Vol.2021-SE-209 No.8
2021/11/25

自然言語仕様書からの試験ケースの自動生成—図表を含む仕様書からの機能試験の抽出と品質試験の統合

吉井　亮介[1]　若松　大雅[1]　久代　紀之[1]

概要：システム開発において，テストは重要な工程である．テスト設計者は，システム仕様書の記述文（図・表を含む）を精読し，各記述文から“条件 –> 期待動作“の組を抽出し，これをベースに試験ケースを設計する．これに加え，製品ドメイン毎に蓄積された品質試験項目を追加し，機能試験項目と品質試験項目からなる試験ケースを設計する．本研究では，仕様書に含まれる文・図・表を形式記述へと変換し，“条件 –> 期待動作“を抽出することで，試験ケースを自動生成するアルゴリズム・ツールを開発する．さらに，ドメイン毎の品質試験ケースを形式記述化したパターンとして蓄積し，機能試験項目と統合することで品質試験ケースを含む試験ケースの生成を試行する．

Automatic generation of test cases from natural language specifications - Extraction of functional tests from specifications containing diagrams and integration of quality tests

Abstract: Testing is an important process in system development. The test designer carefully reads the description of the system specification (including figures and tables), extracts the ”condition –> expected behavior” pairs from each description, and designs the test cases based on these pairs. In addition to this, we add the quality test items accumulated in each product domain, and design test cases consisting of functional test items and quality test items. In this study, we develop an algorithm and tool to automatically generate test cases by converting sentences, figures, and tables in the specification into formal descriptions and extracting ”condition –> expected behavior. In addition, we will try to generate test cases including quality test cases by accumulating quality test cases for each domain as patterns with formal descriptions and integrating them with functional test items.

1. はじめに

システム開発では，システムに求める機能動作を文書化した “システム仕様書” を作成する必要がある．プログラマなどの技術者はこれを熟読し，システムに求められる機能動作を実装していくことになる．また，開発されたシステムが仕様書通りに動作するかどうかを確認する，“テスト” の工程は重要であり，ここで機能要求や品質要求を保証することになる．

機能要求は開発において重要な要素であり，正しく認識されていなければならない．テスト設計者は，仕様書から試験項目を抽出し，それらを “条件 –> 期待動作” の組として記述することで，機能要求のテストケースを作成可能である．一方で品質要求は，ISO25000 シリーズ [1] で定められている規格のような，セキュリティ，使用性といったシステムの品質について定めた要素であり，機能要求を満たすだけでなく，適切な品質要求が十分に満たされていることも重要である．

現状，仕様書を精読し，テストケースの生成を行うという手作業が行われるために，生成されるテストケースはテスト設計者の知識に依存してしまう課題点がある．また，仕様書は自然言語で記載されるため，自然言語が持つ曖昧さにより解釈を誤る可能性もある．仕様書に含まれる要素は文・図・表があり，それぞれからシステムの機能を洗い出すため，技術者の読解能力や経験によって正確さが変動することや，そもそも読解・テスト設計に時間がかかることも問題である．

さらに，品質テストについても，テスト設計者が製品ドメインにおける品質要求の知識を保有している場合があ

[1] 九州工業大学

り，それが言語化・共有されていないという観点から，品質要求のテストケースの設計もテスト設計者の知識に依存してしまうという課題もある．

本研究では，これらの課題を解決するために，自然言語で記載された仕様書に含まれる文・図・表を，表記に依存しない形式記述に変換し，テストケースを自動生成するアルゴリズムを開発する．また，テスト設計者の知識として蓄積されている品質試験の知識・経験をパターンとして抽出し，知識の蓄積や品質試験への再利用が可能になるよう形式化する．さらには，これらの処理をツールとして実装し，機能および品質のテスト設計の効率化を図る．

2. 研究の全体像

仕様書からテストケースへの流れ，品質試験との統合の全体像を説明する．中間報告のように，研究１と研究２に分けて説明する．

図１に研究の全体像を示す．まずシステム仕様書が存在し，そこに記載されている文・図・表を抜粋し，それぞれを形式記述化する．これは，それぞれの要素が曖昧さをもつ場合があり，論理構造を明らかにする必要があるために行う．次に形式化された記述から試験ケース（テストケース）を生成し，これをシステムのテストに用いることで機能要求を確認することができる．ここで，生成されたテストケースや形式記述をレビューした結果，それらに修正が必要だった場合は，その修正内容を仕様書へ反映させ，仕様書を更新する．この変換処理は，仕様書と修正後のテストケースや形式記述の間での論理的な等価性を保持するためである．ここまでで生成されたテストケースは仕様書から変換されたものであり，仕様書に含まれる機能試験項目を含んでいる．ここに，品質試験項目を追加していく．品質試験についてはまずパターン [2] の蓄積を行う．技術者へのヒアリングなどにより情報を言語化し，パターンとして蓄積する．そしてそれを適用したいシステムに対して適切な表記へと調整した上で，論理関係を維持しながら機能試験項目と統合していく．それにより，最終的に生成されるテストケースは機能試験項目と品質試験項目の両方を持つものになり，これをシステムのテストに用いることで，機能要求と品質要求を確認することができるようになる．

この全体像のうち，仕様書に含まれる図や表を形式記述への変換の部分と，品質試験パターンの蓄積および機能試験との統合の部分が本研究の取り組みであり，それぞれ“研究１”，“研究２”とする．以降の章ではこの２つに分けて説明する．

3. 研究１：仕様書の形式化と試験ケース生成

システム開発で作成される仕様書は，Excel や Word のようなバイナリベースの仕様書は，バージョン管理や再利用が困難という課題点がある．これの解決策として，Asciidoc[3] 等のマークダウン記述（テキストベース）への以降が有効である．なぜなら，テキストベースのファイルは Git 等を用いたバージョン管理が可能であり，それに伴い再利用も容易であるためである．

本研究における仕様書は，テキストベースの仕様書として Asciidoc を用いる．仕様書を記述するときには Asciidoc の文法に従い，それを pdf ファイルなどに出力することで仕様書を作成することができる．仕様書中の図の表現には PlantUML[4] を用いる．

3.1 自然言語文章の形式化

自然言語で記述された仕様書は，論理関係に曖昧さがあるために，テスト設計者の解釈に誤りが生じる場合がある．[5] では，自然言語の文をセミ形式記述と呼ぶ表現で論理記述することにより，論理関係の曖昧さを排除するアルゴリズムを提案している．

セミ形式記述では，日本語文を単文化し，各単文を「関係語（主体, 対象, 制約）」という形式（以降命題プリミティブと呼ぶ）で表現することで，テストの各操作・確認項目のパラメータを明確化する．また，命題プリミティブ同士を命題論理の記号 “!”（否定），“&”（論理積），“|”（論理和），“–>”（論理包含）を用いて接続することで，厳密な論理関係の記述を行う．

セミ形式記述への変換はルールベースのアルゴリズムとして実現されており，形態素解析器 Juman++ [6]（v2.0.0-rc2[*1]）と日本語構文解析器 KNP [7]（v4.1.9[*2]）によって日本語文を解析した結果得られる feature [8] のデータ（格や品詞などの文法データ）を用いて文節ごとに変換する．このルールは，例えば，用言の形態素を含む文節を命題プリミティブにおける関係語にする，体言でありかつガ格の助詞を含む文節を命題プリミティブにおける主体にする，用言の活用形が条件形の場合には係り先の命題プリミティブと論理包含で結合する，といったものが Python スクリプトとして記述されている．

[5] の変換アルゴリズムを用いることで，例えば「ユーザがボタンを押すと，ポットはランプを 3 秒間点灯させる.」という自然言語の文は，[5] のアルゴリズムによって「押す（ユーザ, ボタン） –> 点灯する（ポット, ランプ, 3 秒間）」と変換される．

3.2 図表のセミ形式化

図や表の形式化は，図 2 の流れで変換される．図や表の中で記述される自然言語文については，3.1 と同様にセミ形式記述へと変換する．また，2 で説明したように，セミ形式記述から図・表への逆変換も行う．逆変換を行うことで，図・表とセミ形式記述の間で論理的に等価であること

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

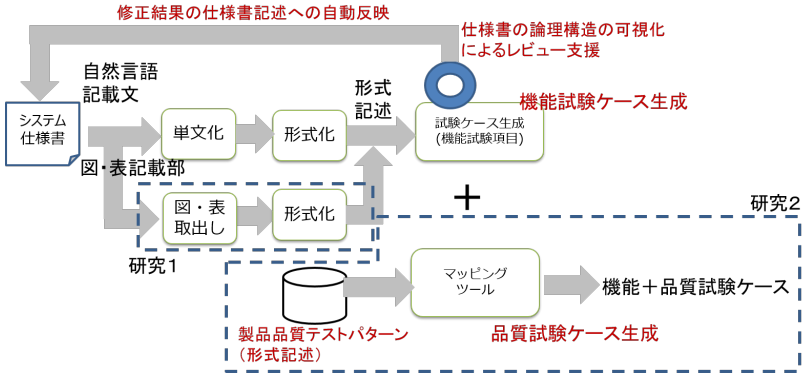

図 1　仕様書からテストケースまでの流れ
Fig. 1　Flow from specifications to test cases

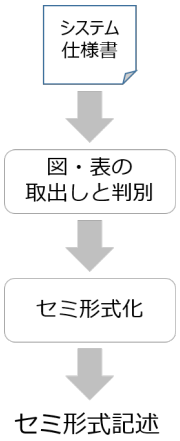

図 2　図・表の形式記述の流れ
Fig. 2　Flow of formal description of figures and tables

を保つことができる．他にも，セミ形式記述をレビューし，内容の修正・追加等を行った場合に，その内容を仕様書へと反映し，仕様書の更新を容易にしてくれるという利点もある．

仕様書に記載される図にはいくつか種類があり，状態遷移図やシーケンス図など，さまざまであり，それぞれは異なった文法に従い，図へと変換されるようになっている．仕様書から図の記載部を抽出したあと，図の種類についての判別が必要であり，PlantUML の文法をもとに区別を行う．

表については，表のどのセルにどんな情報が記載されているかを判別する必要がある．３種類の仕様書を対象とした調査により，その仕様書中に含まれる 238 個の表のうち，77%が 2 種類の形式に絞られることが判明した．よって，多くの表は各セルの記載情報が限定的であるため，ルールベースでの変換が可能である．

以降に図・表についての変換ルールや変換例を示す．

3.2.1　表の変換

図 3 は表の変換のイメージである．Asciidoc のテキスト，出力される表，セミ形式記述の間の対応付けが示されている．ここで記述されている表は，表頭に条件や期待値などの項目カテゴリがあり，それ以下のセルに具体的な値が記載されている．自然言語として解釈すると，「“条件” が “条件 A” のとき，“期待値” が “期待値 B” である．」となる．これがセミ形式記述へと変換され，逆変換時にはこのセミ形式記述（または修正したもの）から表が生成される．

3.2.2　図の変換

仕様書で用いられる UML の図にはいくつかあり，それぞれで PlantUML の文法が異なる．図 4 は状態遷移図についての変換イメージである．Asciidoc のテキスト，出力される図，セミ形式記述の間の対応付けが示されている．状態遷移図には “状態定義” と “状態遷移の” ２つがあり，これらとセミ形式記述の対応を表 1 に示す．状態遷移は「“状態 A のとき，トリガーが起こり，ガード条件を満たしているとき，エフェクトを実行して状態 B になる”」という解釈になり，これをセミ形式記述で表したものが表 1 で示されている．トリガー・ガード・エフェクトについてはそれぞれ省略が可能であり，省略された要素が省かれたセミ形式記述へと変換される．セミ形式記述において関係語の末尾に “Trigger” などと記述があるが，逆変換時に各命題プリミティブがどの要素に該当するかを判断するためのもので，逆変換時にはこの記述は省略される．

3.3　デシジョンテーブル生成

3.1 や 3.2 によって，セミ形式記述が生成された．このセミ形式記述をデシジョンテーブルへと変換することで，網羅的な論理関係の組み合わせでテストケースを生成する．セミ形式記述は，命題論理の記号を用いて仕様が記述されているため，各命題プリミティブが取りうる真理値を求めることでテストケースが生成できる．この事実に基づき，命題プリミティブの充足可能な真理値を網羅的に求め，デシジョンテーブルを生成する [5] の手法により，セミ形式記述からデシジョンテーブルを生成する（図 5）．

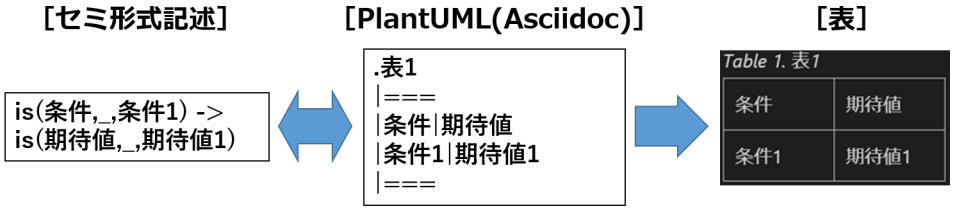

図 3 表の変換イメージ
Fig. 3 Example of table transformation

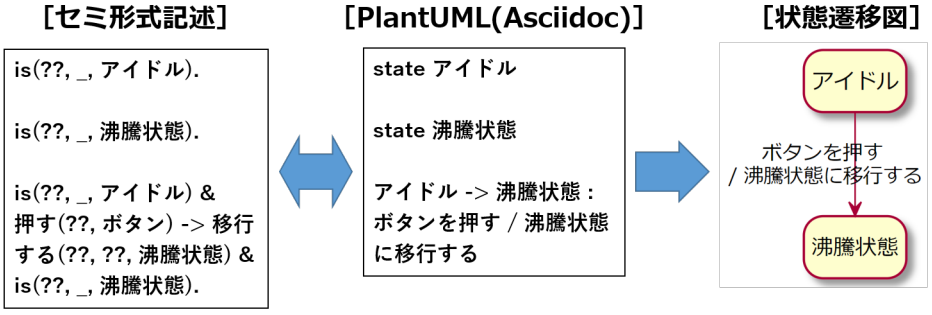

図 4 状態遷移図の変換イメージ
Fig. 4 Example of state transition diagram transformation

4. 研究２：機能試験と品質試験の統合

自然言語で記載された仕様書から生成されるテストケースは，仕様書に明示的に記載されるものは機能要求に関するものであることから，品質試験項目についての情報は不十分である．品質要求についての知識は，テスト設計者の知識に依存するものであり，共有する必要がある．そこで，品質試験項目についての知識をパターンとして共有する．

パターンとは，ソフトウェア設計において頻出する問題に対して適用される解決策のことである．本研究では，テスト設計がテスト設計者の知識に依存しないよう，製品ドメインの品質テストのパターンを言語化し，蓄積する．製品ドメインごとに蓄積されたパターンは，複数の製品の品質テストのために再利用することができ，テスト設計の作業効率化を可能とする．以降は，このパターンを記述・蓄積し，それを機能試験（研究１）の項目と統合する流れについて説明する．

4.1 パターンの記述・蓄積

パターンの再利用のためにはまず，ドメイン内で発生した頻出課題を言語化し，テンプレートに従い記述する必要がある．このパターンの抽出は，そのドメイン内での品質試験の知識・経験が深く身についている技術者へのヒアリングまたは，技術者本人のパターン記述によって実現が可能である．

パターンのテンプレート（各項目の説明）を図 2 に示す．“問題分類” や，“問題”，“状況” を参照することで，そのパターンが試験を行いたいシステムに必要かどうかを選定することができる．特に重要な項目は “テストケース” の欄である．この欄はそのパターンにおける試験の内容をセミ形式記述で記載しており，これを参照しながら品質試験を行うことになる．研究２においては，このセミ形式記述を仕様書から出力された機能試験の内容（セミ形式記述）と統合し，最終的なテストケースの生成を目指す．

表 3 は [9] で記述されているテスト内容より抽出したパターン記述例である．内容としてはシステムのエラーメッセージについて確認する試験項目となっている．ユーザによる文字・数字の入力操作があり，画面出力が存在するシステムのテストでは活用できる可能性があると考えられる．

4.2 機能試験項目と品質試験項目の統合

3.1 節で，仕様書からセミ形式記述への変換を行い，4.1 で，パターンの記述および蓄積を行った．ここでは，２つのセミ形式記述を論理的に結合し，１つのセミ形式記述へと変換する処理を行う．まず，結合の前に各セミ形式記述の表記を統一する．なぜなら，パターンはドメイン内での再利用のために汎用的でなければならないため，表記が適用したいシステムに特化したものではないからである．パターン（または仕様書側）のセミ形式記述を構成する単語を調整し，自動で論理的な結合ができるようにする．

統合の流れを以下に示す．ここでの単語とは，セミ形式記述における関係語，主体，対象，制約に該当する文字列を示している．

(1) 仕様書（セミ形式）とパターン（セミ形式）から１文ずつ選出する．
(2) 各セミ形式記述を単語ごとに分解する．

表 1　状態遷移図の変換ルール
Table 1　Transformation rules for state transition diagrams

	PlantUML	セミ形式記述
状態定義	state 状態 A	is(??, _, 状態 A).
状態遷移	状態 A –> 状態 B : Trigger [Guard] / Effect	is(??, _, 状態 A) & XX"Trigger"(X1, X2) & YY"Guard"(Y1, Y2) –> ZZ"Effect"(Z1, Z2) & is(??, _, 状態 B).

		antecedent=T 0	antecedent=F 1	antecedent=F 2
condition	is(_, _, 状態A)	T	F	T
condition	押す(??, ボタンX)	T	–	F
action	is(_, _, 状態B)	T	–	–

図 5　デシジョンテーブル
Fig. 5　Decision table

表 2　パターンの記述方法
Table 2　Pattern description method

項目	内容
名前	パターンの内容を指す造語
状況	このパターンが適用されるべき状況
問題	起こりうる問題，具体的な確認事項
問題分類	パターンの大まかな分類
テストケース	試験内容をセミ形式記述でまとめたもの
フォース	本品質試験パターンの適用の際に考慮すべき事柄

(3) 各単語に対し，変換後の単語があればユーザが指定する．
(4) 各単語を指定された単語に変換し，新たなセミ形式記述として出力する．

パターン（セミ形式）として “点滅する（マシン, ランプ, 赤色）.” を選出したと仮定する．パターン（セミ形式）の関係語 “点滅する” を “点灯する” に，主体 “マシン” を “ポット” に変換したいという指定を行った場合，これに従い各セミ形式記述を変換すると，パターンは “点灯する（ポット, ランプ, 赤色）.” へと変換される．この操作は必要に応じて仕様文（セミ形式記述）に対しても同様に行うことができる．

以下に，セミ形式記述の論理的結合の順序について示す．
(1) 2 つのセミ形式記述を選出する．
(2) それぞれのセミ形式記述に含まれる命題プリミティブから 1 つずつを選出し，比較する．命題プリミティブを構成する要素のうち，関係語，主体，対象がすべて一致した場合，それらは結合可能と判断し，両命題プリミティブに含まれる制約の全要素を制約に持つ命題プリミティブを構成する．
(3) 2 つのセミ形式記述の論理構造に対して，表 4 を参照しながら適切な結合を行い，1 つのセミ形式記述を出力する．表 4 は，2 つのセミ形式記述の論理構造によって，結合結果が変化することを示している．また，各セミ形式記述を構成する命題プリミティブ（A1，A2，A3，B，C）については，以下に詳細を示す．

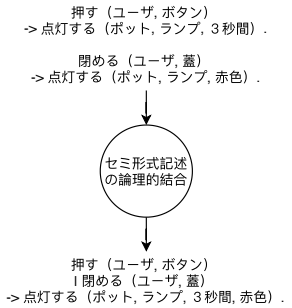

図 6　セミ形式記述の結合例
Fig. 6　Combination example of semi-formal description

- A1，A2 は，関係語，主体，対象がすべて一致している命題プリミティブである．
- A3 は，A1，A2 と関係語，主体，対象がすべて一致しており，A1，A2 それぞれが持つ制約をすべて制約として持っている．
- B，C は，A1，A2，A3 と関係語，主体，対象の少なくとも 1 つは不一致（B，C 間も同様）である．

結合可能な命題プリミティブが選出できなかった場合，2 つのセミ形式記述は結合不可能とし，2 つのセミ形式記述をそのまま出力し，本処理を終了する．

図 6 は，論理的結合の例である．どちらも “–>”（論理包含）を含む場合の結合例である．2 つのセミ形式記述はそれぞれ “–>” の後件部分が一致しており，前件部分は論理和で結合され，後件部分は制約が並べられた状態で 1 つのセミ形式記述に結合されていることを確認できる．

5. ツールの開発

3 と 4 で示したアルゴリズムを実装したツールをそれぞれ開発した．それぞれのツールの実行例について説明する．

5.1 研究 1：仕様書の形式化と試験ケース生成

図 7 は，3 において開発したツールの実行の様子である．図 7 では， Visual Studio Code（実際は任意のエディタ

表 3 パターンの記述
Table 3 Pattern description example

項目	内容
名前	エラーメッセージ
状況	・ユーザの入力に対する処理が正しいことを確認する． ・誤ったユーザ入力に対し，適切なエラーメッセージが表示されることを確認する
問題	以下のような入力に対し、誤った入力であれば適切なエラーメッセージが 1 回表示されることを確認する。 型の異なる入力（整数が有効なボックスに実数・文字を入力するなど)、長い入力（許容を超える長さの文字列を入力する)、 数値の境界（各データ型で定義されている数値の範囲を超えたり、0 を入力（正負の境界値）したりする）
問題分類	UI, IO
テストケース	入力する（ユーザ, 文字・数値, 型の異なる入力, 長すぎる入力, 数値の境界）–> 表示する（ソフトウェア, エラーメッセージ, 適切な）
フォース	システムは，ユーザのどんな操作に対しても適切に動作する必要があり，想定外の挙動があってはならないため．

表 4 セミ形式記述の論理的結合
Table 4 Logical combination of semi-formal descriptions

	A1	A1 & B	A1 |B	A1 –> B	B –> A1
A2	A3	A3 & B	A3 |B	A3 –> B	B –> A3
A2 & C	-	A3 & B & C	A3 & C |B	A3 & C –> B	B –> A3 & C
A2 |C	-	-	A3 |C |B	A3 |C –> B	B –> A3 |C
A2 –> C	-	-	-	A3 –> B |C	B –> A3 –> C
C –> A2	-	-	-	-	B |C –> A3

で良い）上で 3 つのタブを開いており，左から，仕様書（Asciidoc)，セミ形式記述（変換後)，仕様書のプレビューを確認できる．仕様書やセミ形式記述をレビューし，仕様書の修正を行った場合はセミ形式記述が，セミ形式記述の修正を行った場合は仕様書が自動で修正・反映されるようになっている．

5.2 研究２：機能試験と品質試験の統合

話題沸騰ポット第 7 版 [10] から 3 文を抜粋し，テストケースの生成までの具体的な実行例を示す．抜粋した自然言語仕様文は以下のとおりである．

- 蓋センサが 3sec 以上 on となったら，蓋が閉じられたと判断する．
- 蓋が閉じられ，水量が適正な場合，沸騰行為をする．
- 蓋が閉じられても，水量が異常な場合，状態はアイドルのままである．

5.2.1 自然言語仕様の形式化

図 8 は，[5] にて開発された変換ツールによって 3 文をそれぞれセミ形式化したものである．“#” で始まる行はコメントとして扱われる．

5.3 パターン記述

表 2 に従い作成したパターン記述を表 5 に示す．このパターンは，開発したツールでパターンの追加が行えることを説明するために作成したものである．

5.3.1 機能試験項目と品質試験項目の統合

表 5 の「解決（セミ形式記述)」の内容と各仕様文（セミ形式）を統合する様子を図 9 に示す．この画面の表体にユーザ入力を行う．入力されているセルには「水量」と書かれており，これにより，パターンのセミ形式記述の単語が「ポット」から「水量」へと変換され，関係語，主体，対象が同一である命題プリミティブが仕様文とパターンの両方に含まれることとなり，論理的結合へと進めることが可能になる．これらの単語が変換される様子は，図 9 のプレビュー画面で確認できる．右のテキストボックスは論理的結合を行ったあとのプレビュー画面である．結合は論理構造をもとに機械的に行っているため，結合結果が正しいかどうかは確認が必要である．そこで，図 9 のように論理記号や制約といった，結合によって変化した部分を赤字で表記し，ユーザに確認を促している．これらの箇所をユーザが確認・修正した後，ファイル出力を行い，本ツールの実行を終了する．

表 5 パターン記述
Table 5 Pattern description

項目	内容
名前	異常を検出したときの伝達
状況	ポットが異常を検出したときの動作を確認する
問題	ポットが異常事態を検出とき，ブザーを 1sec ごとに鳴らし，ユーザに異常を知らせる
問題分類	API
テストケース	is（ポット, _, 異常） –> 鳴らす（ポット, ブザー, 1sec ごとに）．
フォース	想定外の使用が行われたときにユーザに伝える必要があるため

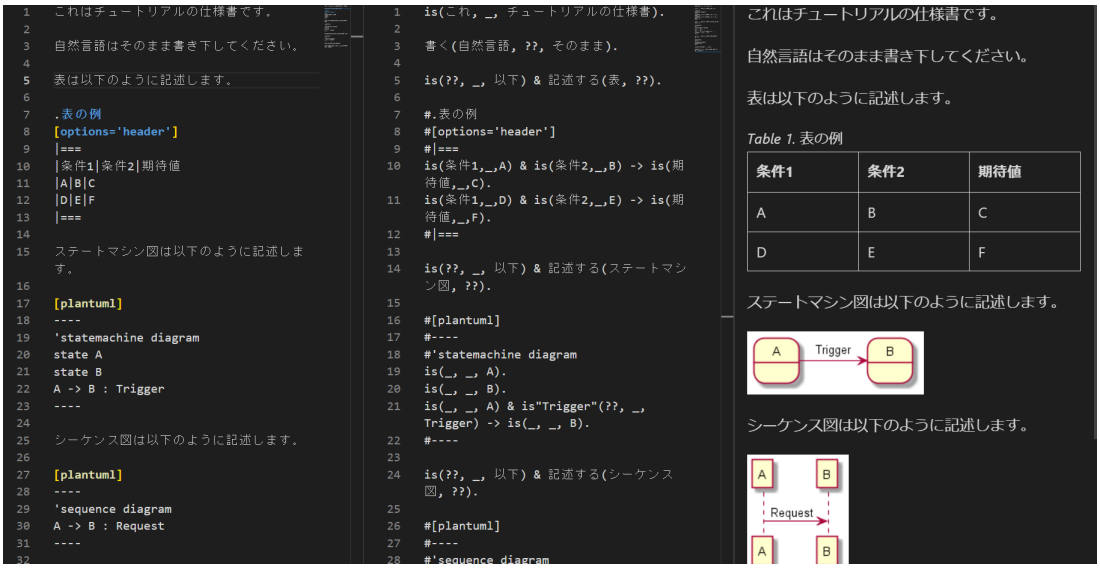

図 7 トレースツールの実行結果
Fig. 7 Trace tool execution results

```
# 蓋センサが3sec以上onとなったら、蓋が閉じられたと判断する。
# ========================================================
成る(蓋センサ, ??, 3sec以上onと) -> 閉じる(蓋, ??) & 判断する(??, ??).

# 蓋が閉じられ、水量が適正な場合、沸騰行為をする。
# ========================================
閉じる(蓋, ??) & is(??, _, 適正) -> する(水量, 沸騰行為).

# 蓋が閉じられても、水量が異常な場合、状態はアイドルのままである。
# ============================================================
閉じる(蓋, ??) -> is(水量, _, 異常) -> is(状態, _, アイドルのまま).
```

図 8 自然言語仕様の形式化
Fig. 8 Semiformalize

5.3.2 デシジョンテーブルの自動生成

論理的結合によって 1 つに結合されたセミ形式記述をデシジョンテーブルに変換する．[5] で開発されたツールに入力することで図 11 のような結果を得られる．

6. まとめ

本研究では，システム仕様書に記載される文・図・表から変換された試験ケース（形式記述）と品質試験ケース（形式記述）を論理的に結合し，試験ケースを自動生成するアルゴリズムを開発した．機能試験は，セミ形式記述や試験ケースから仕様書への逆変換により，論理の一貫性を保ちながら自動生成を行うことができる．品質試験は，それまで技術者が知識として保有していたものを言語化して蓄積することで，その知識を共有，再利用することができ，さらに機能試験との統合により，機能試験項目と品質試験項目の両方を持つ試験ケースを生成することができる．これらをツールで自動化することにより，技術者の知識や技術への依存や，工数がかかる問題を解決することができると考えられる．

今後の予定として，実際のシステム開発現場の中で行われる試験に本ツールを投入し，設計プロセスの明示化と作業効率化，および品質試験のパターン化によるテスト設計のノウハウの蓄積と再利用に貢献できることについての評価を行う予定である．また，評価結果をもとに，ツールの使いやすさやわかりやすさの向上など，システム開発現場での有用性を高めるためのツールの精緻化を行っていく．

参考文献

[1] JIS X 25010:2013 システム及びソフトウェア製品の品質要求及び評価（SQuaRE）－システム及びソフトウェア品質モデル, http://kikakurui.com/x25/X25010-2013-01.html
[2] 鷲崎弘宜. ”ソフトウェアパターン－ 時を超えるソフトウェアの道－: 1. ソフトウェアパターン概観.” 情報処理 52.9 (2011): 1119-1126.
[3] AsciiDoc Home Page, https://asciidoc.org/
[4] シンプルなテキストファイルで UML が書ける、オープンソースのツール https://plantuml.com/ja/
[5] 青山裕介，黒岩丈瑠，久代紀之：テストケース生成のためのシステム仕様書の論理記述変換アルゴリズム, 情報処理学会論文誌， Vol. 61, No. 3, pp. 521–534 (2020).
[6] Tolmachev, A., Kawahara, D. and Kurohashi, S.: Juman++: A Morphological Analysis Toolkit for Scriptio Continua, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 54–59 (2018).
[7] Kawahara, D. and Kurohashi, S.: A fully-lexicalized probabilistic model for Japanese syntactic and case structure analysis, *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, Association for Computational Linguistics, pp. 176–183 (2006).
[8] 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.
[9] J.A.Whittaker：“How to Break Software: A Practical Guide to Testing”, Addison-Wesley, 2007
[10] 話題沸騰ポット要求仕様書（GOMA-1015 型）第 7 版， http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification_v7.PDF

図 9 パターン統合
Fig. 9 Pattern integration

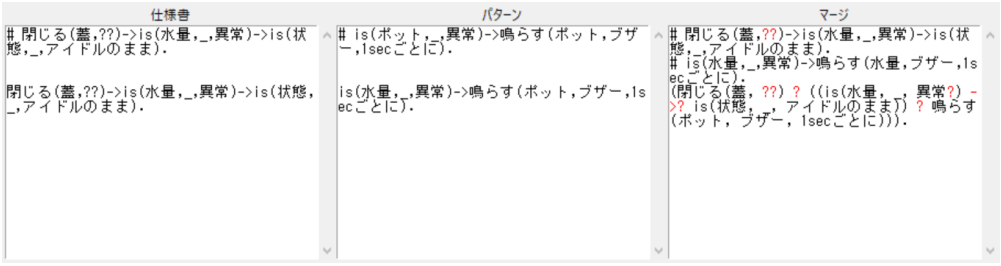

図 10 パターン統合のプレビュー（左：仕様書，中央：パターン，右：結合後）
Fig. 10 Preview of pattern integration(Left: Specification, Center: Pattern, Right: After merging)

		antecedent=T 0	antecedent=T 1	antecedent=T 2	antecedent=F 3
condition	閉じる(蓋, ??)	T	T	T	F
action	is(水量, _, 異常)	F	T	T	-
action	鳴らす(水量, ブザー, 1secごとに)	-	F	T	-
action	is(状態, _, アイドルのまま)	T	-	T	-

図 11 デシジョンテーブルの生成
Fig. 11 Generation of decision table