ストップウォッチの状態遷移図

図 4. ストップウォッチの GUI 設計

4.2. ソースコードの生成

コード生成法によって，図 3 からスケルトンコードを得ることができる．例として OMT に基づく手法から生成されるコード（C++）の一部を以下に掲載する．

```
//---------------------------------------
// 状態クラスの継承元（抽象化クラス）
class BaseState {
  public:
    virtual void pushStartStopButton() { };
    virtual void pushResetButton() { };
    virtual void terminate() { };
};
//---------------------------------------
// 状態クラス
class MeasuringState : public BaseState {
  public:
    void pushStartStopButton();
};
class PauseState : public BaseState {
  public:
    void pushStartStopButton();
    void pushResetButton();
};
class IdleState : public BaseState {
  public:
    void pushStartStopButton();
    void terminate();
};
//---------------------------------------
// 遷移メソッド
void MeasuringState::pushStartStopButton() {
  delete state;
  state = new PauseState;
}
void PauseState::pushStartStopButton() {
  delete state;
  state = new MeasuringState;
}
void PauseState::pushResetButton() {
  delete state;
  state = new IdleState;
}
void IdleState::pushStartStopButton() {
  delete state;
  state = new MeasuringState;
}
void IdleState::terminate() {
  delete state;
  exit(0);
}
//---------------------------------------
```

ストップウォッチのクラス図を図 5 に示す．斜体で表したものは，実装を持たない（すなわち抽象化クラスや仮想関数である）ことを意味する．市販の統合開発環境を用いて作成した GUI モジュールに，生成したソースファイルをインクルードして実装を行った．

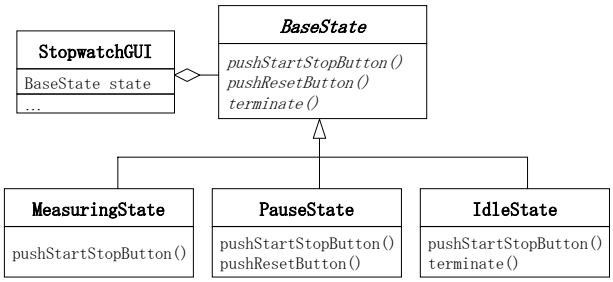

図 5. ストップウォッチのクラス図

4.3. 利用モデルの作成

一通りの実装とデバッグが終了した後，試用を行う．試用は，利用モデル作成のための実行履歴を取得するだけでなく，仕様がユーザのイメージと一致することの確認も兼ねることができる．

3 節の 3 において，実行履歴を取得する 2 つの方法を紹介した．その 1 つは，実行した状態列やイベント列を出力するルーチンを組み込む方法であった．OMT に基づくスケルトンコードにおいては，例えば，状態クラスのコンストラクタあるいはデストラクタに現在の状態名を出力する機能を実装することが考えられる．この場合，図 6 に示すような実行履歴（状態列）が得られ，図 7 の利用モデルが導かれる．

ただし，図 7 の利用モデルはサンプル数（実行履歴の量）が少ないため，統計的に有意とはいえない．実行履歴の収集に先立って，例えば「すべての遷移を最低 x 回実行する」というような基準を定めておくとよい．

```
Idle, Measuring, Pause, Idle, Measuring, Pause,
Measuring, Pause, Idle, Measuring, Pause, Idle
```

図 6. 実行履歴の例（CSV 形式）

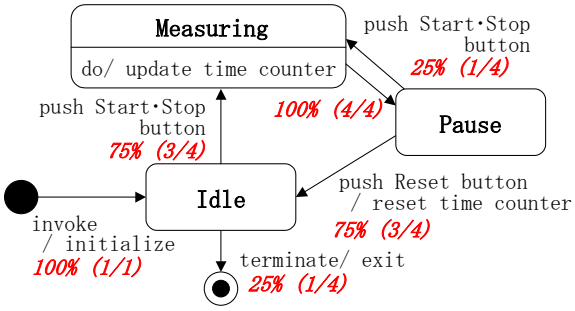

図 7. ストップウォッチの利用モデル例

5. 利用モデルの分析

本節では利用モデルの分析方法を紹介する．利用モデルは単にテストケースの生成だけでなく，テストの見通しを得たり，設計の評価にも用いる．

5.1. テストケースの生成

利用モデルから機能テストのためのテストケースを機械的に生成することができる．テストケースは，利用モデルの遷移確率に基づいて作成される．従って，十分な量のテストケースを用いれば，大数の法則によって，テストの分布が利用モデルの分布と一致することは明らかである．

テストケースの表記法に関しては XML（eXtensible Markup Language）形式のものが提案されている[8]．以下は，4 節で示したストップウォッチに対するテストケースの表記例である．

```
<?xml version="1.0" encoding="Shift_JIS"?>
<title>Testcase for Simple Stopwatch</title>
<pathGroup>
  <path>
    <state>
      <stateName>Initial State</stateName>
    </state>
    <transition>
      <eventName>invoke</eventName>
      <activity>initialize</activity>
    </transition>
    <state>
      <stateName>Idle</stateName>
    </state>
    <transition>
      <eventName>terminate</eventName>
      <activity>exit</activity>
    </transition>
    <state>
      <stateName>Final State</stateName>
    </state>
  </path>
</pathGroup>
```

5.2. 確率的解析

マルコフ連鎖は，確率論における確率過程の一分野として確立されており，種々の方法で解析することができる[1][2][9]．以下に示す応用例は，ソフトウェア開発において有用であると考えられる．

- 1 実行当たりの平均入力回数 h を求める．もし入力回数が少ないほどソフトウェアが使いやすいとすれば，h がより小さくなるように設計を変更することによって操作性を改善できる．
- すべての状態あるいは遷移を網羅するために必要なテストケース数 x を確率的に推定する．テス

ト環境における 1 入力当たりの時間 i が明らかであれば，テストの実行に要する時間 T ($=x \times h \times i$) を予測できる．

- 遷移確率に基づいて，各状態の相対的な使用頻度を明らかにする．使用頻度の低いオプショナルな機能を開発から除外することによって，開発期間の短縮と信頼性の向上が期待できる．

6. テスト支援システム

統計的テストは，利用モデルを作成したり，信頼性評価を行う上で十分な量のテストケースを作成したりする必要があるため，従来のテスト方法より時間がかかる．そこで，作業を自動的に行うためのテスト支援システム（図 8）を試作中である．現在の主な機能を以下に示す．

- UML に基づく状態遷移図の編集
- ソースコードの生成と編集
- 実行履歴からの利用モデル作成
- 利用モデルからのテストケース生成

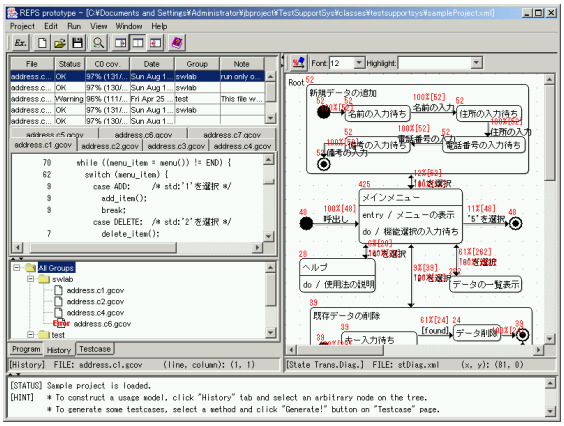

図 8. 試作システムの GUI

7. 考察

本手法には以下に挙げる 4 つの特徴がある．

- ユーザを長時間拘束しない
  ユーザの手を借りるのは，利用モデルのための実行履歴を作成する時だけである．後は開発者によって，ユーザの使用特性を反映した実行系列をいつでも好きなだけ生成できる．しかも利用モデルは一旦作成すれば，仕様が変更にならない限り再使用できる．
- テスト範囲を限定できる
  コード生成法によって，仕様を正確に反映した状態遷移のロジックを得ることができる．例えば，図 3 の Measuring 状態において，push Reset button イベントを受理しないことをテストする必要はない．
- 適用可能な対象が広い
  開発するプログラムを状態遷移図として記述可能で，かつ，ロードモジュールから実行履歴を取得できる場合は，本手法を適用できる可能性がある．これは，プログラムの形態（GUI，CUI，組み込みなど）に依らない．
- コードの変更に対応しやすい
  状態や遷移がプログラム要素と 1 対 1 で対応しているため，仕様変更やバグによるコードの修正，また，それに伴う回帰テストの設計が容易になることが期待できる．

8. おわりに

ソースコード生成法の性質に着目することによって，利用モデルは容易に作成できる．本手法は，ユーザを長時間拘束する必要がなく，テスト範囲を限定できるなどの特徴がある．また，利用モデルを分析することによって，開発に有用な情報が得られる．

今後の研究においては，テストの実施や信頼性評価なども含めて考察する．また，現在試作中のテスト支援システムを用いて，本手法の客観的な有効性を評価する予定である．

文　献

[1] J. A. Whittaker and M. G. Thomason : "A Markov Chain Model for Statistical Software Testing", IEEE Transactions on Software Engineering, Vol.20, No.10, pp.812-824, October 1994.

[2] G. H. Walton, J. H. Poore, and C. J. Trammell: "Statistical Testing of Software Based on a Usage Model", Software Practice and Experience, Vol.25, No.1, pp.97-108, January 1995.

[3] 高木智彦，古川善吾："UML 状態図を用いたテストケース作成支援システムの試作"，情報処理学会研究報告，Vol.2002，No.23，pp.79-86.

[4] 高木智彦，古川善吾："プログラムの実行履歴を用いた利用モデルの作成方法について"，ソフトウェアテストシンポジウム 2003 予稿集，pp.41-48.

[5] Object Management Group : "OMG Unified Modeling Language Specification Version 1.5", March 2003 http://www.uml.org/.

[6] Boris Beizer : "ソフトウェアテスト技法"，日経 BP 出版センター，1994.

[7] アリ ジョハル，田中二郎："オブジェクト指向方法論（OMT）に基づく動的モデルからの Java コード生成"，情報処理学会論文誌，Vol.39，No.11，pp.3084-3096，1998.

[8] 山本修二："組み込みソフトウェアの上流設計における試験表現に関する研究 －状態遷移モデルを基にしたモデルベース試験－"，ソフトウェアテストシンポジウム 2003 予稿集，pp.55-60.

[9] 渡部隆一："数学ワンポイント双書 31 マルコフ・チェーン"，共立出版，1979.