部分例示手法に基づく定理自動証明に関する研究

山本　雅人

目 次

図 目 次

表 目 次

第 1 章

序　論

1.1　背　景

近年の計算機技術の急速な進歩に伴って，人間が行ったとすれば知的な能力を必要とするであろう仕事を計算機で行おうとする要求が増えつつある．例えば，プログラムを書くとか，質問に答えるとか，定理を証明するといった仕事である．人工知能は計算機にこのような仕事をさせようという計算機分野の一つである．1960 年代後半は，人工知能の分野において急速に定理自動証明に対する興味が高まった時期であった．このことは，論理的な演繹が人間の知能の中心部分になっているという見方が強くなってきたことだけでなく，定理自動証明についての技術が非常に進歩してきたことによるものであろう．

定理自動証明は，ある論理式が他の論理式から論理的に帰結できることを示すために，ある論理式が恒偽（または，恒真）であることを示すものである．その応用として，質問応答システム，プログラムの分析や合成などに適用されている [Chang 73]．1936 年に Church と Turing は任意の第一階述語論理式が恒偽であることを示すための一般的な手続きは存

在しないことを証明した．しかし，論理式が実際に恒偽であるときにはそれを証明する手続きは存在し，恒偽でない論理式に対しては，これらの証明手続きは一般には停止する保証がないことも明らかとなった．

定理自動証明の研究の基礎は 1930 年に Herbrand によって築かれた．Herbrand は与えられた論理式を偽にするような解釈を見つけだす手続きを考案した．彼の方法は，論理式が実際に恒真であれば，有限回の試行後に停止するものであったが，この方法は計算機が発明されるまで実行は不可能であった．Gilmore は Herbrand の方法を計算機ではじめて実行した．Gilmore の方法は，実行中に命題論理式を次々に発生し，その充足可能性を一定間隔ごとに検査する方法であったが，命題論理式の充足可能性を検査する部分の効率が悪く，後に Davis と Putnam によって効率化がなされた [Davis 60]．しかし，この改良も十分なものではなかった．

画期的な突破口を切り開いたのは，Robinson による導出原理であった [Robinson 65]．この導出原理に基づく手続きは，それまでの手続きよりはるかに効率的である．導出原理が発表されて以来，この方法を改良した方法が数多く提案されおり，現在の定理自動証明手続きは，この導出原理に基づくものがほとんどである．

最近になって，定理証明を命題論理式の充足可能性問題を繰り返し解くことに帰着するリダクション手法が再び注目されてきた．その理由として，計算機技術の進歩と命題論理式の充足可能性を判定する手続きの効率化が挙げられる．実際，命題論理式の充足可能性問題は，ある特定の性質をもつ難しい問題以外は，かなり高速に解けることが報告されている．

リダクション手法としては，Lee らが提案している Hyper- Linking Strategy，及び，Jeroslow によって提案された部分例示手法がある [Lee 92] [Jeroslow 88]．部分例示手法

は，論理式中の一部の変数に Herbrand 空間の要素を代入した論理式を次々に生成し，その充足可能性を繰り返し判定して定理証明を行う手法である．特に，充足可能性問題を繰り返し解くことで得られる解釈を用いる点に特徴がある．しかしながら，1988 年の Jeroslow の急逝により，部分例示手法に関する研究は多くの重要な問題点が未解決のまま中断した．その研究は，関数記号を関数記号を含まない論理式を対象としたものでしかなく，導出原理に基づく方法に比べて，より小さいクラスの問題にしか適用できない段階であった．また，命題論理式の充足可能性問題を解く際の計算効率が悪いなどの欠点があった．

1.2 本研究の目的

本論文は，Jeroslow の提案した部分例示手法に基づいて，Jeroslow の方法の欠点について検討し，それを改善した新しい定理自動証明手続きを構築することを目的とする [山本 94] [山本 95]．本論文で提案する手続きは，関数記号を含む節形式の論理式に対して適用可能である．また，本論文で提案する手続きが完全性をもつことの証明を与えている．完全性とは，実際に恒偽である論理式に対しては，その論理式が恒偽であることを証明することが可能であることをいい，定理自動証明手続きがもつべき重要な性質の一つである．さらに，導出原理に基づく定理自動証明手続きとの比較実験を行い，その有効性について検討している．その成果は，以下のように要約できる．

(1) Jeroslow の方法と比較して，より大きなクラスに含まれる問題に適用可能である．

(2) 節形式を前提としているため，命題論理の充足可能性を判定するための効率のよいアルゴリズムを用いることができる．

(3) 導出原理に基づく方法では証明が困難な問題の中で，提案した手続きでは容易に証

明可能な問題が存在することを示している．

本論文は全 6 章で構成されている．各章の概要は以下の通りである．

第 1 章は序論であり，本研究の背景とその目的について述べている．

第 2 章では，本論の準備として記号論理の基礎について述べている．本論文で提案する定理自動証明手続きは，命題論理式の充足可能性問題を繰り返し解くことにより定理証明を行うものである．このため，記号論理におけるもっとも基本的な体系である命題論理に関して，特に，解釈や充足可能性について説明している．また，第一階述語論理の基礎として，任意の論理式が充足可能性を変えることなしに節集合へ変換が可能であることを示している．そして，節集合に対して充足可能性を判定するための重要な定理である，Herbrand の定理などについて説明している．本論文で提案する手続きや現在まで提案されているほとんどの定理自動証明手続きは，この Herbrand の定理を基礎としている．

第 3 章では，定理自動証明に対するこれまでの研究の中で，最も一般的な導出原理に基づく方法を説明している．導出原理は 2 つの節から新たな節を生成するための推論規則である．導出原理に基づく定理自動証明手続きは，この推論規則を節集合に対して繰り返し適用し矛盾を意味する空節を導くことを目的とする反駁手続きである．また，本論文が基礎としている部分例示手法の基本的概念について述べ，Jeroslow が提案している方法について詳しく述べている．そして，Jeroslow の方法を用いて実際にコンピュータによる定理証明を行う際の問題点について指摘し，本論文で提案する手続きの目的を明確にしている．

第 4 章では，節形式の論理式に対して適用可能な部分例示手法に基づく定理自動証明手続きを考案し，その詳細について述べている．手続きは，(1) 命題論理式の充足可能性問題を解く，(2) 純障害物と呼ばれる原子式の組を見つける，(3) その純障害物を解消するために節集合に新たな節を加える，の 3 つの処理を繰り返すものである．本章では，これらの

処理の詳細についてそれぞれ例題を交えて述べている．さらに，Jeroslow の方法との相違点などについて論じ，今回提案した手続きでは，Jeroslow の方法では不可能であった関数記号を含む論理式を扱うことが可能となることを示している．また，提案した手続きの完全性を証明している．

第 5 章では，提案した手続きの効率化のため，命題論理式の充足可能性問題に関して，(1) いかに高速に解くか，(2) どのような解が有効であるか，の検討を行なっている．その際，導出原理に基づく定理自動証明手続きとして有名な OTTER との計算機実験を行い，提案した手続きの特徴を明らかにしている．

第 6 章では，本研究の結論及び今後の展望について述べている．

第 2 章

記号論理の基礎

本論文で扱う定理自動証明は，第一階述語論理の枠組みにおいて行われる．また，提案する手続きは，定理証明を命題論理の充足可能性問題を繰り返し解くことに帰着するリダクション手法の一つであるため，命題論理及び第一階述語論理の概念は非常に重要なものである．

本章では，定理自動証明手続きについて述べるための準備として，記号論理学の最も簡単かつ基本的な体系である命題論理と，より一般的な体系である第一階述語論理について順に述べる．特に，定理自動証明において重要な解釈，充足可能性などの用語や性質について述べる．

2.1 命題論理

2.1.1 序 論

命題論理においては，真 (true) か偽 (false) の値をとり，その両方の値をとることのない言明文を扱う．このような言明文は，命題 (proposition) と呼ばれる．例えば,「海は青い」や,「天気は晴である」,「ソクラテスは人間である」等は命題である．命題に与えられた「真」または「偽」の値は，真理値 (truth value) と呼ばれる．命題を表すのに，通常 $P, Q, \cdots$ や $P_1, P_2, \cdots$ 等の文字を用い，これらの文字を原子論理式 (atomic formula)，または単に原子式 (atom) と呼ぶ．また，真理値を表すのに，T(真)，F(偽) を用いる．

このような原子式を用いることでもっと複雑な言明文，すなわち命題をつくることができる．例えば,「もし海は青いならば天気は晴である」や,「海は青いかつソクラテスは人間である」等である．このとき,「もし … ならば …」(含意) や「… かつ …」(連言) は，論理結合子 (logical connectives) と呼ばれる．命題論理において用いられる論理結合子は以下の 5 種類とする．ただし，() 内は，それを表す記号とする．否定 ($\sim$)，連言 ($\wedge$)，選言 ($\vee$)，含意 ($\rightarrow$)，同値 ($\leftrightarrow$)．

以下では，論理式 (formula) を定義する．

定義 2.1.1

論理式は以下のように帰納的に定義される．

(1) 原子式は論理式である．

(2) P が論理式ならば，$(\sim P)$ も論理式である．

(3) P，Q が論理式ならば，$(P \wedge Q)$，$(P \vee Q)$，$(P \rightarrow Q)$，$(P \leftrightarrow Q)$ も論理式である．

(4) 以上の規則から生成されるもののみが論理式である.

例えば, $(P \to (Q \vee R))$ や $((P \vee Q) \wedge ((\sim P)))$ は論理式であるが, $(P \to)$ や $(P\wedge)$ などは論理式ではない.

また, 論理結合子に以下のように左から高い優先順位を付け, 優先順位の高い論理結合子の結合を優先することで無駄な括弧を省略することにする.

$$\sim , \wedge , \vee , \to , \leftrightarrow$$

例えば, $P \to Q \vee R$ は $(P \to (Q \vee R))$ を, $(P \vee Q)\wedge \sim P$ は $((P \vee Q) \wedge ((\sim P)))$ を意味する.

2.1.2 命題論理式の解釈

論理式の真理値は, その論理式中の原子式の真理値にのみ依存して決まる. P, Q を原子式とし, その原子式の真理値と論理式 $(\sim P)$, $(P \wedge Q)$, $(P \vee Q)$, $(P \to Q)$, $(P \leftrightarrow Q)$ の真理値の対応は以下の表 2.1 のようになる. このような表は, 真理値表 (truth table) と呼ばれる.

この対応から, どんな複雑な論理式の真理値もその論理式中の原子式の真理値のみによって決定できる. 例えば, P, Q, R の真理値をそれぞれ T, T, F とすると, 論理式 $(P \to (Q \vee R))$ の真理値は T となることが容易にわかる.

P, Q, R に割当てた真理値の組は論理式 $(P \to (Q \vee R))$ に対する解釈 (interpretation) と呼ばれる. 以下で, 解釈を定義する.

P	Q	$\sim P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

表 2.1: 原子式の真理値と基本的な論理式の真理値との関係

定義 2.1.2

命題論理式 G が与えられ，$A_1, A_2, \cdots, A_n$ を G 中の原子式とする．このとき，$A_1, A_2, \cdots, A_n$ に T か F のどちらか一方を割当てたものを G の解釈という．

解釈を表すのに，$\{m_1, m_2, \cdots, m_n\}$ を用いることがある．これは，$m_i (1 \leq i \leq n)$ が T となるような原子式への真理値の割当てである．ただし，m_i は A_i か $\sim A_i$ のどちらかである．例えば，P，Q，R に対してそれぞれ T，T，F を割当てる解釈は，$\{P, Q, \sim R\}$ となる．一般に，論理式が n 個の原子式を含むならば，その論理式に対する解釈は 2^n 個存在する．

2.1.3 命題論理式の充足可能性

論理式 $P \vee \sim P$ を考えてみよう．この論理式に対する解釈は，$I_1 = \{P\}$ と $I_2 = \{\sim P\}$ の二つである．この二つの解釈に対して，論理式の真理値は共に真となる．このように，すべての解釈において真となる論理式のことを恒真式 (valid formula) または，トートロジー (tautology) と呼ぶ．

反対に論理式 $P \wedge \sim P$ のようにすべての解釈で偽となる論理式は，恒偽式 (inconsistent formula)，または，矛盾 (contradiction) と呼ぶ．

定義 2.1.3

ある論理式がすべての解釈で真 (T) であるならば，そのときに限ってその論理式は恒真 (valid) であるという．

定義 2.1.4

ある論理式がすべての解釈で偽 (F) であるならば，そのときに限ってその論理式は恒偽 (inconsistent)，または，充足不能 (unsatisfiable) であるという．また，それが充足不能でないとき，そのときに限ってその論理式は充足可能 (satisfiable) であるという．

従って，充足不能と恒偽は同じ意味であるが，充足可能と恒真とは同じ意味ではない．また，論理式 G がある解釈 I において真となるとき，I は G を充足する (satisfy)，または，G は I によって充足されるという．逆に，論理式 G がある解釈 I において偽となるとき，I は G を偽にする (falsify)，または，G は I によって偽になるという．さらに，論理式 G が解釈 I によって充足されるとき，I は G のモデルであるともいう．

2.1.4 命題論理式の標準形

任意の論理式は，論理式間のある等価な変換を繰り返し用いることで，標準形と呼ばれる形にすることができる．以下で，標準形の定義を与える．

定義 2.1.5

P を原子式とするとき，P または，$\sim P$ のことをリテラル (literal) という．また，P を正のリテラル，$\sim P$ を負のリテラルという．

定義 2.1.6

$l_{i_1}, l_{i_2}, \cdots, l_{i_m}$ をリテラルとするとき，F_i が

$$F_i = l_{i_1} \vee l_{i_2} \vee \cdots \vee l_{i_m}$$

という形をしていて，F が

$$F = F_1 \wedge F_2 \wedge \cdots \wedge F_n \qquad (n \geq 1)$$

という形をしているとき，F の形式の論理式を**連言標準形** (conjunctive normal form) という．

F_i は，**節** (clause) と呼ばれ，連言標準形は，**節形式** (clausal form) とも呼ばれる．また，$F = \{F_1, \cdots, F_n\}$ は，**節集合** (set of clauses) と呼ばれる．これと同様に選言標準形も定義できるが，連言標準形を用いる方が一般的であるので，ここでは連言標準形のみ定義する．

任意の論理式は，節形式に変換することができるが，そのためには以下の等価な式を繰り返し用いた変換を行う．ただし，F，G は任意の論理式であるとする．等価の定義も併せて与える．

定義 2.1.7

F，G が論理式であるとする．F と G のそのいかなる解釈のもとでも真理値が同じであれば，F と G は**等価** (equivalent) であるという．

$$F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F) \tag{2.1}$$

$$F \rightarrow G = \sim F \vee G \tag{2.2}$$

$$\sim (\sim F) = F \tag{2.3}$$

$$\sim (F \vee G) = \sim F \wedge \sim G \tag{2.4}$$

$$\sim (F \wedge G) = \sim F \vee \sim G \tag{2.5}$$

$$F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H) \tag{2.6}$$

これらの式を用いて，任意の論理式を節形式に変換するためのアルゴリズムを以下に示す．

アルゴリズム 2.1.1

step 1 : (2.1), (2.2) を用いて，論理結合子 $\leftrightarrow$, $\rightarrow$ を消去する．

step 2 : (2.3), $\cdots$, (2.5) を用いて，否定記号を基本命題の直前にもってくる．

step 3 : (2.6) を用いることによって，節形式に変換される．

2.1.5 論理的帰結

以下で，論理的帰結と呼ばれる重要な概念を定義する．

定義 2.1.8

$G, F_1, F_2, \cdots, F_n$ をそれぞれ論理式とする．$F_1 \wedge F_2 \wedge \cdots \wedge F_n$ が真となるすべての解釈において G が真となるとき，またそのときに限って，G を $F_1, F_2, \cdots, F_n$ の論理的帰結 (logical consequence) という．このとき，$F_1, F_2, \cdots, F_n$ を G の公理 (axiom)，または仮定 (postulates) という．

従って，G が $F_1, F_2, \cdots, F_n$ の論理的帰結であることを示すためには，$F_1 \wedge F_2 \wedge \cdots \wedge F_n$ が真となるすべての解釈において G の真理値を調べなければならない．しかし実際には，以下の定理によって，ある論理式が恒真であるか，または，恒偽であるかを判定する問題に帰着でき，その問題を解くことで示すことができる．定理 2.1.1は，演繹定理とも呼ばれる．

定理 2.1.1

論理式 $G, F_1, F_2, \cdots, F_n$ が与えられたとき，G が $F_1, F_2, \cdots, F_n$ の論理的帰結であることと，論理式 $((F_1 \wedge F_2 \wedge \cdots \wedge F_n) \rightarrow G)$ が恒真であることは同値である．

定理 2.1.2

論理式 $G, F_1, F_2, \cdots, F_n$ が与えられたとき，G が $F_1, F_2, \cdots, F_n$ の論理的帰結であることと，論理式 $(F_1 \wedge F_2 \wedge \cdots \wedge F_n \wedge \sim G)$ が恒偽であることは同値である．

G が $F_1, F_2, \cdots, F_n$ の論理的帰結であるとき，論理式 $((F_1 \wedge F_2 \wedge \cdots \wedge F_n) \rightarrow G)$ を定理 (theorem) と呼び，G を定理の結論 (conclusion of the theorem) と呼ぶ．この定理が恒真であることを証明するのが定理証明の手続きである．また，一般的には定理 2.1.2より，$(F_1 \wedge F_2 \wedge \cdots \wedge F_n \wedge \sim G)$ が恒偽であるかどうか，すなわち，充足可能性問題を解くことによって定理証明を行う．

2.2 第一階述語論理

2.2.1 序 論

命題論理では，最も基本的な要素である原子式を使ってさまざまな表現が可能である．しかしながら，もっと複雑な内容の表現は命題論理のみでは不十分である．例えば，以下のような言明文を考える．

$A1$: すべての人間は死ぬ運命にある．

$A2$: 孔子は人間である．

$A3$: 孔子は死ぬ運命にある．

A1，A2 から A3 が結論できることは明らかである．しかし，A1，A2，A3 の言明文をそれぞれ命題 P, Q, R で表すとすると，命題論理の範囲では，R は $P \wedge Q$ からの論理的帰結とはならない．

このように命題論理のみで扱えないような表現を扱うために，述語論理が用いられる．以下では，第一階述語論理においても，命題論理と同様に原子式，充足可能性などの定義を与える．

原子式の作るのに，以下の 4 種類の型の記号を使う．

(1) 定数：$a, b, c, John$

(2) 変数：$x, y, z, \cdots$

(3) 関数記号：$f, g, father, \cdots$

(4) 述語記号：$P, Q, MAN, \cdots$

本論文では，関数記号を小文字，述語記号を大文字を用いて区別する．また，定数は引数をもたない関数であると考えることもできる．例えば，述語記号を用いた $MAN(John)$, $LOVE(John, Mary)$ はそれぞれ，$John$ は人間である，$John$ は $Mary$ を愛している，などを表し，関数記号を用いた $father(John)$ は $John$ を $John$ の父に写像する関数を表す．n 個の引数をもつ関数記号や述語記号は，それぞれ $n-$変数関数記号，$n-$変数述語記号と呼ばれる．

定義 2.2.1

項 (term) は次のように帰納的に定義される．

(1) 定数は項である．

(2) 変数は項である．

(3) f が $n-$変数関数，$t_1, \cdots, t_n$ が項であれば，$f(t_1, \cdots, t_n)$ は項である．

(4) 以上の規則から生成されるもののみが項である．

以下では，原子式を定義する．

定義 2.2.2

P を $n-$変数述語記号，$t_1, \cdots, t_n$ を項とすると，$P(t_1, \cdots, t_n)$ は**原子式** (atomic formula) である．

この原子式と前節の命題論理と同様の論理結合子を用いて，新しい論理式を組み立てることができる．第一階述語論理においては，変数を用いるため，これらの変数を特徴づける 2 つの限定記号 $\forall$, $\exists$ が使われる．x を変数とすると，$(\forall x)$ は,「すべての x について」，$(\exists x)$ は,「ある x について」を意味する．これらは，それぞれ**全称記号** (universal quantifier)，**存在記号** (existential quantifier) と呼ばれる．

例えば，$(\forall x)(MAN(x))$ は,「すべての x について，x は人間である」を意味する．また，$(\exists x)(LOVE(x, Mary))$ は,「ある x について，x は $Mary$ を愛する」を意味する．

上で述べた，原子式，論理結合子，限定記号を使って，以下で第一階述語論理における論理式を定義する．

定義 2.2.3

第一階述語論理式における**論理式** (formulus) は，次のように帰納的に定義される．

(1) 原子式は論理式である．

(2) F, G が論理式ならば，$(\sim F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ も論理式である．

(3) F が論理式，x が F 中の自由変数[1]であれば，$(\forall x)F$, $(\exists x)F$ は論理式である．

(4) 以上の規則を有限回適用することによって生成されるもののみが論理式である．

[1] その変数を限定する限定記号が変数の出現位置を束縛していない

これらから，A1，A2，A3 の文は以下のように表現される．ただし，MAN，$MORTAL$ は 1−変数述語記号，$confucious$ は孔子を表す定数である．

$$\text{A1: } (\forall x)(MAN(x) \rightarrow MORTAL(x))$$

$$\text{A2: } MAN(confucious)$$

$$\text{A3: } MORTAL(confucious)$$

2.2.2 第一階述語論理式の解釈

命題論理における解釈は，原子式に対する真理値の割当てであった．しかし，第一階述語論理における解釈は，論理式が変数を含むためにこれだけでは不十分である．第一階述語論理式の解釈を定めるには，定義域と，論理式中に現れる定数，関数記号，述語記号への割当てを定めなければならない．以下で，第一階述語論理における解釈を定義する．

定義 2.2.4

第一階述語論理の論理式 F の解釈 (interpretation) I は，空でない定義域 D と以下の割当てからなる．また，このとき I は D 上の解釈と呼ばれる．

(1) F 中の各定数に対して，D の要素を一つ割当てる．

(2) F 中の各 n−変数関数記号に対して，D^n から D への写像を割当てる．

(3) F 中の各 n−変数述語記号に対して，D^n から $\{T, F\}$ への写像を割当てる．

例えば，論理式 $(\forall x)(P(x) \rightarrow Q(f(x), a))$ に対する解釈の一つは，以下の表 2.2 で表される．

論理式の解釈を定めると，論理式の真理値を定めることができる．論理式 G，H の真理値が計算されているとすると，

定義域　$D = \{1, 2\}$

定数への割当て

a
1

関数への割当て

$f(1)$	$f(2)$
2	1

述語への割当て

$P(1)$	$P(2)$	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
F	T	T	T	F	T

表 2.2: 論理式 $(\forall x)(P(x) \to Q(f(x), a))$ に対する解釈

1. 論理式 $\sim G$, $G \wedge H$, $G \vee H$, $G \rightarrow H$, $G \leftrightarrow H$ の真理値は命題論理の場合と同様に求めることができる.

2. 論理式 $(\forall x)G$ は, 定義域 D のすべての要素 x について G の真理値が真となるとき真となり, それ以外は偽となる.

3. 論理式 $(\exists x)G$ は, 定義域 D の少なくとも一つの要素 x について G の真理値が真となるとき真となり, それ以外は偽となる.

例えば, 論理式 $(\forall x)(P(x) \rightarrow Q(f(x), a))$ の真理値は, 表 2.2 の解釈において, 以下のように計算することができる.

(1) $x = 1$ のとき,

$$\begin{aligned} P(1) \rightarrow Q(f(1), a)) &= F \rightarrow Q(2, 1) \\ &= F \rightarrow F \\ &= T \end{aligned}$$

(2) $x = 2$ のとき,

$$\begin{aligned} P(2) \rightarrow Q(f(2), a)) &= T \rightarrow Q(1, 1) \\ &= T \rightarrow T \\ &= T \end{aligned}$$

(1), (2) より, 定義域 D 上のすべての要素について $(P(x) \rightarrow Q(f(x), a))$ は真 (T) となるから, 論理式 $(\forall x)(P(x) \rightarrow Q(f(x), a))$ の真理値は真となる.

命題論理の場合と同様に充足可能, 充足不能についても定義できる.

定義 2.2.5

G が真となる解釈 I が存在するとき，またそのときに限り，G は**充足可能** (satisfiable) であるという．また，その解釈 I は論理式 G を充足するという．さらに，論理式 G を充足する解釈が存在しないとき，またそのときに限り，G は**充足不能** (unsatisfiable) であるという．

定義 2.2.6

G のすべての解釈が G を充足するとき，またそのときに限り，論理式 G は**恒真** (valid) であるという．

定義 2.2.7

すべての解釈 I について，$F_1, F_2, \cdots, F_n$ が I のもとで真であれば，G もまた I のもとで真となるとき，またそのときに限り，論理式 G を $F_1, F_2, \cdots, F_n$ の**論理的帰結** (logical consequence) であるという．

定理 2.1.1，定理 2.1.2 で述べたことは，第一階述語論理に対しても同様に成立する．

2.2.3　第一階述語論理式の標準形

命題論理においては，標準形として連言標準形について述べた．第一階述語論理においては，連言標準形に加えて，冠頭標準形と呼ばれる標準形とスコーレム標準形と呼ばれる標準形がある．これらについて順に説明する．

以下では，まず冠頭標準形の定義を与える．

定義 2.2.8

第一階述語論理式 F が,

$$(Q_1x_1)\cdots(Q_nx_n)M$$

の形をしているならば，またそのときに限って，冠頭標準形 (prenex normal form) であるという．ここで，各 (Q_ix_i) $(i=1,\cdots,n)$ は $(\forall x_i)$，または $(\exists x_i)$，M は限定記号を含まない論理式である．$(Q_1x_1)\cdots(Q_nx_n)$ を前置部 (prefix)，M を母式 (matrix) と呼ぶ．

第一階述語論理式においても，すべての解釈のもとで二つの論理式 F と G の真理値が同じであるなら，F と G は等価 (equivalent) であるといい，$F=G$ と表す．

任意の論理式を，等価な式を繰り返し用いて冠頭標準形に変換する手続きが存在する．そのために必要な等価な式を以下に示す．ただし，$F[x]$ や $H[x]$ は論理式 F や H が変数 x を含むことを表し，G は変数 x を含まない論理式を表す．また，Q は $\forall$ かまたは，$\exists$ である．

$$\sim((\forall x)F[x])=(\exists x)(\sim F[x]) \tag{2.7}$$

$$\sim((\exists x)F[x])=(\forall x)(\sim F[x]) \tag{2.8}$$

$$(Qx)F[x]\vee G=(Qx)(F[x]\vee G) \tag{2.9}$$

$$(Qx)F[x]\wedge G=(Qx)(F[x]\wedge G) \tag{2.10}$$

$$(\forall x)F[x]\wedge(\forall x)H[x]=(\forall x)(F[x]\wedge H[x]) \tag{2.11}$$

$$(\exists x)F[x]\vee(\exists x)H[x]=(\exists x)(F[x]\vee H[x]) \tag{2.12}$$

$$(Q_1x)F[x]\vee(Q_2x)H[x]=(Q_1x)(Q_2z)(F[x]\vee H[z]) \tag{2.13}$$

$$(Q_3x)F[x]\wedge(Q_4x)H[x]=(Q_3x)(Q_4z)(F[x]\wedge H[z]) \tag{2.14}$$

以上の式と命題論理における等価な式 (2.2) ～ (2.5) を用いて，任意の論理式を冠頭標準形に変換するアルゴリズムを以下に示す．

アルゴリズム **2.2.1**

step 1： (2.1), (2.2) を用いて，論理結合子 $\leftrightarrow$, $\rightarrow$ を消去する．

step 2： (2.3), $\cdots$, (2.5) と，(2.7), (2.8) を用いて，否定記号を原子式の直前にもってくる．

step 3： 必要な箇所の変数名をつけかえる．

step 4： (2.9) ～ (2.14) を用いることによって，限定記号を論理式の左に移すことで冠頭標準形を得る．

例えば，論理式 $(\forall x)P(x) \rightarrow (\exists x)Q(x)$ は以下のように冠頭標準形に変換できる．

$$\begin{aligned}(\forall x)P(x) \rightarrow (\exists x)Q(x) &= \sim ((\forall x)P(x)) \vee (\exists x)Q(x) \\ &= (\exists x)(\sim P(x)) \vee (\exists x)Q(x)) \\ &= (\exists x)(\sim P(x) \vee Q(x))\end{aligned}$$

次に，冠頭標準形に変換された論理式に対して，スコーレム標準形と呼ばれる標準形を求める手続きについて述べる．定理自動証明手続きのほとんどは，このスコーレム標準形に対して適用される．この標準形への変換は，以下の考え方に基づいている．

1. 任意の第一階述語論理式はそれと等価な冠頭標準形に変換することができる．
2. 冠頭標準形に変換された論理式の母式は限定記号を含まないから，連言標準形 (節形式) に変換することができる．

3. 充足可能性を変えることなく，前置部の存在記号をスコーレム関数を用いることで消去することができる．

(1)，(2) についてはすでに述べたので，ここでは，(3) を行う手続きについて述べる．

冠頭標準形に変換された論理式を $F = (Q_1x_1)\cdots(Q_nx_n)M$ とおく．ここで，M は連言標準形に変換されているとする．Q_r が前置部中の存在記号であるとする $(1 \leq r \leq n)$．もし，Q_r よりも左に全称記号がないなら，M に現れない新しい定数 c をとり，M 中のすべての x_r を c で置き換えて，(Q_rx_r) を前置部から消去する．$Q_{s_1}, Q_{s_2}, \cdots, Q_{s_m}$ が Q_r の左に現れる全称記号 $(1 \leq s_1 < s_2 < \cdots < s_m < r)$ であるときには，M に現れない新しい $m-$変数関数記号 f をとり，M に現れるすべての x_r を $f(x_{s_1}, \cdots, x_{s_m})$ で置き換え，前置部から (Q_rx_r) を消去する．この処理を前置部の存在記号すべてに対して適用した結果得られる論理式をスコーレム標準形 (略して，標準形) という．また，存在記号を置き換えるときに用いられた定数や関数をスコーレム関数という．

例えば，論理式 $(\exists x)(\forall y)(\exists z)P(x, y, z)$ に対する標準形は，$(\forall y)P(a, y, f(y))$ である．

リテラルは，命題論理と同様に定義できる．以下では，節を改めて定義する．

定義 2.2.9

節 (clause) とは，リテラルの論理和 $(\vee)$ である．

リテラルを一つも含まない節を空節 (empty clause) と呼び，解釈によって真となるリテラルを含まないので，常に偽であるような節である．また，リテラルをただ一つもつ節を単位節 (unit clause) と呼ぶ．

節の集合 S は，S の要素であるすべての節の論理積 $(\wedge)$ であるとみなし，S に現れるすべての変数は，全称記号で束縛されているとする．このような節の集合を標準形を表す

節集合，または，単に節集合 (set of clauses) という．以後，本論文で節集合といった場合，標準形を表す節集合を意味するとする．

定理 2.2.1

論理式 F の標準形を表す節集合を S とすると，F が充足不能であることと，S が充足不能であることは同値である．

この定理より，任意の第一階述語論理式の充足可能性問題を解くことは，その論理式の標準形を表す節集合の充足可能性問題を解くことと等価であることが分かる．Davis-Putnam の手続きや導出原理などは，一般に，この標準形を表す節集合に対して適用される．

2.2.4 Herbrand の定理

節集合 S の充足可能性は，すべての定義域上のすべての解釈上での真偽を調べることで判定できる．しかし，すべての定義域上のすべての解釈を考えることは事実上不可能である．そこで，ある定義域 H が存在して，その定義域上のすべての解釈で偽であることと，S が充足不能であることが同値であれば都合がよい．実際に，Herbrand 空間と呼ぶこのような定義域が存在する．以下では，この Herbrand 空間の定義を与える．

定義 2.2.10

H_0 を節集合 S 中に現れる定数の集合であるとする．S に定数が現れないときは，ある定数 a について $H_0 = \{a\}$ であるとする．$n-$変数関数記号 $f(t_1, \cdots, t_n)$ の形をしたすべての項の集合と H_i との和集合を H_{i+1} とする．ただし，$f(t_1, \cdots, t_n)$ は，S 中のすべての $n-$変数関数記号であり，$t_i\ (i = 1, 2, \cdots)$ は H_i の要素である．このとき，H_i を $i-$レベル定数集合といい，H_∞ を S の **Herbrand 空間** (Herbrand universe) という．

例えば，$S = \{P(x), \sim P(a) \vee Q(f(y))\}$ であるとする．このとき，

$$H_0 = \{a\}$$

$$H_1 = \{a, f(a)\}$$

$$H_2 = \{a, f(a), f(f(a))\}$$

$$\vdots$$

$$H_\infty = \{a, f(a), f(f(a)), f(f(f(a))), \cdots\}$$

である．

また，$S = \{P(x), \sim P(y) \vee Q(z)\}$ であるとする．このとき，S は定数を含まないから，$H_0 = \{a\}$ である．また，S は関数記号も含まないから，$H_0 = H_1 = \cdots = H_\infty = \{a\}$ である．

定義 2.2.11

原子式が変数を含まないとき，その原子式を**基礎原子式** (ground atomic formula) という．

定義 2.2.12

節集合 S に含まれる節 C の**基礎例** (ground instance) とは，C 中のすべての変数を S の Herbrand 空間の要素で置き換えて得られる節をいう．

例えば，$S = \{P(x), \sim P(a) \vee Q(f(y))\}$ であるとすると，Herbrand 空間は，$\{a, f(a), f(f(a)), \cdots\}$ となるから，$P(a)$ や $P(f(a))$，$\sim P(a) \vee Q(f(f(a)))$ などは，S の節の基礎例である．

以下では，Herbrand 空間上の解釈で H 解釈と呼ばれるある特別な解釈を定義する．

定義 2.2.13

S を節集合，H を S の Herbrand 空間，I を S の H 上でのある解釈とする．I が S の H 解釈 (H interpretation) と呼ばれるのは I が以下の条件を満たすときである．

(1) I は S のすべての定数をそれ自身に対応づける．

(2) f を n−変数関数記号，$h_1, \cdots, h_n$ を H の要素とする．I において f は $(h_1, \cdots, h_n)$ を $f(h_1, \cdots, h_n)$ に写像する関数に割当てる．

例えば，以下の表 2.3 で示される解釈は，節集合 $S = \{P(x) \vee Q(f(x), a)\}$ に対する H 解釈である．

また，任意の解釈 I からそれに対応する H 解釈を定義することができる．

定義 2.2.14

定義域 D 上のある与えられた解釈 I に対して，I に対応する H 解釈 I^* (H interpretation I^* corresponding to I) とは，以下の条件を満足する H 解釈である．

$h_1, \cdots, h_n$ を H (S のエルブラン空間) の要素とする．すべての h_i を D の要素 d_i に対応づける．もし，$P(d_1, \cdots, d_n)$ が I のもとで $T(F)$ に割当てられていれば，$P(h_1, \cdots, h_n)$ を I^* のもとで $T(F)$ に割当てる．ただし，P は n−変数述語記号である．

このように，任意の定義域上の任意の解釈 I に対応する H 解釈をつくることができる．I に対応する H 解釈は一般に一意に定まらず複数存在するが，これらは以下の性質を満たす．

補題 2.2.1

ある定義域 D 上のある解釈 I が節集合 S を充足するなら，I に対応する H 解釈 I^* はすべて S を充足する．

定義域 (Herbrand 空間)　$H = \{a, f(a), f(f(a)), \cdots\}$

定数への割当て

a
a

関数への割当て

$f(a)$	$f(f(a))$	$\cdots$
$f(a)$	$f(f(a))$	$\cdots$

述語への割当て

$P(a)$	$Q(a,a)$	$P(f(a))$	$Q(a,f(a))$	$Q(f(a),a)$	$\cdots$
T	F	T	T	F	$\cdots$

表 2.3: 節集合 $S = \{P(x) \vee Q(f(x), a)\}$ に対する H 解釈

この補題より，以下の定理が証明可能である．

定理 2.2.2

節集合 S が充足不能であることと，S が S のすべての H 解釈のもとで偽であることとは同値である．

この定理は重要である．なぜなら，節集合 S が充足不能であることを示すには，Herbrand 空間上の解釈，実際には，H 解釈での真偽のみを調べるだけで十分であることを示しているからである．

以下は，ほとんどの定理自動証明手続きが基礎としている Herbrand の定理である．

定理 2.2.3

節集合 S が充足不能であることと，S の節の有限個の基礎例からなる充足不能な集合 S' が存在することは同値である．

第 3 章

定理自動証明に対するこれまでの手続き

本章では，定理自動証明に対するこれまでの方法として，最も一般的な導出原理について説明する．現在まで提案されている定理自動証明手続きのほとんどは導出原理に基づくものがほとんどである．

また，本論文で提案する手続きの基礎となっている Jeroslow の提案した部分例示手法について述べ，その問題点を指摘する．

3.1 導出原理

これまでの定理自動証明に関する研究において画期的な突破口を切り開いた導出原理は，1965 年に Robinson によって提案された [Robinson 65]．導出原理は，二つの節から一つの節を機械的に推論するための推論規則である．これは，二つの節に相補対 (符号のみ異なるリテラルの組) が存在するなら，その相補対を節から除去して，残りのリテラルをつないだ節を作り出すものである．節集合に対して，この操作を繰り返し適用する間に空節 (矛盾) を導き出すことができれば，その節集合は充足不能であることが証明されたことになる．

第一階述語論理における論理式は変数を含むため，相補対を求めるためには原子式の単一化という操作が必要である．以下では，代入，単一化といった概念を定義する．また，表現という用語を，項，項の集合，原子式，原子式の集合，リテラル，節，節の集合を総称して用いることにする．

定義 3.1.1

代入とは，$\{t_1/v_1, \cdots, t_n/v_n\}$ の形をした有限集合である．ここで，v_i は変数，t_i は v_i とは異なる項である．集合中のどの 2 つの要素も / の後にくる変数が同じではない．

定義 3.1.2

$\theta = \{t_1/v_1, \cdots, t_n/v_n\}$ を代入，E を表現であるとする．このとき，$E\theta$ は，E に現れる各変数 $v_i(1 \leq i \leq n)$ を同時に項 t_i に置き換えて得られるものであり，このことを E に θ を代入する (substitute) という．さらに，$E\theta$ を E の例 (instance) という．

例えば，$\theta = \{a/x\}$，$E = P(x,b) \vee \sim Q(x,y)$ に対して，$E\theta = P(a,b) \vee \sim Q(a,y)$ である．以下では，代入の合成を定義する．

定義 3.1.3

$\theta = \{t_1/x_1, \cdots, t_n/x_n\}$, $\lambda = \{u_1/y_1, \cdots, u_m/y_m\}$ を 2 つの代入とする．θ と λ の合成もまた代入であり，$\theta \circ \lambda$ で表される．$\theta \circ \lambda$ は集合

$$\{t_1\lambda/x_1, \cdots, t_n\lambda/x_n, u_1/y_1, \cdots, u_m/y_m\}$$

から，$t_j\lambda = x_j$ となる要素 $t_j\lambda/x_j$ と，y_i が $\{x_1, \cdots, x_n\}$ に含まれる要素 u_i/y_i とを取り除いて得られる集合である．

定義 3.1.4

2 つの項 s, t がある適当な代入 θ によって同一，すなわち $s\theta = t\theta$ となるとき，s と t は**単一化可能** (unifiable) であるといい，代入 θ を**単一化作用素** (unifier) という．この定義は，s と t が原子式のときもそのまま用いる．

定義 3.1.5

σ を 2 つの項 s と t の単一化作用素であるとする．s と t の任意の単一化作用素 θ に対して，ある代入 λ が存在して，$\theta = \lambda \circ \sigma$ となるとき，σ を**最汎単一化作用素** (most general unifier) という．

例えば，原子式 $P(a, y)$ と $P(x, b)$ は単一化が可能であり，最汎単一化作用素は，$\sigma = \{a/x, b/y\}$ である．すなわち，$P(a, y)\sigma = P(x, b)\sigma$ である．また，$P(a, y)$ と $P(a, f(y))$ は，これらを同一にする代入が存在しないので，単一化不可能である．

二つの原子式が単一化可能であるかどうかを判定し，単一化可能であるときには，最汎単一化作用素を求めるアルゴリズムが存在する．このアルゴリズムについて述べる前に，不一致集合を定義する．

定義 3.1.6

表現の空でない集合 W の不一致集合は，W のすべての表現が，左から順に比べて，同じ記号をもたなくなる最初の記号の位置を見つけて，その位置の記号から始まる部分表現を W の各表現から取り出し，この部分表現からできる集合を W の**不一致集合** (disagreement set of W) という．

W を単一化可能かどうかを判定したい二つの原子式の集合であるとする．このとき，以下のアルゴリズムは，単一化可能であるときに最汎単一化作用素を求めるものである．

アルゴリズム 3.1.1

step 1： $k=0$, $W_k=W$, $\sigma_k=\varepsilon$ とする．

step 2： W の要素が一つであれば停止し，σ_k が最汎単一化作用素である．そうでなければ，W_k の不一致集合 D_k を求める．

step 3： D_k 中に変数 v_k を項 t_k が存在し，v_k が t_k 中に現れないとき，**step 4** へ，このような v_k と t_k が存在しなければ停止し，W は単一化不可能である．

step 4： $\sigma_{k+1}=\sigma_k\circ\{t_k/v_k\}$, $W_{k+1}=W_k\{t_k/v_k\}$ とする．

step 5： $k=k+1$ として，**step 2** へ．

このアルゴリズムによって，単一化可能である場合に，必ず最汎単一化作用素が求まることは，以下の定理によって示される．

定理 3.1.1

W が単一化可能な表現の空でない有限集合であるとする．上のアルゴリズムは，必ず停止し，そのとき得られた σ が W の最汎単一化作用素になっている．

二つの原子式の単一化について定義したので，以下では，導出原理を定式化する．

定義 3.1.7 A が原子式であれば，リテラル A と $\sim A$ とは互いに**相補** (complement) であるという．また集合 $\{A, \sim A\}$ を**相補対** (complementary pair) と呼ぶ．

定義 3.1.8

節 C の二つ以上のリテラルが最汎単一化作用素 σ をもつとき，$C\sigma$ を C の**簡約形** (factor) という．

例えば，$C = P(x) \vee P(f(a)) \vee \sim Q(b)$ について，1 番目のリテラルと 2 番目のリテラルが最汎単一化作用素 $\sigma = \{f(a)/x\}$ をもつので，$C' = C\sigma = P(f(a)) \vee \sim Q(b)$ は C の簡約形である．

定義 3.1.9

共通な変数をもたない二つの節を $L \vee C_1$ と $\sim M \vee C_2$ とする．このとき，$(C_1 \vee C_2)\sigma$ をこの二つの節からの**二元導出形** (binary resolvent) という．ただし，σ は原子式 L と M の最汎単一化作用素である．

定義 3.1.10

C_1 と C_2 からの**導出形** (resolvent) とは，次の二元導出形のどれかである．

1. C_1 と C_2 からの二元導出形
2. C_1 と C_2 の簡約形からの二元導出形
3. C_1 の簡約形と C_2 からの二元導出形
4. C_1 の簡約形と C_2 の簡約形からの二元導出形

例えば, $C_1 = P(x) \vee P(f(a)) \vee \sim Q(b)$, $C_2 = Q(y) \vee R(f(y))$ のとき, $C_1' = P(f(a)) \vee Q(b)$ は C_1 の簡約形である. したがって, C_1' と C_2 からの二元導出形 $P(f(a)) \vee R(f(b))$ は, C_1 と C_2 からの導出形である.

定理 3.1.2

二つの節 C_1 と C_2 からの導出形は, C_1, C_2 の論理的帰結である.

定義 3.1.11

節集合 S が与えられたとき, 節の有限系列 $C_1, C_2, \cdots, C_k$ を S からの C の演繹 (deduction) という. ここで, C_i は S 中の節か, または C_i に先行する節からの導出形であり, $C_k = C$ である. S からの空節の演繹を S の反駁 (refutation), または S の証明 (proof) という.

導出原理に基づく方法は, 与えられた節集合 S から導出形を次々に生成し, 空節が導出されたなら, S は充足不能であることを結論するものである.

導出原理を用いた反駁の例を示す. 節集合 S が以下の 3 つの節からなるとする.

(1) $P(x) \vee \sim Q(y)$

(2) $\sim P(a)$

(3) $Q(b)$

この節集合に対して, 以下のような節集合が生成できる.

(4) $\sim Q(y)$ (1) と (2) から

(5) 空節 (3) と (4) から

このように, 空節が導出されたので, S は充足不能である. 充足不能な節集合からは必ず空節が演繹されることは, 以下の完全性に関する定理によって保証される.

定理 3.1.3 (導出原理の完全性)

節集合 S が充足不能であることと，節集合 S から空節が導出されることは同値である．

3.2 部分例示手法

部分例示手法は，1988 年に Jeroslow によって提案された [Jeroslow 88]．部分例示手法の基本的な概念は，定理証明を命題論理式の充足可能性問題を繰り返し解くことに帰着することである．その際，命題論理式を充足可能とするような真理値の割当てを利用することに特徴がある．しかしながら，Jeroslow の方法は関数記号を含まない論理式を対象としている．従って，導出原理に基づく方法と比べて，小さいクラスに含まれる問題にしか適用できない．以下では，Jeroslow の方法の詳細について述べる．

任意の第一階述語論理式は，冠頭標準形への変換，母式の連言標準形への変換，スコーレム関数の導入を適宜行うことで，その充足可能性を変えることなしに，$(\forall x_1)\cdots(\forall x_n)$ $B(x_1,\cdots,x_n)$ という形に変換できる．ここで，$B(x_1,\cdots,x_n)$ は変数 $x_1,\cdots,x_n$ を含み限定記号，関数記号を含まない論理式である．この論理式 $B(x_1,\cdots,x_n)$ は，節形式である必要はなく，任意の論理式でよい．以下では，簡単のために $(\forall x_1)\cdots(\forall x_n)B(x_1,\cdots,x_n)$ を $(\forall \vec{x})B(\vec{x})$ で表す．

$(\forall \vec{x})B(\vec{x})$ が充足可能であるのは，任意の $\vec{h} \in \vec{H}$ について，$B(\vec{h})$ を真とするような原子式への真理値の割当てが存在するときであり，かつそのときに限る，ということが知られている．ただし，$B(\vec{h})$ は $B(\vec{x})$ の変数 $\vec{x} = (x_1,\cdots,x_n)$ に $\vec{h} = (h_1,\cdots,h_n)$ をそれぞれ代入したもので，$\vec{H}$ はその成分が $B(\vec{x})$ に中に含まれる定数 (Herbrand 空間の要素) であるようなベクトルの集合である．

以下では，Jeroslow の手続きに用いる記号および用語の定義を与える．

3.2.1 諸定義

定義 3.2.1

論理式のすべての変数に定数を代入することを**完全例示** (complete instantiation) という．また，一部の変数に定数を代入することを**部分例示** (partial instantiation) という．

定義 3.2.2

論理式 F 中の変数に定数を代入して $\bar{F}$ が得られるとき，$\bar{F}$ を F の**改良** (refinement) と呼ぶ．

$(\forall \vec{x})B(\vec{x})$ の充足可能性を判定する Jeroslow の手続きは，$B_1 = B(\vec{x})$ として開始し，以下のような論理式の連言を扱う．

$$B_1 \wedge B_2 \wedge \cdots \wedge B_t$$

ただし，$B_i(1 < i \leq t)$ は $B(\vec{x})$ 中の変数に定数を代入することによって得られる論理式であり，ある $B_j(1 \leq j < i)$ の改良となっている．

定義 3.2.3

二つの原子式が変数のみ異なるとき，それらは**変種** (variant) であるという．

例えば，論理式 $P(a, y)$ と $P(a, z)$ は変種であり，$P(x, y)$ と $P(a, y)$ や，$P(a, y)$ と $P(x, b)$ などは変種ではない．

定義 3.2.4

変種である原子式にすべて同じ真偽を割当てる真理値の割当てを変種独立割当て (variant independent valuation) という.

以下の定理は，手続きの終了条件となる重要な定理である.

定理 3.2.1 (Jeroslow)

$B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが存在しなければ，$(\forall \vec{x})B(\vec{x})$ は充足不能である.

証明 3.2.1

$B_1 \wedge \cdots \wedge B_t$ 中のすべての変数に対して，ある一つの定数を代入した論理式を $B'_1 \wedge \cdots \wedge B'_t$ とおく．このとき，$B'_1 \wedge \cdots \wedge B'_t$ 中の原子式はすべて基礎原子式となっている．このことから，$B'_1 \wedge \cdots \wedge B'_t$ を真とする原子式への真理値の割当てが存在しないことを示せばよい.

$B_1 \wedge \cdots \wedge B_t$ 中の変種は，$B'_1 \wedge \cdots \wedge B'_t$ 中では同じ原子式になる．仮定より，$B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが存在しないから，$B'_1 \wedge \cdots \wedge B'_t$ を真とする原子式への真理値の割当ては存在しない．(証明終)

この定理によって，$B_1 \wedge \cdots \wedge B_t$ における t の増加に伴って，論理式が充足不能であるかどうかを検査する．以下では，論理式が充足可能である場合を検査するための定理について述べる．そのために，被覆と呼ばれる概念と，障害物と呼ばれる原子式の組について定義する.

定義 3.2.5

$B_1 \wedge \cdots \wedge B_t$ 中の $B_i(1 \leq i \leq t)$ が $B(\vec{x})$ に $\vec{h} \in \vec{H}$ を部分例示して得られるとき，B_i は $\vec{h}$ を被覆 (cover) するという．さらに，B_i の改良のすべてが $\vec{h}$ を被覆しないとき，B_i は $\vec{h}$ を直接被覆 (direct cover) するという.

例えば，$B_1 = B(x, y)$，$B_2 = B(a, y)$ とする．また，B_1 に含まれる定数の集合を $\{a, b\}$ とする．このとき，B_1 は $(a, a), (a, b), (b, a), (b, b)$ を被覆する．しかし，直接被覆するのは，$(b, a), (b, b)$ のみである．なぜなら，$(a, a), (a, b)$ は B_1 の改良である B_2 が被覆するからである．

定義 3.2.6

$B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが与えられているとする．このとき，以下の条件をすべて満たす二つの原子式 R と S を障害物 (blockage) という．ただし，R，S はそれぞれ B_i，B_j に含まれているとする．

(a) R と S は変種独立割当てにおいて異なる真理値が割当てられている．

(b) B_i，B_j にそれぞれ直接被覆される $\vec{h}_1$，$\vec{h}_2$ を $B(\vec{x})$ に例示すると，$B(\vec{h}_1)$，$B(\vec{h}_2)$ 中で R と S は同じ原子式になる．

この障害物という原子式の組は，以下の定理から明らかなように，論理式 $(\forall \vec{x})B(\vec{x})$ を充足可能とするための障害となるものであり，Jeroslow の手続きはこの障害物を取り除くことを目的としながら進められる．

定理 3.2.2 (Jeroslow)

$B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが与えられているとする．この割当てに障害物が存在しなければ，$(\forall \vec{x})B(\vec{x})$ は充足可能である．

証明 3.2.2

すべての $\vec{h} \in \vec{H}$ について，$B(\vec{h})$ を真とする真理値の割当てが存在することを示す．

ある $\vec{h}$ について，$\vec{h}$ を直接被覆する B_i が少なくとも一つは存在する．$B(\vec{h})$ 中のある原子式を R とする．B_i の中には $\vec{h}$ を例示することによって R になる原子式 P が存在する．

P が B_i の中で割当てられている真理値が R の真理値となるように割当てを行う．このように，$B(\vec{h})$ の中のすべての原子式に対して真理値の割当てを行うと $B(\vec{h})$ は真となる．与えられている変種独立割当てには障害物が存在しないから，このような割当てはすべての $B(\vec{h})$ 中で同じ原子式に対して異なる真理値を割当てることはない．ゆえに，これをすべての $\vec{h}$ について考えると，任意の $\vec{h}$ において $B(\vec{h})$ は真となる割当てが存在する．(証明終)

3.2.2 Jeroslow の手続きの概要

前節でも述べたように，論理式 $(\forall\vec{x})B(\vec{x})$ の充足可能性を判定する Jeroslow の手続きは，$B_1 = B(\vec{x})$, $t = 1$ として開始し，$B_1 \wedge \cdots \wedge B_t$ を真とするような変種独立割当てを求めることに基づいている．すなわち，そのような割当てが見つからなければ，定理 3.2.1 より，$(\forall\vec{x})B(\vec{x})$ は充足不能である．また，$B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが見つかったときには，その割当てに障害物が存在するかどうか調べる．もし，障害物が存在しなければ，定理 3.2.2より，$(\forall\vec{x})B(\vec{x})$ は充足可能である．障害物が存在すれば，その障害物を取り除くことを目的として，論理式 $B_1 \wedge \cdots \wedge B_t$ に対して，B_{t+1} を生成し，$B_1 \wedge \cdots \wedge B_t \wedge B_{t+1}$ として論理式を拡大する．この拡大を以下のように行う．

$B_1 \wedge \cdots \wedge B_t$ に存在する障害物を R, S とする．ただし，R は B_iに，S は B_j に含まれる原子式であるとする．障害物の定義より，R と S は単一化可能である．その最汎単一化作用素を σ とする．$i = j$ ならば，$B_{t+1} = B_i\sigma$, $t = t+1$ として論理式を拡大する．$i \neq j$ ならば，$B_{t+1} = B_i\sigma$, $B_{t+2} = B_j\sigma$, $t = t+2$ として論理式を拡大する．この結果，R と S は B_i, B_j の改良が $B_1 \wedge \cdots \wedge B_t$ に加わることで，障害物ではなくなる．

ある変種独立割当てに対して，障害物が複数存在することもあり得る．このとき，すべての障害物に対して上記のような論理式の拡大を行うか，一つの障害物に対してのみ行う

べきかについて，Jeroslow は特に言及していない．

$(\forall\vec{x})B(\vec{x})$ の充足可能性を判定する Jeroslow の手続きの概要を図 3.1 に示す．

3.2.3 問題点

前節では，論理式 $(\forall\vec{x})B(\vec{x})$ の充足可能性を判定する Jeroslow の手続きについて述べた．本節では，この手続きの計算上の問題点について述べる．この手続きにおいて，計算時間に影響を与える要因は，(1) 変種独立割当てを求めること，(2) 障害物を見つけること，(3) 全体の処理のループ回数 (図 3.1 の 3: ～ 8:)，である．以下ではこれらについて順に検討していく．

まず (1) について，変種独立割当てを求める際には，述語論理式を命題論理式として扱い，その充足可能性問題を解く．命題論理式の充足可能性問題は NP 完全問題である．その計算時間は，問題の形式や，命題の数，節の数などによってかなりの違いがみられるが，一般に問題のサイズ (命題の数や節の数) が小さいときには，かなり短い．ただし，問題のサイズが大きくなると極端に長くなる．Jeroslow の手続きでは，論理式 $B(\vec{x})$ を節形式に限定してはいないから，論理式 $B_1 \wedge \cdots \wedge B_t$ は節形式とは限らない．しかし，節形式でない命題論理式に対する充足可能性判定アルゴリズムは，現在まで，効率の良いものが提案されていない．したがって，結局 $B_1 \wedge \cdots \wedge B_t$ を節形式に変換してから解く，などといったことが行われる．このことから，節形式でなくてもよいという特徴を生かすことはできない．また，論理式の拡大が繰り返されるとともに，$B_1 \wedge \cdots \wedge B_t$ 中の原子式の数が極端に増えるため，先程の議論から変種独立割当てを求める際の計算時間が増える可能性が大きい．

次に，(2) について，障害物を見つけるためには，論理式 $B_1 \wedge \cdots \wedge B_t$ 中の任意の二つ

1: **begin**

2: $B_1 = B(\vec{x})$, $t = 1$ とする

3: **while** 充足可能性問題が解けていない **do**

4: **begin**

5: **if** $B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てが存在しない

then $(\forall \vec{x})B(\vec{x})$ は充足不能である (終了)

6: **if** $B_1 \wedge \cdots \wedge B_t$ を真とする変種独立割当てに障害物が存在しない

then $(\forall \vec{x})B(\vec{x})$ は充足可能である (終了)

7: 論理式 $B_1 \wedge \cdots \wedge B_t$ を拡大する

8: **end**

9: **end;**

図 3.1: Jeroslow の手続きの概要

の原子式の組で単一化を試みなければならない．t の値の増加とともに，$B_1 \wedge \cdots \wedge B_t$ 中の原子式の数が増えることから，この単一化を試みる回数は組み合わせ的爆発をおこすことになり，指数的に多くなる．このことは計算時間の増大につながると考えられる．

(3) について，処理のループの回数が増えれば計算時間が増えることは容易にわかる．すなわち，いかに論理式の無駄な拡大を抑えるかが問題になるが，これは難しい問題である．なぜなら，論理式を拡大するときにはそれが無駄かどうかがわからないからである．しかし，障害物の条件を厳しくすることができれば無駄な拡大を抑えることができる可能性は残されている．後に提案する手続きにおいては，論理式を節形式に制限することから障害物の条件を厳しくしている．

このように，Jeroslow の手続きにおいては，論理式 $B_1 \wedge \cdots \wedge B_t$ の増大が，計算時間に大きな影響を与えていることがわかる．また，この Jeroslow の手続きに関する実験などの報告も今までのところなされていない．

以上のような問題点を改善することを目的とし，次章で新しい手続きを提案する．

第 4 章

部分例示手法に基づく定理自動証明手続き

本章では，部分例示手法に基づく定理自動証明手続きを提案する．本手続きは，Jeroslow の方法を基礎としているが，以下の点で異なる．

(1) 関数記号を含む論理式を扱うことが可能である．

(2) 節形式の論理式に対して適用する．

(1) によって，Jeroslow の方法よりもはるかに大きなクラスに含まれる問題を扱うことが可能となる．(2) によって一般性が失われないことは，任意の論理式をその充足可能性を変えることなしに節形式に変換が可能であることから明らかである．

従って，本手続きは導出原理に基づく方法と同様のクラスの問題に対して適用可能なことになる．

以下では，本論文で提案する手続きの詳細について述べる．さらに，本手続きが完全であることの証明を行う．

4.1 諸定義

定理証明が以下のような m 個の節からなる節集合 S に対して，S の充足可能性を調べることに帰着できることはすでに述べた．

$$S = \{C_1, \cdots, C_m\}$$

本論文で提案する定理証明手続きでは，Herbrand の定理に基づき，S が充足不能であることを充足不能な S の節の基礎例を求めることによって示す．以下では，本手続きの記述で必要となる用語や記号について定義する．

定義 4.1.1

節 C が節 C_i の例であるなら，C_i は C を被覆するという．また，C_i が C を被覆し，かつ C_i 自身を除く C_i の例が S に含まれないなら，C_i は C を直接被覆するという．

定義 4.1.2

変種であるすべての原子式に同じ真理値を割り当てる真理値の割当てを変種独立割当てという．

定義 4.1.3

V_k が S_k を真とする変種独立割当てであるとする．また，F，G を l_F，l_G にそれぞれ現れる原子式であるとする．但し，l_F，l_G は S_k 中の節 C_i，C_j にそれぞれ現れるリテラルである．このとき，以下の条件をすべて満足する原子式 F，G の組を純障害物といい，$\langle F, G\rangle_{i,j}$ と表す．

(1) F と G は V_k で異なる真理値が割り当てられている．

(2) l_F と l_G の両方が V_k で真である．

(3) $F\sigma = G\sigma$ となる最汎単一化作用素 σ が存在し，$F\sigma$ のある例が C_i' と C_j' の両方に現れる．但し，C_i'，C_j' は C_i，C_j によってそれぞれ直接被覆されているとする．

これらの定義を例を用いて説明する．

以下の C_1 と C_2 の二つの節からなる節集合 S_1 を考える．また，C_2' は C_2 の基礎例である．

$$C_1 = P(f(a)) \vee \sim Q(x),$$

$$C_2 = \sim P(y) \vee Q(z),$$

$$C_2' = \sim P(f(a)) \vee Q(a),$$

ただし，P，Q は述語記号，x，y は変数，a は定数である．

このとき，C_2 は C_2' を直接被覆している．なぜなら，C_2' は C_2 の例であり，S_1 中には C_2 以外の C_2 の例が含まれていないからである．(図 4.1).

S_1 を真とする変種独立割当ての一つとして，以下の割当て V_1 がある．

$$C_1 = \underbrace{P(f(a))}_{T} \vee \sim \underbrace{Q(x)}_{T},$$

$$C_2 = \sim \underbrace{P(y)}_{F} \vee \underbrace{Q(z)}_{T},$$

これは，原子式 $P(f(a))$，$Q(x)$，$P(y)$，$Q(z)$ に対してそれぞれ，T，T，F，T を割当てることを意味する．$Q(x)$ は $Q(z)$ の変種であるから，これらの原子式には同じ真理値 T が割当ててある．

この V_1 に対して，S_1 中に純障害物 $\langle P(f(a)), P(y)\rangle_{1,2}$ が存在する．なぜなら，以下の 3 つの条件を満たしているからである．

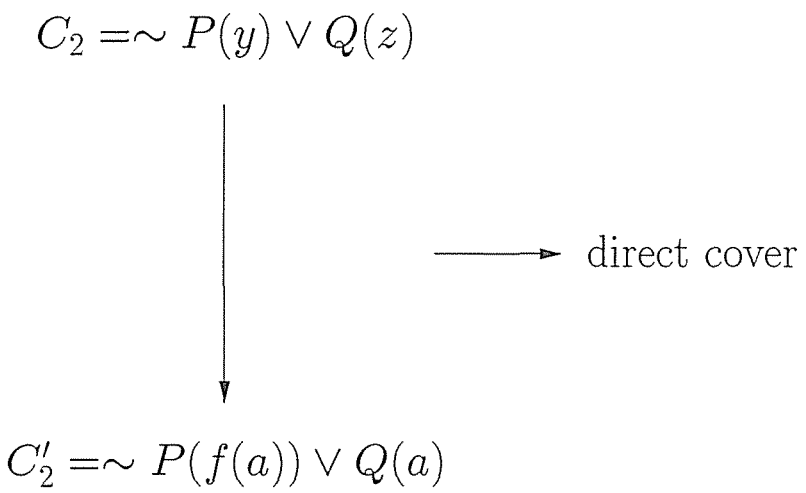

図 4.1: C_2 と C_2' の関係

- V_1 中で，$P(f(a))$ は T，$P(y)$ は F が割当てられている．
- 二つのリテラル $P(f(a))$ と $\sim P(y)$ は共に V_1 で真となる．
- $P(f(a))\sigma = P(y)\sigma$ となる最汎単一化作用素 $\sigma = \{f(a)/y\}$ が存在し，$P(f(a))$ は C_1 が直接被覆している $C_1' = P(f(a))\vee \sim Q(a)$ と C_2 が直接被覆している $C_2' = \sim P(f(a)) \vee Q(a)$ の両方に現れる．

S_1 に $C_3 = C_2\sigma = \sim P(f(a)) \vee Q(z)$ を加えた集合を S_2 とする．このとき，S_2 中では，C_2' は C_2 に直接被覆されなくなる．なぜなら，C_2 の例である $C_2\sigma$ が S_2 中に含まれているからである．(図 4.2).

4.2 手続きの詳細

節集合 S の充足可能性を調べる本手続きの概要を図 4.3 に示す．尚，本手続きにおける k 回目の繰り返し (図 4.3 の Step 4 ~ Step 10) を第 k ループと呼び，第 k ループで扱う節集合を S_k とする．ただし，$S_1 = S$ である．

本手続きは，以下の基本的な三つの処理を節集合 S の充足可能性が判定されるまで繰り返す．

1. [変種独立割当て] S_k を真とする変種独立割当て V_k を求める．(Step 5)
2. [純障害物の発見] V_k に対して，S_k 中のすべての純障害物を求める．(Step 7)
3. [純障害物の解消] S_k 中のすべての純障害物を解消した S_k の拡大 S_{k+1} を生成する．(Step 9)

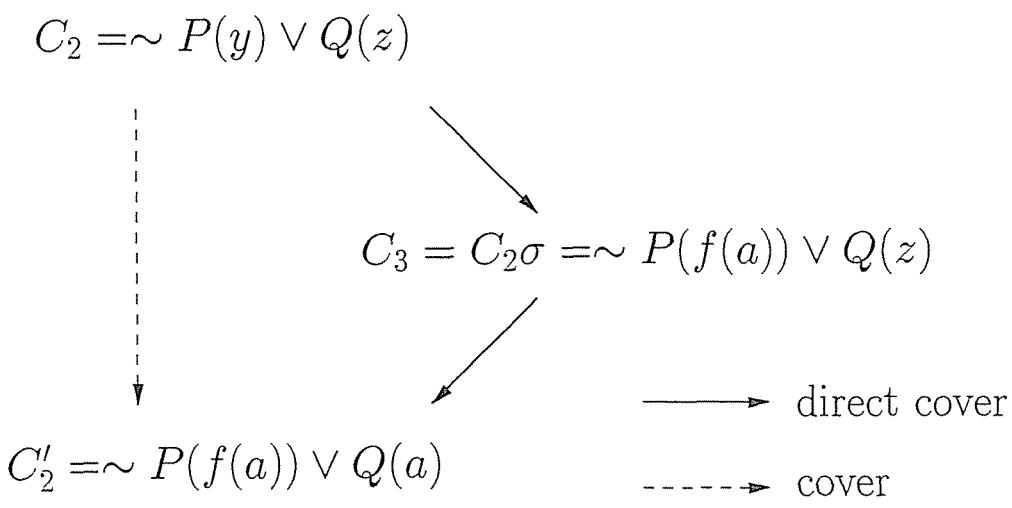

図 4.2: C_2, C_2', C_3の関係

```
1:  begin
2:      k ← 1,  S_k ← S ;
3:      loop
4:          V_k ← φ,  B_k ← φ ;
5:          V_k ← S_k を真とする変種独立割当て ; (図 4.4)
6:          if   V_k = φ   then   S は充足不能 ; (exit)
7:          B_k ← V_k に対して S_k 中に存在するすべての純障害物の集合 ; (図 4.5)
8:          if   B_k = φ   then   S は充足可能 ; (exit)
9:          S_{k+1} ← B_k 中のすべての純障害物の解消の結果 ; (図 4.6)
10:         k ← k + 1 ;
11:     endloop ;
12: end ;
```

図 4.3: 定理証明手続きの概要

上記の 2. および 3. に関して，論理式に関数記号が現れない場合は S の節の基礎例が有限であるため，第 k ループにおいて一つの純障害物を発見し，その一つの純障害物の解消のみを行うことでも完全性が保証できる．一方，論理式に関数記号が現れる場合は S の節の基礎例は有限ではないため，第 k ループにおいて一つの純障害物を発見しそれのみを解消する方法では，充足不能となる基礎例を生成する保証がない．しかし，本手続きのように，V_k に対する S_k 中のすべての純障害物を発見し解消することにより，後で述べるように完全性を保証することができる．

以下では，1. ～ 3. について順次詳細に述べる．

4.2.1　変種独立割当て

本手続きの第 k ループにおいて，S_k を真とする変種独立割当てを求める．この手続きを図 4.4 に示す．

この処理はまず S_k 中のすべての原子式を以下の規則を用いて命題変数に変換し，命題論理式の節集合 S_k^p を得る．

- ある原子式 F の変種であるすべての原子式は F と同じ命題変数にする．
- ある原子式 F の変種でないすべての原子式は F と異なる命題変数にする．

次に，この命題論理式の節集合 S_k^p の充足可能性問題を解く．もし，S_k^p が充足不能なら，明らかに S_k を真とする変種独立割当ては存在しない．S_k^p を真とする解釈が存在すれば，S_k 中の原子式の真理値を，S_k^p 中の対応する命題変数の真理値とする．これが，S_k を真とする変種独立割当てである．

1: **begin**
2: 　S_k を命題論理式の節集合 S_k^p へ変換する ;
3: 　$I \leftarrow \phi$;
4: 　$I \leftarrow S_k^p$ を充足する解釈 ;
5: 　**if**　$I = \phi$　**then**　exit ;
6: 　$V_k \leftarrow I$ から得られた変種独立割当て ;
7: **end**

図 4.4: 変種独立割当ての発見手続き

S_k を真とする変種独立割当ては一般には複数存在すると考えられるが，その中のどの割当てを用いても手続きの完全性は失われない．しかし，証明に要する繰り返し回数 k は，用いる変種独立割当てによって大きく影響されることが予想され，高速に証明を行うためには，変種独立割当てを求める戦略が重要となる．

例 4.2.1

以下のような節集合を考える．

$$S_k = \{P(x), \sim P(a) \vee Q(f(y), b), \sim Q(f(z), b)\}$$

$P(x)$ と $P(a)$ には，変種となる原子式が S_k 中に存在しないので，それぞれ異なる命題変数 p_1，p_2 とする．また，$Q(f(y), b)$ と $Q(f(z), b)$ は互いに変種であるので，同じ命題変数 p_3 とする．従って，節集合 S_k は以下の節集合 S_k^p に変換できる．

$$S_k^p = \{p_1, \sim p_2 \vee p_3, \sim p_3\}$$

ここで，解釈 $\{p_1, \sim p_2, \sim p_3\}$ の下で S_k^p は真となり，S_k を真とする変種独立割当て V_k を得ることができる．

$$S_k = \{\underbrace{P(x)}_{T}, \sim \underbrace{P(a)}_{F} \vee \underbrace{Q(f(y), b)}_{F}, \sim \underbrace{Q(f(z), b)}_{F}\}$$

4.2.2 純障害物の発見

S_k を真とする変種独立割当て V_k が見つかれば，V_k に対して，S_k 中に存在するすべての純障害物を求めなければならない．この手続きを図 4.5 に示す．

S_k 中の異なる二つの節 C_i，C_j 中にそれぞれ現れる原子式 F，G が純障害物となるためには，F と G が単一化可能であることは定義から明らかである．しかし，実際に F と

```
1:  begin
2:    for   S_k 中のすべての C_i, C_j (i ≠ j) の組   do
3:      for   C_i, C_j にそれぞれ現れるすべての原子式の組 F, G   do
4:        if   ( F と G は V_k で反対の真理値を割当てられている   and
5:               l_F と l_G の両方が V_k で真   and
6:               F と G が単一化可能 )
7:        then begin
8:             σ ← F と G の最汎単一化作用素 ;
9:             if   ( C_iσ が S_k の要素でない   and
10:                   C_jσ が S_k の要素でない)
11:            then   B_k ← B_k ∪ {⟨F, G⟩_{i,j}} ;
12:       end ;
13: end ;
```

図 4.5: すべての純障害物の発見手続き

G の単一化を試みるのは F，G が V_k において異なる真理値を割り当てられていて，l_F と l_G が共に V_k で真となるときのみである．ただし，l_F，l_G は F，G がそれぞれ現れているリテラルである．この結果，単一化可能であれば，F と G の最汎単一化作用素 σ を用いて，$C_i\sigma$ と $C_j\sigma$ が共に S_k の要素であるかを調べる．すなわち，$F\sigma$ の例の現れる節が C_i と C_j に直接被覆されているかどうかを調べる．このように，S_k 中のすべての異なる二つの節の組について純障害物を求め，その集合を B_k とする．

例 **4.2.2**

以下の節 C_1，C_2，C_3 からなる節集合 S_k を真とする以下の変種独立割当て V_k が与えられている．

$$C_1 = \underbrace{P(x)}_{T}$$

$$C_2 = \sim \underbrace{P(a)}_{F} \vee \underbrace{Q(f(y),b)}_{F}$$

$$C_3 = \sim \underbrace{Q(f(z),b)}_{F}$$

この V_k に対して，C_1 中の $P(x)$ と C_2 中の $P(a)$ は純障害物である．なぜなら，$P(x)$ と $P(a)$ は共に V_k で異なる真理値を割り当てられており，それらが現われるリテラル $P(x)$ と $\sim P(a)$ は V_k で共に真である．さらに，$P(x)\sigma = P(a)\sigma$ となる最汎単一化作用素 $\sigma = \{a/x\}$ が存在し，$P(x)\sigma$ が C_1 の直接被覆している節 $P(a)$ と C_2 の直接被覆している節 $\sim P(a) \vee Q(f(a),b)$ の両方に現れるからである．

S_k 中には V_k に対して，純障害物は他に存在しないから，純障害物の集合は $B_k = \{\langle P(x), P(a)\rangle_{1,2}\}$ である．

4.2.3　純障害物の解消

純障害物の集合 B_k 中のすべての純障害物を解消するために S_k の拡大 S_{k+1} を生成する．この手続きを図 4.6 に示す．

$\langle F, G\rangle_{i,j}$ を S_k 中の V_k に対する純障害物の一つ，すなわち，B_k の要素の一つとし，σ を F と G の最汎単一化作用素であるとする．本手続きは，節 $C_i\sigma$ と $C_j\sigma$ を生成し，S' に保存していく．ただし，$C_i\sigma = C_i$ のとき，または，$C_j\sigma = C_j$ のときは，それぞれ $C_i\sigma$，$C_j\sigma$ の生成は明らかに無駄となるため行わない．この処理を B_k のすべての要素について行った後，S' と S_k の和集合を S_{k+1} とする．S_{k+1} が S_k の拡大になっていることは容易にわかる．従って，S の拡大になっている．

生成した S_{k+1} には $C_i\sigma$ と $C_j\sigma$ が存在する．この二つの節において，$F\sigma$ と $G\sigma$ は $F\sigma = G\sigma$ であるから，純障害物にはなり得ない．また，F と G も S_{k+1} 中では純障害物にはならない．なぜなら，$F\sigma$ の例が現れる任意の基礎例は，S_{k+1} において $C_i\sigma$ か $C_j\sigma$ に直接被覆されるからである．

例 4.2.3

例 4.2.2 における節集合 S_k を真とする変種独立割当て V_k に対して，

$$B_k = \{\langle P(x), P(a)\rangle_{1,2}\}$$

を解消するために，$C_1\sigma = P(a)$ を生成する．また，$C_2\sigma$ に関しては $C_2\sigma = C_2$ であるから生成しない．そして，$S_k \cup \{P(a)\}$ を S_{k+1} とする．

1: **begin**
2: 　$S' \leftarrow \phi$
3: 　**for**　B_k 中のすべての要素 $\langle F, G\rangle_{i,j}$　**do begin**
4: 　　$\sigma \leftarrow F$ と G の最汎単一化作用素 ;
5: 　　$S' \leftarrow S' \cup \{C_i\sigma, C_j\sigma\}$;
6: 　**end;**
7: 　$S_{k+1} \leftarrow S_k \cup S'$;
8: **end;**

図 4.6: S_k の拡大 S_{k+1} の生成手続き

4.3 完全性の証明

本手続きが正当でかつ完全であることを示すために，節集合 S が充足不能であることと S_p を真とする変種独立割当てが存在しないような有限の p が存在することが同値であることを示す．以下の定義および補題は定理 4.3.1 を示すために必要である．

定義 4.3.1

$T = \{D_1, \cdots, D_n\}$ を充足不能な S の節の基礎例の集合であるとする．また，$U_k = \{E_1, \cdots, E_n\}$ を多重集合とする．但し，各 E_i $(i = 1, \cdots, n)$ は D_i を直接被覆する S_k 中の節であるとする．

補題 4.3.1

T を充足不能な S の節の基礎例の集合であるとする．また，S_k を真とする変種独立割当て V_k が存在するとする．このとき，V_k に対して U_k 中の節に純障害物 $\langle F, G\rangle_{i,j}$ が存在する．ただし，F，G はそれぞれ U_k 中の節 E_i，E_j $(i \neq j)$ に現れるとする．

証明 4.3.1

この補題が成り立たないとすると，U_k 中には純障害物が存在しない．このとき，以下のような割当てを考える．U_k の節の任意の基礎例 E_i' について，E_i' を直接被覆する U_k 中の節 E_i が存在し，E_i は V_k において真である．従って，E_i には V_k で真となるリテラル l が存在する．また，E_i' は E_i の例であるから，E_i' には l の例 l' が存在する．このとき，l に現れる原子式に割り当てられている真理値を l' に現れている基礎原子式の真理値とする．この割当てによって，U_k の節の任意の基礎例 E_i' を真とすることができる．ただし，ある基礎例 E_i' 中で真理値を割り当てられた基礎原子式 P が別の基礎例 E_j' 中で異なる真

理値を割り当てられる場合がある．この場合，上記の割当て方法から，P が現れるリテラルは V_k において，E'_i と E'_j で共に真となっている．従って，P に異なる真理値が割り当てられるのは，E_i，E_j 中に例示すると P となる二つの原子式 F，G が存在し，F，G がそれぞれ現れるリテラル l_F，l_G が V_k において共に真となるときのみ起こる．すなわち，純障害物 $\langle F, G\rangle_{i,j}$ が存在するときである．しかし仮定より，V_k において純障害物は存在しないのでこのような場合は生じない．従って，このような割当てにより U_k 中の節の基礎例からなる任意の節集合は充足可能となる．T は，U_k 中の節の基礎例からなる集合であるから T は充足可能であり仮定に反する．(証明終)

定義 4.3.2

節 C に対して，$\|C\|$ はその節での定数，関数記号，述語記号の出現の個数を表すとする．また，節集合 $S = \{C_1, \cdots, C_m\}$ に対して，$\|S\|$ を $\|S\| = \|C_1\| + \|C_2\| + \cdots + \|C_m\|$ と定義する．

補題 4.3.2

T を充足不能な S の節の基礎例の集合であるとする．また，S_k を真とする変種独立割当て V_k が存在するとする．このとき，本手続きにおける S_k の拡大により，S_{k+1} に対して $\|U_k\| < \|U_{k+1}\|$ となる．

証明 4.3.2

補題 4.3.1 より，U_k 中に純障害物 $\langle F, G\rangle_{i,j}$ が存在する．ただし，F，G は U_k 中の節 E_i，E_j $(i \neq j)$ にそれぞれ現れるとする．このとき，σ を F と G の最汎単一化作用素とすると，S_k の拡大により，S_{k+1} には $E_i\sigma$ と $E_j\sigma$ が含まれる．E_i，E_j に直接被覆されていた節は，それぞれ $E_i\sigma$，$E_j\sigma$ に直接被覆されるから，$U_{k+1} = U_k \cup \{E_i\sigma, E_j\sigma\} - \{E_i, E_j\}$

となる．また，σ は，F か G に現れる変数を変数以外の記号 (定数記号，関数記号) で置き換える代入であるから，$\|E_i\| + \|E_j\| < \|E_i\sigma\| + \|E_j\sigma\|$ である．従って，$\|U_k\| < \|U_{k+1}\|$ である．(証明終)

定理 4.3.1

S が充足不能であることと S_k を真とする変種独立割当てが存在しないような有限の p が存在することは同値である．

証明 4.3.3

($\Rightarrow$)

S が充足不能であるとする．Herbrand の定理より，S の節の有限個の基礎例からなる充足不能な節集合 T が存在する．このとき，ある k において，S_k を真とする変種独立割当てが存在しない場合は証明は完了しているので，S_k を真とする変種独立割当て V_k が存在する場合を考える．補題 4.3.1より，V_k に対して S_k 中には純障害物が存在するから，補題 4.3.2より，S_k の拡大により，S_{k+1} に対して $\|U_k\| < \|U_{k+1}\|$ となる．T に含まれる各節における定数，関数記号，述語記号の出現の個数の総和は有限であるから，ある有限の p において $T = U_p$ となる．T は充足不能な S の節の基礎例の集合であるから，U_p を真とする変種独立割当ては存在しない．U_p のすべての要素は S_p の要素であるから，S_p を真とする変種独立割当ては存在しない．

($\Leftarrow$)

ある有限の p に対して，S_p を真とする変種独立割当てが存在しないとする．S の Herbrand 空間の要素の一つを a とし，節集合 S_p 中のすべての変数に a を代入した節集合を S'_p とする．このとき，S_p 中のすべての変種は S'_p 中ですべて同じ基礎原子式になる．従って，仮定より，S_p を真とする変種独立割当ては存在しないから，S'_p は充足不能である．な

ぜなら，もし，S'_p が充足可能なら，その真理値の割当てから S_p を真とする変種独立割当てを得ることができるからである．明らかに，S'_p 中のすべての節は S 中の節の基礎例になるから，S'_p には，充足不能な S 中の節の有限個の基礎例が存在し，Herbrand の定理より，S は充足不能である．(証明終)

この定理から，以下の系は容易に導ける．この系は本手続きの終了条件の一つになっている．

系 4.3.1

任意の変種独立割当てに対して S_k 中に純障害物が存在しなければ，S は充足可能である．

4.4　例　題

前節までで，部分例示手法に基づく定理自動証明手続きについて述べた．本節では，この手続きの処理の流れを二つの例題を用いて説明する．例 4.4.1 は，充足不能である節集合に対して，例 4.4.2 は，充足可能な節集合に対して適用した例である．それぞれの例において，変種独立割当ての求め方，純障害物の発見や解消，などの手続きの詳細については前節までで述べているので省略し，ここでは全体の流れや問題点について説明する．

例 4.4.1

三つの節 C_1, C_2, C_3 からなる節集合 S が以下のように与えられているとする．

$$C_1 = P(x)$$

$$C_2 = \sim P(a) \vee Q(f(y), b)$$

$$C_3 = \sim Q(f(z), b)$$

ただし，a，b は定数，x，y，z は変数，f，は関数記号，P，Q は述語記号であるとする．

$S_1 = S$ を真とする以下の変種独立割当て V_1 が存在する．

$$C_1 = \underbrace{P(x)}_{T}$$

$$C_2 = \sim \underbrace{P(a)}_{F} \vee \underbrace{Q(f(y), b)}_{F}$$

$$C_3 = \sim \underbrace{Q(f(z), b)}_{F}$$

V_1 に対して，S_1 中には一つの純障害物が存在する．すなわち，$B_1 = \{\langle P(x), P(a)\rangle_{1,2}\}$ である．

図 4.3 の Step 9 では，純障害物 $\langle P(x), P(a)\rangle_{1,2}$ の原子式の最汎単一化作用素 $\sigma_1 = \{a/x\}$ を用いて，以下の節 C_4 を生成する．$C_2\sigma_1 = C_2$ であるから，$C_2\sigma_1$ は生成する必要はない．

$$C_4 = C_1\sigma_1 = P(a)$$

B_1 中のすべての純障害物について処理したので，新しく生成した節の集合 $\{C_4\}$ と S_1 との和集合を S_2 とする．

第 2 ループにおいて，S_2^p は以下のようになる．

$$S_2^p = \{p_1, \sim p_2 \vee p_3, \sim p_3, p_2\}$$

この S_2^p は充足不能なので，S_2 を真とする変種独立割当ては存在しない．従って，定理 4.3.1 より S は充足不能である．

例 4.4.2

以下の二つの節からなる節集合 S を考える．

$$C_1 = P(f(a))$$

$$C_2 = \sim P(x) \vee P(f(x))$$

ただし，x は変数，f は関数記号，P は述語記号である．$S_1 = S$ を真とする以下の変種独立割当て V_1 が存在する．

$$C_1 = \underbrace{P(f(a))}_{T}$$

$$C_2 = \sim \underbrace{P(x)}_{T} \vee \underbrace{P(f(x))}_{T}$$

この V_1 では，すべての原子式が真に割り当てられているため，純障害物は存在しない．従って，系 4.3.1 より，S は充足可能である．

しかし，以下のような S_1を真とする別の変種独立割当て V_1' も存在する．

$$C_1 = \underbrace{P(f(a))}_{T}$$

$$C_2 = \sim \underbrace{P(x)}_{F} \vee \underbrace{P(f(x))}_{T}$$

この割当てに対しては，純障害物 $\langle P(f(a)), P(x)\rangle_{1,2}$ が存在する．この純障害物を解消するために節 $C_3 = C_2\{f(a)/x\} = \sim P(f(a)) \vee P(f(f(a)))$ を生成する．このとき，C_1，C_2，C_3 を真とする以下の変種独立割当てに対しても純障害物 $\langle P(f(f(a))), P(x)\rangle_{2,3}$ が存在するため，処理を続けなければならない．このように，どの変種独立割当てを用いるかによって計算効率が大きく変わる．

$$C_1 = \underbrace{P(f(a))}_{T}$$

$$C_2 = \sim \underbrace{P(x)}_{F} \vee \underbrace{P(f(x))}_{T}$$

$$C_3 = \sim \underbrace{P(f(a))}_{T} \vee \underbrace{P(f(f(a)))}_{T}$$

第 5 章

手続きの効率化

前章では，部分例示手法に基づく定理自動証明手続きを提案した．また，その計算効率が変種独立割当てに強く依存することを例題を用いて示した．

本章では，本論文で提案した手続きを効率化するため，変種独立割当てを求める方法に着目し，いかに高速に求めるか，いかに有効な割当てを求めるか，について検討する．そして，導出原理に基づく方法との比較実験を通して，本論文で提案した手続きの有効性，及び，特徴について検討する．

5.1 効率化の要点

本論文で提案した手続きは，以下の二つの処理に分けることができる．

(1) 節集合 S_k を真とする変種独立割当てを求めるために命題論理式の充足可能性問題 (通常，SAT と呼ばれる) を解く．

(2) 節集合 S_k 中のすべての純障害物を発見し解消する．

従って，これらの処理を効率化することが手続きの効率化につながる．

(1) に関して，SAT をいかに高速に解くかは用いるアルゴリズムに強く依存する．

(2) に関して，すべての純障害物を発見するためには，原子式のすべての組に対して，純障害物となるかどうかの検査が必要である．この際，ある原子式の組が純障害物となるかどうかは前章の例題で示したように，変種独立割当ての影響を受ける．

これらの考察から，本論文で提案した手続きの効率化のためには，変種独立割当てに関して，いかに高速に求めるか，いかに有効な割当てを求めるか，が重要な検討課題であることは明らかである．

以下では，この 2 つの処理の効率化について順に検討する．

5.1.1 高速な解法

SAT は，NP–完全問題の一つとして有名であり，SAT に対してこれまで多くの研究成果が報告されている．SAT の解法として，命題論理においては非常に効率的であるとされている有名な Davis-Putnam の手続きがある [Davis 60] [Blair 86]．Davis-Putnam の手続きの概要を以下に示す．

Davis-Putnam の手続きは，節集合 S の充足可能性を判定するものであり，以下の 4 つ

の規則からなる.

[トートロジー規則]

節集合 S 中のトートロジーであるすべての節を取り除いた節集合を S' とすると，S の充足可能性と S' の充足可能性は変わらない.

[純リテラル規則]

節集合中のあるリテラル l は，$\sim l$ が他の節にあらわれないとき，純一リテラルと呼ばれる．もし，l が純一リテラルであれば，l を含むすべての節を S から取り除いた節集合 S' を考える．このとき，S の充足可能性と S' の充足可能性は変わらない.

[1 リテラル規則]

節集合 S 中に単位節 l が存在すれば，S から l を含むすべての節を取り除き，さらに $\sim l$ をすべて取り除いた節集合 S' を考える．このとき，S の充足可能性と S' の充足可能性は変わらない.

[分割規則]

節集合 S 中に現れる原子論理式 P について，$S \cup \{P\}$ と $S \cup \{\sim P\}$ をそれぞれ S'，S'' とする．S' と S'' の充足可能性を調べ，共に充足不能なら S は充足不能である．このとき，P は分枝変数と呼ばれる.

これらの規則を元の節集合 S に対して用いながら，ある S' が空となれば元の節集合 S は充足可能であり，S' 中に空節が現れれば S は充足不能と判定される.

トートロジー規則と純リテラル規則は，計算時間がかかる割にあまり有効でないことが知られており，一般には用いられない [Dowling 84]．従って，Davis-Putnam の手続きは，本質的に 1 リテラル規則，分割規則の二つの規則からなると考えてさしつかえない.

Davis-Putnam の手続きの計算効率は，分割規則における分枝変数の選択に強く依存することが報告されている．従って，分枝変数選択のための様々な戦略が提案されている．比較的有効とされているものに，Jeroslow-Wang の戦略，Algorithm A における戦略がある [Blair 86] [Dubois 93]．Jeroslow-Wang の戦略は，ある段階の節集合 S' 中に現れる原子式に対して，以下の評価関数を適用する．

$$w(S', j, v) = \sum_{k=1}^{R} N_{jkv} \frac{1}{2^k}$$

ただし，N_{jkv} は S' 中で p_j が正で現れている $(v = 1)$，または，p_j が負で現れているような $(v = 0)$，k 個のリテラルからなる節の個数を表す．また，R は S' 中で最も長い節のリテラルの個数である．

もし，(j^*, v^*) が $w(S', j, v)$ を最大とする場合，もし，$v^* = 0$ なら，p_j を分枝変数として選択し，$S' \cup \{p_{j^*}\}$ の充足可能性を優先して調べる．もし，$v^* = 1$ なら，$\sim p_j$ を分枝変数として選択し，$S' \cup \{\sim p_{j^*}\}$ の充足可能性を優先して調べる．

Algorithm A における戦略は，上記の評価関数を以下の関数で置き換えたものである．

$$w(S', j, v) = \sum_{k=1}^{R} N_{jkv} \frac{2^{2(R-1)}}{2^{2(k-1)}}$$

これらの戦略は，どちらも短い節に多く出現している原子式を優先して分枝するような戦略である．このことは，1 リテラル規則によって取り除かれる節を多くし，1 リテラル規則を適用した結果，さらに単位節が多く現れることを期待するものである．

Davis-Putnam の手続きにおける分枝変数の選択が計算効率にどれほどの影響を与えるかを示すために，Jeroslow-Wang の戦略，Algorithem A における戦略，及び，ランダム戦略を比較した実験を行なった．ランダム戦略とは，ランダムに選択された変数を分枝変数とすることを優先するものである．

実験は IRIS Indigo^2 (CPU:R4000, Memory:64 Mbytes) 上で行なった．プログラムは，Algorithm A のプログラム (ASAT) の分枝変数選択の部分を変更したものを用いた．テスト問題として，ランダム 3-SAT と呼ばれる問題を生成しそれらを用いた．ランダム 3-SAT 問題は，すべての節が 3 つのリテラルからなるランダム問題で，非常に難しい問題として知られ，SAT の解法アリゴリズの評価に良く用いられるものである．しかし実際は，原子式 (命題変数) の数と節の数の比が約 4.3 のとき，ランダム 3-SAT 問題は非常に難しく，それ以外のときはそれほど難しくはないという報告もされている [Crawford 93] [Mitchell 92]. 従って，ここではその比が約 4.3 の問題を用いている．

結果を表 5.1 に示す．表 5.1 における “Random”, “J-W” and “A” では，分枝変数の選択方法としてそれぞれ，ランダム戦略，Jeroslow-Wang の戦略，Algorithm A における戦略を用いた場合の各問題に対する分枝した回数を示している．ただし，各サイズに対して問題を 10 個生成しこれらに適用した場合の分枝の回数の平均をとっている．

表 5.1: 分枝変数選択の戦略の効果

vars	clauses	Random	J-W	A
50	214	538.3	76.6	56.4
80	344	13233.1	169.3	111.3
100	430	52855.3	277.9	171.0
120	516	849262.0	1492.1	1020.0

この結果から，Algorithm A における戦略，及び，Jeroslow-Wang の戦略はランダム戦略に比べて非常に効率が良いことがわかる．つまり，よい分枝変数選択の戦略を伴う Davis-

Putnam の手続きは非常に効率がよい．また，Algorithm A における戦略は Jeroslow-Wang の戦略に比べて効率が良いことがわかる．

最近，SAT の解法として GSAT などの局所探索法が注目を集めている [Selman 92] [Selman 93]．GSAT やその変種が，N-queen 問題や難しいランダム問題に対して，かなりの成果を示していることが報告されている．

GSAT は，原子式へのランダムな解釈を初期値とし，その解釈が節集合を充足しないとき，

- ある原子式の真理値を反転 (真と偽を入れ換える) させたとき，充足する節のの数が最大になるような原子式の真理値を反転させる，

という操作をあらかじめ決められた MAX-FLIPS 回実行する．この間に節集合を充足する解釈が得られれば停止する．もし，充足する解釈が得られなければ，初期値の解釈を新たに生成し同様の操作をあらかじめ決められた MAX-TRIES 回繰り返すものである．GSAT の概要を図 5.1 に示す．

明らかに，GSAT は完全性を持たない．つまり，局所探索法であるために，ある節集合が充足不能であることを示すことはできない．このことは，本論文で提案した定理証明手続きには不向きであることを意味する．

従って，いかに高速に解くか，ということに関しては Algorithm A の分枝変数選択の戦略をもった Davis-Putnam の手続きが現状で最も適していると考えられる．

```
1:  begin
2:    for   i := 1   to   MAX-TRIES
3:    begin
4:      T ← ランダムな解釈 ;
5:      for   j := 1   to   MAX-FLIPS
6:      begin
7:        if   T が S を充足する   then return   T ;
10:       p ← その値を変更することで，S 中の充足する節の数
11:           が最も多くなる変数 ;
12:       T ← p の値を変更した解釈 ;
13:     end ;
14:   end ;
15: end ;
```

図 5.1: GSAT の概要

5.1.2　有効な割当ての発見

前章の例題を通して，ある原子式の組が純障害物かどうかは求められた変種独立割当てに依存し，それが手続きの計算効率に影響を与えることを述べた．従って，ここでは，有効な変種独立割当てを求めるための戦略について検討する．

純障害物の定義から，すべての原子式に同じ真理値が割当てられている変種独立割当てが求まれば，明らかに純障害物は存在しない．そして，このとき手続きは停止する．また，あるループで発見される純障害物の数が多くなると，純障害物の解消によって次のループでの節の数が多くなり，その後の処理にかかる計算時間が長くなることが予測される．これらのことから，あるループでの純障害物の数をなるべく少なくすることが有効であろうと考えられる．

このことから，以下のような戦略が考えられる．

> 真理値の固定されていないすべての原子式の真理値を偽とした解釈を初期値として，分割規則，または，1 リテラル規則によって真理値が真と固定される原子式の数を少なくするような戦略．

1 リテラル規則において，真理値の固定される原子式の数は節の数に強く影響を受ける．従って，上記の戦略を行うためには，分割規則によって分枝する際，なるべく多くの節が取り除かれることが必要である．このような戦略は，Algorithm A の分枝変数選択の戦略そのものである．このことは，いかに有効な変種独立割当てを求めるか，に関しても Algorithm A の分枝変数選択のための戦略が有効であることを示唆している．

従って，本論文で提案した手続きにおいて，変種独立割当てを求めるために命題論理式の充足可能性問題を解く方法としては，Algorithm A の分枝変数選択のための戦略を用い

た Davis-Putnam の手続きを用いることが有効と考えられる.

5.2 実験および考察

本論文で提案した手続きの有効性を示すために，いくつかの問題に適用した実験を行った．前節で述べたように，変種独立割当てを求めるためには，Algorithm A の戦略を用いた Davis-Putnam の手続きを用いた．実験は，Sun SPARC Station IPX (主記憶 32 Mbyte, 28.5 MIPS) を用いて行い，プログラムは，Sun Common Lisp により開発した．また，比較のために，OTTER の計算時間も併せて示した.

OTTER は，McCune によって開発された導出原理を基礎とした定理証明プログラムであり，様々な手法，戦略などがサポートされている著名な定理証明システムの一つである [McCune 89]．OTTER のプログラムは C 言語で書かれ，多種のコンピュータで動作するシステムである．実験では，Ver. 3.0.3 を用い，戦略等はすべての問題においてオート (auto) に設定した.

結果を表 5.2，表 5.3 に示す．表中の Time は各問題に対する実行時間 (単位 [秒]) を，Loop は変種独立割当てを求めた回数を表す．また，表中の「 – 」は，30 分経過しても解が得られなかったため，または，節の爆発的増加により計算続行が不能になったために，計算を中止したことを表す.

表 5.2 において使用した問題は，文献 [Pelletier 86] 中の等式を含まない節形式の論理式のみからなる問題 35 ～ 46 (ただし, 38 は問題の不備のため除いた) である．これらの問題の充足可能性は，問題 42 と 46 のみ充足可能であり，残りの問題はすべて充足不能である．表中では，それぞれ T および F で示す．表 5.3 における問題は，表 5.2 で使用した問題

表 5.2: 実験結果 1

problem	SAT	本手続き		OTTER
		Time	Loop	Time
35	F	0.04	2	0.04
36	F	1.23	4	0.08
37	F	0.12	3	0.09
39	F	0.03	1	0.06
40	F	0.60	5	0.17
41	F	0.20	3	0.08
42	T	0.03	1	0.27
43	F	2.34	9	1.36
44	F	0.07	2	0.12
45	F	1.12	4	0.13
46	T	0.58	4	0.18

表 5.3: 実験結果 2

problem	SAT	本手続き		OTTER
		Time	Loop	Time
35′	T	0.02	1	0.03
36′	T	9.90	6	0.07
37′	T	0.50	5	198.60
39′	T	0.02	1	0.03
40′	T	0.03	1	0.03
41′	T	0.03	1	0.05
42′	T	0.03	1	-
43′	T	0.04	1	0.09
44′	T	0.04	1	0.07
45′	T	0.20	2	0.10
46′	T	0.57	3	0.10

に対して，最後の節を一つ取り除いた節集合を用い，問題番号に「′」を付加して表した．表 5.3 に示すようにすべての問題の充足可能性は充足可能であった．

これらの表から，本論文で提案した手続きはすべての問題が解けていることがわかる．一方，OTTER は，問題 42′ が 30 分経過後も停止しなかった．また，問題 37′ もかなりの時間がかかっていることがわかる．問題 42′ や 37′ は，以下の例のような特徴を問題に含んでいる．

例 **5.2.1**

以下の節 C_1，C_2 からなる節集合を考える．

$$C_1 = P(f(a)),$$

$$C_2 = \sim P(x) \vee P(f(x)),$$

ただし，a は定数，x は変数，f は関数記号，P は述語記号である．この節集合に対して，以下のような変種独立割当てが存在する．

$$C_1 = \underbrace{P(f(a))}_{T},$$

$$C_2 = \sim \underbrace{P(x)}_{T} \vee \underbrace{P(f(x))}_{T},$$

この割当てに対して純障害物は存在しないことは簡単にわかる．なぜなら，すべての原子式に同じ真理値が割当てられているからである．従って，本論文では 系 4.3.1 より，この節集合は充足可能であると結論される．

一方，OTTER では，まず最初に C_1 と C_2 の導出節 $C_3 = P(f(f(a)))$ を生成する．さらに，この C_3 は C_2 との導出により，導出節 $C_4 = P(f(f(f(a))))$ を生成する．以下，同様に導出節 $P(f(f(f(f(a)))))$，$\cdots$，を無限に生成し停止することはない．

この例のように，OTTER のような導出原理に基づく方法では困難な問題で，本論文で提案した手続きでは容易な問題が存在することがわかる．

次に，Algorithm A の戦略を用いた Davis-Putnam の手続きと，ランダム戦略を用いた Davis-Putnam の手続きとの比較によって，Algorithm A の戦略が実際に有効であるかどうかを調べる実験を行なった．

結果を表 5.4，表 5.5 に示す．ランダム戦略では乱数を用いているため，各問題に対して乱数の種を変えて 10 回適用した．表中の solve は，10 回のうち 10 分以内に手続きが停止した回数を表し，unsolve は 10 分以内に停止しなかった回数を表している．Random はランダム戦略を用いた場合の計算時間とループの回数を表している．ただし，10 分以内に停止したもののみの平均をとっている．比較のために，Algorithm A の戦略を用いた場合の表 5.2 と表 5.3 の値を"本手続き"に示した．

この結果から，ランダム戦略を用いた場合は 10 分以内に停止しない問題がいくつかあるのに対して，Algorithm A の戦略を用いた場合はすべての問題が停止しているのがわかる．従って，Algorithm A の戦略が比較的有効な変種独立割当てを求めていることがわかる．

表 5.4: 実験結果 3

Problem	Solve	Unsolve	Random		本手続き	
			Time	Loop	Time	Loop
35	10	0	0.01	2.0	0.04	2
36	10	0	1.21	4.0	1.23	4
37	10	0	0.49	2.8	0.12	3
39	10	0	0.00	1.0	0.03	1
40	10	0	4.63	6.9	0.60	5
41	10	0	0.14	3.0	0.20	3
42	8	2	1.25	1.3	0.03	1
43	10	0	2.52	7.2	2.34	9
44	10	0	0.02	2.0	0.07	2
45	10	0	0.68	3.7	1.12	4
46	10	0	0.31	3.0	0.58	4

表 5.5: 実験結果 4

Problem	Solve	Unsolve	Random		本手続き	
			Time	Loop	Time	Loop
35′	10	0	0.00	1.0	0.02	1
36′	8	2	19.15	8.5	9.90	6
37′	10	0	0.12	3.0	0.50	5
39′	10	0	0.00	1.0	0.02	1
40′	10	0	0.00	1.0	0.03	1
41′	10	0	0.01	1.1	0.03	1
42′	10	0	0.02	1.6	0.03	1
43′	10	0	0.03	1.3	0.04	1
44′	10	0	0.02	1.4	0.04	1
45′	10	0	0.67	2.7	0.20	2
46′	2	8	0.03	1.0	0.57	3

第 6 章

結　論

本論文は，人工知能の分野における基本的な問題である定理自動証明に対して，命題論理の充足可能性問題を繰り返し解くことにより定理証明を行うリダクション手法の一つである部分例示手法に基づいた手続きを新しく構築し，その有用性について検討したものである．また，その手続きが完全性をもつことの証明を与えている．すなわち，本論文で提案した手続きは，与えられた節集合が実際に充足不能であれば，そのことを有限ステップで証明することができる．

6.1　各章の要約

6.1.1　第 1 章

第 1 章は序論であり，定理自動証明に対するこれまでの研究について述べた．また，本研究の目的を明確にした．

6.1.2　第 2 章

第 2 章では，記号論理の基礎として，命題論理および第一階述語論理について説明した．その中で，リダクション手法にとって重要な命題論理式の充足可能性問題についてや，現在まで提案されている多く定理自動証明手続きが基礎としている Herbrand の定理などについて説明した．

6.1.3　第 3 章

第 3 章では，定理自動証明に対するこれまでの研究として，最も一般的な手法である導出原理について説明した．また，本論文で提案した手続きが基礎としている部分例示手法について述べた．その中で Jeroslow が提案した方法について説明し，その問題点を明確にした．

6.1.4　第 4 章

第 4 章では，部分例示手法に基づいた定理自動証明手続きを新しく構築し，その詳細について述べた．また，本論文で提案した手続きが定理自動証明手続きがもつべき重要な性質の一つである完全性をもつことの証明を与えた．さらに，例題を通して手続きの全体の流れを説明し，手続きの計算効率が変種独立割当てに強く依存することを示した．

6.1.5　第 5 章

第 5 章では，本論文で提案した手続きの効率化について検討した．効率化のためには，変種独立割当てを求めるアルゴリズムに関して，いかに高速に解くか，いかに有効な割当て

求めるか，が重要であることを示した．そして，これらを満足する方法として，Algorithm A の戦略をもった Davis-Putnam の手続きを適用し，実験によってその有効性を示した．また，導出原理に基づく方法として有名な OTTER との比較実験により，OTTER が困難な問題で，本論文で提案した手続きでは容易な問題が存在することを示した．

6.2 まとめ

本論文では，定理自動証明に関する研究として以下のことを行った．

1. 定理証明を命題論理式の充足可能性問題を繰り返し解くことによって行うリダクション手法の一つである部分例示手法について，Jeroslow の方法の問題点を検討した．
2. 部分例示手法に基づいた新しい定理自動証明手続きを構築した．
3. その手続きの計算効率が変種独立割当てを求めるアルゴリズムに強く依存することを示し，そのアルゴリズムとして Algorithm A の戦略を用いた Davis-Putnam の手続きが有効であることを示した．
4. 導出原理に基づく方法として有名な定理自動証明システム OTTER との比較実験により，OTTER では困難な問題で，本論文で提案した手続きでは容易に証明可能な問題が存在することを示した．

今後の検討課題として以下のことが残されていると考えられる．

- 有効な変種独立割当てを求める戦略のより詳細な検討．
- 導出原理に基づく方法における支持集合戦略などの技法の取り入れ．

- より大きな問題 (節数の多い) への対応.

- 他のリダクション手法である Hyper-Linking Starategy との関連性について検討.

謝　辞

本論文作成に当たり御指導を頂いた北海道大学工学部情報工学専攻調和系工学分野大内東教授，有益な御助言を頂いた北海道大学工学部情報工学専攻伊達惇教授，和田充雄教授，栗原正仁助教授，様々な面で御協力を頂いた調和系工学分野の三田村保助手に厚く御礼申し上げます．特に，札幌医科大学保健医療学部大柳俊夫助教授には研究の詳細な内容はもとより，熱心な御指導，御助言を頂いた．深く感謝申し上げます．

最後に，両親に深く感謝します．

参考文献

[大柳 92] 大柳，大内. “命題論理充足可能性問題に対する定量的アプローチ”，平成 4 年度日本 OR 学会秋期研究発表会アブストラクト集 (1992)

[大柳 93] 大柳，山本，大内：“陰的列挙法に基づく SAT アルゴリズム”，情報処理学会論文誌, Vol. 34, No. 12, pp. 2464–2473(1993)

[大柳 94] 大柳，山本，大内：“命題論理における β 節充足可能性問題とその解法”，電気学会論文誌 C, Vol. 114-C, No. 7/8, pp. 796–804(1994)

[志村 83] 志村.：“機械知能論”，人工知能シリーズ, Vol. 1, 昭晃堂 (1983)

[長尾 83] 長尾，淵.：“論理と意味”，岩波講座　情報科学, Vol. 7, 岩波書店 (1983)

[山本 94] 山本，大柳，大内.：“部分例示手法に基づく充足可能性判定手続き”，電子情報通信学会論文誌, Vol. J77-A, No. 11, pp. 1577–1584(1994)

[山本 95] 山本，大柳，大内.：“部分例示に基づく定理自動証明”，電子情報通信学会論文誌, Vol. J78-A, No. 7, pp. 864–871(1995)

[Blair 86] Blair C. E., Jeroslow R. G. and Lowe J. K.：“Some Results and Experiments in Programming Techniques for Propositional Logic”, Computers and Operations

Research, Vol. 13, No. 5, pp. 633–645(1986)

[Chang 73] Chang C. -L. and Lee R. C. -T. : "Symbolic Logic and Mechanical Theorem Proving", ACADEMIC PRESS(1973)

[Crawford 93] Crawford J. M. and Auton L. D. : "Experimental Results on the Crossover Point in Satisfiability Problems", In Proceedings of AAAI-93, pp. 21–27(1993)

[Davis 60] Davis M. and Putnam H. : "A Computing Procedure for Quantification Theory", Journal of the ACM, Vol. 7, pp. 201–215(1960)

[Dowling 84] Dowling W. F. and Gallier J. H. : "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae", Journal of Logic Programming, Vol. 3, pp. 267–284(1984).

[Dubois 93] Dubois O., Andre P., Boufkhad Y. and Carlier J. : "Can a Very Simple Algorithm be Efficient for Solving the SAT Problem?", via anonymous ftp from `dimacs.rutgers.edu`: / pub/ challenge/ sat/ contributed/ dubois/ dubois.tex (1993)

[Gallo 89] Gallo G. and Urbani G. : "Algorithm for Testing Satisfiability of Propositional Formulae", Journal of Logic Programming, Vol. 7, pp. 45–61(1989)

[Gent 92] Gent I. P. and Walsh T. : "Towards an Understanding of Hill-Climbing Procedures for Sat", In Proceedings of AAAI-93, pp. 28–33(1992)

[Haken 85] Haken A. : "The Intractability of Resolution", Theoretical Computer Science, Vol. 39, pp. 297–308(1985)

[Hooker 88] Hooker J. N. : "A Quantitative Approach to Logical Inference", Decision Support Systems, Vol. 4, pp. 45–69(1988)

[Jeroslow 88] Jeroslow R. G. : "Computation-Oriented Reductions of Predicate to Propositional Logic", Decision Support Systems, Vol. 4, pp. 183–197(1988)

[Lee 92] Lee S. J. and Plaisted D. A. : "Eliminating Duplication with the Hyper-Linking Strategy", Journal of Automated Reasoning, Vol. 9, pp. 25–42(1992)

[McCune 89] McCune W. W.: "Otter 3.0 Reference Manual and Guide", Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois (1989).

[Mitchell 92] Mitchell D., Selman B. and Levesque H. : "Hard and Easy Distributions of Sat Problems", In Proceedings of AAAI-93, pp. 459–465(1992)

[Robinson 65] Robinson J. A. : "A Machine-Oriented Logic based on the Resolution Principle", Journal of the ACM, Vol. 12, pp. 23–41(1965)

[Pelletier 86] Pelletier F. J. : "Seventy-Five Problems for Testing Automatic Theorem Provers", Journal of Automated Reasoning, Vol. 2, pp. 191–216(1986)

[Selman 92] Selman B., Levesque H. and Mitchell D. : "A New Method for Solving Hard Satisfiability Problems", In Proceedings of AAAI-92, pp. 440–446(1992)

[Selman 93] Selman B. and Kautz H. A. : "An Empirical Study of Greedy Local Search for Satisfiability Testing", Proceedings of AAAI-93(1993)

[Urquhart 87] Urquhart A. : "Hard Example for Resolution", Journal of the ACM, Vol. 34, pp. 209–219(1987)

[Yamamoto 94] Yamamoto M., Ohyanagi T. and Ohuchi A. : "Computational Results for Satisfiability Problems", Proceedings of The Third Conference of APORS(The Association of Asian-Pacific Operational Research Societies within IFORS), pp. 538–545(1994)

[Yamamoto 95] Yamamoto M., Ohyanagi T. and Ohuchi A. : "Theorem Proving based on the Partial Instantiation Technique", Proceedings of the 34th SICE(The Society of Instrument and Control Engineers) Annual Conference, pp. 1183–1186(1995)

付　録 A

プログラムリスト

A.1　Davis-Putnam の手続き

```
(proclaim '(optimize (compilation-speed 0) (safety 0) (speed 3)))

;;; The following part is the Davis-Putnam procedure.
;;; The Davis-Putnam procedure is an algorithm for solving satisfiability
;;; problem (SAT) that determines whether a given formula is satisfiable.
;;; A predicate DP returns a list of variables which must be true if the
;;; formula CLSS is satisfiable. Otherwise, it returns symbol NIL.

(defun dp (clss)
  (dp-aux (tautology-rule clss) '(0)) )

(defun varlist (num)
  (cond ((equal num 0) (list 0))
        (t (cons num (varlist (- num 1))))))

(defun dp-aux (clss val)
  (let ((unit-cls (unit-clause clss)))
    (cond ((null clss) val)
          ((member nil clss) nil)
          (unit-cls (if (> 0 unit-cls) (setf val (remove (- unit-cls) val)))
                    (dp-aux (one-literal-rule unit-cls clss) val))
          (t (let ((branch-var (branch-variable clss)))
               (or (dp-aux (cons (list branch-var) clss) val)
                   (dp-aux (cons (list (- branch-var)) clss) val))) ))))

;;; Tautology is a valid formula. A clause is tautology iff the clause
;;; contains both a variable and its negation.
;;; For example, a clause (1 2 -2) is a tautology.

(defun tautology-p (cls)
  (cond ((null cls) nil)
        ((member (- (first cls)) (rest cls)) t)
        (t (tautology-p (rest cls))) ))
```

```
(defun tautology-rule (clss)
  (remove-if #'tautology-p clss) )

(defun unit-clause (clss)
  (cond ((null clss) nil)
        ((null (rest (first clss))) (first (first clss)))
        (t (unit-clause (rest clss))) ))

(defun one-literal-rule (unit-cls clss)
  (one-literal-rule-aux unit-cls clss nil) )

(defun one-literal-rule-aux (unit-cls clss result)
  (cond ((null clss) result)
        ((member unit-cls (first clss))
         (one-literal-rule-aux unit-cls (rest clss) result))
        ((member (- unit-cls) (first clss))
         (one-literal-rule-aux unit-cls
              (rest clss) (cons (remove (- unit-cls) (first clss)) result)))
        (t (one-literal-rule-aux unit-cls
                (rest clss) (cons (first clss) result))) ))

;;; The following functions are based on the Asat Algorithm.
;;; A function BRANCH-VARIABLE returns the branching-varible that
;;; we should branch on first. A minus sign indicates that we should
;;; branch on the negation of the varible.

(defun func (lit-num lit-max)
  (/ (expt 2.0 (* 2.0 (- lit-max 1))) (expt 2.0 (* 2.0 (- lit-num 1)))) )

(defun max-lit (clss)
  (cond ((null clss) 0)
        (t (max (length (first clss)) (max-lit (rest clss)))) ))

(defun branch-variable (clss)
  (let ((lit-max (max-lit clss))
        (max-point 0)
        (result 0))
    (dotimes (var *new-num* result)
      (let ((temp (sign (+ var 1) clss lit-max)))
        (cond ((> (second temp) max-point)
               (setf max-point (second temp))
               (setf result (first temp))))) ) ))

(defun sign (var clss lit-max)
  (let ((point1 0)
        (point0 0))
    (if (> (dolist (cls clss point1)
             (if (member var cls)
               (setf point1 (+ point1 (func (length cls) lit-max)))))
           (dolist (cls clss point0)
             (if (member (- var) cls)
```

```
        (setf point0 (+ point0 (func (length cls) lit-max))))))
(list var point1)
(list (- var) point0))))
```

A.2 部分例示手法に基づく定理自動証明手続き

```
;;; The following function is the main function of our theorem
;;; proving procedure based on the partial instantiation technique.
;;; A function SAT take a set of clauses and returns a symbol T if
;;; a given set of clauses is satisfiable, otherwise it returns a
;;; symbol NIL.

(defun sat (clss)
  (init)
  (sat-aux (lclss-first clss)) )

(defun sat-aux (lclss)
  (setf *loop* (+ *loop* 1))
  (let ((val (dp (trans-lclss lclss))))
    (print (- (length val) 1))
    (if (null val) (format t "~% NIL (unsatisfiable)  loop=~a" *loop*)
        (let ((blockages (find-blockages lclss val)))
          (if (null blockages) (format t "~% T (satisfiable)  loop=~a" *loop*)
              (sat-aux (extend lclss blockages))) ))) )

(defun extend (lclss blockages)
  (let ((result lclss)
        (new (remove-duplicates (extend-aux lclss blockages) :test #'variant-p)))
    (dolist (cls new result)
      (push (make-lcls (new-label) (rename-var cls)) result)) ) )

(defun extend-aux (lclss blockages)
  (let ((result nil))
    (dolist (blockage blockages result)
      (setf result (append (extend-aux2 lclss blockage) result))) ) )

(defun add (lclss1 lclss2)
  (let ((result lclss2))
    (dolist (lcls lclss1 result)
      (if (not (member (lcls-body lcls) lclss2 :test #'variant-p :key #'second))
          (push lcls result)) ) ) )

(defun extend-aux2 (lclss blockage)
  (let ((l1 (first blockage)) (l2 (second blockage))
        (r (third blockage)) (s (fourth blockage))
        (mgu (fifth blockage)))
    (if (change-p mgu r)
        (if (change-p mgu s) (list (generate-cls lclss l1 mgu)
                                   (generate-cls lclss l2 mgu))
                             (list (generate-cls lclss l1 mgu)))
        (list (generate-cls lclss l2 mgu)) ) ) )

(defun generate-cls (lclss label mgu)
  (let ((cls (lcls-body (find label lclss :key #'first)))
        (subst (filter-mgu (vars-in lclss) mgu)))
```

```
    (setf (gethash label cover)
          (remove-duplicates (cons subst (gethash label cover)) :test #'equal))
    (apply-subst subst cls) ) )

(defun change-p (mgu e)
  (if (variant-p e (apply-subst mgu e)) nil t) )

(defun variant-p (e1 e2)
  (if (or (atomic-p e1)
          (atomic-p e2)) (var-equal e1 e2)
      (and (variant-p (first e1) (first e2))
           (variant-p (rest e1) (rest e2))) ) )

(defun var-equal (x y)
  (if (var-p x) (var-p y)
      (equal x y)) )

(defun find-blockages (lclss val)
  (find-blockages-aux lclss val nil) )

(defun find-blockages-aux (lclss val result)
  (if (null lclss) result
      (let ((blocks (find-blockages-aux2 (first lclss) (rest lclss) val result)))
        (find-blockages-aux (rest lclss) val blocks)) ) )

(defun find-blockages-aux2 (lcls lclss val result)
  (dolist (cl lclss result)
    (let ((blocks (find-blockages-c lcls cl val)))
      (if blocks (setf result (append blocks result)))) ) )

(defun find-blockages-c (lcls1 lcls2 val)
  (let ((result nil)
        (cls1 (lcls-body lcls1)) (l1 (lcls-label lcls1))
        (cls2 (lcls-body lcls2)) (l2 (lcls-label lcls2)))
    (dolist (lit1 cls1 result)
      (dolist (lit2 cls2)
        (let ((mgu (blockage-lit lit1 lit2 val)))
          (cond ((or (null mgu) (eq mgu 'fail)) nil)
                ((unfit-blockage cls1 cls2 l1 l2 mgu) nil)
                (t (push (list l1 l2 lit1 lit2 mgu) result)) )) )) ))

(defun unfit-blockage (c1 c2 l1 l2 mgu)
  (let* ((cov1 (gethash l1 cover))
         (cov2 (gethash l2 cover))
         (mgu1 (filter-mgu (vars-in c1) mgu))
         (mgu2 (filter-mgu (vars-in c2) mgu)))
    (cond ((null mgu1) (if (include-uni mgu2 cov2) t nil))
          ((null mgu2) (if (include-uni mgu1 cov1) t nil))
          (t (if (or (include-uni mgu1 cov1)
                     (include-uni mgu2 cov2))
                 t nil)) )) )
```

```
(defun filter-mgu (vars mgu)
  (cond ((null mgu) nil)
        ((and (member (first (first mgu)) vars)
              (not (var-p (second (first mgu)))))
         (cons (first mgu) (filter-mgu vars (rest mgu))))
        (t (filter-mgu vars (rest mgu))) ) )

(defun instance-of (e1 e2)
  (let ((mgu (unify e1 e2)))
    (and (not (equal mgu 'fail))
         (equal e1 (apply-subst mgu e2))) ) )

(defun include-in-aux (u1 u2)
  (and (equal (first u1) (first u2))
       (instance-of (second u1) (second u2))) )

(defun include-in (u1 u2)
  (if (null u1) t
      (and (include-in-aux (first u1) (first u2))
           (include-in (rest u1) (rest u2))) ) )

(defun include-uni (u cov)
  (cond ((null cov) nil)
        ((include-in u (first cov)) t)
        (t (include-uni u (rest cov))) ) )

(defun blockage-lit (lit1 lit2 val)
  (if (and (opposite-p lit1 lit2) (true-p lit1 val) (true-p lit2 val))
    (unify (get-atom lit1) (get-atom lit2))) )

(defun opposite-p (lit1 lit2)
  (if (negate-p lit1) (not (negate-p lit2))
                      (negate-p lit2)) )

(defun true-p (lit val)
  (let ((truth-value (member (gethash (sub-to-const (get-atom lit)) table) val)))
    (if (negate-p lit) (not truth-value)
                       (not (not truth-value))) ) )

(defun var-p (e)
  (typecase e
    (symbol (char= #\? (char (symbol-name e) 0)))
    (t nil)))

(defun const-p (e) (and (not (var-p e)) (atom e)))

(defun atomic-p (e)
  (or (const-p e) (var-p e)))

(defun composite-p (e)
  (not (atomic-p e)))
```

```
(defun sub-to-const (e)
  (cond ((const-p e) e)
        ((var-p e) 'a)
        (t (cons (sub-to-const (first e))
                 (sub-to-const (rest e)))) ) )

(defun return-num (e)
  (let* ((ground-form (sub-to-const e))
         (value (gethash ground-form table)))
    (if (null value) (setf (gethash ground-form table) (new-num)) value)))

(defun trans-lit (literal)
  (if (negate-p literal) (- (return-num (get-atom literal)))
                         (return-num literal)) )

(defun trans-lclss (lclss)
  (mapcar #'(lambda (lcls)
    (mapcar #'(lambda (literal) (trans-lit literal)) (lcls-body lcls))) lclss) )

(defun negate-p (literal)
  (if (equal 'not (first literal)) t nil) )

(defun get-atom (literal)
  (if (negate-p literal) (second literal) literal) )

;;; The following part is a collection of functions for initialization.

(defvar *new-num* 0)
(defvar *new-label* 0)
(defvar *loop* 0)

(defun init ()
  (setf *new-num* 0)
  (setf *new-label* 0)
  (setf *loop* 0)
  (setf table (make-hash-table :test #'equal))
  (setf cover (make-hash-table))
)

(defun new-var ()
  (gensym "?") )

(defun new-label ()
  (setf *new-label* (+ *new-label* 1))
  *new-label* )

(defun new-num ()
  (setf *new-num* (+ *new-num* 1))
  *new-num* )

(defun vars-in (e)
  (cond ((var-p e) (list e))
```

```
        ((const-p e) '())
        (t (union (vars-in (first e))
                  (vars-in (rest e)))) ))

(defun make-fresh-subst (vars)
  (if (null vars) '()
      (cons (list (first vars) (new-var))
            (make-fresh-subst (rest vars)) )))

(defun rename-var (cls)
  (apply-subst (make-fresh-subst (vars-in cls)) cls))

(defun make-lcls (label body)
  (list label body) )

(defun lcls-label (lcls)
  (first lcls) )

(defun lcls-body (lcls)
  (second lcls) )

(defun lclss-first (clss)
  (mapcar #'(lambda (cls)
              (make-lcls (new-label) (rename-var cls)))
          clss) )

;;; The following function UNIFY is the unification algorithm.
;;; UNIFY accepts two expressions and returns the most general
;;; unifier for them if the two expressions are unifiable, otherwise,
;;; it returns a symbol FAIL.

(defun unify (e1 e2)
  (cond ((atomic-p e1) (atomic-unify e1 e2))
        ((atomic-p e2) (atomic-unify e2 e1))
        (t (let ((z1 (unify (first e1) (first e2))))
             (if (equal z1 'fail) 'fail
                 (let ((z2 (unify (apply-subst z1 (rest e1))
                                  (apply-subst z1 (rest e2)) )))
                   (if (equal z2 'fail) 'fail
                       (composite-subst z1 z2) )))))))

(defun atomic-unify (e1 e2)
  (cond ((equal e1 e2) nil)
        ((var-p e1)
         (if (occurs-in e1 e2) 'fail
             (make-subst e1 e2) ))
        ((var-p e2) (make-subst e2 e1))
        (t 'fail)))

(defun occurs-in (v e)
  (cond ((const-p e) nil)
        ((var-p e) (equal v e))
```

```
        (t (or (occurs-in v (first e))
               (occurs-in v (rest e)) ))))

(defun composite-subst (s1 s2)
  (if (null s1) s2
      (let ((key (first (first s1)))
            (expr (second (first s1))))
        (cons (list key (apply-subst s2 expr))
              (composite-subst (rest s1) s2) ))))

(defun make-subst (v e)
  (list (list v e)))

(defun find-var (v subst)
  (assoc v subst))

(defun apply-subst (subst e)
  (cond ((var-p e)
         (let ((pair (find-var e subst)))
           (if (null pair) e
               (second pair))))
        ((const-p e) e)
        (t (cons (apply-subst subst (first e))
                 (apply-subst subst (rest e)) ))))
```