CB-001

# 自然言語仕様からの機能間の並列・順序動作の抽出と左記テスト環境
# Extraction of Execution Order of Functions from Specifications in Natural Language and Environment for Parallel and Sequential Order Tests

青山裕介† 黒岩丈瑠† 久代紀之†
Yusuke Aoyama Takeru Kuroiwa Noriyuki Kushiro

## 1. はじめに

システムの機能評価では、仕様書に記載された通りに機能が動作するかを確認する機能テストとともに、機能の実行順序を入れ替えた場合や、複数機能が並行して動作した場合にも、その機能が正常に動作することを確認する並列・順序試験の実施が重要である。

一方で、機能の実行順序や並列動作などの振舞いに関しては、システム仕様書に明示的に記載されていることは少なく、試験技術者の知識や経験に基づいて試験ケースが設定されることも多い。この結果、試験ケースの漏れや、逆に過大な試験ケースが設定されてしまうことも往々に発生する。また、機能の実行順序を変更しての実行や、複数機能を並列に動作させることは、その実施に膨大な工数を必要とするほか、試験ケースの実現が実際には困難であることもある。

本研究では、自然言語で記載された仕様書をセミ形式記述に変換し、機能の依存関係と並列関係を可視化することで、順序・並行テストの対象となる試験ケースの抽出を支援するツールを開発する。さらに、可視化された機能間の関係に基づき、モデル検査器用の試験ケース記述言語に変換するツールを開発する。

これらのツール開発にあわせ、実機や実ソフトウェアを対象に直接実行テストを行う並列・順序試験実行装置を構築した。左記装置では、テストケース実行環境のコアに、モデル検査器を搭載し、試験ステップの実行の決定性/非決定性を制御することで、自在な並列・順序試験の実行を可能とする。これら一連のツール、装置の開発により、並列・順序試験のテストケース設計の品質向上とテスト実行効率を向上させることを試行した。

## 2. テスト設計手順の現状とその課題

ソフトウェア開発現場におけるテスト（統合テスト、システムテストを対象とする）設計手順の現状およびそれを実行する上での課題を把握するために、主にテレビ、エアコンなど組込み製品の開発・設計・評価を業務とする企業 2 社のシステム設計・評価技術者にインタビューを行った。

### 2.1 テスト設計手順

上述のインタビュー結果に基づきモデル化したテスト設計手順を図.1 に示す。マクロ的視点からは、テスト設計は、システム仕様書（製品ドメインにより、動作仕様書や制御仕様書とも呼ばれる）の記述をテスト設計書へ変換する作業と見なすことができる。すなわち、システム仕様書に記載された文、図表を、“条件→（期待）動作”の群に変換する作業と見なすことができる[1]。

一方で、システム仕様書に記述された文、図表だけでは、テストケースを実際に設計することは困難である。テスト設計者は、システム設計者から得た当該システムに関するドメイン知識と、自らの専門知識（品質特性を満足するために必要となる試験ケース等）を活用し、テストケースの設計を行う。

設計されたテストケースは、システム設計者を交えたレビューにおいて、適用したドメイン知識、品質特性の優先度付けの妥当性を確認した後に、テスターにより試験が実行される。

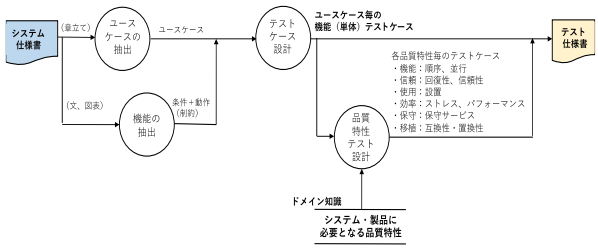


図. 1 現状のテスト設計のプロセス

### 2.2 テスト設計上の課題

インタビューに基づく分析の結果、下記 4 つの課題を抽出した。

(1) テストケースの設計品質がテスト設計者のスキルに大きく依存する。
(2) システム仕様からテストケースへの変換プロセス・ルールが、テスト設計者の頭の中だけにあり、明示化されていない。このため、変換プロセス・ルールを知識として、技術者間で共有ができず、テスト設計者の技術力の向上、育成が難しい。また、上記変換過程の中間成果物も見えないため、最終成果物であるテスト仕様書だけを対象にレビューしなければならず、レビューの品質が上がりにくい。
(3) テスト設計プロセスに手作業が多い。テストケース設計の前提となるユースケースの設定、条件・動作群の抽出などの作業が、システム仕様書の文書からのコピー/ペーストなどの手作業に依存している。手作業による人為的なミス（テストケース抜け、テストケースの誤り等）が多発する。
(4) テスト仕様に基づくテストの実施が、実際には困難（特に並列・順序試験）である、あるいは、テスト仕様の記述の曖昧さによりテスターにより、正しく実行されないことがある。

## 3. 課題解決へのアプローチ

上記課題認識に基づいて、以下のテスト設計プロセスおよび解決ツールの導入を提案する。

† 九州工業大学情報工学部, Kyushu Institute of Technology

### 3.1 提案プロセス

第 2 章で記載したインタビュー結果に基づき、図.2 に示すテスト設計プロセスを提案する。システム仕様書からテスト仕様書までの変換プロセスを定義することで、プロセスそのものとプロセスの生成物を明確化する。さらに、プロセスの入力−出力の変換アルゴリズムを定義し、左記変換作業をツールにより支援することで、人為的なミスの排除を行う。

これら施策により、課題として抽出されたテストケース品質のテスト設計者のスキル依存性、人為的なミスの削減、テストケースのレビューの品質向上、テスト技術者の育成等の課題を解決する。さらに、テスト仕様に基づくテストの実施が困難である等の課題については、並列・順序試験を実行するための試験環境を構築することで解決する。

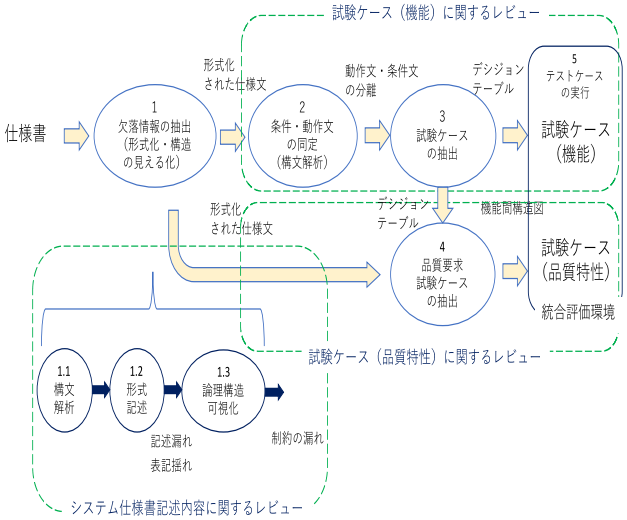


図. 2 提案するテスト設計のプロセス

### 3.2 提案プロセスの詳細とサポートツール

以下に、図.2 に提案したプロセスと支援ツールにより、どのようにシステム仕様をテスト仕様に変換していくかの概要を説明する。なお、本論文では、紙面の都合上、プロセス 4 における並列・順序試験ケース抽出ツール、並列・順序試験実行装置に関し以下の章にて詳述する。他のプロセス、支援ツールに関しては、別報にて報告する予定である。

#### 3.2.1 プロセス 1：欠落情報の抽出

プロセス 1 は、3 つのサブプロセス（構文解析、形式記述、論理構造の可視化）から構成される。サブプロセス“構文解析”においては、構文解析ツール KNP[2]を利用し単文化を行い、継続するサブプロセス“形式記述”において、次のようなセミ形式記述（関係語（主体語、対象語））[3]の形式に機械的に変換する（図.3）。

一般的に、自然言語で記載されたシステム仕様には、多くの記述漏れや表記揺れがあり、上記 2 つのサブプロセス中で、これら欠落情報が発見される（図.3“？”記述部分が欠落情報）。これらの欠落情報を、システム仕様設計者を交えたレビューにより補完する。さらに、補完されたセミ形式記述間の論理構造をサブプロセス“論理構造可視化”プロセスにて、ツールの支援を受け可視化[4,5]し、セミ形式記述化された単文間の論理的な関係（AND、OR、IMPLY 等）を規定する（図．3）。

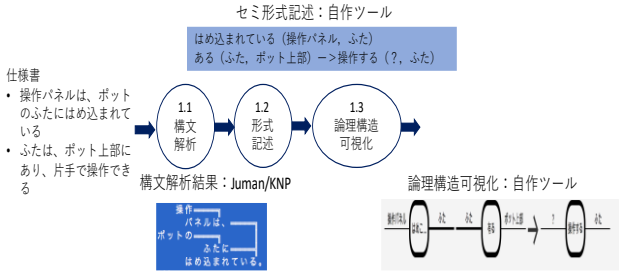


図. 3 システム仕様書の文の単文化と形式記述への変換

例えば、話題沸騰ポット第 7 版[8]の記述例である“操作パネルは、ポットのふたにはめ込まれている”、“ふたは、ポット上部にあり、片手で操作できる”という仕様記述文は、ツールを用い単文化（サブプロセス 1.1）、セミ形式記述（サブプロセス 1.2）に変換される。仕様書の記述漏れや表記揺れにより、欠落部分“？”（図.3）が発生し、ここに“ユーザ”という主体をレビューにより補う。さらに、補完した仕様をもとに、サブプロセス 1.3 にて、各単文間の論理構造を可視化する（図.4）。

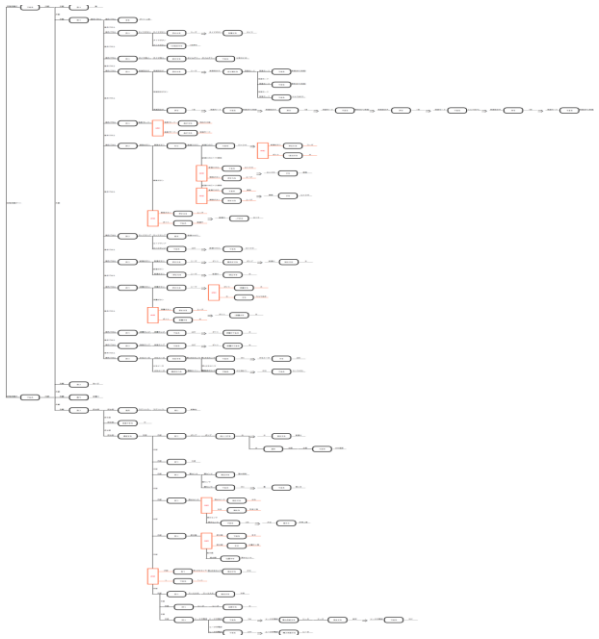

図. 4 話題沸騰ポット仕様書の論理構造の可視化[5]

#### 3.2.2 プロセス 2：条件・動作文の同定

テストケースの生成には、多数の技法があるが、ここでは、増田らにより提案された係り受けと表層格から、条件句と動作句を抽出する手法[1]をベースとして、条件文・動作文の同定を行う。ただし、増田らの提案では、構文解析ツールにより分割された句単位で、条件・動作を同定する。一方で、句単位にまで分割することで、図.5 の下部に示すように、文としての情報を失ってしまう。文としての情報

を失ってしまうことで、“操作パネルがふたにはめ込まれている”∧“ふたが上部にある”ことで、“ユーザが、ふたを操作できる”のような単文間の論理関係が失われてしまうこともある。

本研究では、上記のような分割のしすぎによる単文間の意味を失わないために、サブプロセス 1 にて抽出したセミ形式記述による単文単位で、条件文・動作文の同定を行う。また、上述の表層格の特徴からの条件文・動作文の同定にあわせ、サブプロセス 1 の出力である意味的な関係からの可視化結果をその判定に適用することで、条件・動作文の判定の正確性を向上させることを試行する。

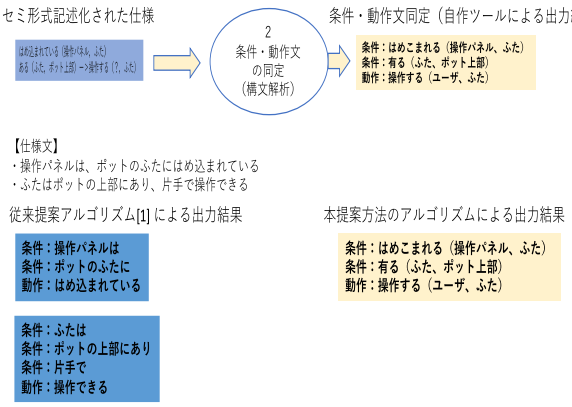


図. 5 表層格からの条件文・動作文の同定

### 3.2.3 プロセス 3：（単体機能）試験ケースの抽出

本プロセスでは、プロセス 2 で同定された条件文・動作文をデシジョンテーブルテスト技法[6]を適用して、単体機能の試験ケース候補を抽出する[1]。

プロセス 2 で、システム仕様書に記述の条件文・動作文が同定される。左記で同定された条件文をデシジョンテーブルの条件欄に、動作文をアクション欄に配置することで、デシジョンテーブルを完成させる。さらに条件文に付随する制約条件をルールの欄に追記することで、試験ケース（単体機能）の候補を抽出する（図.6）。

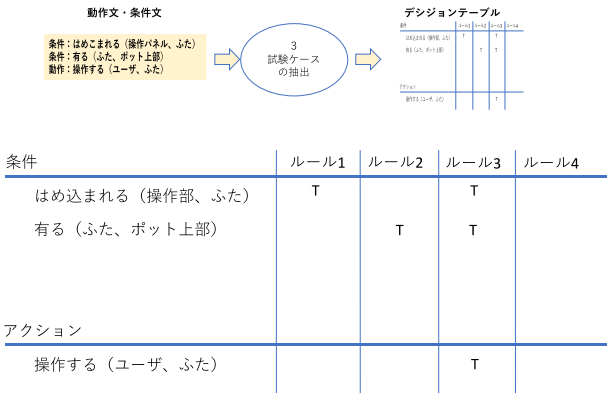


図. 6 デシジョンテーブルによる単体機能試験ケースの抽出

デシジョンテーブルテスト技法により、抽出される試験ケースは膨大な組合せ数になる。例えば、たかだか 2 値の値を取る 3 つの条件文の構成でも、$2^3$ 通りの試験ケースが抽出されてしまう。一方で、テストに投下できる時間・コスト資源は限定されるため、本プロセスで抽出されたすべての試験ケースを実際に実行することは現実的でない。同値・境界値法、ペア構成法[7]の適用によるケース数の縮退作業、およびシステム設計者を交えたケースの重点化を経て、テスト実行フェーズに移行する必要がある。

### 3.2.4 プロセス 4：品質要求試験ケースの抽出

本プロセスでは、ISO9126 に定義された品質要求を満足するために、プロセス 3 で抽出された機能試験ケースに試験ケースを追加する。

各品質要求に対し、具体的にどういう試験ケースを追加するかについては、製品ドメイン、あるいは企業の評価試験の指針に依存する。調査した組込みソフトウェア関連企業の事例では、機能性に対する追加試験として、並列試験、順序試験が、信頼性に関しては、回復性試験、信頼性試験、効率性に関しては、パフォーマンス試験、ストレス試験などが実施されている（図.1 参照）。

例えば、信頼性試験項目として、異常検出、異常中の処理、異常回復後のアクション定義し、これらを確認する試験ケースをプロセス 3 で抽出された単体機能試験ケースに追加する。前述したように、従来これらの試験ケースの追加は、テスト設計者の専門知識や経験に基づき設定されていたが、本提案プロセスでは、プロセス 3 で抽出されたテーブルの条件文毎に否定を取り、条件文が否定された際の動作を定義することで信頼性試験ケースを網羅的に抽出することができる（図.7）。話題沸騰ポットの例では、条件：はめこまれている（操作パネル，ふた）∧条件：ある（ふた，ポット上部）の条件が否定された時（異常時）を想定し、操作パネルやふたが破損した場合に、ユーザがポット動作を停止するアクションを信頼性の試験項目候補として抽出する。プロセス 3 と同様に、ここで候補として抽出される試験ケースも膨大なものとなるため、システム設計者を交えたケースの重点化を経て、テスト実行フェーズに移行することが必要となる。

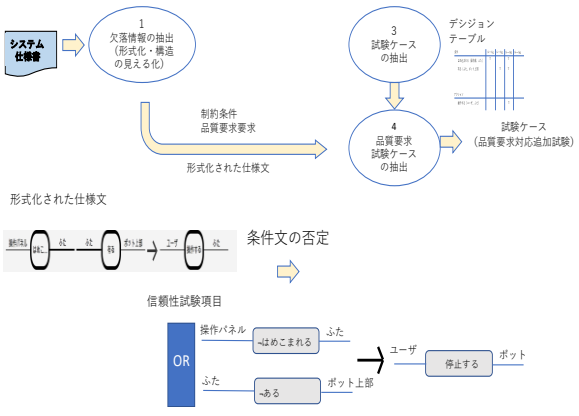


図. 7 品質要求試験対応の試験ケース抽出

## 4. 並列・順序試験抽出ツール

機能の実行順序や並列動作時に正しく動作するかどうかの試験は、機能性の品質要求において重要な試験である。一方で、機能実行順序や並列動作の可能性に関し、システム仕様書に明示的に記述されていることは少ないため、試験ケースとして漏れてしまうことが往々にして発生する。

本研究では、プロセス 1，2 のアウトプットを利用し、並列・順序試験ケースの抽出を行うツール[9,10]を提案する。プロセス 1、2 のアウトプットから、各機能がどんな状態変数を参照、変更するかを抽出し、機能間の参照・変更関係から相互の依存関係と並列動作する機能を可視化する（図.8）。

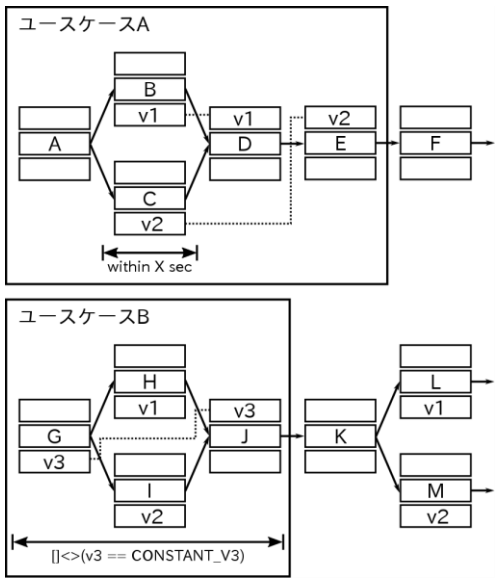


図. 8 機能の実行順序の可視化

図.8 において、3 つの矩形で構成されるノード A，B,…、L は、機能を表す（以下、機能ノードと呼ぶ）。機能ノード間のリンクは、システム仕様に記載されたユースケースを実現するために必要な機能の実行順序を示す。機能ノード間に複数リンク（例えば、図.8 における A→B，A→C，G→H，G→I 等）が出ている場合は、機能 B と C、機能 H と I は、並列動作することを示す。

各機能ノードは、上段(事前)・下段（事後）に、各々その機能が参照・変更する状態変数を記述する。中段（ボディ）には、機能が、事前条件に記載された状態変数を事後条件に記載された状態変数に変換するアルゴリズムを記述する。状態変数を介した機能ノードの事前・事後の接続関係が、機能間の依存関係を表すものとする。例えば、図.8 のユースケース A では、機能ノード B に対して、v1 を事前条件に持つ機能ノード D が依存している。機能ノード C には、 v2 を事前条件に持つ機能ノード E が依存していることを破線のリンクで表現している。

これら可視化の結果を利用し、並行に実行される可能性のある機能については並列試験を、依存性がある機能に関しては、その順序試験を設定することで、品質要求の機能性に関する試験ケースを漏れなく設定する。

機能ノードの事前・事後に用いる状態変数は、プロセス 3 で作成したデシジョンテーブルの条件、アクション部の記載内容から抽出する。例えば、図.6 の例では、状態変数“ふた”を事前条件として、事後条件として“ふた”を抽出する。機能間の並列性については、システム仕様書に明示的に時間的な制約が記載されている機能、あるいは、同時に開始される可能性のあるユースケースを抽出し、ユースケースを構成する機能ノード間に依存関係がある場合を並列試験の実施対象試験ケースとして抽出する。

## 5. 並列・順序試験実行装置

並列するユースケースの数に従って、並列・順序試験として実行すべき試験ケース数は爆発的に増大し、試験ケースの実行には、膨大な工数が必要になる。また、複数機能を並行動作させる試験は、その実行が実機上で困難であることもある。

並列・順序試験の効率化のために、並列・順序試験実行装置を構築した[11,12]。装置の構成の概略を図.9 に示す。実行装置では、プロトコル変換器（プロアナ）を介して被試験機器と直接ネットワーク接続して行うシステム試験と、被試験機器のソフトウェアを実行装置上のエミュレータで模擬動作させて行うソフトウェア試験を実施することができる。ソフトウェア組合せ試験から、システム試験までの広い範囲の試験に対応する。

また、機能を並列に、あるいは順序を入れ替えての実行を容易にするために、図.9 に示すテストケース実行環境は、モデル検査器を内蔵している。オープンソースであり、試験ケースの記述言語として C 言語の埋め込みが可能であることから搭載するモデル検査器のコアとして SPIN[13]を採用し、これを独自拡張した[11]。

モデル検査器では、試験ケースに記述された試験ステップを非決定的に実行することができる[13]。このモデル検査器の非決定的な実行機能を活用し、並列・順序試験を実行する。並行試験は、機能の試験ステップの実行順序の制御することで、機能の順序試験は、機能の実行順序を制御することで、それぞれ実現する。

一方で、従来のモデル検査器においては、時相論理で記述された検査式の真偽をシミュレータ上で確認するしか検証方法が提供されていなかった。このため、検証結果の解析が難しく、実務に適用する際に大きな障壁となっていた。本研究では、モデル検査用の試験ケースから、直接、実機や実機のソフトウェアをネットワークやイベントを介して操作可能なように拡張することで、実際に機器を動作させる実行テストとモデル検査用の試験ケースの共用化と、実行テスト併用による試験結果の解析の容易性を実現した[14]。

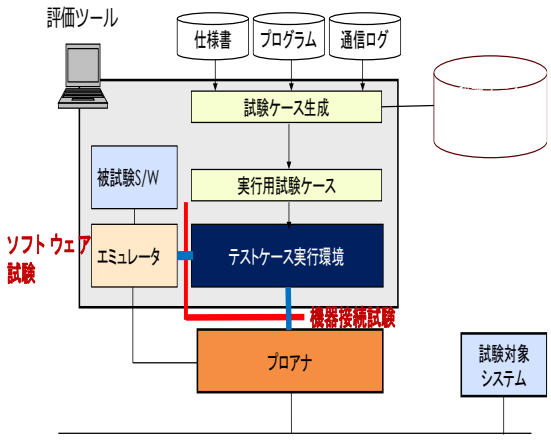


図. 9 並列・順序試験実行装置の概略構成

並列・順序関係実行環境は、図.10 に示す手順で活用される。入力として、モデル検査器 SPIN の試験ケース言語 PROMELA[13]にて記述された試験ケースを受取る。試験ケースには、テンプレートエンジン用のタグが埋め込まれており、シミュレータ上で検査式の真偽を確認するモデル検査を行うのか、実機あるいは実機のソフトウェアを用いた実行テストを行うのかを、同一の試験ケースから選択できる。

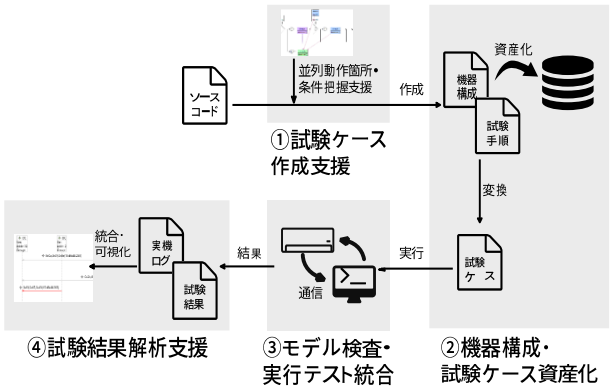


図. 10 並列・順序試験実行環境におけるテスト実施手順

さらに、並列・順序試験の試験ケースは、第 4 章で示した並列・順序試験抽出ツールの出力結果と図.10 に示す試験手順リポジトリから自動生成される。図.8 に示す可視化表現から以下の処理を行い、順序通りの実行と並列な実行を、PROMELA で記述された試験ケースとして出力する（図.11）。

Case1: 順序通り（分岐してない）機能の場合
PROMELA の単一プロセス上に、試験ステップを図の順に並べて統合することで、順序通りの実行を実現する。

Case2: 並列な（分岐している）機能の場合
Step1: 分岐の発生箇所で PROMELA の子プロセスを生成し、ステップの実行を並列化する。
Step2: 子プロセスを生成したプロセスは、子プロセス終了までの判定が False となる判定式を置き、実行を延期する。
Step3: 各子プロセスは、終了時点で終了フラグを立てる。
Step4: 子プロセスの終了後、子プロセスを生成したプロセスの実行を再開する。

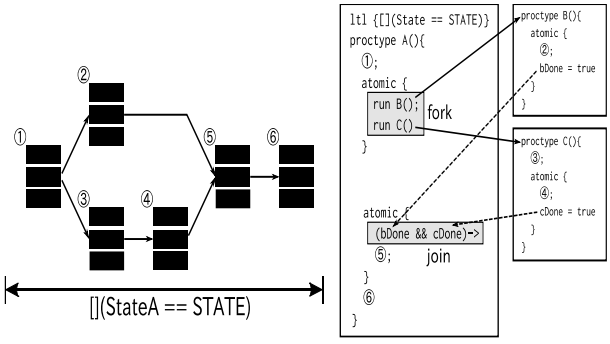


図. 11 並列・順序試験ケースの生成

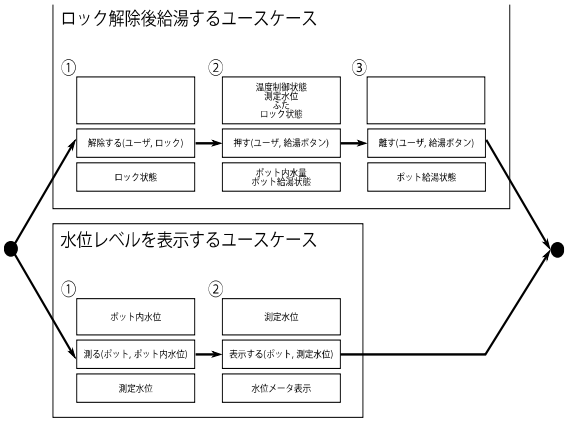


図. 12 機能の並列・順序関係の記述例

前述の話題沸騰ポットの“ロック解除後給湯するユースケース”と“水位レベルを表示するユースケース”の例を用いて、並列・順序試験ケースの抽出の手順を説明する。“ロック解除後給湯するユースケース”と「“水位レベルを表示するユースケース”の 2 つのユースケースが並列に動作している。また、ユースケース内の機能は順序どおりに動作する。図.12 の左側の黒点が開始点であり、ここから 2 つのユースケースが並列動作し、右側の黒点で終了する。

1. 最左の黒点にて 2 つのユースケースの並列実行を開始し、また同時に両者の終了を待つプロセス A、及び、並列動作する両ユースケースの表す 2 つプロセス B、C を追加する。
2. 図 11 の最左の黒点に相当するプロセス A の最初のステップにて 2 つのプロセス B 及び C を子プロセスとして生成する。
3. プロセス A の 2 つめのステップとして、図 12 の最右の黒点に相当する、2 つのプロセスの終了を待つための条件を追加する。今回は 2 つのプロセス B 及び C の終了を表すフラグ変数をそれぞれ bDone、cDone として、(bDone && cDone)なる条件を置く。また、bDone、cDone をグローバル変数として宣言する文を追加する。
4. プロセス B では“ロック解除後給湯するユースケース”を反映する。“ロック解除後給湯するユースケース”内部にある①～③の機能は順序どおりに動作するものであることから、Case1 の方法に従い、プロセス B 中に各機能を順序どおりに追加する。
5. プロセス C は“水位レベルを表示するユースケース”を反映する。このユースケース中には順序どおり動作する機能が①、②の 2 つある。ここから Case1 の方法に従い、プロセス C 中に各機能を順序どおりに追加する。
6. プロセス B 及びプロセス C の各々の最終ステップに、プロセス A にて両プロセスを合流するための条件を成立させる。プロセス B ではステップ bDone=true を追加し、直前の機能の記述と一緒に atomic にて囲む。同様にしてプロセス C の最後の機能の記述の直後に cDone=true なるステップを追加し、直前の最後の機能の記述と一緒に atomic で囲む。

以上の操作により、PROMELA の試験ケースが、図.12 に示す可視化結果より生成される（図.13）。

```
bool bDone, cDone
proctype A() {
 atomic {
  run B()
  run C()
 }
 (bDone && cDone)
}
proctype B() {
 // 解除する(ユーザ, ロック)
 // 押す(ユーザ, 給湯ボタン)
 atomic {
  // 離す(ユーザ, 給湯ボタン)
  bDone = true;
 }
}
proctype C() {
 // 測定する(ポット, ポット内水位)
 atomic {
  // 表示する(ポット, 測定水位)
  cDone = true;
 }
}
```

図. 13 並列・順序試験ケースの生成例

## 6. おわりに

本研究では、自然言語で記載された仕様書をセミ形式記述に変換し、機能の依存関係と並列関係を可視化することで、並列・順序試験の対象となる試験ケースの抽出を支援するツールを開発した。さらに、可視化された機能間の関係に基づき、モデル検査器用の試験ケース記述言語に変換するツールを開発した。これらのツール開発にあわせ、実機や実ソフトウェアを対象に直接実行テストを行う並列・順序試験実行装置を構築し、これら一連のツール、装置の開発により、並列・順序試験のテストケース設計の品質向上とテスト実行効率を向上させることを試行した。

本論文では、紙面の都合上、並列・順序試験抽出ツール、並列・順序試験実行環境をとりあげ説明した。他プロセス・ツールに関しては、別報にて報告する予定である。今後は、第 3 章に示した各ツールの精緻化を進めるとともに、図.2 に示した試験設計のプロセスを実務に適用し、テストケース設計の品質向上とテスト実行効率への寄与を定量的に評価する予定である。

### 謝辞

本研究は、科研費 16K00100, JST CREST JPMJCR1304 の試験を受けて実施されたものである。

### 参考文献

[1] 増田 聡, 松尾屋 徹, 津田 和彦, ”テストケース作成自動化のための意味役割付与”, ソフトウェア工学の基礎壱ＸＸⅡ, pp.71-pp.76, (2015).
[2] Daisuke Kawahara and Sadao Kurohashi, “A Fully-Lexicalized Probabilistic Model for Japanese Syntactic and Case Structure Analysis”, In Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL2006), pp.176-183, (2006).
[3] Collet Rolland and Camille Ben Achour, “Guiding the construction of textual use case specifications”, Data & Knowledge Engineering, Vol. 25, Issues 1–2, pp.125–160, ELSEVIR, (1998)
[4] Noriyuki Kushiro, Ryoichi Torikai and Tatsutya Ehira, Trial for System Design with Logic Visualization Tool, International Joint Conference on Artificial Intelligence Workshops, July, 2015
[5] 久代 紀之、鳥飼 諒一：システムデザインにおける論理的枠組みとデータ活用戦略、HI 学会学会誌、May.2015
[6] ISO/IEC/IEEE JTC I/SC7, “Software and systems engineering – Software testing – Part.4: Test techniques”, pp.70—pp.72, (2015)
[7] Lee Copeland, “A Practitioner’s Guide to Software Test Design”, Artech House, (2004)
[8] 組込みソフトウェア管理者・技術者育成研究会(SESSAME), “話題沸騰ポット第 7 版”, http://www.sesami.jp/
[9] 青山裕介, 黒岩丈瑠,久代紀之, “モデル検査の実行順序制約の図式表現と試験ケースの自動生成”, 電子情報学会論文として投稿中, (2017)
[10] 青山裕介、黒岩丈瑠、久代紀之, “既存ソフトウェア部品を用いたソフトウェア開発におけるソースコード理解支援ツール”, 第 15 回情報科学技術フォーラム), 第 4 分冊, pp.111-118, (2015)
[11] 青山祐介、黒岩丈瑠、久代紀之：ＣＰＳのためのモデル検査・実行テスト統合環境の構築–モデル検査器を用いた試験ケースの資産化、第 16 回情報科学技術フォーラム, (2016)
[12] Takeru Kuroiwa, Noriyuki Kushiro, “Testing Environment for Embedded Software Product Line”, 12th ACS/IEEE International Conference on Computer Systems and Applications AICCSA, (2015)
[13] G.J. Holzmann, The SPIN model checker: Primer and reference manual, vol.1003, Addison-Wesley Reading, (2004)
[14] 吉岡昌己, 青木利晃, 田原康之, “SPIN によるモデル設計モデル検証”, 近代科学社, (2008)
[15] Takeru Kuroiwa, Yusuke Aoyama, Noriyuki Kushiro, “Testing environment for CPS by cooperating model checking with execution testing,” Procedia Computer Science, vol.96, pp.1341–1350, (2016)