情報処理学会論文誌 Vol.61 No.3 521–534 (Mar. 2020)

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

青山 裕介[1,a)] 黒岩 丈瑠[1] 久代 紀之[1]

受付日 2019年6月17日, 採録日 2019年11月29日

概要：システム開発の最終段階に実施されるシステムテストケースの設計では，自然言語で書かれた仕様書からテストに必要な入力項目や確認項目を抽出し，抽出した入力項目・確認項目のパラメータを具体化してテストケースが作成される．一方，自然言語で書かれる仕様書は論理関係に曖昧さを含むため，左記の仕様書をもとに作成されるテストケースは，テスト設計者の論理関係の解釈誤りによって不適当になることがあるという課題がある．本研究では，日本語の仕様書から命題論理の記号を用いることで論理関係の曖昧さを排した，セミ形式記述と呼ぶ表現へ変換するアルゴリズムを開発した．セミ形式記述の仕様を対象に入力・確認項目間の論理関係をレビューすることで正しい論理関係へ修正する．論理関係を修正したセミ形式記述は，本研究で開発したテストケース生成アルゴリズムにより，自動的にデシジョンテーブル形式のテストケースへと変換される．本研究で開発したアルゴリズムを電気ポットの仕様文に適用した．提案アルゴリズムにより日本語の仕様文の曖昧さを排除したセミ形式記述へと変換されること，セミ形式記述から自動的にデシジョンテーブルが生成されることを確認した．

キーワード：セミ形式記述，テストケース生成，デシジョンテーブル

Algorithm to Convert System Specification Written in Natural Language to Logical Description for Test Case Generation

Yusuke Aoyama[1,a)] Takeru Kuroiwa[1] Noriyuki Kushiro[1]

Received: June 17, 2019, Accepted: November 29, 2019

Abstract: In the system test case design, the last phase of development, input items and confirmation items for tests are extracted from a natural language specification, and test cases are created by assigning the items to concrete values. The specification written in a natural language often includes ambiguous logical relations among the items. Test designers create invalid test cases due to the misunderstanding of logical relations. In this research, we developed an algorithm that converted Japanese sentences of a specification to a semi-formal description. The semi-formal description describes the logical relations of the items with operators of the propositional logic to remove the ambiguities of the logical relations, and the wrong logical relations of the items are corrected by reviews on the semi-formal description. We also developed an algorithm that converted specification written in the semi-formal description to test cases in the form of the decision table to generate test cases without human-errors. The algorithms were applied to the Japanese specification of an electrical pot system for evaluation. We confirmed that the algorithm automatically converted the Japanese specification to the semi-formal description, which removed the ambiguities of the Japanese specification. We also confirmed the semi-formal description was converted to decision tables.

Keywords: semi-formal description, test case generation, decision table

1. まえがき

テストケースの各入力・確認項目の設計には，システム仕様書に書かれた機能の入力項目と確認項目の間の論理関

[1] 九州工業大学
Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[a)] q794051y@mail.kyutech.jp

係を正しく把握する必要がある．しかし，システム仕様書の記述に広く用いられる自然言語表現では，入力項目・確認項目間の正確な論理関係の記述・把握が難しく，論理関係の解釈の誤りから誤ったテストケースを設計してしまうことがあるという課題がある．

本研究ではまず，機能の入力項目・確認項目間の論理関係を厳密に表現するセミ形式記述を用いてシステム仕様を記述し，解釈の曖昧さなく論理関係をレビュー可能にする．セミ形式記述からデシジョンテーブル形式のテストケースへと自動的に変換することによって，操作ミスによる誤りが混入しないテストケースの生成を実現する．この実現のために，日本語のシステム仕様書の文章をセミ形式記述へ変換するアルゴリズムと，セミ形式記述からデシジョンテーブルへ変換するアルゴリズムを開発した．

以降，2 章で機能の入力項目・確認項目の論理関係を厳密に記述するためのセミ形式記述について説明する．3 章で自然言語仕様書をセミ形式記述へ変換するアルゴリズムについて説明し，4 章でセミ形式記述をデシジョンテーブル形式のテストケースへ変換するアルゴリズムについて述べる．5 章では本変換アルゴリズムの評価について述べ，6 章で評価結果を考察する．7 章で関連研究を紹介し，8 章で本論文をまとめる．

2. セミ形式記述

テストケース設計では，自然言語で書かれた仕様書から入力項目・確認項目を抽出した後に，左記の入力項目・確認項目のパラメータを具体化し，具体化されたパラメータを入力項目・確認項目間の論理関係に基づいて組み合わせることで，テストケースが作成される．一方，自然言語表現は入力項目・確認項目間の論理関係を複数通りに解釈可能という曖昧さを含むので，テスト設計者が論理関係の解釈を誤ると，誤ったテストケースを作成してしまうことがある．

本研究では，自然言語仕様書から抽出した各入力項目・確認項目とその間の論理関係を厳密に表現するために，セミ形式記述（**図 1**）で仕様を記述する．セミ形式記述は，形式的な仕様記述手法 [1] をテスト設計への活用のために本研究において拡張したものである．セミ形式記述では，1 つ 1 つの入力項目・確認項目を「関係語（主体語，対象語，制約語）」という形式で表現する（図 1 の clause に相当し，以降，命題プリミティブと呼ぶ）．日本語では主体語や対象語が省略される場合があり，また，関係語が自動詞の場合，対象語を持たないことがある．セミ形式記述では，省略により主体語や対象語が不明な場合には「?」を用いて表現し，対象語を持たない場合には「_」を用いて表現する．

関係語には動詞の終止形か「is」を記述し，主体語・対象語・制約語間の関係を表現する．「is」は，**図 2** の「アイドル中に」のように用言を持たず，静的な状態（付帯条件）を表す句に用い，「is（?，_，アイドル中）」のように記述する．また，「A は B だ」，「A は B である」といった，A と B の間の静的な同値関係を表す「……だ」，「……である」といった判定詞も「is」を用いて統一的に表現する．

テストに際しては，関係語・主体語・対象語の 3 項目により各入力項目・確認項目で行われる操作やシステムの状態を決定し，続く制約語により入力項目・確認項目のパラメータを列挙する．

命題プリミティブ間の論理関係は，4 つの論理演算子 —— Not（記号：“!”），And（記号：“&”），Or（記号：“|”），Imply（記号：“–>”）——を用いて厳密に記述する．命題論理の記号を用いて入力項目・確認項目を記述することで，論理関係を一意に解釈させる．

自然言語による機能仕様は，「入力項目が成立するならば，確認項目をもたらす」というように，入力項目・確認結果が「ならば（Imply）」に対する前件・後件として記述される．セミ形式記述でも入力項目・確認項目間の論理関係を論理演算子 Imply（“–>”）を使い，図 2 のように「入力項目 –> 確認項目」と表現する．

命題プリミティブ間の論理関係の修正に先立ち，欠落語の補完を行って命題プリミティブの曖昧さを排除する．セミ形式記述上では，元の仕様書に記述されていない欠落語は「?」として明示されるので，「?」にあてはまる語をレビューで埋めることで欠落語を補完できる．

```
/* Written in Extended Backus–Naur Format (EBNF) */
statement = expression, "." ;
expression = "(", expression, ")" |
  expression, "&", expression |
  expression, "|", expression |
  expression, "–>", expression |
  "!", expression |
  clause ;
clause = verb, "(", subject, ",", object,
  { ",", verb_constraint }, ")" ;
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
Fig. 1 The syntax of the Semi-Formal Notation.

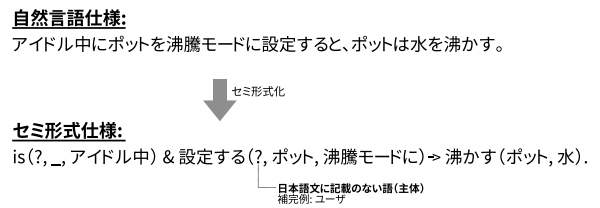


図 2 セミ形式記述の例
Fig. 2 An example of Semi-Formal Specification.

3. 自然言語仕様書からセミ形式記述への変換

3.1 セミ形式記述への変換アルゴリズム

自然言語からセミ形式記述への変換の教師データ（対訳データ）を見つけることができないため，自然言語からセミ形式記述への変換には，機械学習によるアプローチではなく，ルールベースのアプローチを採用した．ルールの構築には形態素解析器 Juman++ [2]（v2.0.0-rc2 [*1]）および，日本語構文解析器 KNP [3]（v4.1.9 [*2]）から得られる feature [4] のデータ（格，品詞，係り受けなどを含む文法データ）と並列な係り受け構造の検出結果を用いる．本論文で提案する日本語の仕様文からセミ形式記述への変換プロセスは**図 3** である．

図 3 の前処理ステップは，形態素解析の正確さの向上と変換ルールの簡単化のために入力された日本語仕様書の記述を統一するものであり，以下を行う．

(1) Juman++で未定義語となる文字種の置換（例：半角カナから全角カナへ置換）
(2) 文の終了スタイルの統一（「……すること.」のように「こと」で文を終えるスタイルを「……する.」のように用言で文を終えるスタイルに統一）

図 3 の変換ステップでは，変換ルールをもとに Algorithm 1 によって日本語の仕様文をセミ形式記述の仕様文へと変換する．Algorithm 1 では，まず各文節を再帰的にセミ形式記述に変換した後（Algorithm 1 の line 4），子文節[*3]から変換されたセミ形式記述と親文節から変換されたセミ形式記述を再帰的に結合していく（Algorithm 1 の Conversion 2–3），という流れで最終的に日本語文全体をセミ形式記述に変換する．子文節と親文節のセミ形式記述の結合に際して，子文節の兄弟文節が And や Or などの論理関係にある場合には，左記の論理関係を維持してセミ形式記述に変換するために，兄弟文節から変換されたセミ形式記述どうしを先に結合（Algorithm 1 の Conversion 1）し，兄弟文節の結合結果を親文節から変換されたセミ形式記述と結合させる．

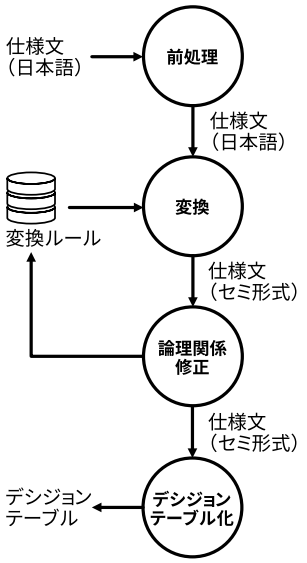


図 3 セミ形式記述への変換プロセス

Fig. 3 A process to the conversion to the Semi-formal Description.

[*1] https://github.com/ku-nlp/jumanpp/releases
[*2] http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP
[*3] 本論文では，文節 c が文節 p に係るとき，p から見た c を子文節，c から見た p を親文節，同じ親文節を持つ子文節どうしを兄弟文節と呼ぶ．

Algorithm 1 An Algorithm to Convert a Japanese Sentence to the Semi-Formal Description

Definition:
- children(b): 文節 b にかかる子文節のリストを返す．
- push(l, e): リスト l に要素 e を追加する．
- pop(l): リスト l から末尾の要素 e を取り除き，e を返す．
- last(l): リスト l の末尾の要素を返す．
- reversed(l): リスト l を逆順にして返す．
- applyF(op, s_t, s_f): オペレータ op の $func_f$ を s_t，s_f を引数に呼び出す．
- applyS(op, s_t, s_f): オペレータ op の $func_s$ を s_t，s_f を引数に呼び出す．
- NewSemiElement(b): 文節 b を句・命題へ変換した結果を返す．
- NewOperator(s_t, s_f): セミ形式記述と文節のペア $s_t = (d_t, B_t)$ と $s_f = (d_f, B_f)$ を結合するためのオペレータを返す（なお，d_t と d_f は句・命題，またはセミ形式記述へ変換過程のデータ構造であり，d_f が d_t にかかる．B_t と B_f はそれぞれ d_t と d_f の変換元となった文節のリストである）．

Input:
b: 文末文節

Output:
d: 命題（命題プリミティブまたは二項演算オブジェクト）

```
 1: procedure Convert(b)
 2:     stack ← []
 3:     C ← children(b)
 4:     s_t ← (NewSemiElement(b), [b])      ▷ 文節のセミ形式記述変換ルール
 5:     s_f ← nil
 6:     while true do
 7:         if s_f = nil then
 8:             if |C| = 0 then
 9:                 break
10:             c ← pop(C)
11:             s_f ← (Convert(c), [c])
12:         op_{s_f} ← NewOperator(s_t, s_f)   ▷ セミ形式記述結合ルール
13:         if |stack| = 0 ∨ op_{s_f} is prior to last(stack) then
14:             push(stack, (op_{s_f}, s_f))
15:             s_f ← nil
16:             continue
17:         (op_last, s_last) ← last(stack)
18:         if applyS(op_last, s_f, s_last) ≠ nil then
19:             s_f ← applyS(op_last, s_f, s_last)        ▷ Conversion 1
20:             continue
21:         s_t ← applyF(op_last, s_t, s_last)            ▷ Conversion 2
22:         s_f ← nil
23:     for all (op, s) in reversed(stack) do
24:         s_t ← applyF(op, s_t, s)                      ▷ Conversion 3
25:     return d_t of s_t where s_t = (d_t, B_t)
```

本変換アルゴリズムは，変換ルールの蓄積を容易化するために，日本語の文節に対する直接の変換処理を担う変換処理部（Algorithm 1 の Conversion 1–3）と，Juman++/KNP から得た構文解析結果のデータをもとに変換処理を選択するルール部（Algorithm 1 の文節のセミ形式記述変換ルール，セミ形式記述結合ルール）とから構成される．変換処理の実体はオペレータと呼ぶ単位で変換アルゴリズムに組み込まれており，ルール部で選択したオペレータを日本語文に適用することで，セミ形式記述への変換が実現される．ルール部と変換処理部を分離して構成することで，直接の変換処理とは独立にルールを追加して漸進的に変換の正確さを改善可能にした．

変換処理を担うオペレータは，2 つのセミ形式記述を 1 つのセミ形式記述に結合する関数のペア $(func_s, func_f)$ であり，$func_s$，$func_f$ は次の記号を使って定義される関数 $func_s : s_t, s_f \mapsto s_r?$ と $func_f : s_t, s_f \mapsto s_r?$ である（「?」のついた変数「x?」は，x または nil の値を表すものとする）．

- $s_t = (d_t, B_t)$：結合先のセミ形式記述と文節リストのペア
- $s_f = (d_f, B_f)$：結合元のセミ形式記述と文節リストのペア
- $s_r? = (d_r, B_r)|\text{nil}$：$d_t$ と d_f が結合可能なときは結合結果のセミ形式記述と文節リストのペア，結合不可のときは nil
- B_t，B_f，B_r：それぞれ s_t，s_f，s_r に対応する文節オブジェクトのリスト
  - 文節オブジェクト：KNP の構文解析結果のオブジェクト．日本語文中の文節のテキスト表現に加え，日本語文中で何番目に出現するかや，係り受け構造が並列か否か，前述の feature データといった文法上のデータをプロパティとして有する．以降では文節オブジェクトのことを単に文節と呼ぶ．
- d_t，d_f，d_r：セミ形式記述のオブジェクト．句[*4]，命題，または変換過程のデータ構造（句の二項関係ペア結合オペレータなどで用いる）に相当する．
  - 句：文節か命題を要素として持つリスト
  - 命題：次のいずれか
    - 命題プリミティブ $= (w_{\text{verb}}?, w_{\text{subj}}?, w_{\text{obj}}?, W_{\text{constraints}}, neg)$：$w_{\text{verb}}$，$w_{\text{subj}}$，$w_{\text{obj}}$ はそれぞれ関係語・主体語・対象語の句，$W_{\text{constraints}}$ は制約語を表す句のリスト，neg は w_{verb} が否定表現であるとき true をとる真偽値．
    - 二項演算オブジェクト $= (biop, d_l, d_r)$：$biop$ は二項演算子の種類（「and」,「or」,「imply」），d_l，d_r はそれぞれ二項演算子の左，右オペランドに相当する命題．
  - （変換過程のデータ構造は，左記データ構造を利用するオペレータと合わせて説明をする）．

変換アルゴリズムは厳密には日本語文を上記セミ形式記述のオブジェクトへ変換する．上記オブジェクトは次のように文字列化することで図 1 の文法に沿ったテキスト表現と対応づくため，以降の説明ではセミ形式記述のオブジェクトのことも単にセミ形式記述と呼ぶ．

- 句：
  (1) 句の中に含まれる全文節をリスト化し，日本語文中での出現順にソートする．
  (2) 各文節を日本語文中の表記に従って文字列化し，文節どうしを結合する．
- 命題プリミティブ：
  (1) 句 $w_{\text{subj}}?$，$w_{\text{obj}}?$ が nil のときは「?」そうでなければ左記の句を文字列化したものを命題プリミティブの主体語・対象語に設定する．
  (2) 句のリスト $W_{\text{constraints}}$ 中の各要素を文字列化した後，各要素どうしをカンマで結合したものを制約語に設定する．
  (3) 句 $w_{\text{verb}}?$ が nil のときは「is」を関係語に設定し，そうでなければ
    (a) w_{verb} の句の中に含まれる全文節をリスト化し，日本語文中での出現順にソートする．
    (b) 最後の文節以外の各文節を日本語文中の表記に従って文字列化し，最後の文節の用言を終止形にする．
    (c) 文節どうしを結合して関係語に設定する．
  (4) $neg = true$ のときは関係語の先頭に「!」を付ける．
- 二項演算オブジェクト：
  (1) d_l，d_r を文字列化してそれぞれ二項演算子の左オペランド，右オペランドに設定する．
  (2) d_l が二項演算オブジェクト，かつ d_l の二項演算子 $biop_l$ が $biop$ よりも低優先度のとき，左オペランドを丸括弧で囲む（ただし二項演算子の優先順位は「and」>「or」>「imply」）．
  (3) d_r が二項演算オブジェクト，かつ d_r の二項演算子 $biop_r$ が $biop$ よりも低優先度のとき，右オペランドを丸括弧で囲む．
  (4) $biop$ を次の対応で文字列化する（「and」:「&」,「or」:「|」,「imply」:「–>」）．
  (5) 左右オペランドの間に $biop$ の文字列を置く．

$func_s$，$func_f$ は，それぞれ次の 2 種類の係り受け関係にある文節から変換されたセミ形式記述どうしを結合する．

(1) 兄弟文節（$func_s$）：Algorithm 1 の Conversion 1 に相当する．**図 4** の ① では，兄弟文節にあたる「沸騰モード中か」と「給湯 LOCK 中に」の間に Or の論理関係があり，この論理関係を反映した結合を実現する．
(2) 親子文節（$func_f$）：Algorithm 1 の Conversion 2，3

[*4] 命題プリミティブの各項（関係語・主体語・対象語・制約語）を本論文では句と呼ぶ．

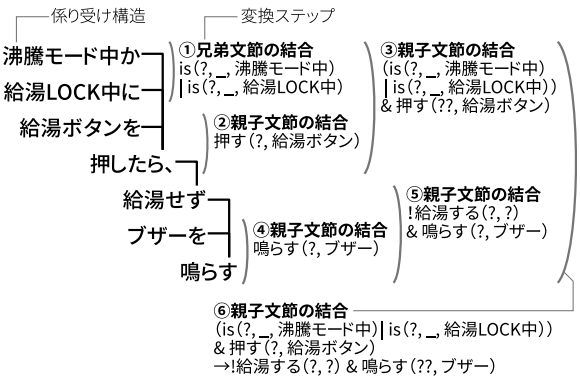


図 4 係り受けの構造の例と変換ステップ

Fig. 4 An example of dependency structure and conversion steps.

に相当する．図 4 の ② では，「押したら」を命題プリミティブとした「押す（?，?）」の対象語に，「給湯ボタンを」を結合している．

本アルゴリズムで定義したオペレータの一部を次にあげる．$func_s$，$func_f$ に相当する関数は，それぞれオペレータ名の末尾に「S」，「F」を付けた関数として表記する．紙面の節約のため，任意の入力で nil を返す関数の定義は省略する．結合の例では，表記を簡潔にするためにオペレータの引数のうち一部記号のみを記載する．例文中の「/」は，s_f，s_t に相当する部分を区切るための説明用の記号とし，「/」の前後からそれぞれ s_f，s_t が構成されるものとする．

句の結合オペレータ： 連体修飾による句の統合と，命題プリミティブの項への挿入を行う．

- アルゴリズム：

Definition:
- insertPhrase(d_t, d_f): 命題 d_t 中の命題プリミティブに句 d_f を挿入した結果を返す．表層格により主体・対象を判定し，ガ格の句を主体語に，ヲ格の句を対象語に，それ以外の句を制約語の位置に挿入する．判定の誤りはレビューで修正する．

procedure 句の結合オペレータ F(s_t, s_f)
  if d_f が句でない **then**
    return nil
  if d_t が句 **then**
    $d \leftarrow$ 句 ($[d_f, d_t]$)　▷ 句のインスタンス
    return (d, $B_f \cup B_t$)
  if d_t が命題 **then**
    $d \leftarrow$ insertPhrase(d_t, d_f)
    return (d, $B_f \cup B_t$)
  return nil

- 例 1：「赤色の/LED を……」
  - 入力：d_f：赤色の，d_t：LED
  - 出力：d_r：赤色の LED
- 例 2：「再沸騰ボタンを/押す」
  - 入力：d_f：再沸騰ボタンを，d_t：押す（?，?）
  - 出力：d_r：押す（?，再沸騰ボタン）

命題の結合オペレータ： 命題を二項演算子（And・Or・Imply）で結合する．

- アルゴリズム：

Definition:
- *biop*: 二項演算子（NewOperator で設定する）
- join(d_t, d_f, *biop*): $(biop, d_f, d_t)$ なる二項演算オブジェクトを返す．

procedure 命題の結合オペレータ S(s_t, s_f)
  return 命題の結合オペレータ F(s_t, s_f)
procedure 命題の結合オペレータ F(s_t, s_f)
  if d_f が命題でない **then**
    return nil
  if d_t が句 **then**
    $d \leftarrow$ 句 ($[d_f, d_t]$)　▷ 句のインスタンス
    return (d, $B_f \cup B_t$)
  if d_t が命題 **then**
    $d \leftarrow$ join(d_t, d_f, *biop*)
    return (d, $B_f \cup B_t$)
  return nil

- 例：「給湯ボタンを押すか/再沸騰ボタンを押すと，……」
  - 入力：d_f：押す（?，給湯ボタン），B_f：[給湯ボタンを，押すか]，d_t：押す（?，再沸騰ボタン），B_t：[再沸騰ボタンを，押すと]
  - 出力：d_r：押す（?，給湯ボタン）| 押す（?，再沸騰ボタン）

句の二項関係ペア結合オペレータ： 句を二項演算子とともにペアにする．

- アルゴリズム：

Definition:
- *biop*: 二項演算子（NewOperator で設定する）

procedure 句の二項関係ペア結合オペレータ S(s_t, s_f)
  if d_f と d_t が句 **then**
    return { 二項演算子：*biop*，句のペア：(d_f, d_t)}
  return nil
procedure 句の二項関係ペア結合オペレータ F(s_t, s_f)
  if d_f と d_t が句 **then**
    return { 二項演算子：*biop*，句のペア：(d_f, d_t)}　▷ 辞書形式（key:value）のデータ
  if d_f が句かつ d_t が命題 **then**
    $d \leftarrow$ insertPhrase(d_t, d_f)
    return (d, $B_f \cup B_t$)
  return nil

- 例：「給湯ボタンか/再沸騰ボタンを……」
  - 入力：d_f：給湯ボタンか，d_t：再沸騰ボタンを
  - 出力：d_r：{二項演算子：“or”，句のペア：(d_f, d_t)}

二項演算子を持つ句のペアの分割挿入オペレータ： 句のペアを命題プリミティブに分割して挿入する．

- アルゴリズム：

Definition:
- clone(d): 命題 d の複製を返す.

1: **procedure** 二項演算子を持つ句のペアの分割挿入オペレータ F(s_t, s_f)
2: **if** d_t が句 **then**
3: $d \leftarrow$ 句 ($[d_f, d_t]$) ▷ 句のインスタンス
4: **return** (d, $B_f \cup B_t$)
5: **if** d_t が命題 **then**
6: **if** d_f のデータ構造 $\neq$ { 二項演算子: $biop$, 句のペア: ($d_{f_{d_f}}, d_{t_{d_f}}$)} **then**
7: **return** nil
8: $d'_t \leftarrow$ clone(d_t)
9: $d \leftarrow$ insertPhrase(d_t, $d_{f_{d_f}}$)
10: $d' \leftarrow$ insertPhrase(d'_t $d_{t_{d_f}}$)
11: $d'' \leftarrow$ join(d', d, $biop$)
12: **return** (d", $B_f \cup B_t$)
13: **return** nil

- 例：「給湯ボタンか再沸騰ボタンを/押す」
  - 入力：d_f：{二項演算子："or"，句のペア：(給湯ボタンか，再沸騰ボタンを)}，d_t：押す（?，?）
  - 出力：d_r：押す（?，給湯ボタン）| 押す（?，再沸騰ボタン）

親文節スキップオペレータ： 親文節との結合をスキップする（「……場合」や「……とき」などの付帯的な状況を表現する体言に命題プリミティブが係るとき，命題の結合オペレータにより句とするのではなく，命題プリミティブとして維持するためのオペレータ）.

- アルゴリズム：

1: **procedure** 親文節スキップオペレータ F(s_t, s_f)
2: **return** (d_t, $B_f \cup B_t$)

- 例：「給湯ボタンを押す/とき，……」
  - 入力：d_f：押す（?，給湯ボタン），d_t: とき
  - 出力：押す（?，給湯ボタン）

文節の結合順序を制御するために，オペレータ間には優先順位を設定する．オペレータ間の優先順位は，アルゴリズム内に静的に設定されており，句の二項関係ペア結合オペレータ > 句の結合オペレータ > 二項演算子を持つ句のペアの分割挿入オペレータ > 親文節スキップオペレータ > 命題の結合オペレータである．本優先順位に従うオペレータの適用は，Algorithm 1 では，lines 13–17 で優先順位の高いオペレータを選択し，lines 18–21，23–24 でそれぞれ選択された方を適用することで実現されている．

命題の結合オペレータどうしは，文節内の読点の有無によってさらに優先度を付け，読点がない文節 > 読点がある文節とした．これにより，読点によって二項演算子の結合順序が順序付けられた「b_a かつ，b_b または b_c」のような文を「$b_a \wedge (b_b \vee b_c)$」と変換する．

ルール部について，Algorithm 1 の文節のセミ形式記述変換ルールでは 1 つの文節を句または命題プリミティブへ変換するルールを規定し，セミ形式記述結合ルールではオペレータの選択を行うルールを規定する．以下にルールの例をあげる．

文節のセミ形式記述変換ルール：

Input:
- b: 文節

1: **procedure** NewSemiElement(b)
2: **if** b が用言 **then**
3: $d \leftarrow$ 命題プリミティブ（verb=句（[b]））▷ 関係語に句（[b]）を持つ命題プリミティブのインスタンス
4: **if** b の feature に「否定表現がある」 **then**
5: (neg of d) $\leftarrow$ true ▷ 否定の命題プリミティブにする
6: **return** d
7: **if** b が「場合」,「時」,「中」のいずれかで終わるか，左記に助詞・句読点がついて終わる **then**
8: $d \leftarrow$ 命題プリミティブ(constraints=[句([b])]) ▷ 句（[b]）を制約の先頭に持つ命題プリミティブのインスタンス
9: **return** 句（[b]） ▷ 句のインスタンス

例：
- 文節「ボタンを」は句「ボタンを」に変換される．
- 文節「押す」は命題プリミティブ「押す（?，?）」に変換される．

セミ形式記述結合ルール：

Definition:
- GetBioperator(s_t, s_f): セミ形式記述と文節のリストのペア s_t, s_f から二項演算子決定ルールに従う二項演算子を返す（ただし，$s_t = (d_t, B_t)$, $s_f = (d_f, B_f)$, d_t, d_f はセミ形式記述の句・命題，B_t，Bf はそれぞれ d_t，d_f に対応する文節のリスト）.

Input:
- s_t, s_f: $s_t = (d_t, B_t)$, $s_f = (d_f, B_f)$. d_t, d_f はセミ形式記述の句・命題，または変換過程のデータ構造，B_t，B_f はそれぞれ d_t，d_f に対応する文節のリスト

Output:
- op: オペレータ

1: **procedure** NewOperator(s_t, s_f)
2: **if** d_f が命題 **then**
3: $biop \leftarrow$ GetBioperator(s_t, s_f)
4: **if** $biop$ = nil **then**
5: $biop \leftarrow$"and" ▷ デフォルト値．レビュー時に修正する．
6: **return** 命題の結合オペレータ（$biop$）▷ 命題の結合オペレータのインスタンス
7: **if** d_f が句 **then**
8: $biop \leftarrow$ GetBioperator(s_t, s_f)
9: **if** $biop$ = nil $\wedge$ B_f の末尾の文節が B_t 中の文節と並列な係り受け構造にある **then**
10: $biop \leftarrow$"and" ▷ デフォルト値
11: **if** $biop \neq$ nil **then**
12: **return** 句の二項関係ペア結合オペレータ（$biop$）
13: **return** 句の結合オペレータ（）

例：「赤色の/LED を……」

- 入力：s_f：(d_f：句「赤色の」，B_f：[赤色の])，s_t：(d_t：句「LED を」，B_t：[LED を])
- 出力：句の結合オペレータ

セミ形式記述結合ルール中に定義する二項演算子決定ルールの例をあげると次である．

二項演算子決定ルール：

```
1: procedure GetBioperator(s_t, s_f)
2:     b ← B_f の末尾の文節
3:     if b が接続助詞「か」∨b が接続助詞「または」 then
4:         return “or”
5:     if b が接続助詞「かつ」 then
6:         return “and”
7:     if b が用言 ∧ last(B_f) の活用形が条件形 then
8:         return “imply”
9:     if b の末尾が「場合」,「時」,「中」のいずれかで終
       わるか，左記に助詞・句読点がついて終わる then
10:        return “imply”
11:    return nil
```

変換ルールには，複数の文法データを組み合わせてオペレータの選択条件を記述する．オペレータの選択条件を厳密な論理関係で表現するために，本研究ではスクリプト言語 Python を用い，KNP の Python バインディングである PyKNP [*5] を用いて**図 5** に示すようにルールを記述する．

図 5 の `new_semi_element(b)` が Algorithm 1 の NewSemiElement(b) に相当する．`new_operator(dt, bt, ds, bs)` が NewOperator(s_t, s_s) に相当し，s_t = (`dt`, `bt`)，s_s = (`ds`, `bs`) である．`b` は PyKNP より得られる文節のデータ構造，`bt`，`bs` は文節のデータ構造のリストを表す．文節のデータ構造からは，`features` という属性から辞書の形式で文法データが取得できる（line 2）．`dpndtype` という属性からは係り受けの構造が取得でき（line 15），この値が`'P'` のとき，並列な係り受け構造を持つことを示す．`get_bioperator(dt, bt, ds, bs)` が二項演算子決定ルールに相当する．`last_mrph` は PyKNP より得られる形態素のデータ構造であり，`bunrui` という属性で形態素の分類が得られ，`midasi` という属性で形態素の見出し語が得られる．

3.2 自動変換されたセミ形式記述の修正支援

自然言語仕様書は入力項目・確認項目間の論理関係の曖昧さや誤りといった欠陥を含みうる．このため，自然言語仕様書から変換されたセミ形式記述の仕様書は，元仕様書に含まれていた論理関係の欠陥を引き継いでしまう．また，自然言語の仕様書からセミ形式記述への変換ルールの不足により，変換結果が誤っていることがある．これらの欠陥の修正にあたっては，セミ形式記述に記載された入力項目・確認項目間の論理関係を把握する必要がある．一方で，テキストベースで表現されるセミ形式記述から複雑な論理関係を把握することは困難である．

[*5] http://nlp.ist.i.kyoto-u.ac.jp/index.php?PyKNP

```
1  def new_semi_element(b):
2    if b.features.get('用言'):
3      ph = 句([b])
4      return 命題プリミティブ(verb=ph)
5    # 以降省略
6
7  def new_operator(dt, bt, dc, bc):
8    if isinstance(dc, 命題):
9      # 二項演算子決定ルール
10     op = get_bioperator(dt,bt,dc,bc)
11     op = 'and' if not op else op
12     return 命題の結合オペレータ(op)
13   if isinstance(dc, 句):
14     # 並列な係り受け構造
15     if bc[-1].dpndtype == 'P':
16       # 二項演算子決定ルール
17       op = get_bioperator(dt,bt dc,bc)
18       op = 'and' if not op else op
19       return 句の二項関係ペア結合オペレータ(op)
20   # 以降省略
21
22 def get_bioperator(dt, bt, dc, bc):
23   # 句読点を除く末尾の形態素を得る
24   mrph = get_last_mrph(bc[-1])
25   if (mrph.bunrui == '接続助詞' and
26       mrph.midasi in ('か', 'または')):
27     return 'or'
28   # 以降省略
```

図 5 Python での文節のセミ形式記述変換ルール，セミ形式記述結合ルール記述の例

Fig. 5 An example of conversion rules from bunsetsu to Semi-Formal Description and concatenation rules of Semi-Formal Description written in Python.

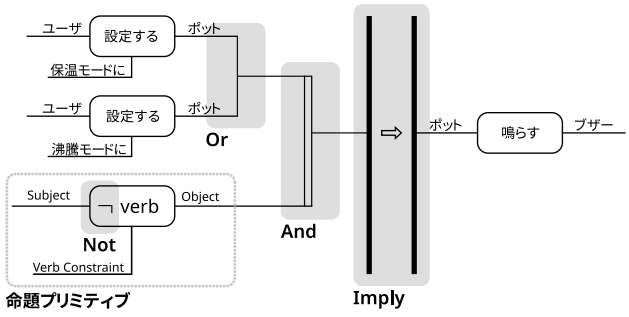


図 6 命題ネットワークの例

Fig. 6 An example of propositional network.

本研究では，命題間の論理関係の把握を支援するために，自動変換されたセミ形式記述を，本研究で開発した命題ネットワークと呼ぶ図式表現で可視化する（**図 6**）．図 6 では，連続する And，Or のオペランド（図 1 における expression）を並列に並べることで，命題のまとまりを可視化する．また，入れ子になった And，Or を階層的に図式化することで，階層的な論理構造を可視化する．

命題ネットワーク上でのレビューにより発見された入力項目・確認項目間の論理関係の誤りを元のセミ形式記述上で修正することによって，論理関係の誤りが除去されたセミ形式記述を得る．また，セミ形式記述では，機能の仕様を「入力項目→確認項目」と表現するため，「→」を持たない文については，入力項目・確認項目の欠落を補完し，機能を表現する Imply の形式のセミ形式記述へと修正する．

自然言語仕様書からセミ形式記述への変換アルゴリズムは，1 文単位で変換を行うため，2 文以上にまたがって記述される仕様（例：「A は，B である．ただし，C が D のときに限る．」）には対応していない．このため，1 つの仕様が 2 文以上のセミ形式記述に変換されることがある．命題ネットワーク上でのレビューでは，論理関係の誤り修正に加え，2 文以上に分かれている仕様の統合も行う．

セミ形式記述への変換は，ルールベースで行うので，変換結果の誤りには，元の仕様書と変換ルールの不備の 2 種類がありうる．修正にあたって，元の自然言語仕様書に含まれている論理関係がセミ形式記述に反映されていない場合（誤変換や欠落している場合）には，変換ルールの問題とし，変換ルールの正確さの向上のためにルールの追加・修正を行う．

4. セミ形式記述からのテストケース生成

本章では，セミ形式記述からテストケースを生成する際に，テスト設計者がテストする入力項目の組合せを見落とすことを防ぐために，セミ形式記述からテストケースを自動生成するアルゴリズムについて述べる．セミ形式記述からのテストケース生成は**図 7** のプロセスで実現され，デシジョンテーブル [5] 形式のテストケースを生成する．

セミ形式記述ではシステム仕様を命題論理の演算子を用いて表現する．したがって，テスト可能な入力項目の組合せは，機能の仕様（入力項目 → 確認項目）を満足する真理値割当てを求めることで得られる．入力項目・確認項目の真理値割当ての算出は，命題論理式を充足する真理値割当てを算出可能な PyEDA [6] を用いて実現する（図 7 の②）．③ で，入力項目の真理値割当てを条件欄，確認項目の真理値割当てを動作欄として埋め，デシジョンテーブルとして出力する．得られたデシジョンテーブルにはとりうる入力項目の組合せが網羅されるので，漏れのない入力項目の組合せでテストを実行できる．

図 7 セミ形式仕様書をデシジョンテーブル化する Data Flow Diagram（DFD）

Fig. 7 The data flow diagram (DFD) to convert Semi-Formal Specification to decision table.

また，膨大になりがちな入力項目の組合せへの対策として，Don't Care と呼ぶ，真と偽のいずれの値をとっても論理式全体の真偽に影響しないことを意味する割当て値を使い，テーブルサイズを縮小する．

本章で提案するアルゴリズムを用いると，図 4 から**表 1** に示すデシジョンテーブルが生成できる．

5. 評価

本章では，本論文で提案する日本語で書かれた仕様書からセミ形式記述への変換アルゴリズム，およびセミ形式記述からテストケースへの変換アルゴリズムの評価のために，ケーススタディによる詳細な変換ステップの評価と，仕様書に変換アルゴリズムを適用した場合の Precision と Recall による定量評価を行う．

5.1 ケーススタディ

ケーススタディでは，架空の電気ポットの仕様 3 文を用いて，次に示す項目を確認する．

(1) 変換ルールの追加により変換の正確さを改善できること

(2)「入力項目→確認項目」の形式で記述されたセミ形式記述がデシジョンテーブル形式のテストケースへ自動変換されること

上記項目の確認のために，次に示す架空の電気ポットの 3 つの仕様文を対象に変換を実施した．

Sentence A：入力項目・確認項目を 1 つずつを持つ仕様

日本語の仕様文：「給湯ボタンを押したら，お湯を排出する．」

対応するセミ形式の仕様文：「押す（?，給湯ボタン）–> 排出する（?，お湯）．」

Sentence B：論理関係に曖昧さを含む仕様

日本語の仕様文：「沸騰モード中に給湯ボタンを押すか，再沸騰ボタンを押したら，ブザーを鳴らしてお湯を排出しない．」

対応するセミ形式の仕様文：「is（?，_，沸騰モード中）

表 1 図 4 から生成したデシジョンテーブル

Table 1 A decision table generated from Fig. 4.

		テストケース				
		前件真		前件偽		
命題		1	2	3	4	5
条件	is (?, _, 沸騰モード中)	F	T	F	F	T
	is (?, _, 給湯 LOCK 中)	T	-	F	T	-
	押す (?, 給湯ボタン)	T	T	-	F	F
動作	給湯する (?, ?)	F	F	-	-	-
	鳴らす (?, ブザー)	T	T	-	-	-

表中の記号は，F: 真，T: 偽，-: Don't Care である．

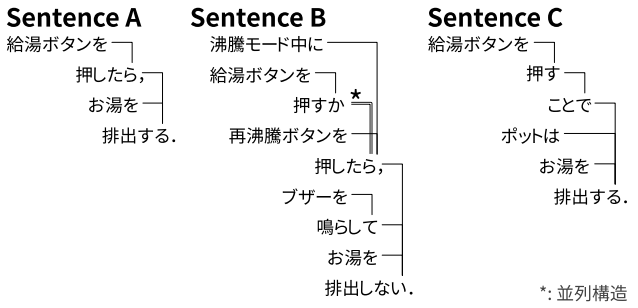


図 8 評価用の文の係り受け構造

Fig. 8 The modification structure of Japanese sentences for evaluation.

& (押す (?, 給湯ボタン) | 押す (?, 再沸騰ボタン)) –> 鳴らす (?, ブザー) & !排出する (?, お湯).」

Sentence C：ルールを追加しないと不正確な変換がなされる仕様

日本語の仕様文：「給湯ボタンを押すことで，ポットはお湯を排出する.」

対応するセミ形式の仕様文：「押す (?, 給湯ボタン) –> 排出する (ポット, お湯).」

上記機能の仕様を Juman++/KNP で構文解析を行うと，**図 8** のような係り受け構造となる．3.1 節のルールにより，Sentence A は次のステップで変換される．

(1)「排出する」が用言なので，文節のセミ形式記述変換ルールに従い，命題プリミティブ「排出する (?, ?)」へ変換 (Algorithm 1 line 4).

(2)「押したら」が用言なので，文節のセミ形式記述変換ルールに従い，命題プリミティブ「押す (?, ?)」へ変換 (Algorithm 1 line 4).

(3)「押したら」の子文節「給湯ボタンを」が体言なので，文節のセミ形式記述変換ルールに従い，句「給湯ボタンを」に変換 (Algorithm 1 line 4).

(4)「給湯ボタンを」が句なので，セミ形式記述結合ルールに従い，句の結合オペレータを選択 (Algorithm 1 line 12).

(5) 句の結合オペレータにより，句「給湯ボタンを」を命題プリミティブ「押す (?, ?)」と結合し，命題プリミティブ「押す (?, 給湯ボタン)」を生成 (Algorithm 1 line 24).

(6) 命題プリミティブ「押す (?, 給湯ボタン)」が命題なので，セミ形式記述結合ルールに従い，命題の結合オペレータを選択 (Algorithm 1 line 14).

(7)「お湯を」が体言なので，文節のセミ形式記述変換ルールに従い，句「お湯を」に変換 (Algorithm 1 line 4).

(8)「お湯を」が句なので，セミ形式記述結合ルールに従い，句の結合オペレータを選択 (Algorithm 1 line 12).

(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，句「お湯を」を命題プリミティブ「排出する (?, ?)」と結合し，命題プリミティブ「排出する (?, お湯)」を生成 (Algorithm 1 line 24).

(10) 命題の結合オペレータにより，命題プリミティブ「押す (?, 給湯ボタン)」を命題プリミティブ「排出する (?, お湯)」と結合し，命題プリミティブ「押す (?, 給湯ボタン) –> 排出する (?, お湯)」を生成 (Algorithm 1 line 24).

Algorithm 1 に従うと，Sentence B，Sentence C は，次のように変換される．

Sentence B：「is (?, _, 沸騰モード中) & 押す (?, 給湯ボタン) | 押す (?, 再沸騰ボタン) –> 鳴らす (?, ブザー) & !排出する (?, お湯).」

Sentence C：「排出する (ポット，お湯，給湯ボタンを押すことで).」

Imply の形式で記述されている Sentence A，Sentence B のセミ形式記述をデシジョンテーブルに変換した結果を**表 2**，**表 3** に示す．

表 2 Sentence A のセミ形式記述から生成したデシジョンテーブル

Table 2 A decision table generated from the Semi-formal Description of sentence A.

		テストケース	
		前件真	前件偽
	命題	1	2
条件	押す (?, 給湯ボタン)	T	F
動作	排出する (?, お湯)	T	-

表 3 Sentence B のセミ形式記述から生成したデシジョンテーブル

Table 3 A decision table generated from the Semi-formal Description of sentence B.

		テストケース				
		前件真			前件偽	
	命題	1	2	3	4	5
条件	is (?, _, 沸騰モード中)	F	T	T	F	T
	押す (?, 給湯ボタン)	-	F	T	-	F
	押す (?, 再沸騰ボタン)	T	T	-	F	F
動作	鳴らす (?, ブザー)	T	T	T	-	-
	排出する (?, お湯)	F	F	F	-	-

5.2 セミ形式記述への変換アルゴリズムの定量評価

定量評価のために，電気ポットの仕様書 [7] の 1 章–7 章のうち，図表，数式を含む文を除く 98 文を対象にセミ形式記述への変換を実施した．本論文の著者 1 名による手作業での変換結果を正解データとし，変換結果を比較して，命題プリミティブの抽出・論理関係の同定のそれぞれについて，Precision (式 (1)) と Recall (式 (2)) を求めた．

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2)$$

命題プリミティブの抽出結果における TP（True Positive），FP（False Positive），FN（False Negative）は，98 文の仕様文をそれぞれ S_i（$1 \leq i \leq 98$）とし，変換アルゴリズムによる S_i の変換結果に含まれる命題プリミティブの集合を P_{a_i}，人手による S_i の変換結果に含まれる命題プリミティブの集合を P_{m_i} として，それぞれ次にあたるものを $1 \leq i \leq 98$ にわたって集計した．

- TP：$p \in P_{a_i} \wedge p \in P_{m_i}$ となる p の数
- FP：$p \in P_{a_i} \wedge p \notin P_{m_i}$ となる p の数
- FN：$p \notin P_{a_i} \wedge p \in P_{m_i}$ となる p の数

論理関係の同定における TP，FP，FN は，変換アルゴリズムによる S_i の変換結果に含まれる論理演算子（Not・And・Or・Imply）の集合を O_{a_i}，人手による S_i の変換結果に含まれる論理演算子の集合を O_{m_i}，ある命題 x に含まれる関係語の集合を W_x，x に単項演算子がついているかを表す関数を has_not(x)，$x \neq y$ なる命題 y との論理関係（And・Or・Imply）を $r_{x,y}$ で表すとき，それぞれ次にあたるものを $1 \leq i \leq 98$ にわたって集計した．

- TP：次の項目の合計
  - $(x \in P_{a_i} \wedge y \in P_{m_i}) \wedge (W_x = W_y \wedge \text{has\_not}(x) \wedge \text{has\_not}(y))$ を満たす x の数
  - $(x, y \in P_{a_i} \wedge x \neq y) \wedge (t, u \in P_{m_i} \wedge t \neq u) \wedge (W_x = W_t \wedge W_y = W_u) \wedge (r_{x,y} = r_{t,u} \wedge r_{x,y} \in O_{a_i} \wedge r_{t,u} \in O_{m_i})$ を満たす $r_{x,y}$ の数
- FP：次の項目の合計
  - $(x \in P_{a_i} \wedge y \in P_{m_i}) \wedge (W_x = W_y \wedge \text{has\_not}(x) \wedge \neg\text{has\_not}(y))$ を満たす x の数
  - $(x, y \in P_{a_i} \wedge x \neq y) \wedge (t, u \in P_{m_i} \wedge t \neq u) \wedge (W_x = W_t \wedge W_y = W_u) \wedge (r_{x,y} \in O_{a_i} \wedge r_{t,u} \in O_{m_i} \wedge r_{x,y} \neq r_{t,u})$ を満たす $r_{x,y}$ の数
  - $\max(|\{o|o \in O_{a_i} \wedge o \neq \text{Not}\}| - |\{o|o \in O_{m_i} \wedge o \neq \text{Not}\}|, 0)$
- FN：次の項目の合計
  - $(x \in P_{a_i} \wedge y \in P_{m_i}) \wedge (W_x = W_y \wedge \neg\text{has\_not}(x) \wedge \text{has\_not}(y))$ を満たす x の数
  - $(x, y \in P_{a_i} \wedge x \neq y) \wedge (t, u \in P_{m_i} \wedge t \neq u) \wedge (W_x = W_t \wedge W_y = W_u) \wedge (r_{x,y} \in O_{a_i} \wedge r_{t,u} \in O_{m_i} \wedge r_{x,y} \neq r_{t,u})$ を満たす $r_{x,y}$ の数
  - $\max(|\{o|o \in O_{m_i} \wedge o \neq \text{Not}\}| - |\{o|o \in O_{a_i} \wedge o \neq \text{Not}\}|, 0)$

集計の結果，人手での変換結果に含まれる 212 個の命題プリミティブ，128 個の論理演算子についての変換の Precision と Recall は**表 4** に示すとおりとなった．

表 4 セミ形式記述への変換の Precision と Recall
Table 4 Precision and Recall of the conversion of Semi-formal Description.

	TP	FP	FN	Precision	Recall
命題プリミティブの抽出	147	65	62	0.69	0.70
論理関係の同定	103	23	26	0.82	0.81

6. 考察

6.1 ケーススタディの結果について

本節では，5 章の評価結果について考察する．

まず，セミ形式記述により元の日本語文中の論理関係の曖昧さが除去されたことを示す．5 章の Sentence A をテストする際には，機能の入力項目として「給湯ボタンを押す」という操作を行い，確認結果として「お湯を排出する」ということを確認する必要がある．入力項目・確認項目をそれぞれ 1 つずつ持つ Sentence A は，論理演算子 Imply を使って，「押す（?，給湯ボタン）–> 排出する（?，お湯）.」のように変換された．変換されたセミ形式記述は，元の仕様文にある「給湯ボタンを押す」と「お湯を排出する」という入力項目・確認項目を反映し，それぞれ Imply の前件・後件に期待どおりに記述された．入力項目・確認項目は，Imply の前件・後件を見ることで一意に把握できる．

ここから，入力項目・確認項目をそれぞれ 1 つずつ持つ最も単純で形式的な仕様文がセミ形式記述で表現でき，論理関係の曖昧さなく Imply の前件・後件から入力項目・確認項目を把握可能なことを確認した．

続いて論理関係に曖昧さを含んだ仕様文である 5 章の Sentence B について考察する．Sentence B では，付帯状況を表す「沸騰モード中」という表現が，「給湯ボタンを押すとき」と「再沸騰ボタンを押すとき」の両方を制約するのか，あるいは「給湯ボタンを押すとき」のみを制約するのかが曖昧になっている．解釈可能な論理関係を命題ネットワークで表現すると，**図 9** の 2 通り存在する．

解釈 B からデシジョンテーブルを生成すると，**表 5** になる．解釈 A から生成された表 3 の 1 列目のテストケースと表 5 は 3 列目のテストケース（太字部分）は，ともに入力項目の「is（?，_，沸騰モード中）」が偽であるが，確認項目が異なっている．セミ形式記述では機能を Imply で表現し，入力項目が成立しないとき（前件偽のとき）の確認項目は真偽が不定であることを示すために don't care で表現する．Sentence B の入力項目が不成立のときに確認項目が成立しない場合，解釈 A の 1 列目では，確認項目が成立することを確認するテストケースとなるが，一方，解釈 B の 3 列目は確認項目が成立しないことを確認するテストケースとなる．解釈 A・解釈 B のいずれか誤った解釈のもとでテストケースを作成した場合，上記項目のテスト

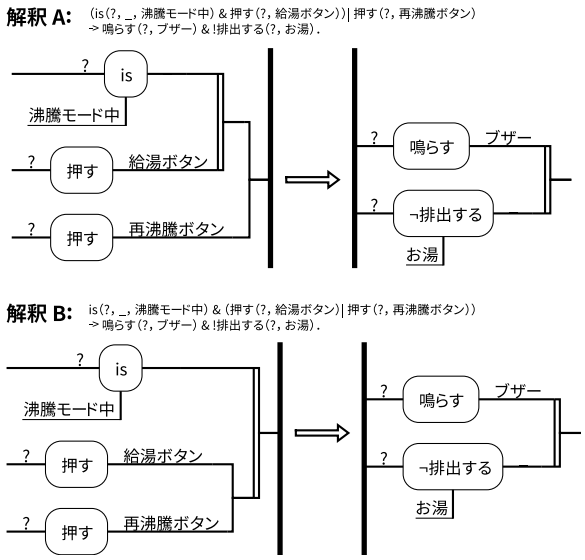


図 9 5 章の Sentence B の解釈
Fig. 9 Interpretations of sentence B in section 5.

表 5 図 9 の解釈 B から生成したデシジョンテーブル
Table 5 A decision table generated from the Semi-formal Description of interpretation B of Fig. 9

		テストケース			
		前件真		前件偽	
	命題	1	2	**3**	4
条件	is（?, _, 沸騰モード中）	T	T	**F**	T
	押す（?, 給湯ボタン）	F	T	-	F
	押す（?, 再沸騰ボタン）	T	-	-	F
動作	鳴らす（?, ブザー）	T	T	-	-
	排出する（?, _, お湯）	F	F	-	-

ケースでは不具合を見落としてしまう．

図 9 では，解釈 A では「is（?, _, 沸騰モード中）& 押す（?, 給湯ボタン））」，解釈 B では「(押す（?, 給湯ボタン）| 押す（?, 再沸騰ボタン））」と異なる命題プリミティブがペアになっていることが可視化されており，セミ形式記述上でも各解釈ごとに異なる論理関係となっている．ここから，セミ形式記述で表現することで論理関係の曖昧さが除去され，一意に解釈できる仕様が記述されている．このセミ形式記述上で正しい解釈に修正し，テストケースを生成することで，論理関係の解釈の誤りによって誤ったテストケースを作成することを防げる．

表 2，3 のデシジョンテーブルは，4 章の方法により自動生成された．これにより，セミ形式記述上で一意に解釈可能となった機能の仕様から人為的な操作ミスによる誤りの混入なく，テストケースを生成できる．

次に，ルールの追加により Sentence C の日本語の仕様文をセミ形式記述へ変換可能となることを示す．Sentence C のセミ形式記述への誤変換は，3.1 節のセミ形式記述結合ルール中に，「用言 p+ことで」という表現について，p を命題プリミティブに変換するルールがないために起こった．「給湯ボタンを」は体言であり，かつ「押す」は用言であるため，両者は文節のセミ形式記述変換ルールにより，句「給湯ボタンを」と命題プリミティブ「押す（?, ?)」となる．句「給湯ボタンを」はセミ形式記述結合ルールにより句の結合オペレータが選択されることから，句「給湯ボタンを」と命題プリミティブ「押す（?, ?)」の結合の結果までは「押す（?, 給湯ボタン)」という命題プリミティブに変換される．

一方，「押す（?, 給湯ボタン)」の親文節「ことで」は体言のため句となっており，ここに命題プリミティブの「押す（?, 給湯ボタン)」が命題の結合オペレータで結合されることで，「『押す（?, 給湯ボタン)』＋『ことで』」の句となってしまう．この句が「ポットはお湯を排出する」の部分から生成された命題プリミティブ「排出する（ポット，お湯)」と結合されて，誤ったセミ形式記述「排出する（ポット，お湯，給湯ボタンを押すことで)」となってしまう．

上記の誤った変換をルール追加により改善可能なことを示す．「給湯ボタンを押すことで」を，「押す（?, 給湯ボタン）->」と変換するためには，「用言+ことで」の結合において，次を実現するルールが必要となる．

(1)「用言」部分から生成される命題プリミティブ p を維持すること
(2)「ことで」を二項演算子決定ルールで Imply として解釈すること

これはセミ形式記述結合ルールと二項演算子決定ルールに次のようなルールを追加することで実現できる．

- 追加のセミ形式記述結合ルール：

```
Definition:
Output:
1: procedure NewOperator(s_t, s_f)
2:     if d_f が命題プリミティブであり，d_t が「ことで」
       という文節から構成される句である then
3:         return 親文節スキップオペレータ（）
```

追加の二項演算子決定ルール：

```
1: procedure GetBioperator(s_t, s_f)
2:     if B_f の末尾の文節が「ことで」である then
3:         return "imply"
```

上記のルールの追加により，変換アルゴリズムにより対応可能な日本語文を拡張することができる．ここから，提案する変換アルゴリズムが，変換ルールの追加により，漸進的に変換の正確さを改善できることを示した．

以上，ルールベースの変換アルゴリズムにより，日本語の仕様文をセミ形式記述に変換することで日本語の仕様文の論理関係の曖昧さを除去できることを示した．また，変換されたセミ形式記述により，システム仕様からテスト

ケースへの人為的な誤りの混入を防ぐ自動的なテストケース生成が実現されていることを示した．

6.2 定量評価の結果について

本節では，5.2 節の結果について考察する．

表 4 では，提案するセミ形式記述への変換アルゴリズムにより，命題プリミティブの抽出・論理関係の同定それぞれについて過半数が正解データと一致した．以降では変換が正しくなされなかったものについて考察する．

自動変換の結果と人手による変換結果を比較すると，次の項目に起因する変換誤りが存在した．

(1) 変換ルール・オペレータの不足による誤り
  (a) 2 文節からなる複合語を 1 つの句にまとめるルール・オペレータが存在しない．
  (b) 論理関係の同定ルールが不足している．
(2) 論理関係の結合順序が誤っている．
(3) Juman++/KNP による構文解析結果の誤り

(1) の誤りは，本評価時点でのルール・オペレータの不足に基づくものであるため，次のようにルール・オペレータを追加することで解決することができる．

(1a) の例は，「システム全体と/して，/以下の/動作仕様を/満たさなければなりません.」という文が，「する（?, ?, システム全体と）& 満たす（?, 以下の動作仕様）.」と変換されたものである（「/」は文節区切りを表す）．正解データでは，「満たす（?, 以下の動作仕様 システム全体として）.」のように，「として」を 1 まとまりで英単語の「as」に相当する意味を表現した．一方，変換アルゴリズムの結果では「と」と「して」をまとめて複合語として扱うルール・オペレータがないため，「do with a whole system」における「do with」という意味に誤変換された．

この誤りは，次のようなオペレータを変換アルゴリズムに追加することによって解決できる．

```
1: procedure 複合語結合オペレータ F(s_t, s_f)
2:     d ← 句 ([d_f, d_t])                ▷ 句のインスタンス
3:     return (d, B_f ∪ B_t)
```

本オペレータを命題の結合オペレータよりも高い優先順位に設定することで，「して」と別の命題を結合する前に，複合語の句として結合することができる．

このオペレータの追加により，次のようなルールをセミ形式記述結合ルールの先頭に追加することで，本例のような複合語に対応できる．

```
Output:
    • op: オペレータ
1: procedure NewOperator(s_t, s_f)
2:     if B_f 末尾の文節が「と」で終わる ∧ B_t 先頭の文節が「して」で始まる then
3:         return 複合語結合オペレータ ()
```

(1b) の例は，「ロック中は，/給湯ボタンを/押しても/水は出ません.」という文が，「is（?, _, ロック中）-> 押す（?, 給湯ボタン）& !出る（水，?）.」と変換されたものである．正解データとしては，「is（?, _, ロック中）& 押す（?, 給湯ボタン）-> !出る（水，?）.」のように，「is（?, _, ロック中）」と「押す（?, 給湯ボタン）」が Imply で結合され，「押す（?, 給湯ボタン）」と「!出る（水，?）」が And で結合されるものであった．これらは，二項演算子決定ルールの不足によるものであり，「is（?, _, ロック中）」と「押す（?, 給湯ボタン）」が Imply で結合された誤りは，「ロック中は」のように，「中」を含む文節のみから二項演算子を判断していたことが原因であり，「押す（?, 給湯ボタン）」と「!出る（水，?）」が And で結合されたものは，「用言連用形 + も」という表現について Imply と解釈するルールがなく，二項演算子のデフォルト値の And が採用されたことが原因である．これらは，次のようなルールを二項演算子決定ルールの先頭に追加することで解決できる．

```
1: procedure GetBioperator(s_t, s_f)
2:     b_f ← B_f の末尾の文節
3:     b_t ← B_t の末尾の文節
4:     if b_f が用言 + 副助詞「も」 then
5:         return “imply”
6:     if b_t が用言 + 副助詞「も」∧ b_f の末尾が「場合」,「時」,「際」,「中」のいずれかで終わるか，左記に助詞・句読点がついて終わる then
7:         return “and”
```

(2) の例は，「温度制御可能な/水位ならば/沸騰状態に/移行し，/ポット内の/水を/加熱します.」という文が，「(is（?, _, 温度制御可能な水位）-> 移行する（?, ?, 沸騰状態に)) & 加熱する（?, ポット内の水）.」と変換されたものである．本例は論理演算子の結合順序が誤っており，正解データとしては，「is（?, _, 温度制御可能な水位）-> 移行する（?, ?, 沸騰状態に）& 加熱する（?, ポット内の水）.」と変換されるべきものであった．自然言語で書いた文章では論理関係の結合順序を厳密に定義することができないため，変換結果には本例のような解釈の誤りが生じうる．この解消には人手による意味の解釈が必要なため，命題ネットワーク上でのレビュー時に修正する．

(3) の誤りは，「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.」について，「is（?, _, タイムアウト時）& is（?, _, 設定したタイマの沸騰状態終了時）-> 鳴らす（ユーザ, ブザー，3 回）.」と変換されたものである．正解データは，「is（?, _, タイムアウト時）& is（?, _, ユーザが設定したタイマの沸騰状態終了時）-> 鳴らす（?, ブザー，3 回）.」と設定した．「ユーザ」は Juman++/KNP による構文解析の結果，「鳴らします」に係ると判断されるため，変換アルゴリズムによる変換の結果では，「ユーザ」が誤って「鳴ら

す（ユーザ，ブザー，3 回）」の主体語とされてしまった．

本変換アルゴリズムでは，構文解析の結果をもとにルールを構築するため，構文解析自体が誤っている場合，提案する変換アルゴリズムでは回避ができない．このため，構文解析の失敗による変換誤りについては変換結果のレビュー時に手動での修正を要する．

本評価で用いた 98 文の仕様文では，14 文に構文解析結果の誤りが存在した．構文解析結果の誤りのある文を除くと，命題プリミティブの抽出については Precision 0.80，Recall 0.82，論理関係の同定については Precision 0.82，Recall 0.86 でセミ形式記述への変換が実現されることを確認した．

以上，提案する変換アルゴリズムを仕様書に適用したところ，変換の Precision と Recall を求めた．Precision と Recall を悪化させる原因について考察し，この原因を解消するルール・オペレータ，および本アルゴリズムでは対応できない原因とその理由について考察した．

7. 関連研究

本研究におけるセミ形式記述の元となった文献 [1] は，自然言語で書かれた仕様から，曖昧さを除去するための修正のガイドラインを提示する．文献 [1] では，自然言語の仕様の各操作を命題プリミティブのような中間表現——主体語や対象語などの記述すべき語を明確化した中間表現——へ変換した後に，主体語などの不足語を検出し，ユーザによる修正・補完を促す．修正・補完された中間表現を自然言語の表現に戻すことで，最終成果物として曖昧さが除去された自然言語の仕様記述を得る．

本研究では，最終成果物としてテストケースを得ることを目的とする．このため，本研究では，自然言語の仕様書からセミ形式記述として抽出した各操作を自然言語ではなくテストケースへの変換を行うための文法拡張を行った．具体的には，命題プリミティブ間の論理関係を命題論理により表現する拡張を行い，論理の明確化と，真理値割当てによるデシジョンテーブルの自動生成を可能とした．

自然言語の曖昧さを排して仕様記述を実現する手法として，Z 記法 [8] や VDM++ [9] といった仕様の形式記述手法がある．これらは厳密な仕様記述の記法を提供し，記述した仕様に対する自動検証機能も提供する．一方で，これらの形式記述言語による仕様記述のためには，形式記述手法の習得と自然言語仕様書自体の書き換えが必要となる．

本研究で提案するセミ形式記述は，自然言語仕様を置き換えるのではなく，曖昧さを排した仕様解釈支援のための中間表現として位置付ける．このため，仕様書は従来どおり自然言語で書かれるものとして，自然言語仕様書からセミ形式記述への変換パスを提供する．セミ形式記述の記法には元の日本語仕様書の表現を流用することで習得の容易化を図りつつ，命題論理の演算子を用いて仕様を記述することで曖昧さを除去した仕様記述との両立を図った．

日本語文を論理式に変換する手法として，文献 [10] が提案されている．文献 [10] は日本語文の知識表現のために，日本語の文を一階述語論理を拡張した論理式へ変換する．一方，本研究では，最終的にテストケースを生成することを目的とした論理式変換を行う．したがって，たとえば「Condition 中に Action する」のような表現において，テストの入力項目を取り出す本研究では，「Condition」を機能の入力項目，「Action する」を機能の確認項目として，テストの入力・確認に必要な異なる項目として分けて取り出す．一方，文献 [10] では，述語「Action する」の知識表現のために，「Condition 中に」は「Action する」の項として併合される．

本論文での提案手法と同じく，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文献 [11] がある．文献 [11] では，日本語の仕様文から句を抽出し，句がデシジョンテーブル中の条件にあたるのか，動作にあたるのかを自動的に判定することができる．一方で，句の間に存在する論理関係の同定や，句の真理値割当てを求める手法については提案されていない．対して本研究では，日本語文中からテスト設計者が入力・確認すべき各項目として命題プリミティブを抽出し，命題プリミティブ間の論理関係の同定も行う．デシジョンテーブル生成時には，命題プリミティブ間の論理関係を反映した真理値割当ても自動的に行うため，日本語の仕様文の誤った論理関係解釈から不適当なデシジョンテーブルを作成することを防げる．

8. まとめと今後の課題

自然言語で記述される仕様書では，論理関係解釈の曖昧さが原因で誤ったテストケースが作成されることがあるという課題がある．

本研究では，曖昧さを含んだ仕様書をもとにしたテストケース設計による誤ったテストケース作成を防ぐために，日本語の仕様文をセミ形式記述に変換するアルゴリズムを開発した．変換アルゴリズムはルールベースのアーキテクチャを採用し，ルールの蓄積により漸進的に変換の正確さを改善可能にした．また，セミ形式記述で記述された仕様から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，人為的なテストケースへの変換誤りのないテストケース生成を実現した．

本変換アルゴリズム，およびデシジョンテーブルの自動生成アルゴリズムについて，架空の電気ポット仕様に適用したところ，セミ形式記述へ変換することで元の仕様書の論理関係の曖昧さが除去されること，セミ形式記述の入力項目・確認項目を反映するテストケースが自動生成されることを示した．加えて，変換処理から独立したルール追加により変換の正確さを改善可能なことを確認した．また，

変換アルゴリズムを 98 文の電気ポット仕様文に適用したところ，Precision 0.70，Recall 0.81 という結果を得た．

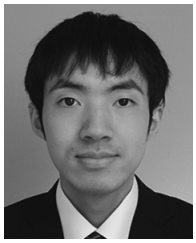
本論文では，自然言語の文章として記述された仕様からセミ形式記述へ変換するアルゴリズムを述べた．本変換アルゴリズムは日本語構文解析器による係り受け・格の情報をもとに，1 文ずつセミ形式記述へと変換を行う．このため，本アルゴリズムが適用可能な仕様の表現は，文節のセミ形式記述変換ルールによって命題プリミティブに変換される文節を 1 つ以上含む日本語文であり，かつ仕様が 1 文で完結している必要がある．2 文以上にまたがる仕様については，現在は命題ネットワーク上でのレビュー時に合わせて統合・修正することを想定している．

また，係り受けや格の情報の得られない表現，たとえば単語をリスト記述するような表現（例：「右記条件で XXX を実行：aaa，bbb，ccc」）や，表や図式など自然言語の文章以外の方法で形式化された仕様についても対応しない．このため，これら表現を含む仕様書については，前処理として文章形式の自然言語表現に書き直したり，形式的に表現された表・図式からセミ形式記述へ変換したりするステップが別途必要になる．上記表現からの変換については今後の課題である．

謝辞 本研究は，JSPS 科研費 16K00100，16H01836 の一部支援を受けたものである．

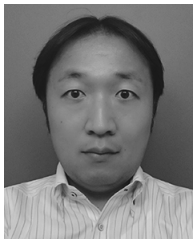
参考文献

[1] Rolland, C. and Achour, C.B.: Guiding the construction of textual use case specifications, *Data & Knowledge Engineering*, Vol.25, No.1-2, pp.125–160 (1998).
[2] Tolmachev, A., Kawahara, D. and Kurohashi, S.: Juman++: A Morphological Analysis Toolkit for Scriptio Continua, *Proc. 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp.54–59 (2018).
[3] Kawahara, D. and Kurohashi, S.: A fully-lexicalized probabilistic model for Japanese syntactic and case structure analysis, *Proc. 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).
[4] 河原大輔：KNP で付与される feature 一覧 (2013), available from ⟨http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP⟩（参照 201-05-31）.
[5] Beizer, B.: *Software testing techniques, second edition*, pp.269–276, Dreamtech Press (2003). 小野間彰，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).
[6] Drake, C.: Python library for Electronic Design Automation (PyEDA) (2011), available from ⟨https://media.readthedocs.org/pdf/pyeda/v0.28.0/pyeda.pdf⟩ (accessed 2018-05-23).
[7] 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 6 版 (2018)，入手先 ⟨http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm⟩（参照 2018-05-23）.
[8] Spivey, J.M. and Abrial, J.: *The Z notation*, Prentice Hall Hemel Hempstead (1992).
[9] Diirr, E. and Van Katwijk, J.: VDM++: A Formal Specification Language for Object Oriented Designs, *Proc. 6th Annual European Computer Conference, Compeuro*, pp.214–219 (1992).
[10] 高柳俊祐，上條敦史，石川 勉：日本語文から拡張型述語論理式への自動変換ツール：CONV，人工知能学会論文誌，Vol.27, No.5, pp.271–280 (2012).
[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（オンライン），DOI: 10.11309/jssst.34.2_16 (2017).



青山 裕介 （学生会員）

2016 年九州工業大学大学院情報工学府博士前期課程修了．同年同大学院博士後期課程に進学．非決定性を含むシステムのテスト技術の研究に従事．



黒岩 丈瑠 （学生会員）

2005 年東京大学大学院情報理工学系研究科修士課程修了．2005 年三菱電機（株）入社．現在，三菱電機（株）で業務用空調機器の開発に従事するとともに，九州工業大学大学院情報工学府博士後期課程で，組み込みシステムのテスト技術の研究に従事．

久代 紀之

三菱電機（株）を経て，現在，九州工業大学大学院情報工学研究院情報創成工学研究系教授．東京大学大学院工学系研究科修了．博士（工学）．住宅，ビル，工場等の環境制御システムを構築するための要素技術を専門とする．