CB-003

# テストケース自動生成のための自然言語の形式変換アルゴリズム
# Semi-formalizing Algorithm for Automated Test Case Generation from Documents in Natural Language

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

## 1 はじめに

システム開発の最終工程で実施されるシステムテストケースの設計は、自然言語で記述されたシステム仕様書を元にして行われる。テスト設計者は、システム仕様書に記載された文や表を注意深く読み解き、これら各文から動作条件・確認項目を抽出し、動作条件・確認項目を具体化することでテストケースを作成する。一般に、この作業には、膨大な工数が必要となり、筆者らの調査では、600 ページ程度の仕様書で 500 人時–1000 人時の工数を必要とする。

一方で、システム仕様書には、往々にして誤りや漏れが存在することも知られている。また、自然言語による仕様記述には曖昧さが含まれるため、テスト設計者の解釈によっては、仕様書に記載された意図とは異なるテストケースを作成してしまうこともある。これらの修正・補完のため、システム設計者とテスト設計者間でのレビューが必須となる。

本研究は、上記課題の解消のために次を実施した。

1. 自然言語仕様書からの動作条件・確認項目の抽出作業を支援するために、日本語で記述されたシステム仕様書からセミ形式記述と呼ぶ論理記述に変換するアルゴリズム・ツールの開発
2. レビューにおけるシステム設計者・テスト設計者による誤りの発見・修正を支援するために、セミ形式記述で記載された仕様書の論理構造をわかりやすく可視化するツールの開発
3. 上記レビュー中のシステム設計者・テスト設計者間のコミュニケーションから指摘された事項を仕様に追加し、修正されたセミ形式記述仕様から、デシジョンテーブル形式でテストケースを自動生成するツールの開発

これら開発したツール・アルゴリズムを使って、企業の一線でテスト設計を行っている技術者らに「話題沸騰ポット」を題材にしたテスト設計を実施させ、その有用性を確認した。

## 2 セミ形式記述

自然言語で記述されたシステム仕様書の動作条件・確認項目の間の論理関係を厳密に記述するために、システム仕様を図 1 に示すセミ形式記述で記述する。セミ形式記述は、命題論理を用いた仕様記述手法 [1] を元に、テストケース生成を行うための拡張を行ったものであり、次のように記述する。

**自然言語仕様:**「沸騰モードに設定したら、ポットは水を沸かす。」

```
/* Written in Extended Backus–Naur Format (EBNF) */
statement = expression "." ;
expression = "(", expression, ")",
  ["<", constraint, { ",", constraint }, ">"] |
  expression, "&", expression |
  expression, "|", expression |
  expression, "–>", expression |
  "!", expression |
  clause;
clause = verb, "(", subject, ",", object,
  {",", verb_constraint}, ")",
  ["<", p_constraint, {",", p_constraint }, ">"];
p_constraint = ["!"], "O", "<", group, ">"| /* One */
  ["!"], "E", "<", group, ">" | /* Exclusive */
  ["!"], "I", "<", group, ">" | /* Inclusive */
  ["!"], "R", "<", direction, ",",
  group, ">"; /* Require */
direction = "s" | "d"; /* Source or Destination */
group = ident;
verb = ident;
subject = ident;
object = ident;
verb_constraint = ident;
ident = /* Japanese and English Letters */;
```

図 1 セミ形式記述の文法

**セミ形式仕様:**「設定する（?, ?, 沸騰モードに）–> 沸かす（ポット, 水）.」

セミ形式記述では、「関係語（主体、対象、制約）」を命題プリミティブと呼んで一つの操作・確認項目の単位とする。命題プリミティブ同士を論理演算子――Not（記号: !）、And（記号: &）、Or（記号: |）、Imply（記号: –>）――で接続することで厳密な論理関係で仕様を記述する。命題プリミティブは、関係語・主体・対象の 3 つ組によってテストで行うべき操作や確認すべき項目を表現し、続く制約によってテストのパラメータを列挙する。命題プリミティブ中の「?」は、省略により自然言語仕様に対応する語がないことを意味する。自動詞など対象がない場合には「_」を用いて表現する。

加えて、排他のような論理演算子での記述が難しい論理関係の記述の容易化のために、原因結果グラフ [2] から 4 つの命題間制約―― One（ただ 1 つの命題が成り立つ）, Exclusive（多くとも 1 つの命題が成り立つ）, Inclusive（少なくとも 1 つの命題が成り立つ）, Require（ある命題 A の充足には命題 B の充足が必要である）――を図 1 の p_constraint として導入する。

動作条件・確認項目の間の論理関係は Imply を用いて「動作条件 –> 確認項目」と表現する。前述のセミ形式記述の例では、「設定する（?, ポット, 沸騰モードに）」という行為（動作条件）が成立すると、「沸かす（ポット, 水）.」という動作結果（確認項目）がもたらされる

†九州工業大学 Kyushu Institute of Technology

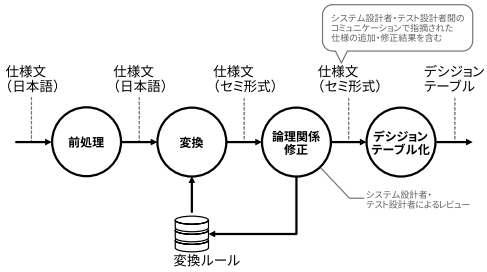



図 2　セミ形式記述への変換プロセス

仕様を記述している。

## 3　セミ形式記述変換アルゴリズム

本節では、自然言語で書かれたシステム仕様書を、セミ形式記述に変換するアルゴリズムについて述べる。本変換アルゴリズムは、図 2 に示すプロセスで構成され、形態素解析器 Juman++ [3]（v2.0.0-rc2[1]）及び，日本語構文解析器 KNP [4]（v4.1.9[2]）から得られる feature [5] のデータ（格や品詞，係り受けなどを含む文法データ）及び並列構造の検出結果から構築したルールを元に、システム仕様書から動作条件・確認項目の抽出と動作条件・確認項目間の論理関係を同定する。

図 2 の前処理ステップは，日本語仕様書の記述を統一することで、形態素解析の正確さの向上と、変換ルールを簡単化を図るものであり、以下を行う．

1. Juman++で未定義語となる文字の置換（例: 半角カナを全角カナへ置換）
2. 文の終了スタイルの統一（「……すること.」の「こと」を取り除き、「……する.」のように用言で文を終えるスタイルに統一）

図 2 の変換ステップでは，Algorithm 1 により日本語の仕様文をセミ形式記述へ変換する。 Algorithm 1 は，ルール部（Algorithm 1 の Rule 1，2）が feature データを元にして、文節ごとに変換処理の方法を決定し、選ばれた変換処理方法で各文節をセミ形式記述へ変換する（Algorithm 1 の Conversion 1–3）という構成になっている。ルール部で選択される各変換処理の方法はオペレータと呼び、変換アルゴリズムに組み込まれている。オペレータの一部を以下に示す。

**句の結合オペレータ:** 連体修飾している句——文節を複数結合したもの——の結合と、命題プリミティブへの句の挿入を行う。命題プリミティブへの句の挿入に際しては、本研究では、表層格から主体・対象・制約を判定し、ガ格の句を主体、ヲ格の句を対象、それ以外の修飾句を制約として扱い、命題プリミティブに挿入する。

**命題の結合オペレータ:** 命題——命題プリミティブ及び命題プリミティブを論理演算子で結合したもの——同士を二項演算子で結合する。

オペレータ同士には結合の優先順位をつけ、ボトムアップにセミ形式記述に変換する。上記オペレータの例では、句の結合オペレータ > 命題の結合オペレータという優先順位になっており、各命題プリミティブの項の挿入が完了した後で、命題同士が結合される（Algorithm 1 の line 5）。

[1] https://github.com/ku-nlp/jumanpp/releases
[2] http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP

**Algorithm 1** 日本語文からセミ形式記述への変換アルゴリズム

**Definition:**
children($b$)：文節 $b$ の子文節を返す。
push($l, e$)：リスト $l$ に要素 $e$ を追加する。
pop($l$)：リスト $l$ から最後の要素を取り除いて返す。
last($l$)：リスト $l$ の最後の要素を返す。
reversed($l$)：リスト $l$ を逆順にする。
apply($c, e$)：オペレータ $c$ をタプル $e$ 中のセミ形式記述 $d$ に適用し、適用結果のセミ形式記述 $d'$ と $d'$ の変換元となった文節のリスト $B'$ のタプルを返す。（なお、$e = (d, B)$. $B$ は $d$ の変換元となった文節のリストである。）
NewSemiElem($b$): 文節 $b$ をセミ形式記述に変換する。
NewOp($s_t$, $s_s$): 2 つのセミ形式記述 $s_t$ と $s_s$ からオペレータを返す。なお、$s_t = (d_t, B_t)$. $s_s = (d_s, B_s)$. $d_t$ と $d_s$ は、セミ形式記述であり、$d_t$ が $_s$ を修飾する関係とする。$B_t$ と $B_s$ は、それぞれ $d_t$ と $d_s$ の変換の元となった文節のリストである。

**Input:**
$b$: ある日本語文の末尾の文節

**Output:**
$s$: 変換されたセミ形式記述

```
1:  procedure CONVERT(b)
2:      stack ← ()
3:      C ← children(b)
4:      s_t ← (NEWSEMIELEM(b), (b))                          ▷ Rule 1
5:      s_c ← nil
6:      while true do
7:          if s_c = nil then
8:              if |C| = 0 then
9:                  break
10:             end if
11:             c ← pop(C)
12:             s_c ← (CONVERT(c), (c))
13:         end if
14:         op_{s_c} ← NEWOP(s_t, s_c)                       ▷ Rule 2
15:         if |stack| = 0 ∨ comb_{s_c} is prior to last(stack) then
16:             push(stack, comb_{s_c})
17:             s_c ← nil
18:             continue
19:         end if
20:         comb_last ← last(stack)
21:         if comb_last can apply to s_c then
22:             s_c ← apply(comb_last, s_c)                  ▷ Conversion 1
23:             continue
24:         end if
25:         s_t ← apply(comb_last, s_t)                      ▷ Conversion 2
26:         s_c ← nil
27:     end while
28:     for all comb in reversed(stack) do
29:         s_t ← apply(comb, s_t)                           ▷ Conversion 3
30:     end for
31:     return s_t
32: end procedure
```

変換ルールの一部を次に示す。

**Rule 1:** Algorithm 1 の NewSemiElem($b$) に相当する。
1. 文節 $b$ が用言の時、$b$ を関係語にした命題プリミティブに変換する。
2. 文節 $b$ が体言の時、$b$ を句に変換する。

**Rule 2:** Algorithm 1 の NewOp($s_t, s_c$) に相当する。（$s_t = (d_t, B_t), s_c = (d_c, B_c)$. $d_t, d_c$ はセミ形式記述を表し、$B_t, B_c$ はそれぞれ $d_t, d_c$ に対応する文節のリストを表す）
1. $d_c$ が命題の時、命題の結合オペレータを選択する。 命題間の論理関係は次のルールで決定する。
   (a) $B_c$ の末尾の文節が接続助詞「か」、「または」の時、Or とする。
   (b) $B_c$ の末尾の文節が接続助詞「かつ」の時、Or とする。

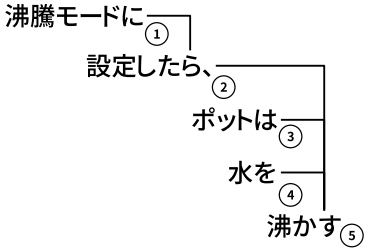



図 3　係り受けの構造の例

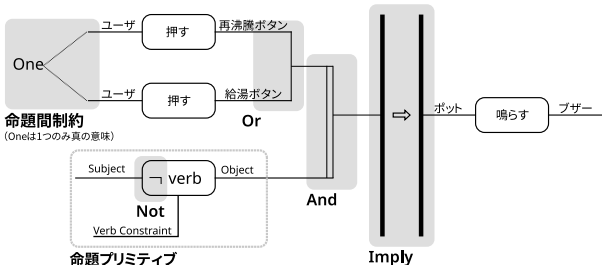



図 4　命題ネットワークの例

(c) $B_c$ の末尾の文節が条件形の用言の時、Imply とする。
(d) 上記いずれでもないときはデフォルト値として、And とする。

2. $d_c$ が句の時、句の結合オペレータを選択する。

上記変換ルール及びオペレータを用いると、図 3 の係り受け構造を持つ日本語文「ポットを沸騰モードに設定したら、ポットは水を沸かす。」は、次のステップでセミ形式記述「設定する（?, ポット, 沸騰モードに）–> 沸かす（ポット, 水）.」へ変換される。

1. 文節⑤「沸かす。」が用言なので，Rule 1 の（1）に従い，を命題プリミティブ「沸かす（?, ?)」へ変換（Algorithm 1 line 4）
2. 文節②「設定したら、」が用言なので，Rule 1 の（1）に従い，を命題プリミティブ「設定する（?, ?)」へ変換（Algorithm 1 line 4）
3. 文節②「設定したら、」の子文節①「沸騰モードに」が体言なので，Rule 1 の（1）に従い，句「沸騰モードに」に変換（Algorithm 1 line 4）
4. 「沸騰モードに」は Rule 2 の（2）に合致するので，句の結合オペレータを選択（Algorithm 1 line 14）
5. 句の結合オペレータにより，句「沸騰モードに」を命題プリミティブ「設定する（?, ?)」と結合し，命題プリミティブ「設定する（?, ?, 沸騰モードに)」を生成（Algorithm 1 line 29）
6. 命題プリミティブ「設定する（?, ?, 沸騰モードに)」が命題なので，Rule 2 の（1）に従い，命題の結合オペレータを選択（Algorithm 1 line 14）
7. 文節③「ポットは」が体言なので，Rule 1 の（1）に従い，句「ポットは」に変換（Algorithm 1 line 4）
8. 「ポットは」は Rule 2 の（2）に合致するので，句の結合オペレータを選択（Algorithm 1 line 14）
9. 命題の結合オペレータよりも句の結合オペレータのほうが優先度が高いので，次の子文節のオペレータの適用を試みる（Algorithm 1 line 11）
10. 文節④「水を」が体言なので，Rule 1 の（1）に従い，句「水を」に変換（Algorithm 1 line 4）
11. 「水を」は Rule 2 の（2）に合致するので，句の結合オペレータを選択（Algorithm 1 line 14）
12. 命題の結合オペレータよりも句の結合オペレータのほうが優先度が高いので，次の子文節のオペレータの適用を試みる（Algorithm 1 line 11）
13. 文節⑤「沸かす。」の子文節のオペレータを適用する。句の結合オペレータにより，句「水を」を命題プリミティブ「沸かす (?, ?)」と結合し，命題プリミティブ「沸かす (?, 水)」を生成 (Algorithm 1 line 29)
14. 句の結合オペレータにより，句「ポットは」を命題プリミティブ「沸かす（?, 水)」と結合し，命題プリミティブ「沸かす（ポット, 水)」を生成（Algorithm 1 line 29）
15. 命題の結合オペレータにより，命題プリミティブ「設定する（?, ?, 沸騰モードに)」を命題プリミティブ「沸かす（ポット, 水)」と結合し，命題プリミティブ「設定する（?, ?, 沸騰モードに） –> 沸かす（ポット, 水)」を生成（Algorithm 1 line 29）

変換されたセミ形式記述は、元仕様書の論理関係の誤り、及び変換ルールの不足が原因で誤った結果が得られることがある。セミ形式記述上に含まれていた誤りは、次節に述べる命題ネットワーク上を用いたレビュー支援により修正される。このレビュー時に発見された変換誤りを防止するように変換ルールを蓄積していくことで、変換の正確さを漸進的に改善する。

## 4　セミ形式記述上の論理関係の修正支援

本論文で提案するアルゴリズムにより自動変換されたセミ形式記述は、次に示す要因から誤りを含みうる。

1. 元の日本語仕様書の論理関係の誤り
2. 変換ルールの不足

上記誤りの修正のために、システム設計者・テスト設計者によるレビューを実施する。

本研究では，上記論理関係の修正を支援するために、命題ネットワーク（図 4）と呼ぶ図式表現によりセミ形式記述に記述された命題プリミティブ間の論理関係を可視化する。図 4 では，テキストベースでは表現・把握の難しい階層的な論理構造図式化したり、同時・選択的に成り立つ命題プリミティブのまとまりを、並列に命題プリミティブを整列したりすることによって論理構造の把握支援を行う。

命題ネットワーク上でシステム設計者・テスト設計者によるレビューを実施することで、上記論理関係の誤りを発見する。レビューで指摘された事項を元のセミ形式仕様に修正・追加することで、動作条件・確認項目間の論理関係の誤りを除去したセミ形式記述を得る。

## 5　セミ形式記述のデシジョンテーブル化

本節では、人手によるセミ形式記述からテストケースへの変換ミスを防ぐために、セミ形式記述の仕様を自動的にデシジョンテーブル [6] 形式のテストケースを生成する手法について述べる。

セミ形式記述では、各動作条件・確認項目を命題プリミティブとして表現し、命題プリミティブ同士を論理演算子で接続した論理式として機能の仕様を表現してい

表 1　セミ形式記述のレビュー結果（表 5）の No. 1 から生成したデシジョンテーブル

| | | テストケース | | |
|---|---|---|---|---|
| | | 前件真 | | 前件偽 |
| | 命題 | 1 | 2 | 3 |
| 条件 | 押す（ユーザ，沸騰ボタン） | T | F | F |
| | 押す（ユーザ，保温ボタン） | T | - | F |
| 動作 | 鳴らす（ポット，ブザー） | T | T | - |

表中の記号は、 F: 真、T: 偽、-: Don't Care である。

表 2　評価実験参加者のプロファイル

| | 経験年数 |
|---|---|
| ID | テスト設計及びテスト実行 |
| A | 13 |
| B | 8 |
| C | 4 |

る。このため、命題プリミティブの真理値割当を算出することで、動作条件・確認項目の取りうる組み合わせを漏れなく導くことができる。

本研究では、命題論理式の真理値割当を導出できるツール PyEDA [7] を用いて各命題プリミティブの取りうる真理値割当を求める。本研究では機能の仕様を Imply を用いて「動作条件 –> 動作結果」と表現するので、Imply の前件に含まれる命題プリミティブの真理値割当をデシジョンテーブルの条件欄に、Imply の後件に含まれる命題プリミティブの真理値割当をデシジョンテーブルの動作欄に埋めることで、自動的にデシジョンテーブルを生成する（例: 表 1）。

上記の方法で算出される動作条件・確認項目の組み合わせは、膨大な数となることがある。この対策として、本研究では、各命題プリミティブが真偽のどちらの値をとっても論理式の真理値を変えない時、この命題プリミティブを Don't Care という値に縮約することで、テーブルサイズを縮小する。

## 6　ケーススタディ

自然言語仕様書からセミ形式記述への変換手法については、[8] にて評価結果を発表済みである。本論文では、セミ形式記述で書かれた仕様のレビューを想定し、セミ形式記述に変換した仕様文から動作条件・確認項目を修正が可能なことを確認するケーススタディを行った。本ケーススタディに当たり、表 2 のプロファイルを持つテスト技術者 3 名からなるグループに、セミ形式記述上で、省略語の補完、及び命題ネットワーク上で論理関係の誤りをレビューにて修正させた。

ケーススタディ対象の仕様には実験参加者の製品ドメインの知識の多寡によりレビューでの指摘事項が増減することを防ぐために、実験参加者の業務とは異なる製品ドメインの仕様書を採用した。本実験では、曖昧さを含む仕様書として公開されている話題沸騰ポット第 3 版 [9] から表 3 に示す 10 文の仕様文を用いた。

## 7　実験結果と考察

実験参加者によるセミ形式記述の修正の結果は表 5 である。実験参加者による補完内容を、[9] から曖昧さを取り除いた仕様書として公開されている第 7 版 [9] の仕様書と比べると、表 5 中の下線部分が不適当な補完内容

表 3　実験で用いた仕様文

| No. | 仕様文 |
|---|---|
| 1 | 第 n 水位センサが on で、かつ満水センサが off の場合、温度制御が可能になります。 |
| 2 | それ以外の場合は、沸騰ボタン・ヒータは動作しません。 |
| 3 | 蓋が開けられると、ヒータは停止します。 |
| 4 | 沸騰ボタンは動作しません。 |
| 5 | ヒータが動作していないときは、沸騰ランプ及び保温ランプは消灯します。 |
| 6 | 保温モードに設定した際、100℃でなかった場合は、必ず一度沸騰させた後、自然に冷やしながら設定温度に保つ動作をします。 |
| 7 | タイマは最大 1 時間まで設定できます。 |
| 8 | ユーザからボタン（タイマ・保温設定・沸騰・解除・給湯の 5 つ）が押された時、ブザーを 1 回鳴らします。 |
| 9 | しかし、上記 2 つの制約時には、沸騰ボタンが押されてもブザーを鳴らさないこととします。 |
| 10 | ユーザが設定したタイマのタイムアウト時、及び沸騰状態終了時には、ブザーを 3 回鳴らします。 |

表 4　セミ形式化した仕様文

| No. | セミ形式の仕様文 |
|---|---|
| 1 | is（第 n 水位センサ，?，on）& is（満水センサ，?，off） –> なる（温度制御，?，可能に）. |
| 2 | is（?，?，それ以外） –> !動作する（沸騰ボタン，?） & !動作する（ヒータ，?）. |
| 3 | 開ける（?，蓋）–> 停止する（ヒータ，?）. |
| 4 | !動作する（沸騰ボタン，?）. |
| 5 | !動作している（ヒータ，?） –> 消灯する（?，沸騰ランプ）& 消灯する（?，保温ランプ）. |
| 6 | 設定する（?，?，保温モードに）& !is（?，?，100℃）–> 沸騰する（?，?，必ず一度）& 保つ（?，?，設定温度に）. |
| 7 | 設定できる（タイマ，?，最大 1 時間まで）. |
| 8 | 押す（ユーザ，沸騰ボタン） –> 鳴らす（?，ブザー，1 回）. |
| 9 | is（?，?，上記 2 つの制約時に）& 押す（?，ボタン“タイマ・保温設定・沸騰・解除・給湯の 5 つ”）–> !鳴らす（?，ブザー）. |
| 10 | タイムアウトする（ユーザが設定したタイマ，?）& is（?，?，沸騰状態終了時）–> 鳴らす（?，ブザー，3 回）. |

となっていた。下線部分のうち、「*」部分は、システムに相当する語であり、実験参加者は、自明であるとして意図的に補完を行わなかった。

本ケーススタディでは、レビューにシステム仕様の設計者が参加していないため正しい仕様確認し、修正することはできなかったものの、システム仕様書中にテスト設計者が補完・修正を実施した箇所が明示された。

上記レビューの結果、修正の誤りは残ったものの、参加者は「?」で示された補完が必要な箇所の候補に対して、補完作業を行うことができた。ここから、実験参加者は、命題プリミティブの主体・対象・制約が表現する内容を理解し、修正作業が可能なことを確認した。

表 5 の内容を命題ネットワークとして可視化し、論理関係の誤りの修正を行ってもらった。図 5 に修正結果の一部を示す。

網掛け部分に示す部分がレビューにて補完された箇所である。図 5 の No. 1 は、動作条件の不足を指摘・補完し、No. 3、4 では、動作条件の記述されていない表 5 の No. 4 に適当な動作条件を議論した結果、No. 3 と同等であると判断され、統合されたものである。No. 5a–No. 5b は、No. 5（表 5 の No. 5 に相当）の確認項目に記載の沸騰ランプと保温ランプを点灯するための動作がないことから追加された仕様である。No. 5b は沸騰ランプと保温ランプを点灯するための仕様として追加されたものであり、No. 5c は No. 5b とは異なるボタンを押したときのランプの点灯動作を表す仕様である。上記から、命題

表５　レビューにより省略語が補われたセミ形式記述

| No. | セミ形式の仕様文 |
|---|---|
| 1 | is（第 n 水位センサ，_，on） & is（満水センサ，_，off） –> なる（温度制御，_，可能に）. |
| 2 | is（水位，_，それ以外） –> !動作する（沸騰ボタン，ブザー） & !動作する（ヒータ，ランプ）. |
| 3 | 開ける（ユーザ，蓋） –> 停止する（ヒータ，ランプ）. |
| 4 | !動作する（沸騰ボタン，ランプ）. |
| 5 | !動作している（ヒータ，ランプ） –> 消灯する（*，沸騰ランプ） & 消灯する（*，保温ランプ）. |
| 6 | 設定する（ユーザ，*，保温モードに） & !is（温度センサ，_，100℃） –> 沸騰する（ヒータ，水，必ず一度） & 保つ（ヒータ，水温，設定温度に）. |
| 7 | 設定できる（タイマ，_，最大 1 時間まで）. |
| 8 | 押す（ユーザ，ボタン”タイマ・保温設定・沸騰・解除・給湯の 5 つ”） –> 鳴らす（*，ブザー，1 回）. |
| 9 | is（*，_，上記 2 つの制約時に） & 押す（ユーザ，沸騰ボタン） –> !鳴らす（*，ブザー）. |
| 10 | タイムアウトする（ユーザが設定したタイマ，_） & is（*，_，沸騰状態終了時） –> 鳴らす（*，ブザー，3 回）. |

上記「*」の記号は、「_」という回答を、本文中での参照のために置き換えたもの。

表６　セミ形式記述のレビュー結果（表 5）の No. 1 から生成したデシジョンテーブル

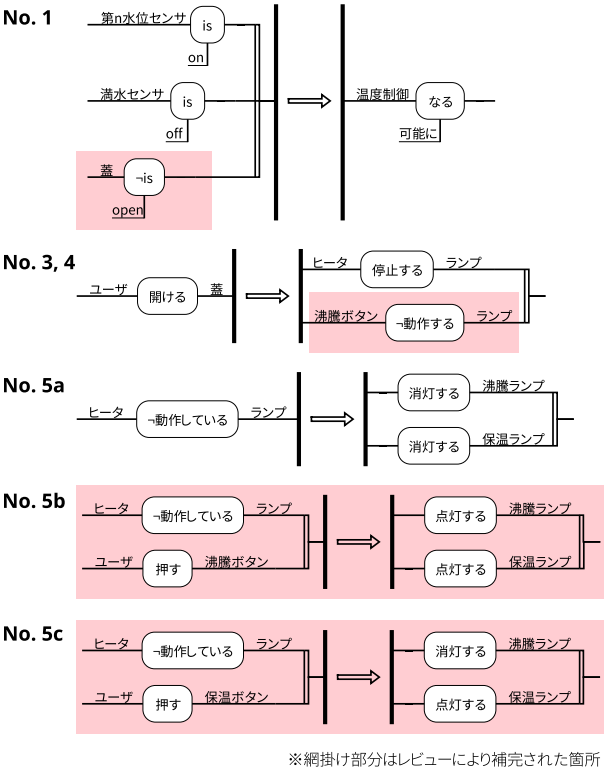
| | | テストケース | | |
|---|---|---|---|---|
| | | 前件真 | 前件偽 | |
| | 命題 | 1 | 2 | 3 |
| 条件 | is（第 n 水位センサ，_，on） | T | F | T |
| | is（満水センサ，_，off） | T | - | F |
| 動作 | なる（温度制御，_，可能に） | T | - | - |

ネットワークを使って動作条件・確認項目の論理関係がレビューでき、誤りを指摘・修正可能なことが確認された。本研究で提案する命題ネットワークによって、従来のテスト設計手法と異なり、自然言語仕様書上の動作条件・論理関係を曖昧さなく表現し、仕様の誤りを明示されることを確認した。

レビュー前後のセミ形式記述からデシジョンテーブルを生成すると、表 6、7 に示すものが得られた。表 7 は命題ネットワーク上でのレビューによる修正を受けて、動作条件「is（蓋，_，open）」を含めたセミ形式記述から自動的に生成された。ここから、セミ形式記述上の動作条件・確認項目の修正結果から、人手によるミスなく網羅的な条件組み合わせのテストケース生成が実現されることを確認した。

本デシジョンテーブルの各列には動作条件・確認項目の間の真偽値の組み合わせが記載されているので、各命題プリミティブのパラメータを真偽値に合わせて設定することで、テストの実行に活用できるテストケースとすることができる。本研究のテストケース生成では、各動作条件・動作項目の組み合わせまでを達成した。命題プリミティブの真偽値に対応するパラメータの具体化については今後の課題とする。

本ケーススタディでテスト設計者により提案された修正内容は、システム設計者への確認が行われていないため、補完された仕様自体は必ずしも正くない（曖昧さの除去された仕様書 [10] に照らすと、図 5 の No. 5b と No. 5c は誤りである）。しかし、セミ形式記述化、命題ネットワークによる可視化により、自然言語仕様書の動作条件・確認項目の間の論理関係の曖昧さが取り除ける




※網掛け部分はレビューにより補完された箇所

図５　実験参加者により修正された命題ネットワーク

表７　命題ネットワークのレビュー結果（図 5）の No. 1 から生成したデシジョンテーブル

| | | テストケース | | | |
|---|---|---|---|---|---|
| | | 前件真 | 前件偽 | | |
| | 命題 | 1 | 2 | 3 | 4 |
| 条件 | is（第 n 水位センサ，_，on） | T | F | T | T |
| | is（満水センサ，_，off） | T | - | F | T |
| | is（蓋，_，open） | F | - | - | T |
| 動作 | なる（温度制御，_，可能に） | T | - | - | - |

こと、また、セミ形式記述、命題ネットワーク上でのレビューが実施できること、さらにレビューにより修正されたセミ形式記述から自動的にテストケースが生成されることを確認した。システム設計者を交えたレビューにより修正内容が保証されれば、自然言語仕様書の論理関係の曖昧さを原因としたテストケースの設計の誤りは防止できると考える。

従来のテスト設計では、テスト設計者が修正・補完した仕様に対するレビューが、テストケース上のレビューで行われる。レビューでの指摘により、テスト設計をやり直す必要からテスト設計に工数がかかる。

一方、本研究で行うテスト設計では、セミ形式記述への変換によって動作条件・確認項目を明らかにしたのち、セミ形式記述上での動作条件・確認項目の修正、命題ネットワーク上での動作条件・確認項目の論理関係の追加・修正というように、段階的にテストケースへの変換していく。セミ形式記述上での動作条件・確認項目の修正や、命題ネットワーク上での論理関係の修正の各々のレビュー結果で誤りが指摘されたとしても、テスト設計全体をやり直す必要がなく、従来より短いループの手戻りとなる。この結果、提案する手法によるテスト設計

では、テスト設計全体にかかる工数を短くできると考える。従来手法と提案手法との間でテスト設計にかかる具体的な時間の差異については今後の評価で明らかにする予定である。

## 8 関連研究

自然言語仕様書からデシジョンテーブル形式のテストケース生成を行う手法として、本研究の他に [11] がある。[11] は、日本語の仕様文からデシジョンテーブルの条件欄・動作欄に割り当てるべき句の同定を行う。[11] では条件欄・動作欄に割り当てられた句の間の論理関係については同定手法までは提案しないため、デシジョンテーブルを完成させるために、元の自然言語仕様書を読み返す必要がある。対して、本研究では、各動作条件・確認項目を表現を命題プリミティブという単位で、論理関係の同定も行いながら抽出する。このため、同定された論理関係から動作条件・確認項目の間の真理値組み合わせを埋め、デシジョンテーブルを完成できる。

自然言語の曖昧さを取り除いて仕様記述を行う手法としては、Z 記法 [12] や VDM++[13] といった仕様の形式記述手法がある．これら手法は厳密な仕様記述の実現し、記述した仕様の自動検証も実現する一方、自然言語での仕様記述を置き換えが必要となる。本研究で提案するセミ形式記述は、自然言語で書かれた仕様の理解・修正支援と、テストケース生成のための中間表現として扱う。このため、仕様書は従来の通り自然言語で書かれるものと想定し、セミ形式記述への変換アルゴリズムによって、仕様の形式を変換する方式を取る。セミ形式記述の文法は、自然言語の表現を引き継ぎ、また、命題論理の記号を用いて記述することで、習得の容易化と厳密な仕様記述の両立を図った。

## 9 まとめと今後の課題

自然言語で書かれたシステム仕様書を元に実施されるシステムテスト設計では、自然言語で書かれた機能の動作条件・確認項目の間の論理関係が曖昧なことで、誤ったテストケースを作成してしまう課題がある。

本研究では本課題解決のために、自然言語で書かれたシステム仕様書をセミ形式記述という論理記述に変換し、論理関係を厳密に表現する。セミ形式記述で書かれた仕様をレビューすることで、動作条件・確認項目間の論理関係を、解釈の曖昧さなく修正する。また、修正された論理関係を元にデシジョンテーブル形式のテストケースを自動生成することで、システム仕様からの操作ミスのないテストケース生成も実現する。上記実現のために、自然言語で書かれたシステム仕様書をセミ形式記述へ自動変換するアルゴリズム・ツールと、変換されたセミ形式記述上で動作条件・確認項目のレビューを支援するための可視化ツール、修正されたセミ形式記述からデシジョンテーブルを生成するツールを開発した.

開発したツールを使い、電気ポットの仕様書を対象に、熟練のテスト設計者によるレビューを実施した。レビューの結果、セミ形式記述に変換されたシステム仕様から動作条件・確認項目の論理関係の誤りを発見・修正可能なことを確認した。

本ケーススタディでは、従来のレビュー手法との比較を行っておらず、指摘される誤りの数や時間、効率などが明らかになっていない。従来手法と比較した評価は今後の課題とする。

また、本手法で生成したテストケースは、各動作条件・確認項目の論理的な組み合わせの算出までであるため、テスト実行に活用するためには、各動作条件・確認項目のパラメータの具体化を行う必要がある。今後は本パラメータの具体化の支援も行い、テスト実行まで活用可能なテストケース生成の実現を予定する。

### 謝辞

本研究は、JSPS 科研費 16K00100 の一部支援を受けたものである。

### 参考文献

[1] Colette Rolland and Camille Ben Achour. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering*, Vol. 25, No. 1-2, pp. 125–160, 1998.

[2] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*, pp. 65–84. John Wiley & Sons, 2 edition, 2004. 長尾真 and 松尾正信（訳）. ソフトウェア・テストの技法, 近代科学社, 2006.

[3] Arseny Tolmachev, Daisuke Kawahara, and Sadao Kurohashi. Juman++: A morphological analysis toolkit for scriptio continua. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 54–59, 2018.

[4] Daisuke Kawahara and Sadao Kurohashi. A fully-lexicalized probabilistic model for japanese syntactic and case structure analysis. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pp. 176–183. Association for Computational Linguistics, 2006.

[5] 河原大輔. KNP で付与される feature 一覧. `http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP`, 2013. Last accessed: May 31, 2019.

[6] Boris Beizer. *Software testing techniques*, pp. 269–276. Dreamtech Press, second edition, 2003. 小野間彰 and 山浦恒央（訳）. ソフトウェアテスト技法, 日経 BP 出版センター, 1994.

[7] Chris Drake. Python library for electronic design automation (PyEDA). `https://media.readthedocs.org/pdf/pyeda/v0.28.0/pyeda.pdf`, 2011. Last accessed: May 23, 2018.

[8] 村上響一, 青山裕介, 村上神龍, 久代紀之. 自然言語で記載された仕様書からのテストケース自動生成アルゴリズムの構築. 研究報告ソフトウェア工学 (SE), Vol. 2019, No. 9, pp. 1–8, 2019.

[9] 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 3 版. Last accessed: May 23, 2018.

[10] 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. Last accessed: May 23, 2018.

[11] 増田聡, 松尾谷徹, 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方法. コンピュータ ソフトウェア, Vol. 34, No. 2, pp. 16–27, 2017.

[12] J Michael Spivey and JR Abrial. *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[13] Eugene Diirr and J Van Katwijk. VDM++: A formal specification language for object oriented designs. In *Proceedings 6th Annual European Computer Conference, Compeuro*, pp. 214–219, 1992.