DEIM Forum 2022 K43-2(day4 p40)

表記揺れを考慮した同義な命題論理式の判別

金子　卓矢†　　横山　昌平††,†††

† 東京都立大学システムデザイン学部　〒 191–0065 東京都日野市旭が丘 6-6
†† 東京都立大学大学院システムデザイン研究科　〒 191–0065 東京都日野市旭が丘 6-6
††† 東京大学生産技術研究所　〒 153–8505 東京都目黒区駒場 4-6-1
E-mail: †kaneko-takuya@ed.tmu.ac.jp, ††shohei@tmu.ac.jp

あらまし　論理演算はプログラミングやディジタル回路などの分野で用いられる．したがって，情報科学において論理学の考えは重要である．論理学の分野の一つである命題論理は，命題変数，論理記号，括弧からなる命題論理式で表現される．大学の論理学の授業では，小テストや最終試験などが行われている．しかし，命題変数は受講者が自由に定義できるため，同じ意味の論理式でも使われる変数は様々である．また，同値な式でも表現の方法は複数ある．そのため，採点者が一つ一つ確認を行うのは時間がかかってしまう．そこで，本研究では変数の置き換えを除いて論理的同値であることを同義として，命題論理式が同義であるかの判別や，標準形であるかの判別を自動で行うことで採点の補助を行うシステムの作成を行った．

キーワード　評価支援システム，論理学，命題論理，パーサ，構文木

1　はじめに

数理論理学は情報科学の分野において重要な役割を持っている．例えば，プログラミングやディジタル回路などでは論理演算が用いられている．また，ソフトウェア開発において「形式手法」という手法が用いられている．形式手法とは，ソフトウェアやハードウェアの仕様，設計，および検証の数学的に厳密な手法とツールである．数学的に厳密とは，情報システムが要件を満たしているかを論理的に推論することである．

数理論理学の分野の一つの命題論理は複雑な命題の構造を解析しその審議をどのように定めるかを研究する数学の分野である [1]．命題論理では命題変数，論理記号，括弧からなる有限の長さの文字列である命題論理式が用いられる．

大学ではプログラミングの基盤となる論理学の講義が行われている．講義では，命題論理，述語論理，ブール代数などを学ぶ．論理式は同じ意味でも表現の方法が異なる場合がある．例えば，式 (1) と式 (2) は使われている記号は異なるが意味は同じである．

$$A \wedge B \vee C \tag{1}$$

$$X \text{ and } Y \text{ or } Z \tag{2}$$

式の表し方は様々なため，試験の採点をするときは表記揺れを考慮する必要がある．表記揺れにはいくつか種類があり，一つは論理的同値だが異なる式である．例えば式 (3) と式 (4) は表現は異なるが論理的同値な式である．

$$\neg(A \wedge B) \tag{3}$$

$$\neg A \vee \neg B \tag{4}$$

次に，変数の表記揺れがある．命題変数は受講生が自分で定義することが可能なため，同じ問題でも使われる変数は答案ごとに異なる．例として，式 (5) と式 (6) は使われている変数は異なるが，P と X，Q と Y がそれぞれ同じ意味を表しているならば，2 つの式は同義であるといえる．

$$P \to Q \tag{5}$$

$$X \to Y \tag{6}$$

さらに，演算子についても表記揺れが発生する．論理学において AND の演算子は「∧」が用いられているが，学生が提出する答案には「&」や「AND」が用いられる場合がある．そのため，人が記述する式には自然言語のように表記揺れや書き間違えなどが発生し，採点者はそのような式が正解か不正解かを一つ一つ確認していくのに時間がとられる．また，プログラミングの場合は，定義されていない記号を使用した場合はコンパイラやインタープリタがエラーを出すが，数式をフォームに入力する場合は，エラーなどを出さないため，解答者は異なる記号を使っていても気づかずに提出することがある．試験などでは，変数が自由に定義されても意味が同じであるかで採点を行う．そこで，本研究では変数の置き換えを除いて論理的同値であることを同義として，命題論理式が同義であるかの判別や，標準形であるかの判別を自動で行うことで採点の補助を行うシステムの作成を行った．

2　関連研究

本章では関連研究について述べる．本研究は同じ意味の論理式を判別するものであるため，数式検索と関連する．また，情報論理学の授業の採点の補助を行うことを目的としているため，課題の自動採点と関連する．

2.1 数式検索

大橋ら [2] はあらかじめ用意した値を数式に代入し，計算した値を特徴量として検索インデックスに利用し，表記が異なっていても外延的に同値な関数を検索する手法を提案した．数式の検索には，数式を木構造で表し，pq-gram [3] などの木構造の類似度計算を用いることができる．しかし，同義であるが表記や木の構造が異なる数式を検索することはできない．事前に生成した乱数を代入して得られる実数値を特徴量とすることで，表記が異なり pq-gram では類似していないとみなされるが，変数の書き換えを除いて数学的に同値な式も検索でき，表記ではなく数学的意味に基づいた検索ができていることが示されている．

小田切ら [4] は数式の部分の形や関数の引数などの条件を利用して問い合わせ言語を実装し MathML[1]を用いた数式検索を行った．例えば「sin を含む積分」を検索したい場合，テキスト検索では「sin AND $\int$」というクエリを使用するが，「$\sin x \times \int_a^b x dx$」のような数式も条件に合致する．構造を持った条件を記述することができる問い合わせ言語を設計することで「sin を含む積分」などの問い合わせを MathML を用いて検索することを可能にした．

MathML は数式を電子化するためのマークアップ言語だが，本研究で扱う式のように人が入力する場合は，決められた文法で記述されないため，パーサの作成から行う必要がある．

2.2 自動採点システム

小西ら [5] は手書きされた数式を認識し，それを採点するシステムを提案した．教師側は模範解答を手書きでシステムに登録し認識した数式を CSV 形式で保存する．生徒は問題に対して手書きで解答し認識した数式を CSV 形式で保存する．提出された数式と正答の数式が同値であるかを Wolfram Alpha Web Service API[2]を利用して判定している．

立石 [6] らは教員が作成した模範解答から採点用のプログラムを自動で生成することで GUI プログラミング課題の採点を自動化するシステムを実現した．GUI プログラミングの課題を手作業で採点する場合はレイアウトの確認やマウスでの操作を必要とし多くの時間が必要である．事前に教員が作成した模範解答のプログラムからレイアウトの情報やイベントを取得し，XML 形式で保存し，提出された課題を採点する際は同様にレイアウトの情報を取得し模範解答の XML と比較を行うことで採点を行う．

3 準　　備

本章では，提案手法の前提知識について述べる．

3.1 命題論理

命題論理式は次のように，帰納的に定義された式である．

（1） 命題変数は，命題論理式である．

（2） T を「真」，F を「偽」とする定数を導入する．

（3） φ，ψ が命題論理式ならば，φ の否定 $\neg\varphi$，論理積 $\varphi \wedge \psi$，論理和 $\varphi \vee \psi$，含意 $\varphi \to \psi$ も命題論理式である．

（4） 以上の規則から生成される式だけを命題論理式という．

以上で定義された命題論理式に対して，真理値表は表 1，2 のように定義する．

表 1　否定の真理値表

X	$\neg X$
F	T
T	F

表 2　論理積，論理和，含意の真理値表

X	Y	$X \wedge Y$	$X \vee Y$	$X \to Y$
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	T

φ，ψ はともに n 個の命題変数 $X_1, X_2, \ldots, X_n$ から成る命題論理式とするとき，命題変数にどのように真理値を代入しても，つねに φ と ψ が同じ真理値を持つとき，φ と ψ は論理的同値であるという．

3.2 木の編集距離 (TED: Tree Edit Distance)

Tree Edit Distance [9] とは，1 つの木構造を別の木構造に変換するために必要な編集操作の最小の回数である．木に対して行う操作は次の 3 種類ある．

（1） 1 つのノードを削除（図 1）

（2） 1 つのノードを挿入（図 2）

（3） 1 つのノードの名前を変更（図 3）

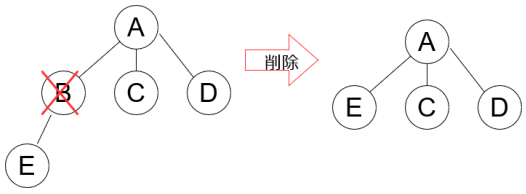

図 1　ノードの削除

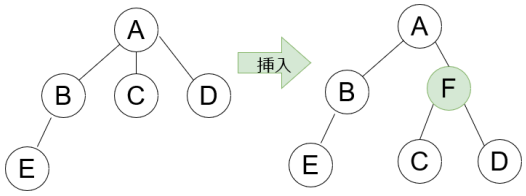

図 2　ノードの挿入

TED を求める計算量は，現在知られている最も効率の良いアルゴリズムでも，ノード数 n に対して $O(n^3)$ の時間計算量がかかる．

1：https://www.w3.org/Math/

2：https://products.wolframalpha.com/api/

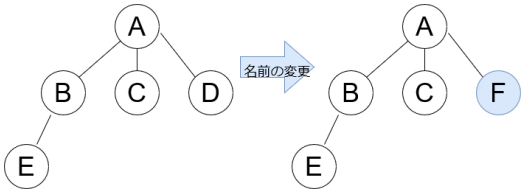

図 3　ノードの名前の変更

4　提 案 手 法

4.1　正規表現を用いた演算子の統一

論理式の表記揺れには大きく 2 種類ある．演算子の表記揺れと変数の表記揺れである．本研究では，はじめに演算子の統一を行う．

例えば，AND の演算子 $\wedge$ を&，AND などで表すことがある．したがって，次の式は全て同じ意味である．

$A \wedge B$　　A AND B　　$A\&B$

本研究で作成したシステムでは，このような演算子の表記揺れを表 3 に示した正規表現を用いて演算子を照合して置換することで統一を行った．

表 3　正規表現を使用した演算子の統一

変換前	変換後
∧|[Aa][Nn][Dd]|&&?	and
∨|[Oo][Rr]|||?	or
¬|[Nn][Oo][Tt]|~|!	not
→|[-=]>|[Ii][Mm][Pp][Ll][Ii][Ee][Ss]	implies

正規表現を使用した表現の統一は，粥川らが開発した数式検索システム [7] でも用いられている．また，自然言語の表記揺れの吸収にも正規表現が用いられることがある．箕輪らの研究 [8] で用いられている人狼ゲームにおける対話システムでも，表記揺れ，言い換えに対して正規表現で，表記揺れの吸収を行っている．そのため，演算子の表記揺れに対して正規表現を使用して，表記を統一することが適していると考えた．

4.2　論理式のパース

演算子を統一させた式に対してパースをする．論理式のパーサは Python の pyparsing[3] モジュールを使用して作成した．pyparsing は単純な文法を定義し，パーサを作成することが可能である．

まずは演算子とその優先順位，結合性を表 4 のように定義した．

次に，変数の定義をする．論理式において変数は大文字のアルファベットを用いることが多い．しかし，変数に用いる記号に決まりはなく，ひらがな，漢字などを変数として用いても良い．そこで，本研究で作成したシステムではアルファベット大文字，小文字，数字，アンダースコアを組み合わせた識別子を変数として定義し，さらに括弧およびスペース以外の Unicode で表された識別子も変数として扱えるように定義した．

3：https://pyparsing-docs.readthedocs.io/en/latest/

表 4　演算子の優先順位

演算子	優先順位	結合性
not	1	右
and	2	左
or	3	左
implies	4	左

以上で定義した変数，演算子をもとに論理式のパースを行う．式をパースした結果は木構造で表現される．今回のシステムでは，パースした結果をリストで表現した木構造に変換する．そのため，「(not A) or (not B)」という式は「[['not', ['A']], 'or', ['not', ['B']]]」と表現して処理を行う．

4.3　値 の 代 入

2 つの論理式が同値であるかを判別するために，実際に値を代入して計算を行う．命題変数には True または False の 2 種類の値が入る．そのため n 変数の命題論理式には 2^n 通りの値の組み合わせがある．命題変数に真理値をどのように代入しても常に 2 つの式が同じ真理値を持つとき論理的同値である．前節でパースした結果を用いて，変数に値を代入して計算を行う．

しかし，使われる変数に表記揺れがある場合は単純に値を代入させて計算するだけでは同値であるかを判別することはできない．そこで，対応させる変数のパターンも変えていく．例えば変数 X,Y,Z を (A,B,C) に対応させる方法は次の 6 通りある．

（1）　(X, Y, Z)
（2）　(X, Z, Y)
（3）　(Y, X, Z)
（4）　(Y, Z, X)
（5）　(Z, X, Y)
（6）　(Z, Y, X)

n 変数の論理式の変数の対応させる方法は $n!$ 通りある．そこで，変数の順列を作成し，それぞれについて命題変数に真理値を代入していく．論理的同値となる変数の対応方法があれば，2 つの式は同義であるとする．

4.4　標　準　形

命題変数を正のリテラルといい，命題変数の前に否定記号が一つだけ付いた命題論理式を負のリテラルという．正または負のリテラルを単にリテラルと呼び，$L_1, L_2, \ldots, L_k$ をリテラルとする．$L_1 \wedge L_2 \wedge \cdots \wedge L_k$ をリテラル積，$L_1 \vee L_2 \vee \cdots \vee L_k$ をリテラル和と呼ぶ．リテラル積の論理和の形の式を論理和標準形 (DNF: Disjunctive Normal Form)，リテラル和の論理積の形の式を論理積標準形 (CNF: Conjunctive Normal Form) という．論理式が論理和標準形や論理積標準形になっているかの判定を行うために，まず入力された論理式を標準形に変換する．与えられた論理式を論理的に同値な標準形は次のアルゴリズムで求められる．

（1）　$A \to B \equiv (\neg A) \vee B$ という同値関係を用いて含意記号 $\to$ を取り除く．

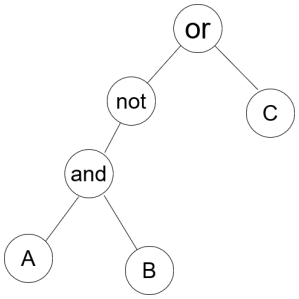

図 4　入力した式の木構造

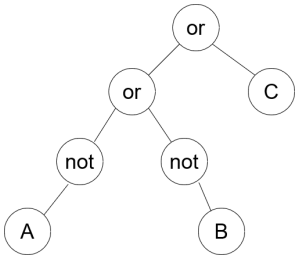

図 5　CNF へ変換した式の木構造

（2）　ド・モルガンの法則を用いて，論理和あるいは論理積の外側にある否定記号を内側に入れる．

（3）　分配法則を用いて，論理和標準形あるいは論理積標準形に整える．

（4）　二重否定の除去を用いて，命題変数の前にある否定記号を高々1 つにする．

本研究では，与えられた論理式を論理型計算モデルの Prolog を用いて標準形へ変換を行っている．論理型計算モデルは論理式の集合からなる論理プログラムが公理となり，プログラムのもとで入力データが成り立つことを証明する過程を計算として捉えるモデルである．標準形を求めるアルゴリズムを Prolog に公理として与え，入力された論理式を標準形に変換する．

次に，入力された式と標準形に変換された式を比較することで論理式が標準形であるかを判別する．式の比較は Tree Edit Distance (TED) [9] を用いた．まず，2 つの式をそれぞれパースして木構造で表現する．入力された式が標準形でない場合は，Prolog で標準形に変換されたものと構造が一致しないため TED は 1 以上となる．入力された式が標準形の場合は，Prolog で標準形に変換する際に適用するルールが無いため，そのまま出力されるため，式の構造が一致し TED は 0 となる．そのため，TED を用いて標準形であるかを判定する．

「not(A and B) or C」という式が与えられた場合，CNF に変換すると「(not A) or (not B) or C」となる．与えられた式の木構造（図 4）と変換した式の木構造（図 5）は異なるため，標準形ではないと判断できる．

5　アプリケーション

5.1　開 発 環 境

本研究の開発環境について述べる．使用 OS は Windows10，GUI (Graphical User Interface) の開発言語は C#，開発環境は Visual Studio 2022 Community を使用した．CSV ファイルを読み込み採点する部分は Python 3.8.10，式を標準形に変換する部分は SWI-Prolog 8.2.4 を用いた．

5.2　アプリケーションの概要

本章では作成したアプリケーションについて説明する．図 6 は作成したアプリケーションの画面である．模範解答と採点するファイルを選択をすると自動で採点が行われ，結果をファイルの保存する．模範解答と学生が提出した解答は図 8 のように

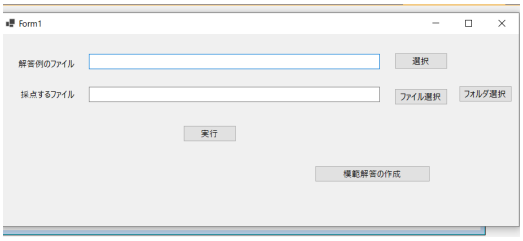

図 6　アプリケーション

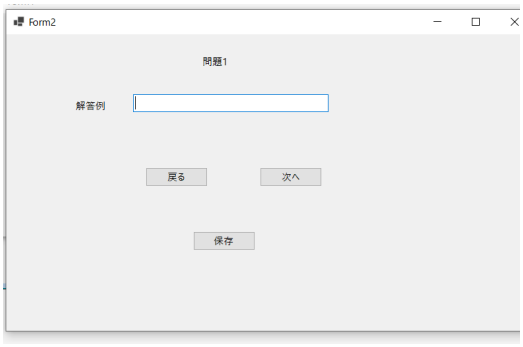

図 7　模範解答の作成画面

```
1,A & B
2,A and B or C
3,not(あ ∨ い)
4,A→B→A
5,X∧(¬Y)∧Z
6,(¬A→¬B)→(A→B)
```

図 8　提出された解答の CSV

```
1,A & B,True,True,True
2,A and B or C,True,False,True
3,not(あ ∨ い),True,False,False
4,A→B→A,False,False,False
5,X∧(¬Y)∧Z,True,True,True
6,(¬A→¬B)→(A→B),True,False,False
```

図 9　採点した結果

CSV 形式でファイルに保存する．

CSV ファイルには，問題の番号，論理式が記述されている．CSV ファイルを読み込み，式が同義であるか，標準形であるかを順番に見ていき，結果を図 9 のような CSV 形式で出力する．

図 7 は模範解答を作成する画面である．模範解答となる論理式を入力し，保存をすると問題番号と論理式が CSV 形式でファイルに保存される．

6　評　　価

6.1　検 証 結 果

実装したアプリケーションが正しく動作することを確認するために，テストケースとして 6 組の論理式を作成し，検証した．同義の判別結果を表 5，標準形の判別結果を表 6 にまとめた．

6.2　考　　察

表 5 より CNF や DNF であることは自動採点で正しく判別を行うことができた．しかし，同義であることの判別は正しく行えない場合がある．式 (7) と式 (8) は使われている変数は同じだが，式の構造は異なる．しかし，本システムでは変数の意味は考慮せず，同値となる変数の対応方法があるかを調べることで同義性を判定している．そのため，式 (7) の Y を式 (8) の Z に対応させ，式 (7) の Z を式 (8) の Y に対応させると 2 つの式は同じ意味となるため同義と判定している．

表 5　同義の判別結果

論理式 1	論理式 2	自動採点	実際の正誤
A ∧ B	A & B	○	○
A ∧ B ∨ C	X and Y or Z	○	○
¬ (A ∨ B)	(not あ) and (not い)	○	○
A → (B → A)	A → B → A	×	×
X ∧ Y ∧ (¬ Z)	X ∧ (¬ Y) ∧ Z	○	×
(¬ A →¬ B) → (A → B)	((¬ A) ∧¬ B) ∨ (¬ A ∨ B)	○	○

表 6　標準形の判別結果

	自動採点		実際の正誤	
論理式	CNF	DNF	CNF	DNF
A & B	○	○	○	○
X and Y or Z	×	○	×	○
(not あ) and (not い)	○	○	○	○
A → B → A	×	×	×	×
X ∧ (¬ Y) ∧ Z	○	○	○	○
((¬ A) ∧¬ B) ∨ (¬ A ∨ B)	×	○	×	○

採点のミスは，学生の評価にかかわるため，今回作成したアプリケーションは採点の補助として用いて，人が確認を行う必要がある．

$$X \wedge Y \wedge (\neg Z) \tag{7}$$

$$X \wedge (\neg Y) \wedge Z \tag{8}$$

7 おわりに

本研究では，命題論理式の採点システムの実装において，値を代入して比較をすることで表記揺れがあるが同義な式の判別を提案した．また，論理式が標準形であるかを判定するために，与えられた式を標準形に自動で変換し，与えられた式と変換した式を比較することで，標準形になっているかの判定を提案した．

今後の課題としては，意味を考慮した命題論理式の判別や，述語論理の判別などが挙げられる．現段階では，論理式の意味は考慮していないため「自然言語で表された文を論理式で表せ」というような問題の採点はできない．また，命題論理式の採点にしか対応していないため，述語論理への対応が求められる．さらに，論理式はプログラミングにも用いられているため，if 文の比較への応用が期待される．プログラミングの授業において，提出された解答と模範解答と比較を行う際，if 文が複雑だと人が判別するのは大変なため，自動で判別を行うことで採点の補助を行うことができる．

謝　辞

本研究の一部は JSPS 科研費 19K11982 の助成を受けたものです．

文　献

[1] 坂井昌典．”数学のかんどころ 31 情報理論のための数理論理学”．共立出版．初版第 2 刷（2021）
[2] 大橋駿介，高須淳宏，相澤彰子．”表記が異なる同義の数式の高速な検索法”．DEIM Forum 2014．C7-1
[3] Augsten, Nikolaus, Michael Böhlen, and Johann Gamper. ”The pq-gram distance between ordered labeled trees.” ACM Transactions on Database Systems (TODS) 35.1 (2008): 1-36.
[4] 小田切 健一, 村田 剛志, MathML を用いた数式検索, 人工知能学会全国大会論文集, 2008, JSAI08 巻, 第 22 回 (2008), セッション ID 1F1-3, p. 23
[5] 小西渉, 佐々木進亮, 松下朋永, レ・ドゥックアイン, 中川正樹. (2016). 手書き数式認識を用いた算数・数学自動採点システム. 研究報告コンピュータと教育 (CE), 2016(7), 1-7.
[6] 立石良生，井上潮．”模範解答を利用した GUI プログラミング課題の自動採点システム”．DEIM Forum 2020．I1-1
[7] 粥川佳哉, 宮崎佳典. ”数式検索システムを応用した学習項目ならびに公式抽出機能の実装 (新技術と教育情報を活用した教育学習環境の設計/一般).” JSiSE 研究会研究報告 32.5 (2018): 97-102.
[8] 箕輪 峻, 狩野 芳伸, 人狼ゲームにおける対話システムのための中間表現および自然言語から中間表現への変換, 人工知能学会論文誌, 2020, 35 巻, 1 号, p. DSI-F_1-13
[9] Tai, Kuo-Chung. ”The tree-to-tree correction problem.” Journal of the ACM (JACM) 26.3 (1979): 422-433.