述語論理型処理システムの高速化に関する研究

昭和５８年１２月

山　口　高　平

序　　文

本論文は、筆者が大阪大学大学院工学研究科(通信工学専攻博士後期課程)において在学中に行なった"述語論理型処理システムの高速化に関する研究"をまとめたもので、全文は次の6章から構成されている。

第1章は緒論であり、本研究の歴史的背景及び意義について概説し、従来の諸研究との関連について述べる。

第2章では、本研究において高速化の対象となっている定理証明プログラムとその実行理論である導出原理について述べる。更に、システムの効率改善策として、これまで提案されてきた制限付導出法と証明戦略及び定理証明プログラムにおける内部構造について述べ、改良単一化アルゴリズムについても論及する。

第3章では、定理証明システムＳＥＮＲＩの構成について述べる。ＳＥＮＲＩは、従来のシステムと比べて、オプションの指定を単に変更するだけで様々な定理証明法を構成できると共に、設定された戦略の不適切さの為反駁が得られない場合ユーザに助言を与える事を新しい特色としている。また、システム内部の新しい特色としては、述語論理式(節)の表現に適する様にデータ圧縮したリスト構造を採用すると共に、ＬＡＶＳ(List of AVailable Space)の再構成として、ＧＣ(Garbage Collection)を使用せずに、将来必要とならない記憶領域をポインタの再設定だけで再利用するという新しいＬＡＶＳの管理法を採用して、ＧＣと比較してその有効性を確認している。

第4章では、定理証明プログラムの内部構造の新しい実現方式ＲＳＧ方式を提案している。ＲＳＧ方式は、冗長な情報を再構成しない様に従来の置換型を改良した方式である。本方式では、リテラルリスト構造値を利用して、単一化計算を行なうペアを高速に抽出しながら、単一化操作を実行し、一時的な束縛を作成する事により、置換操作を一度だけで済ませているため、実行効率は改善される。また、節記録は、束縛環境及び同型のリテラルリスト構造を蓄える必要がないため、記憶効率も改善される。

第５章では、定理証明プログラムの実行理論である導出操作の並列処理について述べる。本章で提案する Parallel Resolution Algorithm は、単一化計算を並列処理する事に特徴がある。この単一化アルゴリズムの時間計算量を導出すると、単一化計算が成功する場合、従来の単一化アルゴリズムが、入力系列の要素数 n に対して $O(n^2)$ の時間計算量になるのに対して、本アルゴリズムの単一化計算の並列処理では、$O(n)$ の時間計算量になる。また、本アルゴリズムに特有の処理であるクラスタリング及びクラスタリング情報の作成を高速に処理する手順を検討した後、効率比較実験により、本アルゴリズムは、リテラルペアが複数のクラスタに分割され、単一化の成否を決定するまでに多くの代入が施される場合、特に有効である事を示す。

第６章は結論で、本研究で得られた結果についての評価を行ない、あわせて今後の課題や研究方針についても言及する。

関　連　発　表　論　文

A．　掲載論文

（1）　山口、西岡、打浪、手塚：" 定理証明システムＳＥＮＲＩの構成 "，情報処理学会論文誌，(論文)，＜Vol．２５，No．１（１９８４)に掲載予定＞．

（2）　山口、淡、打浪、手塚：" 定理証明プログラムにおける単一化計算の並列処理について "，信学論(Ｄ)，(論文)，＜採録決定＞．

B．　専門研究会発表論文

（1）　山口、西岡、打浪、手塚："定理証明システムＳＥＮＲＩとその応用" 信学技報ＥＣ８０－２６(１９８０－０７)．

（2）　山口、西岡、打浪、手塚：" 定理証明システムＳＥＮＲＩの構成 "，情報処理学会人工知能と対話技報研究会資料，２２－１(１９８１－０９)．

（3）　山口、打浪、手塚　：" 並列処理向き単一化アルゴリズムとその効率について "，信学技報ＡＬ８２－０７(１９８２－０７)．

（4）　石川、山口、打浪、手塚　：" 論理プログラミングの内部構造－同型共有構造－の提案と評価 "，信学技報ＡＬ８２－２８(１９８２－０９)．

（5）　山口、石川、打浪、手塚　：" 論理プログラミングシステムＳＥＮＲＩ/ＬＰの構成について "，信学技報ＡＬ８２－８９(１９８３－０１)．

（6）　淡、山口、打浪、手塚　：" 並列処理向き単一化アルゴリズムの定量的評価について "，信学技報ＡＬ８３－１７(１９８３－０６)．

（7）　山口、淡、打浪、手塚　：" Parallel Resolution Algorithm for Logic Machine "，情報処理学会ソフトウェア基礎論研究会資料，６－１(１９８３－０９)．

C．　学会、大会等口頭発表論文

（1）　山口、西岡、打浪、手塚　：" 拡張された定理証明システムＳＥＮＲＩについて ",昭５４信学会情報システム部門全大４１(１９７９－１０).

（2）　山口、西岡、打浪、手塚　：" 定理証明システムＳＥＮＲＩのＳＮＬ導出法の拡張について ",昭５５信学総全大１２００(１９８０－０３).

（3）　山口、西岡、打浪、手塚　：" 各種導出法を統合化した定理証明システムＳＥＮＲＩ ",昭５６信学会情報システム部門全大３４(１９８１－１０).

（4）　山口、打浪、手塚　：" 単一化（ＵＮＩＦＩＣＡＴＩＯＮ）の高速化に関する一考察 ",昭５６関西連大Ｇ８－３３(１９８１－１１).

（5）　山口、打浪、手塚　：" 並列処理向き単一化アルゴリズムに関する一考察 ",第２４回情学全大３Ｂ－３(１９８２－０３).

（6）　山口、阿部、打浪、手塚：" 論理プログラムシステムＳＥＮＲＩ/ＬＰの内部構造について ",昭５７信学総全大１４８５(１９８２－０３).

（7）　山口、石川、打浪、手塚：" 論理プログラミングシステムＳＥＮＲＩ/ＬＰの制御構造について",第２５回情学全大６Ｄ－６(１９８２－１０).

（8）　山口、石川、打浪、手塚：" 論理プログラミングシステムの支援ユーティリティに関する一考察 ",昭５７関西連大Ｇ８－４(１９８２－１２).

（9）　山口、石川、打浪、手塚　：" 論理プログラミングの内部構造－同型共有構造－の改善とその評価について ",昭５８信学総全大１５６５(１９８３－０４).

(10)　淡、山口、打浪、手塚　：" 効率比較実験による並列処理向き単一化アルゴリズムの評価について ",昭５８信学総全大１５６６(１９８３－０４).

(11)　山口、淡、打浪、手塚　：" 述語論理型言語における導出操作を並列処理する効果について ",昭５８信学会情報システム部門全大Ｓ４－４(１９８３－１０).

(12)　山口、淡、打浪、手塚："The Evaluation of Parallel Resolution Algorithm ",第２７回情学全大４Ｐ－１(１９８３－１０).

その他の関連発表論文

（１） 山口、西岡、打浪、手塚 ：" 仮想関係を含む質問文を関係ＤＢで評価する手法について ",昭５５関西連大Ｇ８－２９(１９８０－１２).

（２） 山口、西岡、打浪、手塚 ：" 演繹能力を付加したＤＢＭＳについて ", 京大数理解析研講究録(１９８１－０２).

（３） 山口、西岡、打浪、手塚 ：" 演繹能力を付加したＤＢＭＳについて ", 昭５６信学総全大１３０７(１９８１－０４).

目　　次

第１章　緒　　論

現在の計算機は、人間が組み込んだアルゴリズムを忠実に実行する機械であり、数値計算を代表とする数値処理及びデータベースシステムを代表とするデータ処理が主な利用形態である。しかしながら、来たるべき知識情報化社会の到来を考えると、従来人間にとってのみ可能であった創造的な知的問題解決活動を支援するシステム、換言すれば知識情報処理システム（ＫＩＰＳ：Knowledge Information Processing System)の実現が期待される。我が国においても、国家プロジェクトとしてＫＩＰＳを目標とする第５世代コンピュータシステム(ＦＧＣＳ：Fifth Generation Computer System)の研究開発が１９８２年より始まっている。

この知識処理の実現は、計算機に人間と同じ様な思考能力を与える事につながり、これは人工知能(Artificial Intelligence)の研究分野である。特に、定理証明システム及びＦＧＣＳの核言語であるPrologを代表とする論理プログラミングシステムを含む述語論理型処理システムと関連が深い。

以上の背景より、本論文では、述語論理型処理システムの重要な問題点である効率(実行効率及び記憶効率)を重点的に取り上げ検討している。まず、従来の効率を改善するための研究を眺めてみると、以下の３項目に分けられる。

（１）　制限付導出法及び証明戦略

（２）　定理証明プログラムの内部構造

（３）　改良単一化アルゴリズム

（１）の問題に関しては数多くの提案がなされてきたが、これらの成果を実働化する従来の定理証明システムは、異なった定理証明法の比較や反駁が得られない場合の支援が不備であったり、また効率にも問題があった。そこで、筆者は、以上の問題点を解決する使いやすくて高速な定理証明システムＳＥＮＲＩの構成法について検討し、第３章においてその構成法を述べる。

（２）の実現方式としては、導出の実行時に節レベルで全く新しい構造を作り

出す置換型方式(Structure Generating(SG)方式)と導出の実行時に束縛だけを作成する環境評価型方式がある。現在、述語論理型処理システムにおいては、環境評価型が効率が良いという点から多く採用されている。しかしながら、環境を頻繁に参照するプログラムや導出形の内容を強制的に変更して再実行したい場合においては、環境評価型では、効率が悪くなる危険性がある。環境評価型のこの種の欠点は、節を間接的に表現する事に起因していると考えられる。

そこで第4章においては、従来のSG方式に改良を加えたRSG方式を提案し、環境評価型と比較して、ほぼ同程度の効率が得られると共に、節の修正等に強い事を示す。

(3)については、1965年にJ. A. Robinsonが基本単一化アルゴリズムを提案して以来、M. S. Paterson及びA. Martelli等によって改良がなされているが、いずれも逐次処理における改善に留どまっている。そこで、第5章においては、単一化計算を並列処理する事により、効率改善の可能性について考察し、更にこの考察を導出レベルまで拡張し、導出操作を並列処理しうるParallel Resolution Algorithmを提案する。

以上述べた如く、本論文は、将来実現が期待されるKIPSに関連する諸問題について考察したものである。

第２章　定理証明プログラムの基礎概念

２．１　緒　言

本章では、本研究において種々の考察の対象となっている定理証明プログラムの諸事項について述べる。

定理証明プログラムは、一階述語論理の一標準形である節を用いて表現されるが、一階述語論理は、数学の大部分と日常会話の多くの叙述を表現できる論理体系であるため、プログラム化できる問題領域は広いと言える。また、定理証明プログラムの実行理論は、一階述語論理体系における導出原理に基づいているが、これは、導出原理が完全性及び単純性の両性質を兼ね備えている強力な推論規則となっているためである。

以上の背景から、導出原理に基づく一階述語論理の処理システム(述語論理型処理システム)の作成が始まったわけであるが、効率(実行効率及び記憶効率)が大きな問題点として浮かび上がってきた。そこで、その対策として、無駄な導出を省いて探索空間を狭くするための手続き「制限付導出法」及び「証明戦略」が数多く考案され、その中でも、節集合の特別なクラスの一つであるHorn節集合に対して完全性を保持する制限付導出法に興味が集まった。また、定理証明プログラムの内部構造においても改良が考案され、更に、導出操作の中心的役割を果たす単一化計算のアルゴリズムにおいても逐次処理の範囲で改良が加えられた。

次節以下においては、まず定理証明プログラムの syntax と semantics 及び導出原理と基本単一化アルゴリズムについて述べ、更に、効率改善策として、制限付導出法と証明戦略・定理証明プログラムの内部構造の改良・改良単一化アルゴリズムについて言及する。

２．２　定理証明プログラムの統語論と意味論

一階述語論理は、数学の大部分と日常会話の多くの叙述を表現できる論理体系であり、推論を行なうための言語システムに適用されうる。

本節では、一階述語論理に関する文法定義と意味定義を行なう。

２．２．１　定理証明プログラムの文法定義

定理証明プログラムは、一階述語論理式の一標準形である節(clause)を用いて表現される。そこで、一階述語論理に関する諸定義を与えると共に、一階述語論理式から節形式への変換手順について述べる。

(定義２－１)　記号　(symbol)

（１）　区切記号　　「，」「(」「)」

（２）　限定記号　　全称記号　「∀」　存在記号「∃」

（３）　論理記号　　否定　「~」　含意　「→」
　　論理和　「∨」　論理積　「∧」

（４）　定数記号　　個々の対象の記述　a，b，c，・・・

（５）　変数記号　　限定記号の範囲の指示、又は
　　代入の為の記述　x，y，z，・・・

（６）　関数記号　　「 f^n 」(n≧１)　n 項から値への写像の記述

（７）　述語記号　　「 P^n 」(n≧１)　n 項の関係の記述

（８）　真理値記号　真　「T」　偽　「F」

(定義２－２)　項　(term)

（１）　定数記号は項である。

（２）　変数記号は項である。

（３）　f が n変数関数記号で、$t_1,t_2,\ldots,t_n$ が項であれば、
　　$f(t_1,t_2,\ldots,t_n)$は項である。

（４）　すべての項は、(１)~(３)を適用して作られる。

(定義２－３)　原子論理式　(atomic formula)

（１）　命題記号は原子論理式である。

（２）　$t_1, t_2, \ldots, t_n$ が項であれば、$P^n(t_1, t_2, \ldots, t_n)$は、原子論理式である。

（３）　すべての原子論理式は、(１)(２)を適用して作られる。

(定義２－４)　完全論理式　(well－formed formula ： wff)

（１）　原子論理式は、完全論理式である。

（２）　ＡとＢが完全論理式であれば、

$(\sim A)$，$(A \vee B)$，$(A \wedge B)$，$(A \rightarrow B)$，$(A \leftarrow B)$

は、すべて完全論理式である。

（３）　$A(x)$が完全論理式で、xが$A(x)$において自由変数ならば、$\forall x A(x)$及び$\exists x A(x)$も完全論理式である。

限定記号のうち、$\forall$を全称記号、$\exists$を存在記号と呼び、$\forall x, \exists x$をそれぞれ、全称作用素、存在作用素、両方併せて限定作用素と呼ぶ。

(定義２－５)　リテラル　(literal)

原子論理式、またはその否定をリテラルと呼ぶ。

(定義２－６)　節　(clause)

リテラルの論理和のみで構成される完全論理式を節と呼ぶ。一つのリテラルも持たない節は空節(empty clause)と呼ばれ、□で表わされる。

(定義２－７)　スコーレム関数　(Skolem　function)

存在作用素$\exists y$が、全称作用素$\forall x$に束縛されている時、変数yの値はxに依存する。従って、yはxの関数として、$y=g(x)$と表わすことができる。この時にできる関数gをスコーレム関数と呼ぶ。

(定義２－８)　スコーレム標準形　(Skolem　normal　form)

$x_1, x_2, \ldots, x_n$ を変数、$C_1, C_2, \ldots, C_m$を節とするとき、

$$\forall x_1 \forall x_2 \ldots \forall x_n \,[\, C_1 \wedge C_2 \wedge \cdots \wedge C_m \,]$$

の形の閉論理式をスコーレム標準形と呼ぶ。このとき、$\forall x_1 \forall x_2 \ldots \forall x_n$ を前置記号(prefix)、それ以外を母式(matrix)と呼ぶ。

スコーレム標準形は、節集合$\{C_1, C_2, \cdots, C_m\}$として簡単に表わす。

以下に、任意の完全論理式を節集合に変換する手続きを示す。

〈ＳＴＥＰ１〉　含意記号「→」の排除　：「Ａ→Ｂ」 ↔ 「～Ａ∧Ｂ」

〈ＳＴＥＰ２〉　各否定記号の範囲が、それぞれ一つの述語記号だけとなるように変形する。

"ド・モルガンの法則"　「～(Ａ∧Ｂ)」 ↔ 「～Ａ∨～Ｂ」

「～(Ａ∨Ｂ)」 ↔ 「～Ａ∧～Ｂ」

"全称記号と存在記号の変換規則"　「～(∀ｘＡ)」 ↔ 「∃ｘ(～Ａ)」

「～(∃ｘＡ)」 ↔ 「∀ｘ(～Ａ)」

"二重否定の法則"　「～(～Ａ)」 ↔ 「Ａ」

〈ＳＴＥＰ３〉　各限定作用素の中の束縛変数をすべて異なるものにする。

〈ＳＴＥＰ４〉　存在作用素の中の束縛変数をスコーレム関数で置き換え、存在作用素を除去する。

〈ＳＴＥＰ５〉　全称作用素を式の先頭に移す。

〈ＳＴＥＰ６〉　matrixの部分を積標準形に変換する。

〈ＳＴＥＰ７〉　prefixを除去する。

〈ＳＴＥＰ８〉　論理積記号「∧」を排除して、節集合に変換する。

以上のような手続きによって、完全論理式を節集合に変換することにより、一階述語論理に対する次節で述べる導出原理のアルゴリズムの入力データを作成することができる。

２．２．２　定理証明プログラムの意味定義

定理証明プログラムの意味論が、与えられた解釈において、節がどのような真理値をとるかという事に基づいている。そこで、解釈に関する諸定義を与え、節の真偽決定法について言及する。

(定義２－９)　解釈　(interpretation)

完全論理式Ｆの解釈とは、次の操作によりＦと問題定義域の様相を組み合わせる事である。

Ａ．空でない定義域Ｄを決める。

Ｂ．Ｆの中の定数記号、変数記号、関数記号、そして述語記号に以下の様に値を割り当てる。

(1)　各定数記号a_iに、Ｄのある要素を割り当てる。

(2)　各n変数関数記号f_i^nに、Ｄ上で定義された写像を割り当てる。

$$f_i^n : D^n \rightarrow D$$

(3)　各n変数述語記号P_i^nに、Ｄ上で定義された関係を割り当てる。

$$P_i^n : D^n \rightarrow \{T, F\}$$

与えられた完全論理式Ｆのある解釈における真理値は、Ｆに含まれるすべての原子論理式のその解釈における真理値から決定できる。

(定義２－１０)　充足性　(satisfiability)

完全論理式集合Ｓの充足性とは、ある解釈のもとで、集合Ｓに含まれるすべての完全論理式が真となることを言う。このとき、Ｓは充足可能(satisfiable)であると言い、その解釈をモデルと言う。

(定義２－１１)　充足不可能性　(unsatisfiability)

完全論理式集合Ｓの充足不可能性とは、いかなる解釈によっても、集合Ｓに含まれるすべての完全論理式を同時に真とすることができないことを言う。このとき、Ｓは充足不可能(unsatisfiable)であると言う。

(定義２－１２)　論理的帰結　(logical　consequence)

完全論理式集合Ｓを充足するあらゆる解釈が、ある完全論理式Ｗを充足するとき、ＷはＳの論理的帰結であると言う。

定義２－１１，２－１２より、ＷがＳの論理的帰結であることと集合Ｓ∪{～Ｗ}の充足不可能性とが同値であることがわかる。

なぜなら、ＷがＳの論理的帰結であるならば、Ｓを充足するあらゆる解釈に対して～Ｗは偽となるので、集合Ｓ∪{～Ｗ}は充足不可能となり、その逆も言えるからである。

通常の証明問題では、Ｓを公理としＷをＳから導かれる定理とし、定理証明問題を充足不可能性を導く問題に置き換えることができる。

２．３　単一化計算と導出原理

本節では、定理証明プログラムの実行理論である導出原理に関する諸定義及び諸性質について述べる。まず、導出原理の中心的操作の役割を果たす単一化計算について述べる。

(定義２－１３)　代入・代入の合成

$1 \leqq i \leqq n$なるすべてのiについて、v_i がすべて変数で、t_i がすべてv_i と異なる項であり、しかも$v_1, v_2, \cdots, v_n$ がすべて異なる時、$\{t_1/v_1, t_2/v_2, \cdots, t_n/v_n\}$は代入と呼ばれる。また、Ｅを任意の式とした時、$E\{t_1/v_1, t_2/v_2, \cdots, t_n/v_n\}$は、$1 \leqq i \leqq n$の範囲の各iについて、Ｅの中に現われる変数$v_i$ をすべて項t_i に置き換える事によって得られる式を意味する。

更に、$\theta = \{t_1/x_1, t_2/x_2, \cdots, t_n/x_n\}$と$\lambda = \{u_1/y_1, u_2/y_2, \cdots, u_m/y_m\}$という二つの代入に対して、次の様に定義される代入$\theta \circ \lambda$を$\theta$と$\lambda$の合成という。すなわち、集合 $\{t_1\lambda/x_1, t_2\lambda/x_2, \cdots, t_n\lambda/x_n, u_1/y_1, u_2/y_2, \cdots, u_m/y_m\}$から $t_j\lambda = x_j$ となる要素と、y_i が$\{x_1, x_2, \cdots, x_n\}$のいずれかである様な要素 u_i/y_i を削除する事によって得られる代入を$\theta \circ \lambda$と定義する。

(定義２－１４)　単一化

式$E_1, E_2, \cdots, En$と代入σについて$E_1\sigma = E_2\sigma = \cdots = En\sigma$が成立する時、$\sigma$は$E_1, E_2, \cdots, En$を単一化するといい、$\sigma$を$E_1, E_2, \cdots, En$の単一化作用素(unifier)という。また、単一化作用素が存在するとき、$E_1, E_2, \cdots, En$は単一化可能であるという。更に、式$E_1, E_2, \cdots, En$の単一化作用素σに関して、$E_1, E_2, \cdots, En$の任意の単一化作用素θについて$\theta = \sigma \circ \lambda$となる様な代入$\lambda$が存在する時、$\sigma$を最も一般的な単一化作用素(mgu)という。

$\{E_1, E_2, \cdots, En\}$を表現の集合Wと呼び、このWが単一化可能であるかどうかを決定し、可能であるならばそのmguを求めるためのアルゴリズムを単一化アルゴリズムと呼ぶ。

以下、１９６５年にＪ．Ａ．Robinsonが示した基本的な単一化アルゴリズム[30]について述べる。

(定義２－１５)　不一致集合 (disagreement　set)

表現の集合Wの元である$E_1, E_2, \cdots, En$の各表現をそれぞれ左から始まる記号列として見たとき、これらを初めて異なる記号列とする各$Ei(i = 1, \cdots, n)$の部分表現からなる集合Ｄを不一致集合と呼ぶ。

図２－１に基本単一化アルゴリズムを示すが、このアルゴリズムについては、以下の定理が成立する。

(定理２－１)　単一化定理 (Unification Theorem)

Wが有限で空でない単一化可能な表現の集合であるならば、単一化アルゴリズムは常にＳＴＥＰ２で終了し、最後に得られる単一化作用素σ_kは、Wの最も一般的な単一化作用素mguである。

(証明は、文献(８)参照)

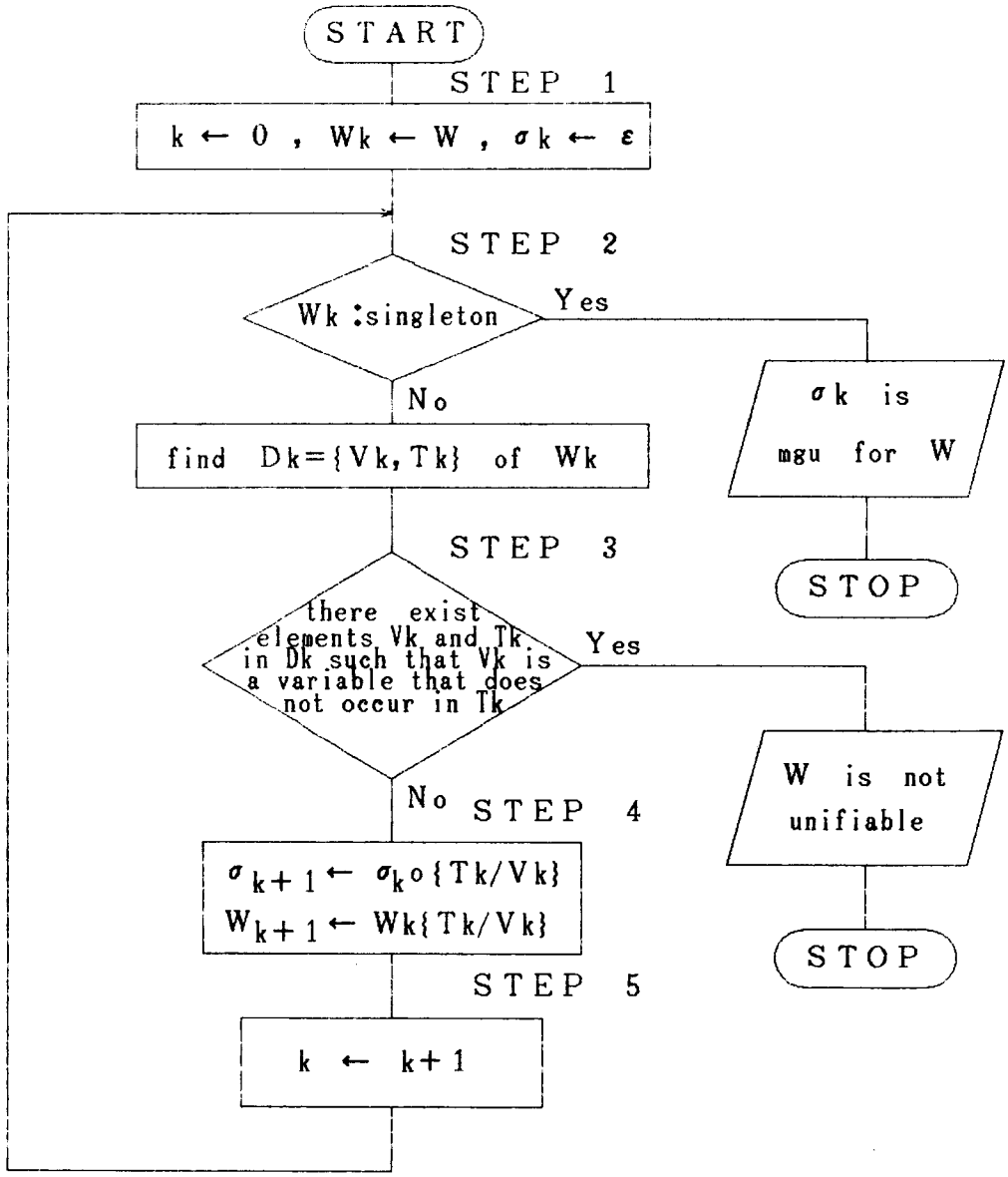


図２－１　基本単一化アルゴリズム

次に、導出原理の基本操作である導出形作成のための定義を示す。

(定義２－１６) 簡約形（因子、 factor ）

節Ｃの中の同符号の二つ以上のリテラルに対して、mgu ： σ が存在する時、$C\sigma$はＣの簡約形(因子)という。

(定義２－１７) ２項導出形 (binary resolvent)

C_1とC_2を共通変数を持たない二つの節とし、L_1とL_2をそれぞれC_1とC_2の中にあるリテラルとする。L_1と$\sim L_2$がmguを持つとき、

$$C=(C_1\sigma-L_1\sigma)\cup(C_2\sigma-L_2\sigma)$$

をC_1とC_2の２項導出形という。このとき、 C_1とC_2をＣの親節 (parent clauses)といい、L_1とL_2を被導出リテラル(literals resolved upon) という。

(定義２－１８) 導出形 (resolvent)

節Ｃ１とＣ２を親節とする導出形とは、Ｃ１もしくはは Ｃ１の簡約形とＣ２もしくはＣ２の簡約形を親節とする２項導出形のことである。

(定理２－２) 導出原理の完全性

(Completeness of the Resolution Principle)

節集合Ｓが充足不可能であることと、Ｓから空節□を導く演繹 (反駁 、refutation)が存在することは同値である。

以上の定理より、導出原理は証明可能な定理は必ず証明できることがわかる。この好ましい性質のために、導出原理が定理証明に使用されるわけであるが、実際には、効率も重要な問題となる。次節では、導出原理の効率改善を図る制限付導出法及び証明戦略について述べる。

２．４　制限付導出法と証明戦略

与えられた節集合に対して、導出原理をすべての節ペアに適用する事は、証明したい定理に関係のない内容まで推論してしまい、実行効率・記憶効率共に悪い。そこで、導出原理の適用範囲(探索空間)を限定し、効率改善を図る手続きとして、制限付導出法及び証明戦略が数多く考えられてきた。まず、制限付導出法について述べる。

２．４．１　制限付導出法

制限付導出法としては、一般の節集合に対して完全性を保持しているものと、後で述べるHorn節集合に対して完全性を保持するものに分ける事ができる。前者の代表的なものとして、線型導出法がある。

(定義２－１９)　線型導出法(Linear　Resolution)

C_0を節集合Ｓの節とする時、C_0を上端節(top　clause)に持つＳからの節C_nの線型導出とは、以下の条件を満足する導出である。

(ｉ)　$i=0,1,\cdots,n-1$に対してC_{i+1}は中心節(center clause)と呼ばれるC_iと側節(side clause)と呼ばれるB_iとの導出形である(図２－２参照)。

(ｉｉ)　各々のB_iは、Ｓの元か、既に導出された中心節である。

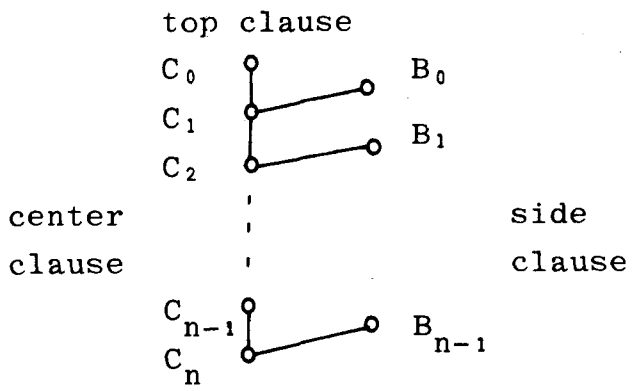


図２－２　線型導出法

線型導出法は、更に改良案が考えられており、その一つとして、線型導出法に順序節(ordered clause ： 異なったリテラル系列を順序節と呼ぶ)と被導出リテラルの情報を取り入れたＯＬ導出法(Ordered Linear Resolution)がある。このＯＬ導出法も一般の節集合に対して完全性を保持する。

この様な一般の節集合に対して完全性を保持する制限付導出法の他に、節集合を特別なクラスに限定し、そのクラスで完全性を保持する制限付導出法が研究されてきた。この特別なクラスとして代表的なものが、Horn節集合である。このHorn節集合に興味が集まっているのは、群論・環論・ブール代数等の数学理論の多くが表現可能であり、一般には完全性を持たない制限付導出法でも、Horn節集合に限定すれば完全性を持つ事が多いためである。以下、Horn節集合を定義し、次にHorn節集合に対して完全性を持つ制限付導出法を列挙する。

(定義２－２０) Horn節集合 (Horn Set)

高々一つの正リテラル(positive literal ： 肯定の述語記号を持つリテラル)しか持たない節をHorn節と呼び、Horn節のみからなる節集合をHorn節集合と呼ぶ。

節が単一のリテラルから成るとき、単位節(unit clause)と呼び、導出を行なう前の入力節集合Ｓに属している節を入力節(input clause)と呼ぶ。また、節の中のすべてのリテラルが否定記号を持たない時、その節を正節(positive literal)といい、節の中のすべてのリテラルが否定記号を持つ時、その節を負節(negative literal)という。

(定義２－２１) 単位導出法 (Unit Resolution)

導出の際の二つの親節のうち、少なくとも一方が単位節であるか単位因子を用いている導出を単位導出という。

(定義２－２２) 入力導出法 (Input Resolution)

導出の際の二つの親節のうち、少なくとも一方が入力節である様な導出を入

力導出という。

節集合Ｓからの単位反駁が存在する事とＳからの入力反駁が存在する事は等価である〔6〕。

(定義２－２３)　ＳＮＬ導出法　(ＳＮＬ　Resolution)

節集合ＳのＳＮＬ導出は、次の条件(１)、(２)を満たす導出である。但し、節は順序節である。

(１)　上端節C_1はＳに属する負節である。

(２)　節C_{i+1}はC_iの最右リテラルを被導出リテラルとするようなC_iと入力節B_iの負の導出形であるか、またはC_iの因子である　($1 \leq i \leq n-1$)。但し、B_iは側節である。

(定義２－２４)　ＳＰＵ導出法　(ＳＰＵ　Resolution)

節集合ＳのＳＰＵ導出は、次の条件(１)、(２)を満たす導出である。

(１)　ＳはHorn節集合であり、節は順序節と考える。

(２)　導出の際の一方の親節は正単位節(positive unit clause)であり、他方の親節の被導出リテラルは、節の最右リテラルである。

節集合ＳからのＳＰＵ反駁が存在する事とＳからのＳＮＬ反駁が存在する事は等価である〔22〕。

２．４．２　証明戦略

証明戦略は、一般の導出法及び制限付導出法に付加されて、探索空間を更に狭くするために用いられ、以下の３種類の戦略に分類する事ができる。

(１)　証明の過程に不必要な節及び操作を消去する戦略　(消去戦略と呼ぶ)

(２)　導出を行なう節のペアを選ぶ場合に優先順位を決め、順位の高いものから順に導出を実行させる戦略　(優先戦略と呼ぶ)

(3) 導出を行なう時に、リテラル・節・導出の深さ等に制限を加えて、制限を満足しない場合は、導出を実行させない戦略 (制限戦略と呼ぶ)

以下、各証明戦略について述べる。

【消去戦略】

(i) 恒真節の除去 (tautology deletion)
恒真値をとる節を除去する。

(ii) 論理的に包含される節の除去 (subsumed clause deletion)
他の節に論理的に包含される節を除去する。

(iii) 等位導出形の除去 (equivalent resolvent deletion)
(i)の特別な場合であるが、非空節の導出形が生成された場合、以前生成された導出形と等しければ削除する。

(iv) 純リテラルの削除 (pure literal deletion)
節記録において補リテラル(述語記号が同じで、符号が異なるリテラル)が存在しないリテラル(pure literal)を含む節を除去する。

(v) 簡約操作の除去 (factoring deletion)
簡約形を生成しない。

【優先戦略】

(i) 単位節優先 (unit preference)
単位節を優先して導出する。

(ii) 評価関数 (evaluation function)
節の長さ・関数の長さ等で決まる評価値よって優先順位を決めて導出を実行する。

【制限戦略】

(i) 節の長さ(clause length)制限
節に含まれるリテラルの個数に制限を加える。

(ii) 関数の深さ(function depth)制限

関数の深さとは、関数の入れ子構造の深さを意味し、例えばP(a,f(g(h(a,b))))の関数の深さは3である。この関数の深さに制限を加える。

(ⅲ) 導出の深さ(resolution depth)制限

導出の深さとは、縦型探索(depth first search)において導出回数を意味し、あまりに深い導出は無意味だと考え、制限を加える。

本節で述べた制限付導出法及び証明戦略を適切に組み合わせる事によって、探索空間が大幅に狭くなり、効率改善が図れると考えられる。

2．5 定理証明プログラムの内部構造

今まで述べてきた一般の導出法、制限付導出法及び証明戦略を実際にプログラム化するとき、項・リテラル・節等のデータ表現(内部構造)にも工夫する余地がある。この内部構造の実現方式としては、置換型(Structure Generating(SG)方式)と環境評価型があり、まず、置換型について述べる。

2．5．1 置換型

置換型というのは、導出の実行時に節レベルで全く新しい構造を作り出す方式であり、リスト構造が代表的である。

リスト構造では、導出時に置換・コピー等の操作により、構造を頻繁にたどる必要があるため実行効率が悪く、また、節記録が導出毎に新しく生成され、その節記録がリテラル数等に直接比例して大きくなるため記憶効率も悪くなる。しかし、節が直接的に表現されているため、修正・変更等の操作が容易であるという利点を持つ。

第3章で述べる定理証明システムSENRIの内部構造としては、節の修正の容易さの点から、節を表現するのに適したリスト構造を採用しており、このリスト構造による導出過程を図2－3に示す。

2．5．2 環境評価型

環境評価型というのは、導出の実行時に束縛(binding: 変数への代入情報)

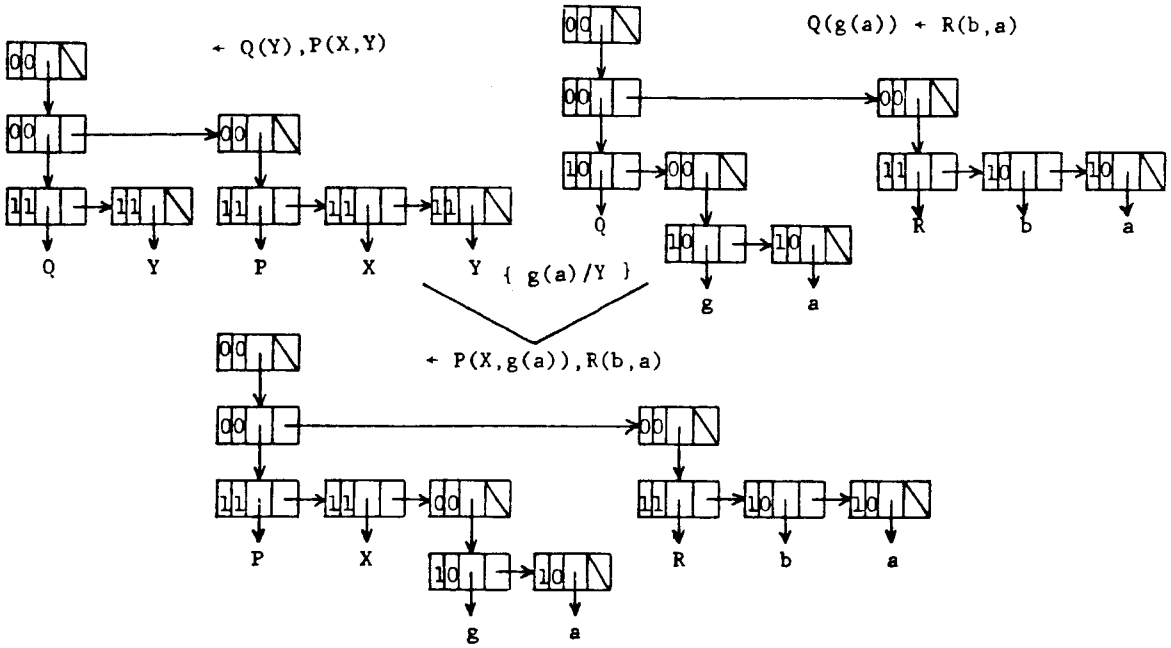


図２－３　リスト構造による導出過程

だけを作成する方式であり、共有構造(Structure Sharing(ＳＳ)方式)〔３〕が代表的である。

共有構造は、実行時に構造をたどる必要がないため、実行効率が良く、また、束縛を利用して節をリテラル数とは無関係にデータ圧縮して表現できるため、リスト構造に比べて大幅に記憶効率が改善される。以上の理由から、現在、述語論理型処理システムの多くは、共有構造を採用している。

しかしながら、共有構造では実行時に束縛をたどる必要があるため、束縛を頻繁にたどるプログラムでは実行効率が悪くなる可能性がある。また、節が間接的に表現されているので、修正・変更等の操作が困難になる危険性がある。

図２－４は、Ｒ．Ｓ．Boyer〔３〕によって提案された共有構造による導出過程(導出例は、図２－３と同様)を示しているが、導出形のタップル表現の各要素は、以下の事項を表わしている。

$$R = < C, i, D, j, NL, MI, \sigma >$$

C ： 左親節Ｃ１の節記録

i ： Ｃ１の被導出リテラル

D ： 右親節Ｃ２の節記録

j ： Ｃ２の被導出リテラル順位

NL： 導出形におけるリテラル数 （Ｃ＋Ｄ－２）

MI： 導出形の最大指数

σ ： 導出形生成時に、単一化計算によって加えられた束縛

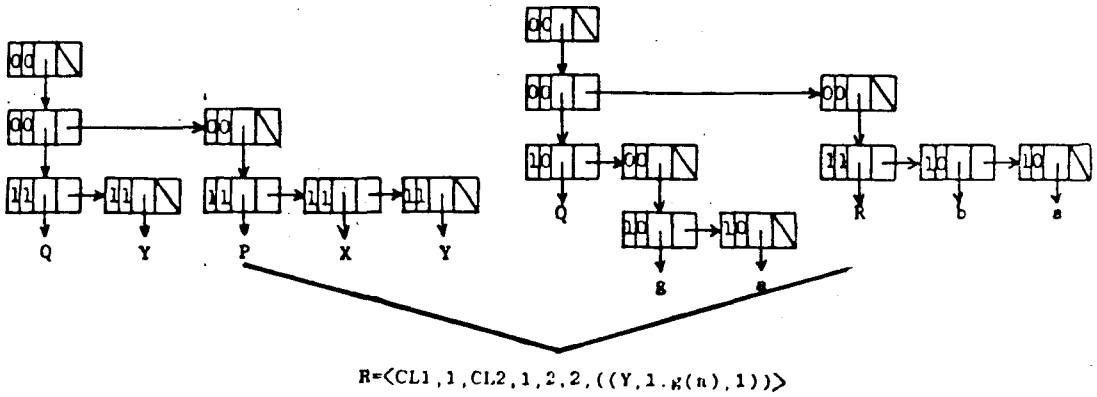


図２－４　共有構造による導出過程

以上、定理証明プログラムの内部構造の代表的な実現方式について述べたが、ＰＲＯＬＯＧを代表とする論理プログラミングシステム〔21〕では、環境評価型において改良が進んでいる。しかしながら、置換型にもまだ改良の余地が残されており、その改良案については第４章で述べる。

2．6　改良単一化アルゴリズム

定理証明プログラムの効率改善を図るには、前節で述べた内部構造を改良する他に、定理証明プログラムを実行する上で、大量に発生する単一化操作を改良する事が考えられる。本節では、逐次計算における改良単一化アルゴリズムとして、A．Martelli（１９７７）[23] と M．S．Paterson(１９７６)[28]の提案について述べる。

2．6．1　A．Martelli　による改良単一化アルゴリズム

A．Martelli　は、R．S．Boyerによって提案された共有構造を修正する事によって、効率的な単一化アルゴリズムを開発している。以下に、主要な修正点を示す。

（１）　変数束縛と共に、カウンタを蓄える。
（２）　束縛は、時々書き替えられる。
（３）　アクセスされないと判明した変数は、その束縛を捨て去る。

（１)の修正によって、初期化が避けられると共に、被導出リテラルのアクセスが高速となる。また、（２)の修正によって、束縛をたどるchainが短くなるため、単一化計算が高速となる。しかしながら、記憶容量がトレードオフ問題として増加する。最後に、（３)の修正によって、束縛を蓄えるデータベースが小さくなる。

本アルゴリズムは、カウンタと束縛の書き替えにより、被導出リテラルと束縛のアクセス時間を短縮する事により、単一化計算の高速化を図るものである。

2．6．2　M．S．Paterson　による改良単一化アルゴリズム

M．S．Paterson　は、表現の集合を関数記号と定数記号及び変数記号によってラベル付けされた節点を持つ有向木(directed acyclic graph ： dag)で表わし、単一化計算をdag上の演算として処理するアルゴリズムを開発している。

本アルゴリズムでは、共通の副表現(図２－５ではX_2)を単一のサブグラフによって表わす事により、データ圧縮されたdagを採用している。また、各ノードで、外に出ていく枝数を outdegree 及び 入ってくる枝数を indegree と定義し、indegree ＝ 0 のノードを特に root node (根節点)と呼ぶ。

本アルゴリズムの重要な特徴は、ノードＳのすべての親ノードが削除され、Ｓがその時点で dag の root node になるまで、Ｓの処理を始めない事である。これは、具体的には単一化計算を独立した関数ノードから優先して実行することを意味するもので、図２－５の場合、まずＦが処理され、次にそれらを削除すれば、唯一 root node となるＧ(X_3)が処理され、通常の単一化アルゴリズムの処理手順とは逆になる。

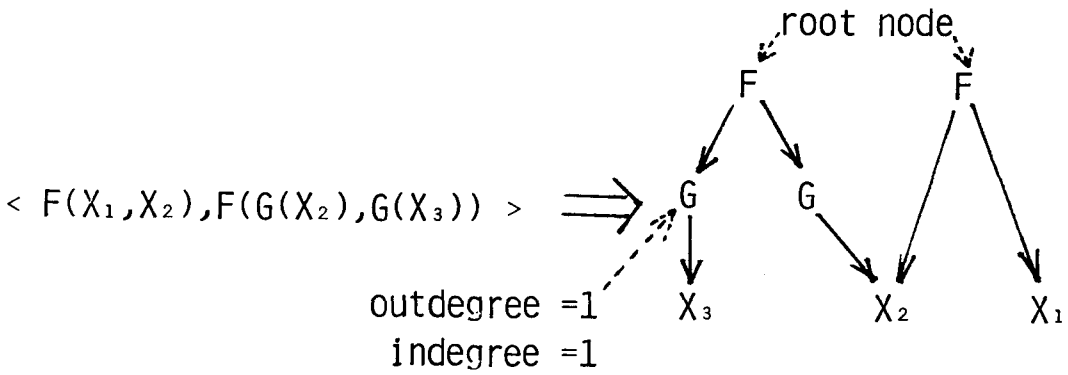


図２－５　dagの具体例

以上述べた単一化アルゴリズムの改良案は、いずれも逐次処理におけるものであり、第５章においては、並列処理による改良案を提案する。

2．7　結　言

定理証明プログラムの実行に関する理論的背景は、一階述語論理体系における導出原理である。これは、導出原理が、証明可能な定理は必ず証明できるという完全性を持つことと、推論規則が導出という唯一の規則から成立しているため、計算機上での実現が容易であることに起因している。導出原理が完全性を保持する事は非常に重要なことではあるが、述語論理型処理システムとしては、効率という点も重要視される。

このため、効率を改善するために以下の3項目が検討された。

（1）　制限付導出法及び証明戦略の開発
（2）　定理証明プログラムの内部構造の開発
（3）　改良単一化アルゴリズムの開発

(1)については、Horn節集合に対して完全性を保持するＳＮＬ導出法及びＳＰＵ導出法等に興味が持たれ、(2)については、導出の実行時に束縛だけを蓄える環境評価型が、効率の良い内部構造の実現方式として提案された。また、(3)については、A．Martelli　及び M．S．Paterson によって、逐次処理における単一化アルゴリズムの改良がなされた。

最後に、これらの検討事項と次章以下との関連を示す。
(1)については、制限付導出法及び証明戦略を適切に組み合わせる事によって、更に効率が改善されると考えられ、第3章においては、これらを容易に組み合わせられる定理証明システムＳＥＮＲＩについて述べる。また、(2)については、環境評価型が多く用いられているわけであるが、置換型にもまだ改善の余地があることを第4章で示す。更に、(3)については、従来の改良単一化アルゴリズムが逐次処理におけるものであるのに対して、並列処理の立場からの改良案を第5章で提案する。

第３章　定理証明システム
ＳＥＮＲＩの構成

３．１　緒　言

第２章においては、導出原理について論じ、また、効率面での改善を目指す制限付導出法と証明戦略について言及した。

一方、これらの成果を実働化する従来の定理証明システムとしては、Ｃ．Ｌ．ChangのＴＰＵ[8]　及び山崎らのシステム[51]等があるが、これらは汎用記号処理言語ＬＩＳＰで構成されており、以下の様な問題点がある。

[システム構成法に起因する問題点]

（１）　ひとつの定理証明法を構成する事に重点が置かれ、異なった定理証明法の比較等が困難である。

（２）　反駁が得られない場合の支援が全く考慮されておらず、戦略設定時にユーザの負担が大きい。

[記述言語ＬＩＳＰに起因する問題点]

（３）　リスト構造が冗長である。

（４）　記憶領域の管理にGarbage　Collection(ＧＣ)を使用しており、実行効率に問題がある。

本章では、以上の問題点を検討し、高速で使い易い定理証明システムＳＥＮＲＩ(System to Evaluate Non−numeRical Informations)の構成について論じる。

３．２　システム構成

３．２．１　システムの概観

ＳＥＮＲＩの概観を図３－１に示す。ＳＥＮＲＩでは、まずユーザの選択した導出法と戦略及び入力節集合を入力モジュール（ RESOLUTION&STRATEGY SELECTOR, INPUTTER）によって読み込み、選択された制限付導出法実行モジュール(SNL EXECUTOR, SPU EXECUTOR, INPUT EXECUTOR, UNIT EXECUTOR ,OL EXECUTOR, TPU EXECUTOR) に制御が移る。以下、戦略モジュール（STRATEGY EXECUTOR）によって導出形及び簡約形を生成し、空節が導出されれば出力モジュール(OUTPUTTER)により反駁木を出力して停止する。

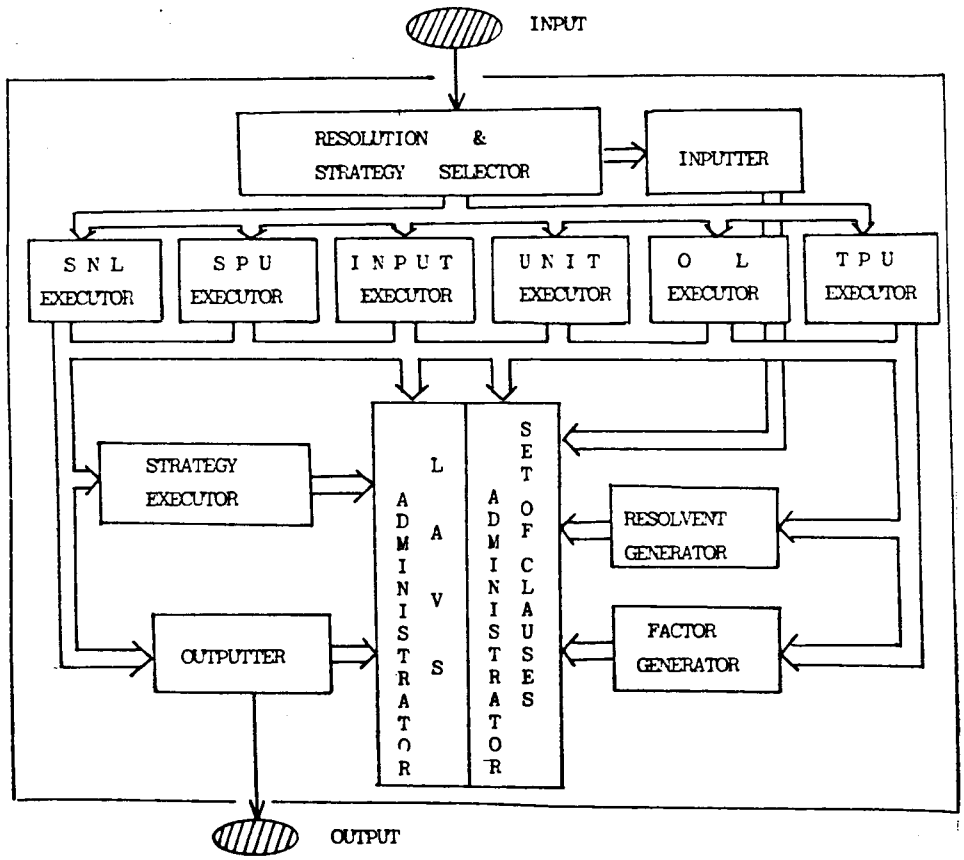
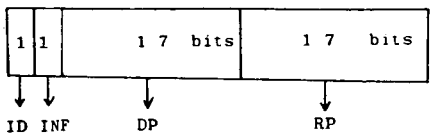


図３－１　　定理証明システムＳＥＮＲＩの構成

リストは、生成順に連結され、節集合リストとして節集合管理モジュール（SET OF CLAUSES ADMINISTRATOR)によって管理されており、ＬＡＶＳはＬＡＶＳ管理モジュール(LAVS ADMINISTRATOR)によって管理されている。また、セルの操作・文字データ領域の管理・リストコピー・代入操作とその応用操作・節の種類判定等のモジュールは図示されていないが、システムの随所で数多く使用されている。

３．２．２　データ構造

ＳＥＮＲＩのデータ構造の基本単位であるセルは、実際には計算機の１ワード(３６bits)で構成されており、図３－２に示す構造を持つ。入力された節は、図３－３の具体例で示す通り、セルを基本単位とするリスト構造によって表現される。



ID部 ⟶ セルの識別情報をもつ.

セルの種類	値
リストセル	0
データセル	1

INF部 ⟶ データセルのとき，データの型情報をもつ.

データの型	値
定　数	0
変　数	1
関数記号	0
述語記号	0
補の述語記号	1

注) 同値のデータの型の区別は，セルの位置を情報としてソフトウェアで行う．
また，枠付きリテラル(framed literal)を表現するときは，リストセルにおいて1をセットする.

DP部 ⟶ データセルのときは，文字データを指すためのポインタをもち，
リストセルのときは，下連結のためのポインタをもつ.

RP部 ⟶ 右連結のためのポインタをもつ.

図３－２　ＳＥＮＲＩのセル構造

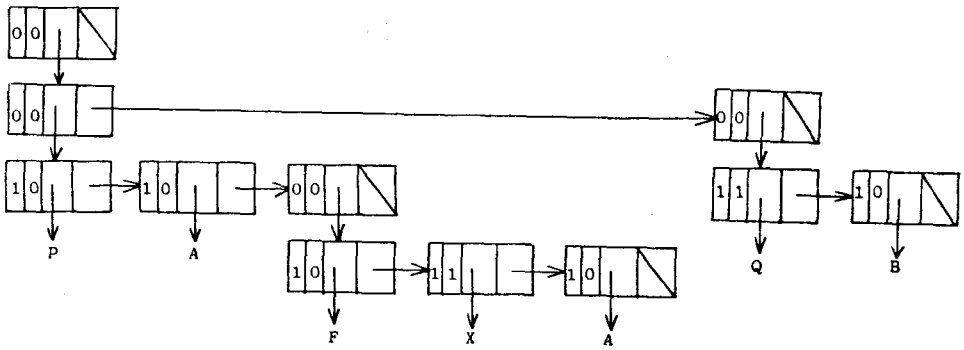


図３－３　入力節の内部構造

```
<INPUT>::=<RESOLUTION>(<STRATEGY SEQUENCE>)<SET OF CLAUSES>.
<RESOLUTION>::= SNL|SPU|TPU|OL|INPUT|UNIT
<STRATEGY SEQUENCE>::=<STRATEGY>=<PARAMETER>|<STRATEGY>=<PARAMETER>,<STRATEGY SEQUENCE>
<STRATEGY>::= SUPPORT|TAUTO|EQRSLVNT|NOFACTOR|PDEPTH|RDEPTH|CLENGTH|TPUN2|TPUN3|TPUN4|
              TPUSUPPORT|OCLENGTH|ANSWER|NOTANS|PRINTRSLVNT
<PARAMETER>::= Y|N|<UNSIGNED INTEGER>|<SUPPORT SET>|<TPU SUPPORT SET>
<SUPPORT SET>::=(<ELEMENTS>)|()
<ELEMENTS>::=<UNSIGNED INTEGER>|<UNSIGNED INTEGER>,<ELEMENTS>
<TPU SUPPORT SET>::=(<TPU ELEMENTS>)
<TPU ELEMENTS>::=()|(),<TPU ELEMENTS>|(<ELEMENTS>)|(<ELEMENTS>),<TPU ELEMENTS>
<SET OF CLAUSES>::=<NON-EMPTY CLAUSE>|<NON-EMPTY CLAUSE>;<SET OF CLAUSE>
<NON-EMPTY CLAUSE>::=<LITERAL SEQUENCE>
<LITERAL SEQUENCE>::=<LITERAL1>|<LITERAL1><OR><LITERAL SEQUENCE>
<LITERAL1>::= #<LITERAL2>|<LITERAL2>
<LITERAL2>::=<ATOMIC FORMULA>|<NOT><ATOMIC FORMULA>
<ATOMIC FORMULA>::=<PREDICATE SYMBOL>(<TERM SEQUENCE>)|<PREDICATE SYMBOL>
<TERM SEQUENCE>::=<TERM>|<TERM>,<TERM SEQUENCE>
<TERM>::=<CONSTANT>|<VARIABLE>|<FUNCTION>
<FUNCTION>::=<FUNCTION SYMBOL>(<TERM SEQUENCE>)
<VARIABLE>::= $<IDENTIFIER>
<CONSTANT>::=<IDENTIFIER>
<PREDICATE SYMBOL>::=<IDENTIFIER>
<FUNCTION SYMBOL>::=<IDENTIFIER>
<OR>::= /
<NOT>::= -
```

図３－４　ＳＥＮＲＩのシンタックス

３．２．３　入力モジュール

ＳＥＮＲＩにおける入力形式は、図３－４の文法で示され、導出法・戦略・節集合の順に指定する。導出法と戦略は、ユーザのオプションとして表３－１から選択できる。戦略には、支持集合の指定(１)・インタラクティブに最適値を決定する必要が無いもの(２－４)・その必要があるもの(５－７)・各導出法に専用のもの(８－１２)があり、その他、出力条件及び停止条件に関するもの(１３－１５)がある。戦略の各パラメータについては、各々暗黙値が存在するためユーザが指定しない場合には、この値がとられる。入力の具体例を図３－５に示すと共に、表３－２に本モジュールの概要を示す。

表３－１　ＳＥＮＲＩのユーザオプション

	指定形式	指定内容
導出法	SNL	SNL 導出法の実行
	SPU	SPU 導出法の実行
	TPU	TPU システムのシミュレート
	OL	OL 導出法の実行
	INPUT	INPUT 導出法の実行
	UNIT	UNIT 導出法の実行
戦略	1 SUPPORT=(m,..)	支持集合の指定
	2 TAUTOLOGY=Y	恒真節の除去
	3 EQRSLVNT=Y	等位導出形の除去
	4 NOFACTOR=Y	簡約操作 (factoring) の除去
	5 FDEPTH=m	関数の深さを *m* に制限
	6 RDEPTH=m	導出の深さを *m* に制限
	7 CLENGTH=m	節の長さを *m* に制限
	8 TPUN2=m	TPU システムにおいてパラメータ Ｎ２ を *m* に指定
	9 TPUN3=m	TPU システムにおいてパラメータ Ｎ３ を *m* に指定
	10 TPUN4=m	TPU システムにおいてパラメータ Ｎ４ を *m* に指定
	11 TPUSUPPORT=((m,..),..)	TPU システムにおいて支持集合の指定
	12 OCLENGTH=m	OL 導出法において非枠付きリテラル (non-framed literal) の数を *m* に制限
	13 ANSWER=Y	unit answering clause での停止要求
	14 NOTANS=Y	述語名が NOTANS の負単位節での停止要求
	15 PRINTRSLVNT=Y	生成された導出形の出力要求

```
SWI(NOFACTOR=Y,FDEPTH=3,RDEPTH=4,EQRSLVNT=Y,CLENGTH=3,PRINTRSL=Y)
/*     EXAMPLE OF GROUP THEORY (TPU 1)    */
P(G($X,$Y),$X,$Y) ; -P(K($X),$X,K($X)) ; P($X,H($X,$Y),$Y) ;
P($X,$V,$W)/-P($X,$Y,$U)/-P($Y,$Z,$V)/-P($U,$Z,$W) ;
P($U,$Z,$W)/-P($Y,$Z,$V)/-P($X,$Y,$U)/-P($X,$V,$W) .
```

図３－５　ＳＥＮＲＩの入力例

表３－２　入力モジュールの概要

RESOLUTION & STRATEGY SELECTOR	
TPINPT (IR, IST, IS, ISUP, IPNT)	ユーザの指定した導出法と戦略を読み込む．IR と IST にそれらの値をセットし，IS には以下に示す SCLIN を呼び出し，その引数値がセットされる．支持集合の指定がある場合は，ISUP (TPU を実行するときは IPNT に支持集合の管理情報) にその値をセットする．
INPUTTER	
CLIN(*N*)	入力節を節リストに変換して，その番地を *N* にセットする．
SCLIN(*N*)	CLIN によって節を読み込み，節集合リストを生成し，その番地を *N* にセットする．

３．２．４　制限付導出法実行モジュール

本モジュールは、表３－１に示す制限付導出法を実行するためにあり、表３－１の戦略において、１・４・６・８・９・１１・１３－１５の処理が、この中に組み込まれている。表３－３に本モジュールの概要を示す。

３．２．５　推論モジュール

本モジュールは、導出形及び簡約形を生成するためにあり、生成される導出形のリストと簡約形のリストは、節集合管理モジュールで統一的に処理されるために、図３－６と図３－７に示す様な同一構造を持つ。表３－４に本モジュールの概要を示す。

３．２．６　戦略実行モジュール

本モジュールは、制限付導出法実行モジュールの中で取り扱われていない戦略を実行するためにあり、表３－５にその概要を示す。

表３－３　制限付導出法実行モジュール

SNL EXECUTOR	
SNL (IST, IS, ISP)	戦略 IST (支持集合の指定は ISP) の下で，入力節集合 IS から SNL 導出法を実行する．
SPU EXECUTOR	
SPU (IST, IS, ISP)	戦略 IST (支持集合の指定は ISP) の下で，入力節集合 IS から SPU 導出法を実行する
TPU EXECUTOR	
TPU (IST, IS, ISP, IPNT)	戦略 IST (支持集合の指定は ISP，支持集合の管理情報は IPNT) の下で，入力節集合 IS から TPU システムを実行する．
OL EXECUTOR	
OL (IST, IS, ISP)	戦略 IST (支持集合の指定は ISP) の下で，入力節集合 IS から OL 導出法を実行する．
INPUT EXECUTOR	
INPUT (IST, IS, ISP)	戦略 IST(支持集合の指定は ISP)の下で，入力節集合 IS から INPUT 導出法を実行する．
UNIT EXECUTOR	
UNIT (IST, IS, ISP)	戦略 IST(支持集合の指定は ISP) の下で，入力節集合 IS から UNIT 導出法を実行する

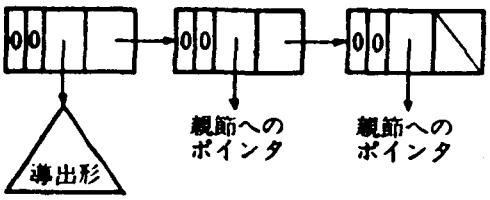


図３－６　導出形のリスト

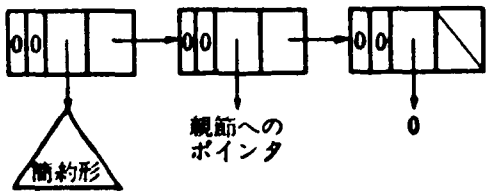


図３－７　簡約形のリスト

表３－４　推論モジュールの概要

RESOLVENT GENERATOR	
RESOLV (LRESLV, IFLAG, ICC 1, N 1, ICC 2, N 2)	節 ICC 1 の N 1 番目のリテラルと節 ICC 2 の N 2 番目のリテラルとから，一般の導出による導出形リストを生成する．非空節を生成した場合は，IFLAG に 1 をセットし，LRESLV にその番地をセットする．空節を生成した場合は，IFLAG に 0 をセットし，LRESLV にその番地をセットする．導出不可能の場合は，IFLAG に −1 をセットし，対応するリテラルが存在しない場合は，IFLAG に −2 をセットする．
SNLRSV (LRESLV, IFLAG, ICC 1, ICC 2, N 2)	節 ICC 1 の最右リテラルと節 ICC 2 の N 2 番目のリテラルとから，SNL 導出による導出形リストを生成する．LRESLV と IFLAG には，RESOLV と同様の値がセットされる．
OLRSLV (LRESLV, IFLAG, ICC 1, ICC 2, N 2, NLC)	節 ICC 1 の最右リテラルと節 ICC 2 の N 2 番目のリテラルとから，OL 導出による導出形リストを生成する．LRESLV と IFLAG には，RESOLV と同様の値がセットされ，NLC には簡約形を生成するときの制御情報がセットされる．
FACTOR GENERATOR	
INFACT (IC)	入力節 IC の簡約形を生成し，それを入力節集合に加える．
OINFCT (IC)	INFACT (IC) と同様であるが，OL 導出を実行するときに使用される．
RSFACT (IFACT, IC, NONNEW, NO)	導出形 IC から生成される簡約形の集合から，NO 番目の簡約形を取り出し，IFACT にその番地をセットする．簡約形が生成されない場合は，IFACT に 0 がセットされる．NONNEW は導出形の nonnew literal* の数である．
ORFACT (IFACT, IC, NLC, NO)	NLC が簡約形を生成するときの制御情報である以外は，RSFACT (IC) と同様であり，OL 導出を実行するときに使用される．

表３－５　戦略実行モジュールの概要

STRATEGY EXECUTOR	
ITAUTO (IC)	節 IC が恒真節ならば関数値を 1 とし，そうでなければ 0 とする．ただし，OL 導出法を実行する場合は，枠付きリテラル (framed literal) の取扱いが必要であるため，IOTAUT (IC) を用いる．
IEQ (IC 1, IC 2)	節 IC 1 と節 IC 2 が等しければ関数値を 1 とし，そうでなければ 0 とする．ただし，OL 導出法を実行する場合は，IOEQ (IC 1, IC 2) を用いる．
LENGTH (IC)	節 IC の長さを関数値とする．OL 導出法を実行する場合は，IOLENG (IC) を用いる．
IDEPTH (IC)	節 IC の関数の深さを関数値とする．

3．2．7　出力モジュール

本モジュールは、証明結果を出力するためにあり、表3－6にその概要を示す。

表3－6　出力モジュールの概要

OUTPUTTER	
CLOUT (IC)	節 IC を出力する.
DTREE (IC)	節 IC の演繹木を出力する. IC が空節ならば，証明結果を出力することになる.

3．2．8　節集合管理モジュール

本モジュールは、以下の4つの機能を持つ。

(1)「初期設定」→入力節集合の最後の節にＩＮＰＵＴとＬＥＮＤという2つの管理変数をセットし、ＩＮＰＵＴは入力節の終わりを示す変数として、ＬＥＮＤは節集合の終わりを示す変数として、以後取り扱われる。

(2)「導出形及び簡約形の付加」→節集合リストの最後に、導出形リスト及び簡約形リストを連結する。

(3)「節の探索」→節集合中の節の順位から、節のリストを取り出す。

(4)「節の削除」→節集合中の最右の節を削除する。

本モジュールの概要を表3－7に示す。

表3－7　節集合管理モジュールの概要

SET OF CLAUSES ADMINISTRATOR						
LCSET (C, L)	節集合の管理を行い，引数Cによってその機能と関数値は以下のようになる.					
	C	機　能	関数値	C	機　能	関　数　値
	'S'	初 期 設 定	LEND	'G'	節の探索	L 番目の節の番地
	'P'	導出形の付加	LEND	'D'	節の削除	0

３．２．９　ＬＡＶＳ管理モジュール

ＳＥＮＲＩではデータ構造としてリスト構造を採用しているため、記憶領域を多く必要とする問題では、ＬＡＶＳの再構成が頻繁に発生してしまい、全体の実行時間に大きく影響する。そこで、本モジュールは、図３－８に示す様に、ＬＡＶＳを将来必要となる記憶領域(節集合領域)とそうでない記憶領域(一時作業領域)に２分割して両方向から使用し、あふれが生じた時、一時作業領域のポインタを再設定する事により、一時作業領域を再利用するというＬＡＶＳの管理を行なっている。また、バックトラックが発生した場合は、それに応じて節集合領域のポインタを再設定している。本モジュールの概要を表３－８に示す。

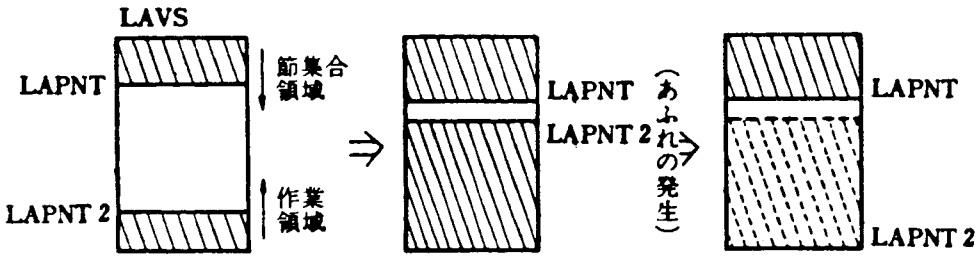


図３－８　ＳＥＮＲＩにおけるＬＡＶＳの管理法

表３－８　ＬＡＶＳ管理モジュールの概要

LAVS ADMINISTRATOR	
GETCEL(L)	LAVS の上位からセルを取り出す.
GETCL 2(L)	LAVS の下位からセルを取り出す.
BACKT	バックトラッキングが発生したとき，ポインタを再設定する.

3．3　システムの効率改善

抽象代数等の数学理論の多くは、Horn節集合のクラスの問題として表現できる。そこで、本節では、ほとんどの問題がHorn節集合となる　ＴＰＵ　ＥＸＡＭＰＬＥ　１－９〔８〕　を証明問題として選び†、ＳＮＬ及びＳＰＵ導出法をそのまま実行する　ＳＥＮＲＩ　ＦＯＲＴＲＡＮ　バージョン　から、徐々に効率改善を図った経過を以下に示す。

最初のバージョンの実行効率は、表３－１０－Ａに示されるが、このバージョンでは、無駄な導出が頻繁に発生し、多くの問題が証明未完了である。そこで、効率改善に影響を与える要因を分析し、次の３項目に着眼し改善を図った(図３－９参照)。

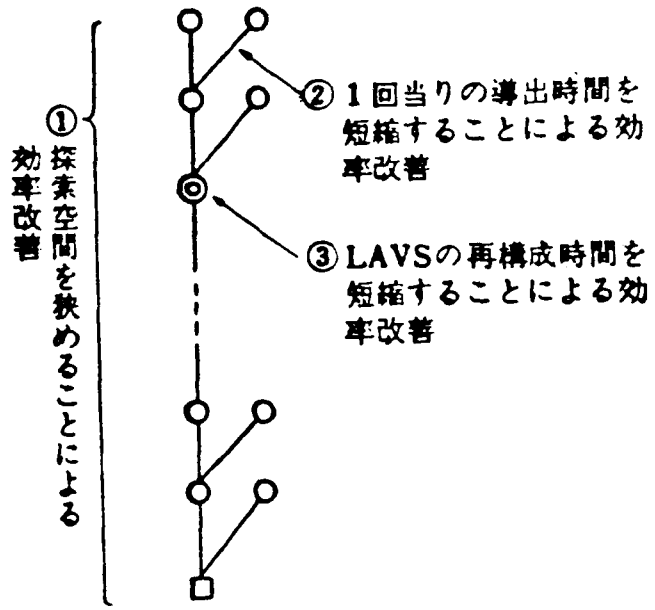


図３－９　効率改善の３要因

注†：　ＴＰＵ　ＥＸＡＭＰＬＥ　９　だけは、renaming操作〔２２〕を加えてもHorn節集合に変換されない。以後　ＴＰＵ　ＥＸ　１－９　と略記する。

[１] 戦略と制限付導出法(ＳＮＬ・ＳＰＵ)を組み合わせる事により、探索空間を狭める。([１]→モード２)

[２] 頻繁に使用するルーチンのアセンブラ・バージョンを作成する事により、１回当たりの導出時間を短縮する。([１]+[２]→モード３)

[３] 記憶領域の管理に、ＧＣを使用せずに３．２．９に述べた手法を用いる事により、ＬＡＶＳの再構成時間を短縮する([１]+[２]+[３]→モード４)。

表３－９ 効率実験モードの概要

効率改善項目 / モード	探索空間の狭化法	システムモジュール	記憶領域の管理法
Mode 1	SNL,SPU	FORTRAN version	Garbage collector
Mode 2	SNL,SPU+strategies	FORTRAN version	Garbage collector
Mode 3	SNL,SPU+strategies	Assembler version	Grabage collector
Mode 4	SNL,SPU+strategies	Assembler version	LAVS administrator

表３－９に各効率実験モードを示すと共に、表３－１０－Ａ～３－１０－Ｄにその実験結果を示す。モード２では、制限付導出法に戦略を付加した結果、探索空間が狭められ、より多くの問題が証明されうる事を示している。モード３では、更にアセンブラ・バージョンにより導出を実行しているため、１回当たりの導出時間が短縮され、モード２と比較して、２．６～４．５倍程度効率が改善されている。更にモード４では、ＬＡＶＳの再構成にＧＣを使用せずに、３．２．９で示した手法を用いる事により、ＧＣが起動される問題(ＴＰＵ ＥＸ－１，３，４，９)では、モード３と比較してほぼ２倍程度効率が改善されている。また、ＧＣの起動のタイミングの影響により、ＳＰＵ導出法で証明未完了であった ＴＰＵ ＥＸ－２，３ も証明を完了している。これは、ＧＣが起動した場合、自由領域を回復するのに、将来必要となるセルのマーク付け及び将来必要とならないセルの再構成という大きな実行コストを要する作業を行なわなければならないのに対し、３．２．９で示した手法では、単に一時作業領域のポインタを再設定するだけで済むためである。但し、バックトラックが生じた時は、節集合領域のポインタも後戻りさせている。

最終モードでは、本実験で用いたほとんどの問題が数秒以内で証明を完了しており、実用的なシステムになったと考えられる。

表３－１０－Ａ　モード１の実行効率

Resolution / Problem	ＳＮＬ	ＳＰＵ
TPU EX-1	*	2924 (m sec)
TPU EX-2	*	Retired
TPU EX-3	*	Retired
TPU EX-4	*	Retired
TPU EX-5	183 (m sec)	7230
TPU EX-6	229	Retired
TPU EX-7	*	1577
TPU EX-8	*	Retired
TPU EX-9	15431	Rejection

＊　：ＬＡＶＳ empty or Stack overflow

Retired: More than ３０ sec

ＬＡＶＳ: １００００ cells (３６ bits／cell)

Machine: Acos system １０００ model ４０

表３－１０－Ｂ　モード２の実行効率

Resolution / Problem	ＳＮＬ	ＳＰＵ
TPU EX-1	1286 (m sec)	785 (m sec)
TPU EX-2	Retired	*
TPU EX-3	1117	*
TPU EX-4	1214	7732
TPU EX-5	31	1286
TPU EX-6	65	5572
TPU EX-7	126	287
TPU EX-8	330	1445
TPU EX-9	937	Rejection

表３－１０－Ｃ　モード３の実行効率

Resolution / Problem	ＳＮＬ	ＳＰＵ
TPU EX-1	400 (m sec)	296 (m sec)
TPU EX-2	Retired	*
TPU EX-3	369	*
TPU EX-4	387	2912
TPU EX-5	8	399
TPU EX-6	15	2082
TPU EX-7	28	67
TPU EX-8	76	597
TPU EX-9	356	Rejection

表３－１０－Ｄ　モード４の実行効率

Resolution / Problem	ＳＮＬ	ＳＰＵ
TPU EX-1	212 (m sec)	119 (m sec)
TPU EX-2	Retired+	5661
TPU EX-3	179	4005
TPU EX-4	196	1162
TPU EX-5	7	216
TPU EX-6	14	853
TPU EX-7	25	59
TPU EX-8	69	223
TPU EX-9	157	Rejection+

注＋：　ＳＮＬ導出法で　ＴＰＵ　ＥＸ－２　の証明を終えていないのは、この問題が縦型探索には、全く適していないためだと考えられる。また、ＳＰＵ導出法で　ＴＰＵ　ＥＸ－９　の証明が拒絶されているのは、この問題がHorn節集合でない事に起因しており、ＳＮＬ導出法では偶然証明されたにすぎない。

３．４　システムの評価

３．４．１　ＳＥＮＲＩとＬＩＳＰで構成したシステムとの比較評価

多くの定理証明システムは、汎用記号処理言語ＬＩＳＰで構成されており、ＳＥＮＲＩを評価する場合、ＬＩＳＰで構成された他のシステムとの比較評価は欠かせない。そこで、ＬＩＳＰで構成されたＴＰＵシステムと同じ導出法と戦略をＳＥＮＲＩで構成し、比較検討を行なった。

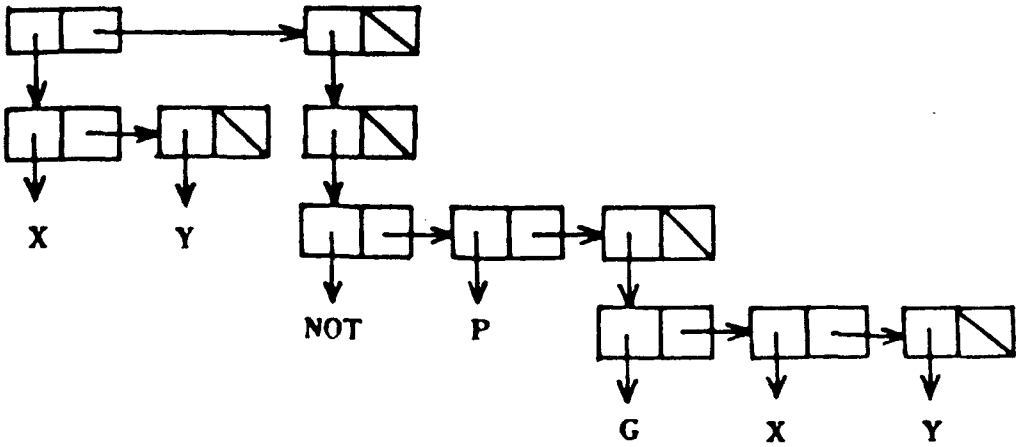
[データ構造の比較]

ＳＥＮＲＩの基本データ構造は、ＬＩＳＰのものと比較して、ＩＮＦ部により述語論理式をコンパクトに表現している。具体的には、変数及び否定の述語記号の取り扱いに関して、ＬＩＳＰで構成された多くのシステムでは特別な部分リストを設けるのに対して、ＳＥＮＲＩではＩＮＦ部に１をセットし、ソフトウェアで管理できる。例えば図３－１０では、ＬＩＳＰのシステムにおけるＸとＹの変数部分リストが、ＳＥＮＲＩにおいては、ＸとＹのデータセルのＩＮＦ部に１をセットする事で表現されている。また、ＮＯＴのデータセルは、Ｐのデータセルの ＩＮＦ部に１をセットする事で表現されている。

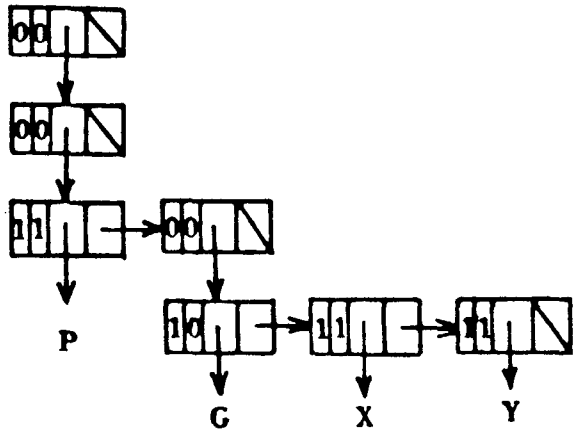
本データ構造は、変数及び否定の述語記号をＩＮＦ部で表現する事によって、多くのリテラルリスト構造が対称的になり、第４章で提案する定理証明プログラムの新しい内部構造の基礎になっている。

[入出力の比較]

ＴＰＵ　ＥＸ－１　を例にとり、原データ及びＬＩＳＰのシステムとＳＥＮＲＩにおける入出力形式を図３－１１～３－１５に示す。図３－１２と３－１３を比較する事により、ＳＥＮＲＩの方が原データにより近い形式で入力できる事がわかる。また、図３－１４と３－１５を比較する事により、ＬＩＳＰのシステムでは単に節番号を出力する形式であるのに対して、ＳＥＮＲＩでは導出の経過が容易に理解できる様に反駁に用いられた節をすべて出力している。



ＬＩＳＰによる論理式の内部表現



ＳＥＮＲＩによる論理式の内部表現

図３－１０　ＬＩＳＰとＳＥＮＲＩによる論理式の内部表現

(1) $P(g(x,y),x,y)$

(2) $P(x,h(x,y),y)$

(3) $\sim P(k(x),x,k(x))$

(4) $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$

(5) $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(u,z,w) \vee P(x,v,w)$.

図３－１１　ＴＰＵ　ＥＸ－１　の原データ

```
(TPU
  @((1(X Y)((P(G X Y)X Y)))
    (2(X Y)((P X(H X Y)Y)))))
  @((3(X)((NOT P(K X)X(K X)))))
  @((4(X Y Z U V W)((NOT P X Y U)(NOT P Y Z V)(NOT P X V W)(P U Z W)))
    (5(X Y Z U V W)((NOT P X Y U)(NOT P Y Z V)(NOT P U Z W)(P X V W))))
  @((3)NIL)
  @5
  @2
  @3
  @0)
```

図３－１２　ＬＩＳＰシステムの入力形式

```
TPU(NSUPPORT=((3),()),TPUN2=2,TPUN3=3,TPUN4=0)
/*     EXAMPLE OF GROUP THEORY (TPU 1)     */
P(G(¥X,¥Y),¥X,¥Y) ; -P(K(¥X),¥X,K(¥X)) ; P(¥X,H(¥X,¥Y),¥Y) ;
P(¥X,¥V,¥W)/-P(¥X,¥Y,¥U)/-P(¥Y,¥Z,¥V)/-P(¥U,¥Z,¥W) ;
P(¥U,¥Z,¥W)/-P(¥Y,¥Z,¥V)/-P(¥X,¥Y,¥U)/-P(¥X,¥V,¥W) .
```

図３－１３　ＳＥＮＲＩの入力形式

```
((6 3 4 4)(11 2 6 2)(15 1 11 1)(CONTRADICTION 1 15)).
```

図３－１４　ＬＩＳＰシステムの出力形式

```
+++ DEDUCTION +++

        *** INPUT CLAUSE(   1) ***
 P(G(¥¥,¥Y),¥X,¥Y)

        *** INPUT CLAUSE(   4) ***
-P(¥X,¥Y,¥U) / -P(¥Y,¥Z,¥V) / -P(¥X,¥V,¥W) / P(¥U,¥Z,¥W)

        *** INPUT CLAUSE(   3) ***
-P(K(¥X),¥X,K(¥X))

        *** RESOLVENT(   6) OF (   4) AND (   3) ***
-P(¥X1,¥X2,K(¥X3)) / -P(¥X2,¥X3,¥X4) / -P(¥X1,¥X4,K(¥X3))

        *** INPUT CLAUSE(   2) ***
P(¥X,H(¥X,¥Y),¥Y)

        *** RESOLVENT(  11) OF (   6) AND (   2) ***
-P(¥X1,¥X2,K(H(¥X2,¥X3))) / -P(¥X1,¥X3,K(H(¥X2,¥X3)))

        *** RESOLVENT(  15) OF (  11) AND (   1) ***
-P(G(¥X1,K(H(¥X1,¥X2))),¥X2,K(H(¥X1,¥X2)))

        *** RESOLVENT(  26) OF (   1) AND (  15) ***
EMPTY


+++ TPU SYSTEM END +++
```

図３－１５　ＳＥＮＲＩの出力形式

[実行効率の比較]

表３－１１にＴＰＵ　ＥＸ１－９をＳＥＮＲＩアセンブラ・バージョンで構成したＴＰＵシステムで証明した時の実行効率の結果を示す。ＬＩＳＰコンテスト〔53〕において、同証明問題を各種ＬＩＳＰで実行した効率結果が報告されている。表３－１１とこれらの結果は、ハードウェアの差異等で直接比較できないが、ＳＥＮＲＩは現在最良と考えられるＬＩＳＰコンパイラと同程度であり、サブルーチンのアセンブラ・バージョンを更に増加したり、開いたサブルーチンを導入したりすれば、ＳＥＮＲＩの効率改善はより一層進むと考えられる。

次に、ＳＥＮＲＩ　ＦＯＲＴＲＡＮ　バージョン　に一括型の　Garbage Collector　と　ＬＡＶＳ　Administrator　を導入した時の実行効率比較実験を行なう事によって、ＬＡＶＳの再構成時間について検討する。表３－１２

と図３－１６より、　ＬＡＶＳ　Administrator　を導入した時の方が、Garbage Collectorを導入した時より常に効率が良く、ＬＡＶＳのサイズが小さい程その事が顕著に現われている事がわかる。すなわち、Garbage Collectorを用いた場合、将来必要となるセルをマークし、将来必要とならないセルを再構成する事により、自由領域を回復するため、１回当たりのＬＡＶＳの再構成時間が大きくなる。従って、ＬＡＶＳのサイズが小さい時、起動回数が多くなり、ほとんどの時間がＧＣに費やされ、全体の実行効率が極端に悪くなる。一方、ＬＡＶＳ　Administrator　を用いた場合、一時作業領域のポインタを単に再設定するだけで、自由領域を回復できるため、たとえＬＡＶＳのサイズが小さく、起動回数が多くなっても、ＬＡＶＳの再構成時間は、ほとんど変化しない。よって、ＬＡＶＳのサイズが大きくとれない時、特に有効性を発揮すると考えられる。

表３－１１　ＴＰＵを実行するＳＥＮＲＩアセンブラ・バージョンの効率

Resolution / Problem	ＴＰＵ
TPU EX-1	131 (m sec)
TPU EX-2	645
TPU EX-3	281
TPU EX-4	279
TPU EX-5	49
TPU EX-6	1715
TPU EX-7	436
TPU EX-8	318
TPU EX-9	206

表３－１２　Garbage Collector 及び LAVS Administrator を使用した時のＴＰＵを実行する　ＳＥＮＲＩ　ＦＯＲＴＲＡＮ バージョンの効率

LAVS reconstructor cells ＼	Garbage Collector	LAVS Administrator
2000	3400 (m sec)	909 (m sec)
4000	1682	911
6000	1588	908
8000	1522	906
10000	1561	920

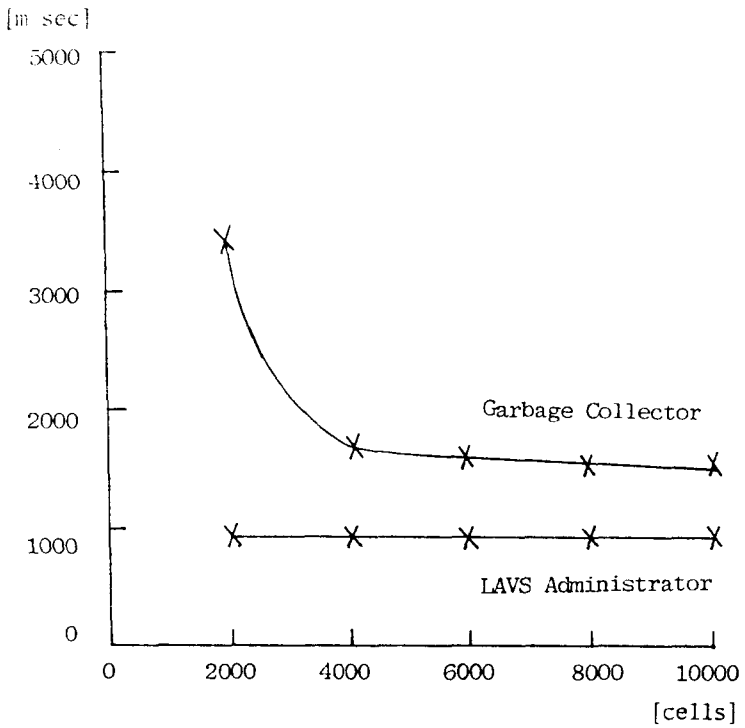


図３－１６　Garbage Collector 及び LAVS Administrator を使用した時のＴＰＵを実行する　ＳＥＮＲＩ　ＦＯＲＴＲＡＮ バージョンの効率比較

３．４．２　他の定理証明システムとの比較評価

従来発表されてきたシステム〔８〕〔５１〕は、ひとつの定理証明法を簡潔に記せる事が主目標であり、異なった定理証明法を比較するには、その数だけプログラムを組まなければならずユーザへの負担が多かった。一方、ＳＥＮＲＩでは、異なった定理証明法を容易に比較できる事が特色であり、表３－１に示したオプションの指定を単に変更するだけで、定理証明法を変更できる。また、戦略は左から順に適用されるため、戦略のオプションは同じでも指定順序が違えば異なった定理証明法を表現する事になり、定理証明法の詳細な変更が可能である。次に、ＳＥＮＲＩは、モジュール別に構成されているので、拡張性が高いと共に、元バージョンは、ＦＯＲＴＲＡＮで作成されているので、移植性も高い。最後に、以前のシステムでは、戦略の不適切さにより反駁が得られない場合、戦略の変更をユーザに一任していたが、ＳＥＮＲＩでは、図３－１７に示す支援システムによって導出過程を分析する事により、適切な戦略再設定の支援を実現している(図３－１８参照)。

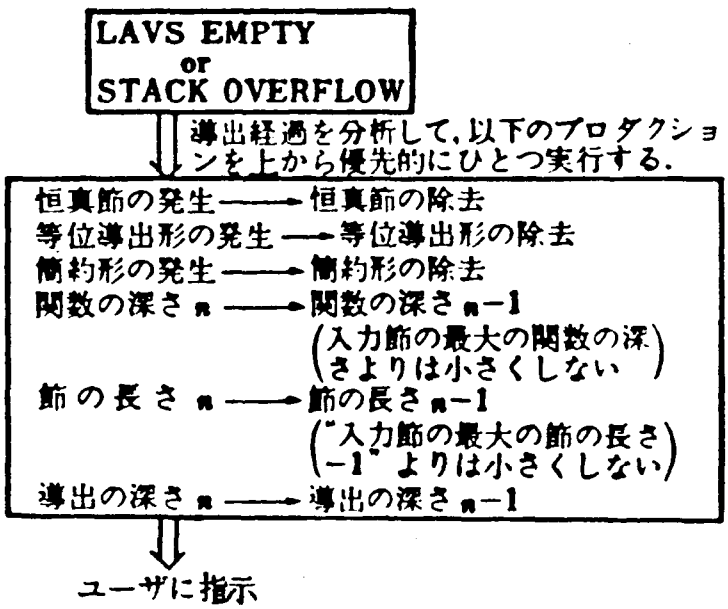


図３－１７　適切な戦略設定を行なうためのＳＥＮＲＩ支援システム

```
        *** THEOREM PROVER SENRI SYSTEM START ***


      +++ SNL RESOLUTION START +++


      +++INPUT CLAUSES+++

              *** INPUT CLAUSE(   1) ***
        P(G(¥X,¥Y),¥X,¥Y)

              *** INPUT CLAUSE(   2) ***
        -P(K(¥X),¥X,K(¥X))

              *** INPUT CLAUSE(   3) ***
        P(¥X,H(¥X,¥Y),¥Y)

              *** INPUT CLAUSE(   4) ***
        P(¥X,¥V,¥W) / -P(¥X,¥Y,¥U) / -P(¥Y,¥Z,¥V) / -P(¥U,¥Z,¥W)

              *** INPUT CLAUSE(   5) ***
        P(¥U,¥Z,¥W) / -P(¥Y,¥Z,¥V) / -P(¥X,¥Y,¥U) / -P(¥X,¥V,¥W)

                                    !
                                    !
                                    !

                    *** FACTOR(  11) OF (   5) ***
              P(¥W,¥Z,¥W) / -P(¥V,¥Z,¥V) / -P(¥X,¥V,¥W)

                    *** FACTOR(  12) OF (   6) ***
              P(¥W,¥W,¥W) / -P(¥W,¥W,¥W)

                                    !
                                    !
                                    !

        +++RESOLVENT+++

                 *** RESOLVENT(  18) OF (   2) AND (   4) ***
          -P(K(¥X1),¥X2,¥X3) / -P(¥X2,¥X4,¥X1) / -P(¥X3,¥X4,K(¥X1))

                                    !
                                    !
                                    !

    *** RESOLVENT(  32) OF (  31) AND (   1) ***
-P(K(¥X1),G(¥X2,G(¥X3,G(¥X4,G(¥X5,G(¥X6,¥X1))))),G(¥X2,G(¥X3,G(¥X4,G(¥X5,G(¥X6,K
(¥X1)))))))
     * LAVS EMPTY AT GETCEL SUBROUTINE *

           +++ AID SYSTEM START +++


           * OCCURENCE OF TAUTOLOGY *
           * ADD STRATEGY TO DELETE TAUTOLOGY *


           +++ AID SYSTEM END +++
```

図３－１８　ＳＥＮＲＩ支援システムの実行例

3．5　結　言

本章では、定理証明システムＳＥＮＲＩを開発していく上で、システム内部的には、データ構造及びＬＡＶＳの再構成に特色を持たせた。ＳＥＮＲＩのデータ構造は、リスト構造を基礎にして、述語論理式(節)の表現に適する様にデータ圧縮したものを採用したが、リスト構造を基礎にしたのは、他のデータ構造と比べて、変更・修正等の操作が柔軟にでき、支援システム等の作成が容易になるからである。また、ＬＡＶＳの再構成法は、最初ＧＣを使用していたが、定理証明においては、将来必要となるセルは節集合の表現に関連したものだけである事に着目し、ＬＡＶＳを双方向から利用して、将来必要とならない記憶領域をポインタの再設定だけで再利用するというＬＡＶＳの管理法を新しく提案し、ＧＣと比較してその有効性を確認した。

次に、ユーザインターフェース的には、従来のシステムが、ひとつの定理証明法を簡潔に構成する事に重点が置かれていたのに対して、ＳＥＮＲＩでは、オプションの指定を単に変更するだけで、様々な定理証明法を構成できると共に、戦略の不適切さにより反駁が得られない場合、ユーザに助言を与える事を新しい特色としている。前者の特色からは、定理証明法の変更が容易となり、証明問題に対して実行効率の良い定理証明法を短時間で設定できると共に、異なった定理証明法の比較が容易になる。また、後者の特色からは、適切な戦略設定が容易になる。

更に、頻繁に使用されるルーチンのアセンブラ・バージョンを作成して実行効率の改善が成され、また元バージョンは　、ＦＯＲＴＲＡＮでモジュール別に作成されている為、移植性及び拡張性が高いという特色を持っている。

以上の事から、定理証明システムＳＥＮＲＩは、従来のシステムと比べて、高速で使い易いシステムになったと考えられる。　また、近年研究が盛んなＰＲＯＬＯＧシステム〔21〕は、ＳＮＬ導出法〔22〕の動作をプログラムの実行過程(計算法)と見立てたものと考えられるが、ＳＥＮＲＩでは、それ以外の計算法も容易に構成できるため、効率の良い論理プログラミングシステムの計算法を開発していく上で役立つと考えられる。

第４章　定理証明プログラムにおける新しい内部構造の実現

４．１　緒　言

導出原理に基く定理証明プログラムの内部構造には、２．５節でのべた様に、導出の実行時に節レベルで全く新しい構造を作り出すもの（置換型：ＳＧ(Ｓtructure　Generating)方式)と導出の実行時に束縛だけを作成するもの(環境評価型)がある。置換型は、リスト構造が代表的であり、環境評価型は、共有構造(ＳＳ(Ｓtructure　Ｓharing)方式)が代表的であるが、現在、述語論理型処理システムにおいては、環境評価型が効率が良いという点から、多く採用されている。しかしながら、環境を頻繁に参照するプログラムや、導出形の内容を強制的に変更して再実行したい場合においては、環境評価型では、効率が悪くなる危険性がある。環境評価型のこの種の欠点は、節を間接的に表現する事に起因していると考えられる。

そこで本章では、従来のＳＧ方式に改良を加えたＲＳＧ方式を提案し、ＳＳ方式と比較して、ほぼ同程度の効率が得られる事を示す。

４．２　ＲＳＧ方式の基本概念

３．２節で示した定理証明システムＳＥＮＲＩのセル構造を基礎にして、定理証明問題(ＴＰＵ　ＥＸ　１－９)〔８〕の証明過程において発生するリテラルリスト数とリテラルリスト構造数を調査した結果が表４－１である。

表４－１より、定理の証明過程において発生するリテラルリスト構造数は、リテラルリスト数に比べて非常に少なく、構造的に同等なリテラルリストが、数多く発生している事が分かる。

(定義４－１)　同　型

点集合をＰ及び辺集合をＥとし、グラフＧを$G=(P,E)$で表現する。２つのグラフ$G_1=(P_1,E_1)$と$G_2=(P_2,E_2)$が同型であるとは、すべての $x,y \in P_1$に対し、$(x,y)\in E_1$ である時に限り、$(\phi(x),\phi(y))\in E_2$ となる様な１対１の写像関数$\phi: P_1 \rightarrow P_2$が存在する事である。

表４－１　証明過程で発生するリテラルリスト数とリテラルリスト構造数

項目 / 問題	リテラルリストの数	リテラルリスト構造の数
TPU-1	39	16
TPU-2	244	1
TPU-3	102	9
TPU-4	106	6
TPU-5	33	5
TPU-6	128	11
TPU-7	177	10
TPU-8	79	9
TPU-9	43	11

定義４－１の用語を使えば、従来のＳＧ方式では、冗長な情報として同型のリテラルリスト構造を生成していたと言える。

次に、リテラルリスト構造の表現法について考察する。

(定理４－１)　２分木構造の同型性

２分木ＴとＴ'の節点を各々行きがけの順に並べる時、$u_1 \cdot u_2 \cdot \cdot \cdot u_n$ 及び $u_1' \cdot u_2' \cdot \cdot \cdot u_n'$ となるものとする。次に、uを任意の節点とする時、

$$l(u)=\begin{cases}1: & u\text{の左の部分木が空でない時}\\ 0: & u\text{の左の部分木が空である時}\end{cases}$$

$$r(u)=\begin{cases}1: & u\text{の右の部分木が空でない時}\\ 0: & u\text{の右の部分木が空である時}\end{cases}$$

と書く事にする。

ＴとＴ'が同型であるのは、以下の条件が満足される場合であり、またその場合に限られる。

$$n=n' \qquad (4-1)$$

$$l(uj)=l(uj') \quad , \quad r(uj)=r(uj') \quad (\text{for} \quad 1 \leqq j \leqq n) \qquad (4-2)$$

（証明は、文献(３９)のpp. １１０，１１１参照）

定理４－１に従って、図４－１の様にリテラルリスト構造に相当する２分木の構造値(リテラルリスト構造値と呼ぶ)を作成すれば、同型の判定操作を高速に実行できる。

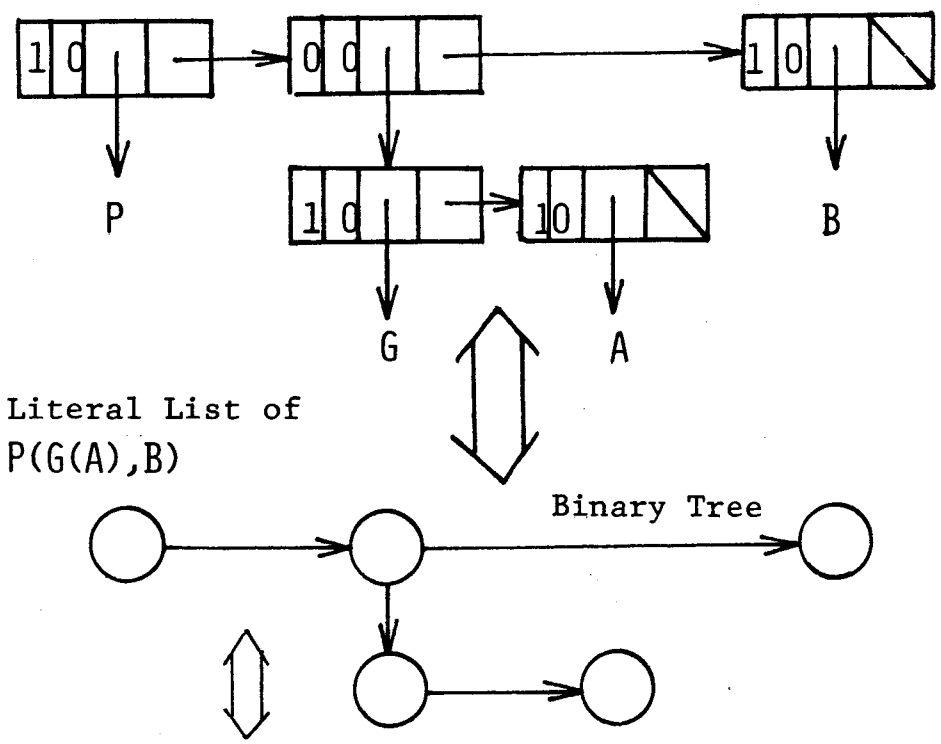


図４－１　P(G(A),B)のリテラルリスト構造値

以上の考察より、定理証明プログラムの各節が持つ情報として、基本的には、リテラルリスト構造値とそれらが持つデータ値のみを考え、これらの情報を高速に処理する内部構造をＲＳＧ(Refined Structure Generating)方式と名付ける。従って、ＲＳＧ方式による節記録の表示法(外部表現)は、リテラル

リスト構造値とそれらが持つデータ値のタップル表現となり、図４－２にその定義を示すと共に、図４－３の各節の外部表現を図４－４に示す。

```
< CLAUSE RECORD >::=< (< STRUCTURE VALUE SEQUENCE >) , (< DATA VALUES OF STRUCTURE VALUE SEQUENCE >) >
< STRUCTURE VALUE SEQUENCE >::=< STRUCTURE VALUE >|< STRUCTURE VALUE >,< STRUCTURE VALUE SEQUENCE >
< STRUCTURE VALUE >::= S< UNSIGNED INTEGER >
<DATA VALUES OF STRUCTURE VALUE SEQUENCE >::= (< DATA VALUE SEQUENCE >)|
                                              (< DATA VALUE SEQUENCE >),< DATA VALUES OF STRUCTURE VALUE SEQUENCE >
< DATA VALUE SEQUENCE >::=< DATA VALUE >|< DATA VALUE >,< DATA VALUE SEQUENCE >
< DATA VALUE >::=< INFORMATION VALUE >< IDENTIFIER >
< INFORMATION VALUE >::= 0 1
< IDENTIFIER >::=< ALPHABET >|< ALPHABET >< LETTER SEQUENCE >
< LETTER SEQUENCE >::=< ALPHABET >|< ALPHABET >< LETTER SEQUENCE >|< DIGIT >|< DIGIT >< LETTER SEQUENCE >
< UNSIGNED INTEGER >::=< DIGIT >|< DIGIT >< DIGIT SEQUENCE >
< DIGIT SEQUENCE >::=< DIGIT >|< DIGIT >< DIGIT SEQUENCE >
< ALPHABET >::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
< DIGIT >::= 0|1|2|3|4|5|6|7|8|9
```

図４－２　ＲＳＧ方式による節記録の表記法

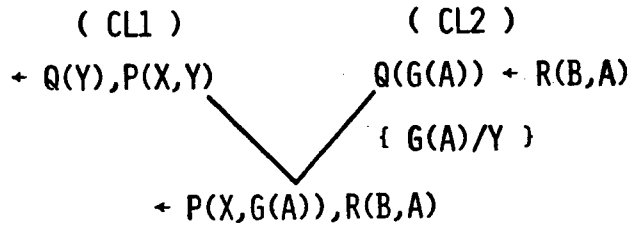


図４－３　定理証明プログラムの実行例

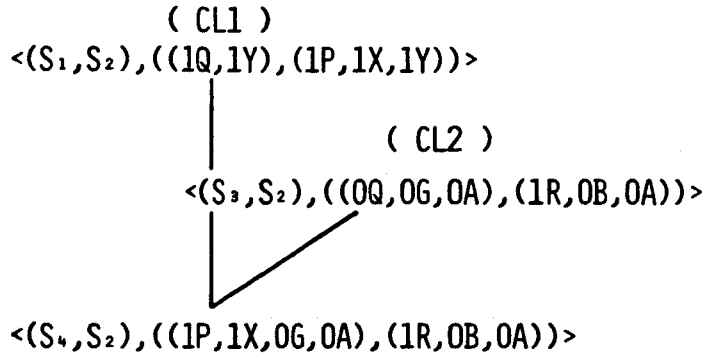


図４－４　ＲＳＧ方式による節記録の表記例

4．3　ＲＳＧ方式のインプリメンテイション

本節では、リテラルリスト構造値とそれらが持つデータ値のタップル表現で表わされた節記録を高速に処理する手順について述べる。

4．3．1　節の内部表現

ＲＳＧ方式による節記録は、基本的には、リテラルリスト構造値とそれらが持つデータ値のタップル表現であるが、その他の付属情報として、両親節の順位と変数を標準化するための最大指数及び出力の制御情報を蓄えている。図４－５に、節の具体例とＲＳＧ方式によるその外部表現及び内部表現を示す。但し、ＬＳはリテラルリスト構造値を蓄え、ＬＣＮＴには、その構造値のビット数と保有データ数が蓄えられている。

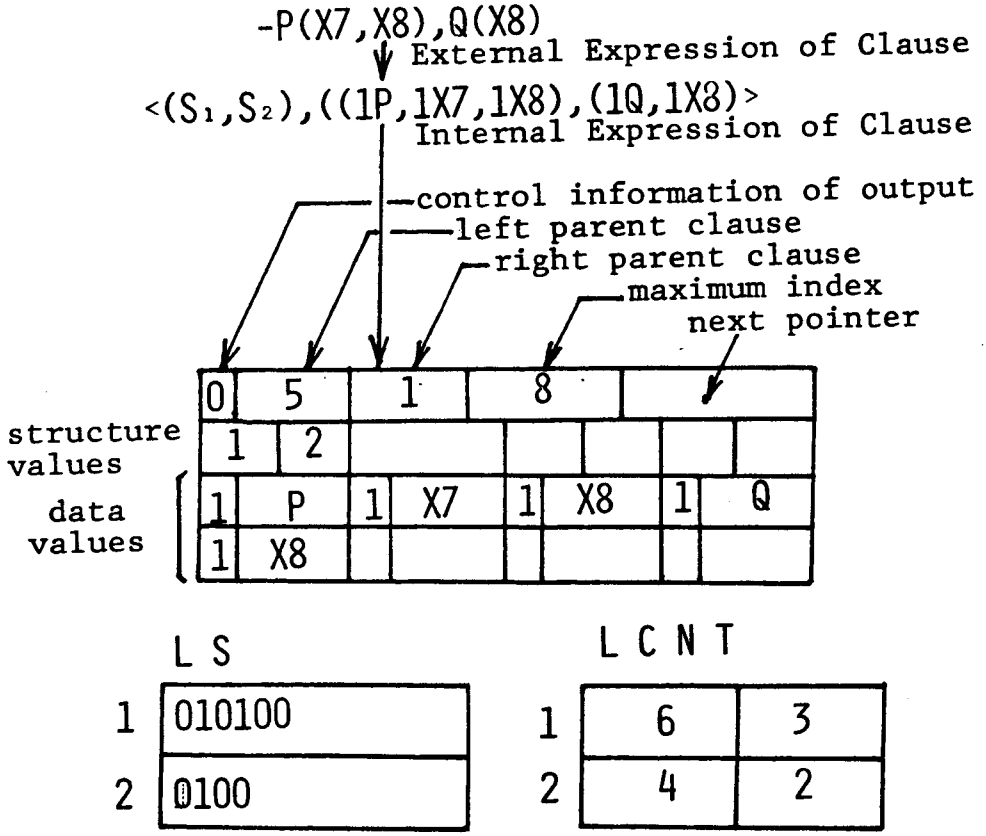


図４－５　ＲＳＧ方式による節の外部表現と内部表現

４．３．２　単一化計算と導出操作

ＲＳＧ方式による単一化計算では、リテラルリスト構造値を利用して、単一化計算を実行するためのペアを高速に抽出する事が可能であり、図４－６にそのアルゴリズムを示す。変数にある値が代入される時は、その度に置換を行なうと実行効率が悪くなるので、一時的な束縛を作成し、導出形生成時にこの束縛を利用して、初めて置換を行なう。また、変数に関数が代入される時は、新しいリテラルリスト構造が発生する可能性があるので、束縛に関数構造値を付加して、導出形生成時に利用する。図４－７にＲＳＧ方式による単一化アルゴリズムを示す。また、導出形作成時には、親節の被導出リテラルを除いたリテラルを連結し、一時的な束縛を参照しながら、リテラルリスト構造値あるいはデータ値を置き換え、導出形の内部表現を生成する。

４．３．３　ＲＳＧ方式による導出過程の具体例

以下、具体例を通して導出過程を説明する。まず、図４－８の導出過程を考えると、親節の内部表現は、図４－９の様になる。各親節の第１リテラルに注目すると、parent clause(６) の第１リテラルでは、リテラルリスト構造値としてＬＳ(１)＝０１０１００を保持しており、ＬＣＮＴ(１,２)＝３ より、３個のデータ値を保有している事がわかる。また、parent clause(２) の第１リテラルでは、リテラルリスト構造値として、ＬＳ(３)＝０１０１１００１００を保持しており、ＬＣＮＴ(３,２)＝４ より、４個のデータ値を保有している事がわかる。

次に、図４－６に示すペア抽出アルゴリズムに従って、図４－１０の様に単一化計算を実行するためのペアが順次取り出され、図４－７に示す単一化アルゴリズムによって、図４－１１に示す様な関数構造値を付加した一時的な束縛が形成される。

最後に、一時的な束縛を参照しながら、親節から被導出リテラルを除いたタップル情報を基にして、図４－１２に示す様な導出形の内部表現を生成する。

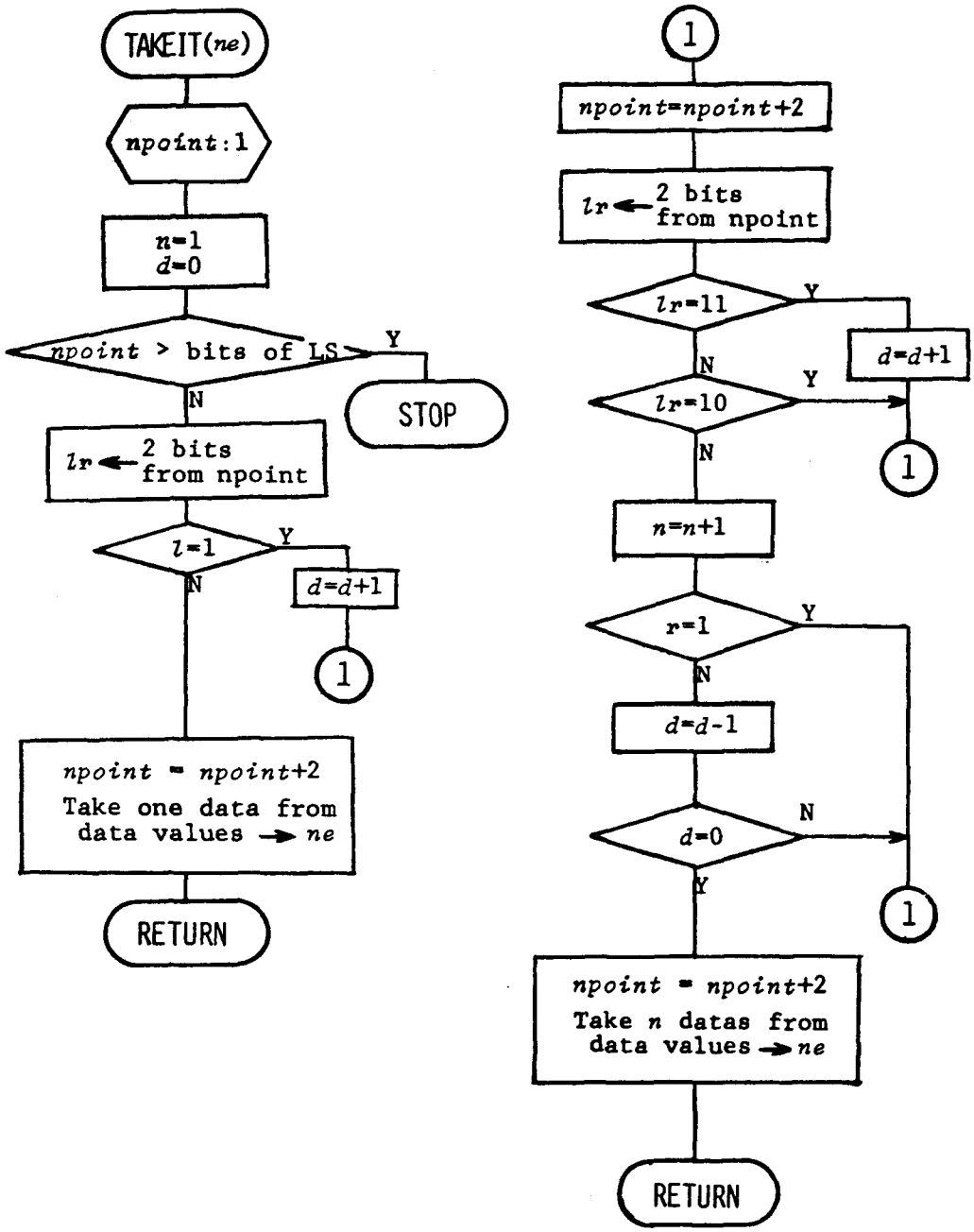


図４－６　単一化計算を実行するためのペア抽出アルゴリズム

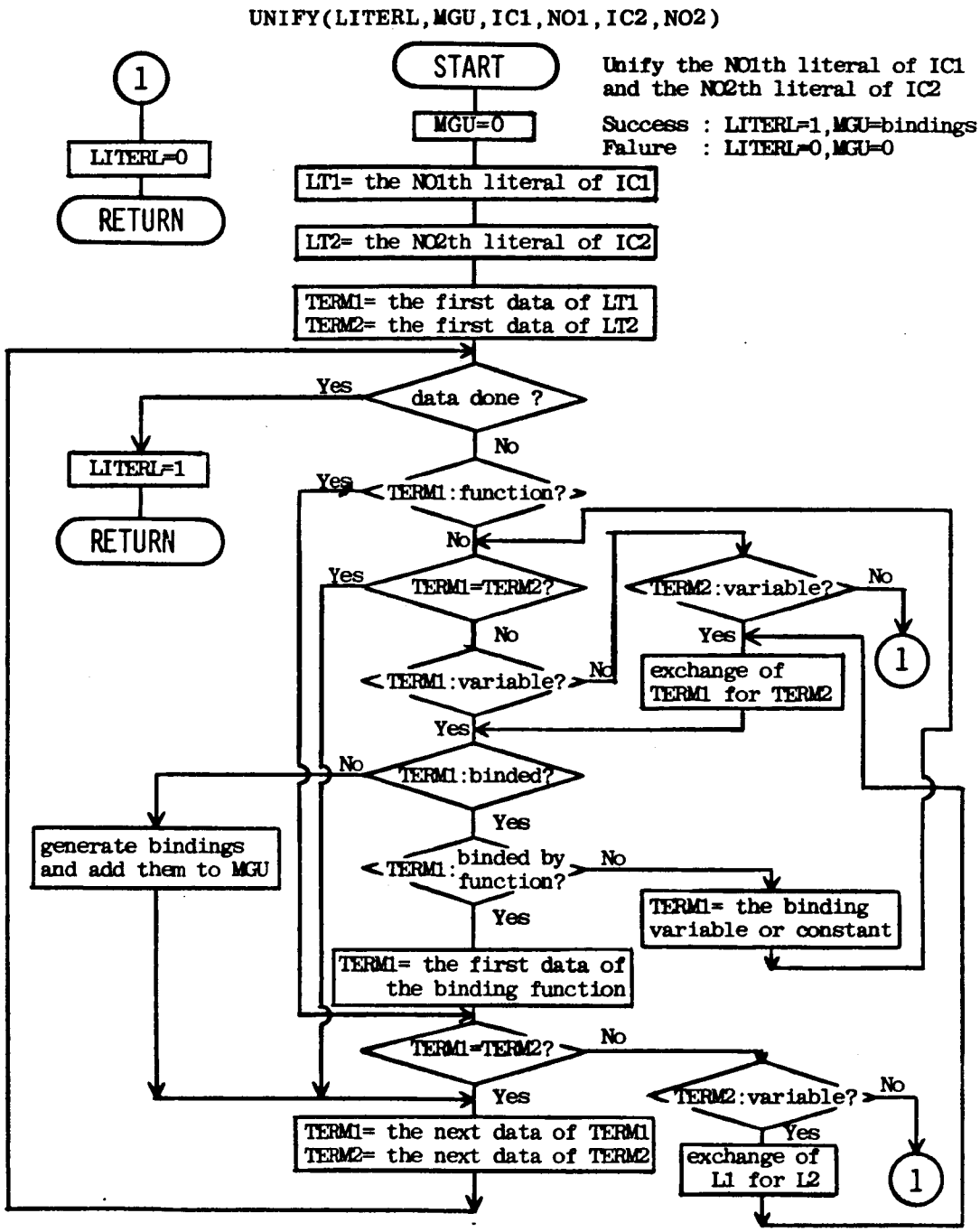


図４－７　ＲＳＧ方式による単一化アルゴリズム

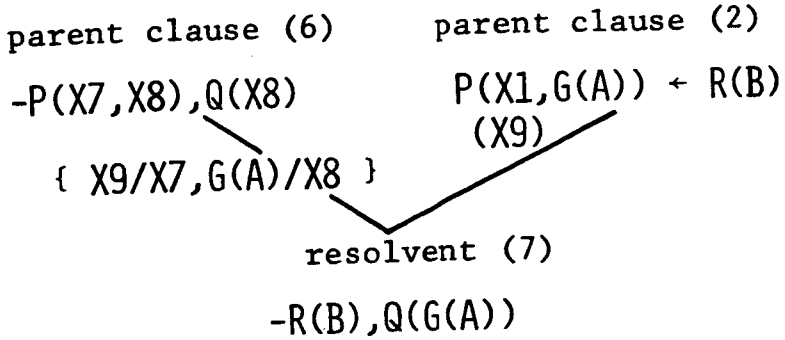


図４－８　導出過程の一例

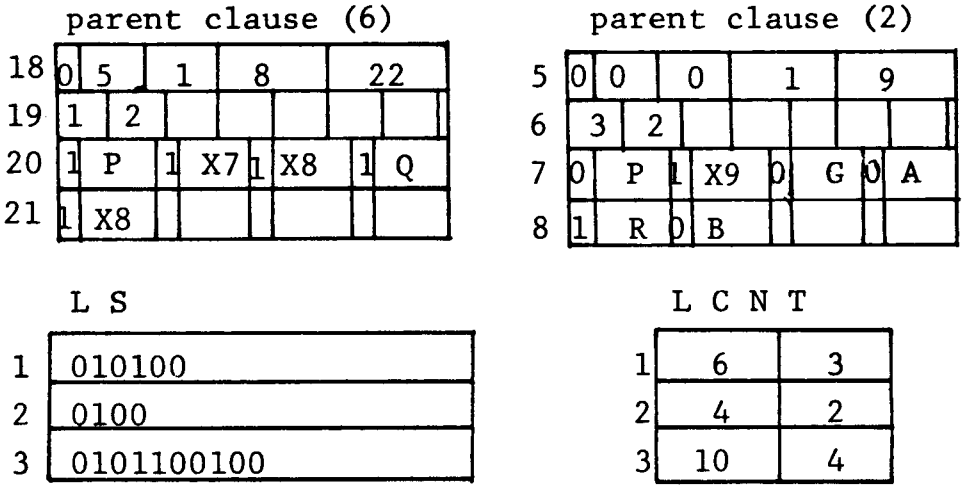


図４－９　各親節の内部表現

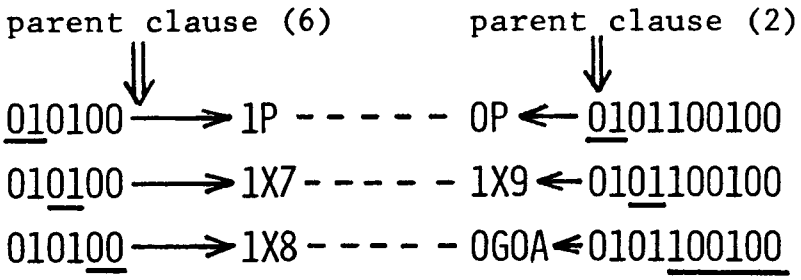


図４－１０　順次取り出された単一化計算を実行するためのペア

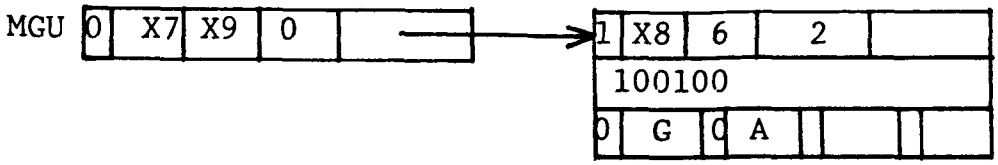


図４－１１　生成された一時的な束縛

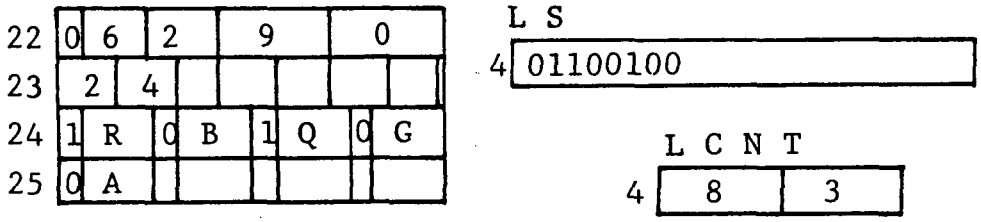


図４－１２　生成された導出形の内部表現

４．４　効率比較実験によるＲＳＧ方式の評価

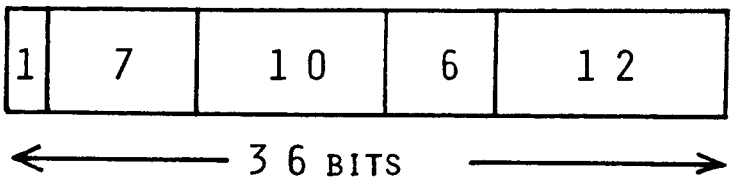
ＲＳＧ方式による単一化計算は、構造をたどる事なく、また束縛を参照する事なく、タップル値を直接操作しながら実行されるため、高速化が期待される。また、節記録は、直接的な表現であり、同型のリテラルリスト構造を蓄えないため、データ圧縮度は高いと考えられる。以上の事を定量的に評価するために、実行効率と記憶効率に関して、リスト構造及び共有構造と効率比較実験を行なった。まず、実験システムについて述べ、次に、実験結果及びその検討について述べる。

４．４．１　実験システムの概要

まず、ＳＧ方式を内部構造とする実験システムは、定理証明システムＳＥＮＲＩを用いた。このシステムでは、導出の実行時に原始的な変数名変換操作を行なったり、変数にある値が代入される度に新しい構造を生成しているため、頻繁に構造をたどる操作が行なわれる。

次に、ＳＳ方式を内部構造とする実験システムは、Ｒ．Ｓ．Boyer〔３〕等によって提案されたものを参考にして作成しており、節記録は図4－１３に示す

様なセル構造となる。また、束縛環境は、連想リストに似た形式で蓄えられている。このシステムでは、導出の実行時に、束縛を参照しながら単一化計算を実行し、新しい束縛の生成および導出形の記録生成が主な操作となる。



1ST FIELD : Whether this clause is a factor or not

2ND FIELD : The order of the (right) parent clause

3RD FIELD : The maximum index

4TH FIELD : The number of literals

5TH FIELD : The pointer to list structure or binding environments

図４－１３　ＳＳ方式を内部構造とする実験システムにおける節記録

最後に、ＲＳＧ方式を内部構造とする実験システムは、４．３節で述べた手順によって生成されており、導出の実行時には、単一化計算を実行するためのペア抽出と一時的な束縛の生成及び導出形の記録生成が主な操作となる。

以上の３つの実験システムにおいては、制御構造及び記憶領域の管理法等、他の環境は全く同じである。また、実験用問題としては、論理プログラムから３題(SORTING,PARSING,4－QUEEN)選出した。

４．４．２　実験結果と検討

表４－２に実行効率の比較と表４－３に生成された導出形に関する記憶領域の比較を示す。

表４－２　各内部構造の実行効率比較実験結果

Internal Structure / Problem	Structure Generating	Structure Sharing	Refined Structure Generating
Sorting	1677	246	134
Parsing	6750	606	458
4－Queen	18892	2054	1615

(m sec)

表４－３　各内部構造の記憶効率比較実験結果

Internal Structure / Problem	Structure Generating	Structure Sharing	Refined Structure Generating
Sorting	567	187	155
Parsing	930	216	243
4－Queen	1241	374	399

(words)

まず、実行効率については、ＳＧ方式は、導出の実行時に構造を頻繁にたどったり、置換操作を繰り返し行なうため最悪である。ＳＳ方式は、導出の実行時に束縛を参照・生成するわけだが、この実行コストは、上記の実行コストに比べてかなり小さいため、実行効率は、大幅に改善されている。最後に、ＲＳＧ方式は、ペアの抽出及び導出生成コストが、小さく押さえられているため、ＳＳ方式より更に効率は改善されている。但し、ＳＧ方式を内部構造とする実験システムでは、リストのコピー方式に効率改善の余地が残され、ＳＳ方式を内部構造とする実験システムでは、束縛の参照は、蓄えられた順に探索しているため、ここに効率改善の余地が残されている。本実験では、平均してＲＳＧ方式は、ＳＧ方式よりも13.0倍効率が改善され、ＳＳ方式と比較しても1.48倍効率が改善されている。

次に、記憶効率については、ＳＧ方式は、節記憶がリテラル等に直接比例して大きくなるため最悪である。ＳＳ方式は、束縛を蓄える必要があるが、１セ

ルで１つの節(導出形)を表現できるため、大幅にデータ圧縮されている。（３つの問題を実行するのに必要な記憶領域で比較すれば、3.52倍改善されている）ＲＳＧ方式は、束縛及び同型のリテラルリストを蓄える必要がなく、またリテラルリストが１ワードで表現できるため、ＳＳ方式とほぼ同程度になっている。(上記と同様に、ＳＧ方式と比較すれば3.44倍改善されている。)

以上の実行効率及び記憶効率の他に、ＲＳＧ方式の特徴としては、節記録が直接的な表現であるため、導出形の内容を強制的に再実行する場合、節の修正操作等は容易に行なえる事と、　２つの節が等しいかどうかを判定する場合(subsumed clause[8]の判定のひとつと考えられる)、リテラルリスト構造値とデータ値を表わすブロック領域の同等性として扱えるため、高速に処理できる事が挙げられる。

４．５　結　言

本章で提案したＲＳＧ方式は、他の定理証明プログラムの内部構造と比較して、以下の特徴を持っていると考えられる。

（１）導出の実行時に、構造をたどる事なく、また束縛環境を参照する事なく、リテラルリスト構造値を利用して、単一化計算を行なうペアを高速に抽出しながら、単一化操作を実行し、一時的な束縛を作成する事により、置換操作を一度だけで済ませているため、実行効率は向上する。

（２）節記録は、束縛環境を蓄える必要がなく、また同型のリテラルリスト構造を蓄える必要がないため、記憶効率は改善される。

（３）節記録は、それだけで節の完全な情報となる直接的な表現であり、導出形の内容を変更する事が容易に実行できる。また、２つの節の同等性が高速に処理できる。

以上の事から、ＲＳＧ方式は、定理証明プログラムにおける新しい置換型の内部構造として位置付けられると考えられる。

第５章　定理証明プログラムにおける導出操作の並列処理

５．１　緒　言

述語論理型処理システムにおいては、幾つかの並列性が内在するが、プログラムの実行過程に相当する導出操作自身の並列性に関する詳しい考察は未だ十分に成されていない。　導出操作は、大きく分けて単一化計算と導出形作成という２つの操作から成立する。このうち実行コストがかかると考えられるのが単一化計算である。この単一化計算の高速化は、J．A．Robinson(１９６５，１９７１)[30][31]に始まり、M．S．Paterson(１９７６)[28]及びA．Martelli(１９７７，１９８２)[23][24]等によって改良が成されているが、いずれも逐次処理における改善にとどまっている。そこで、本章では、単一化計算を並列処理する事により、効率改善の可能性を考察し、導出操作を並列処理する　Parallel　Resolution　Algorithm　を提案する。また、単一化計算に関して、従来の単一化アルゴリズムによって逐次処理する場合と本アルゴリズムによって並列処理する場合の時間計算量の比較を行なうと共に、ソフトウェア　シミュレーションによって本アルゴルズムの有効性を確認する。

５．２　Parallel　Resolution　Algorithm　の特徴

Parallel　Resolution　Algorithm　の特徴は、緒言でも触れた様に、単一化計算を並列処理する事にある。そこで、まず、単一化計算を並列処理するための基礎的考察を行なう。

今、$P(t_1,t_2,\cdots,t_n)$と$\sim P(s_1,s_2,\cdots,s_n)$の単一化問題を考えると、これは、以下の様な連立方程式として表現でき、単一化作用素がこの解と考えられる。

$$\begin{cases} t_1 = s_1 \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ t_n = s_n \end{cases} \qquad (5-1)$$

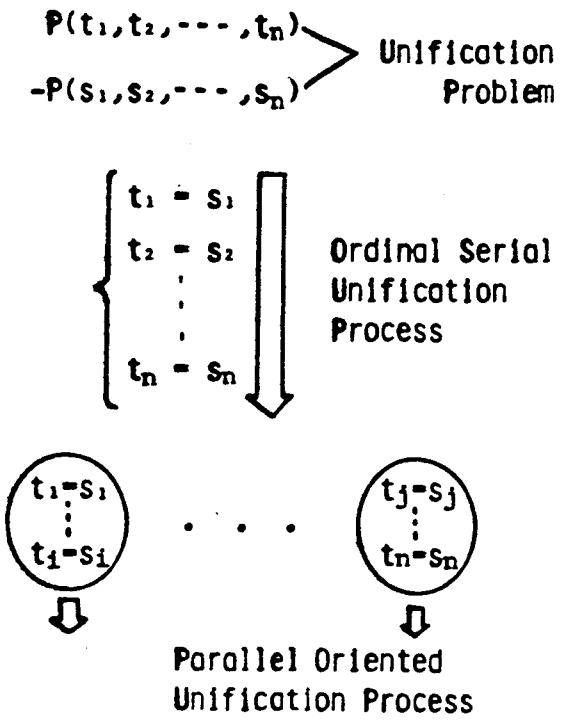


図５－１　単一化計算の逐次処理と並列処理

従来の単一化計算は、(５－１)式の連立方程式を逐次的に解く操作と考えられるが、(５－１)式のn個の方程式は、幾つかの独立な方程式の集合に分割できる可能性がある。すなわち、表現の集合(リテラルペアの引数の対の集合)を(５－２)式の様に表わせば、(５－３)式の様に分割できる可能性がある。

$$W = \{ti=si \ , \ i=1,2,\cdots,n\} \qquad (5-2)$$

$$W = W_1+W_2+\cdots+Wn \qquad (5-3)$$

但し、$Wi \subseteq W$ であり、WiとWj $(i \neq j)$は独立に解ける方程式の集合である。Wiを部分表現集合と呼ぶ。

本アルゴリズムに組み込まれている単一化計算を並列処理する手順は、(5－3)式の可能性に着目するものであり、各部分表現集合が互いに共通な変数を含まない様にWを分割する(Wのクラスタリングと呼ぶ)事によって、(5－3)式を実現する事を考える。この手順は、独立な部分表現集合を作成し、それらを従来の単一化アルゴリズムで計算するため、この手順の正当性及び停止性は、Unification Theorem[8]を使って同様に保証される。

以上の考察を基に、本アルゴリズムの概要を図5－2に示す。前処理として、変数に着目してWのクラスタリングを行ない、単一化計算を並列に実行できる部分表現集合W_1－Wn(クラスタと呼ぶ)を生成する。以下、各クラスタの単一化を並列に実行し、すべてのクラスタが単一化可能であれば、各クラスタのmguを合併する事によってWのmguを得、ひとつでも単一化不可能であれば、Wは単一化不可能とし停止する。Wのmguを得た後、導出形とそのクラスタリング情報を作成して停止する。このクラスタリング情報は、クラスタリングを高速に実行するための情報であり、5．4節で説明する。

図5－2では、比較のために、従来の導出アルゴリズムも掲げたが、○印を付けた処理が、本アルゴリズムに特有の処理である。このうち、Wのクラスタリングと導出形のクラスタリング情報作成の処理時間を小さく押さえる事が重要であると考えられるが、この事項についても5．4節で触れる。

以上説明した本アルゴリズムの概要のうち、単一化計算を並列処理するプロセスの具体例を図5－3に示す。

5．3 単一化計算における時間計算量の比較

本節では、従来の単一化計算を逐次処理する場合と本アルゴリズムで並列処理する場合の時間計算量の比較を行なう。但し、単一化計算が成功する場合に限定している。

まず、各記号を以下の様に定義して、従来の単一化アルゴリズムの時間計算量を導出する。

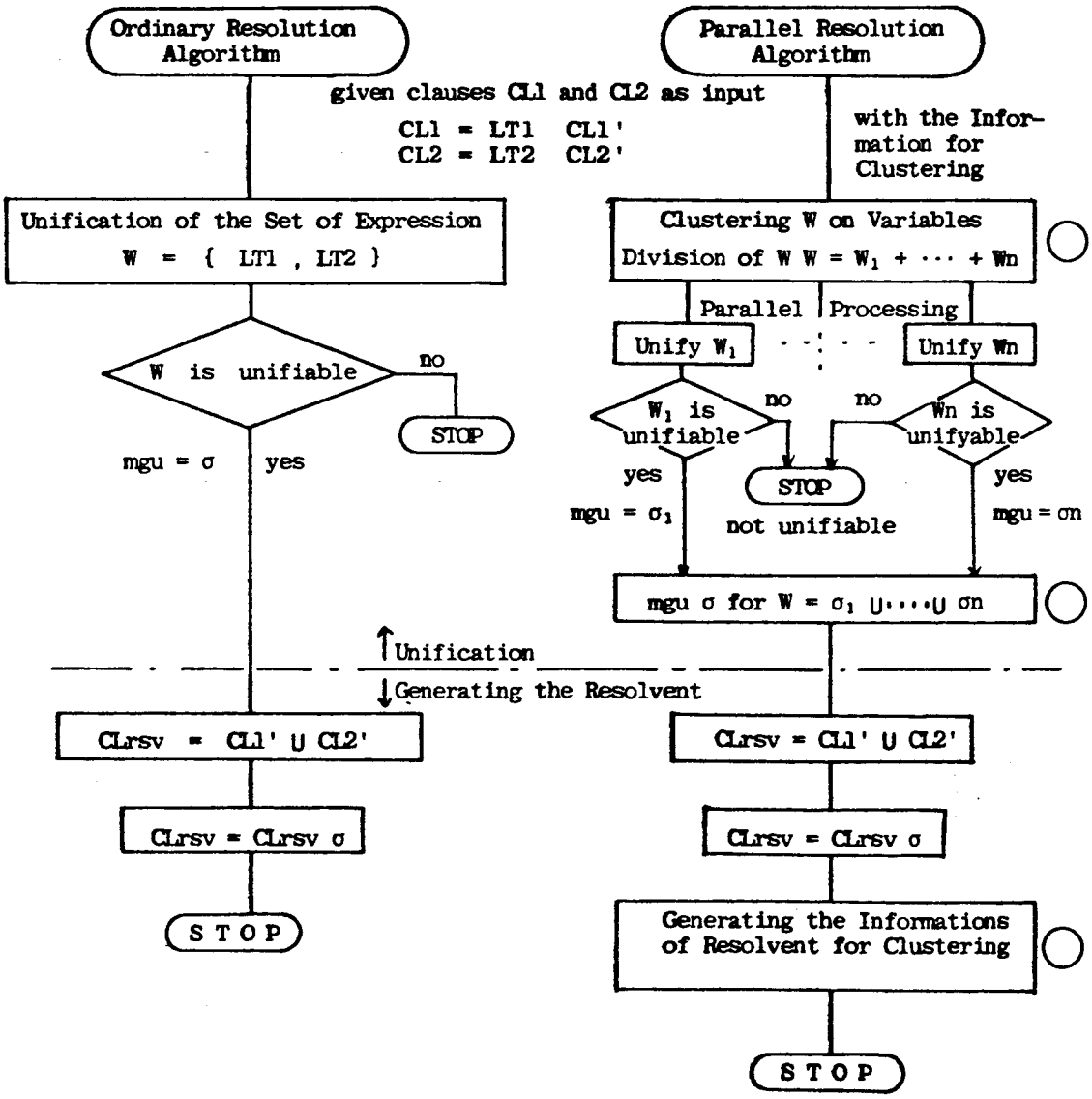


図5－2　Parallel Resolution Algorithm の概要

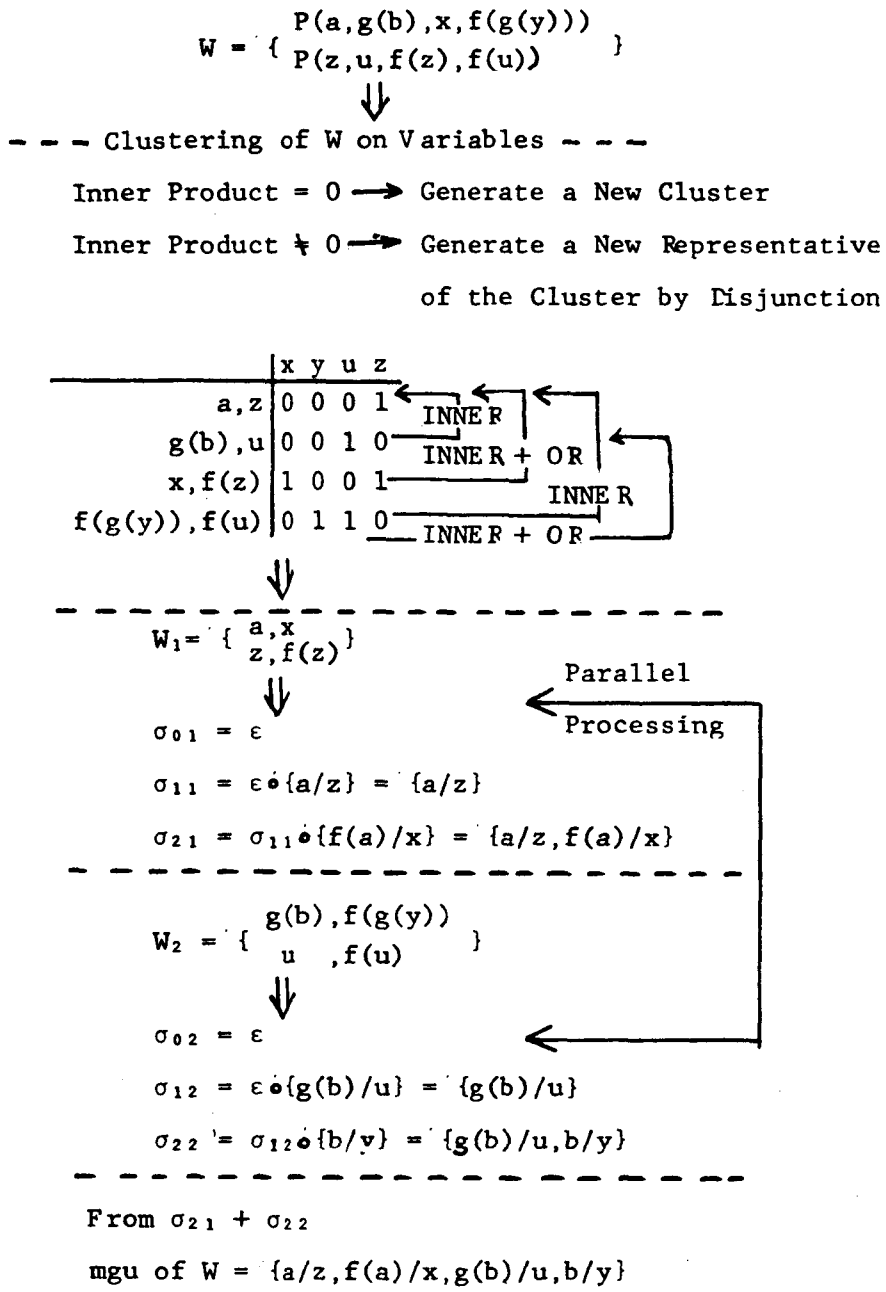


図５－３　単一化計算の並列処理

Ne　：　単一化判定を行なう要素数

Td　：　要素の一致を判定する単位時間

Tsb　：　代入構成要素を作成する単位時間

Tst　：　代入構成要素を用いて、新しい表現の集合を作成する単位時間

Tc　：　代入を合成する単位時間

ある要素が一致するかどうかを判定する操作をBernoulli試行(一つの試行結果が、二つの可能性のどちらかになる独立反復試行)と考えて、要素が一致しない確率をpとする。この場合、i番目より前に発生した不一致の回数がkである確率は、

$$P_s(i,k)={}_{i-1}C_k p^k (1-p)^{i-k-1} \qquad (5-4)$$

で表わせる二項分布となる。従って、kの平均値は、以下の様に表わせる。

$$\bar{k}=\sum_{k=1}^{i-1} P_s(i,k)\cdot k=(i-1)p \qquad (5-5)$$

さて、代入構成要素を用いて、新しい表現の集合を作成する時間は、Neに比例すると考えられ、代入を合成する時間は、kに比例すると考えられる。従って、i番目の要素の単一化に要する時間の平均値を$C_1(i)$とすれば、

$$\begin{aligned} C_1(i)&=T_d+p(T_{sb}+N_eT_{st}+\bar{k}T_c) \\ &=T_d+p(T_{sb}+N_eT_{st}+(i-1)pT_c) \end{aligned} \qquad (5-6)$$

の様に表わせる。

式(5－4)~(5－6)より、従来の単一化アルゴリズムの時間計算量をC1とすれば、

$$C1=\sum_{i=1}^{N_e} C_1(i)$$

$$=N_e{}^2(pT_{st}+\frac{p^2}{2}T_c)+N_e(T_d+pT_{sb}-\frac{p^2}{2}T_c) \qquad (5-7)$$

の様に表わせる。(５－７)式より、従来の単一化アルゴリズムの時間計算量は、$O(Ne^2)$である事がわかる。

次に、本アルゴリズムで単一化計算を並列処理する場合の時間計算量Ｃ２をクラスタリングコストとクラスタリング後の最大の単一化計算のコストの和で求める。

Ｍ個のクラスタに分割される時の最大コストは、図５－４の状況において発生し、(５－８)式で示されると考えられる。但し、各記号は、以下の様に定義する。

Tinner ： 内積を実行する単位時間

Tor ： 論理和を実行する単位時間

$$C(M)_{clus.} = \sum_{k=1}^{M-1}(N_e-k)T_{inner} + \sum_{k=1}^{N_e-M} kT_{inner} + (N_e-M)T_{or}$$

$$= \frac{N_e(N_e-1)}{2}T_{inner} + (N_e-M)T_{or} \qquad (5-8)$$

式(５－８)のＭを一様分布と仮定する事により 、クラスタリングコスト Cclus. を求めると、次式の様に表わされる。

$$C_{clus.} = \sum_{M=1}^{N_e}\frac{1}{N_e}C(M)_{clus.}$$

$$= \frac{N_e(N_e-1)}{2}T_{inner} + \frac{N_e-1}{2}T_{or} \qquad (5-9)$$

以下、クラスタリング後の単一化計算の最大コストについて考察する。不一致集合が、各クラスタに平均して分割されると仮定すれば、各クラスタは並列に処理されるため、実行コストは同じになる。従って、クラスタ数がＭの場合、単一化計算の最大コスト$C(M)_{max\ unify}$は、Ne／Ｍを式(５－７)に代入する事により、以下の様に得られる(図５－５参照)。但し、Tunionは各mguを合併する時間である。

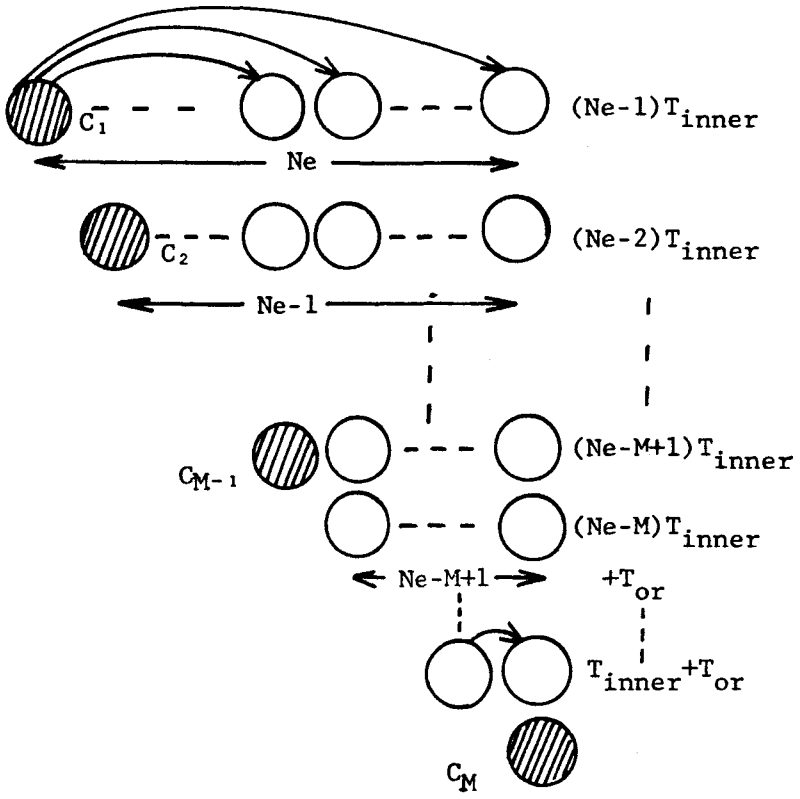


図５－４　Ｍ個のクラスタに分割される状況

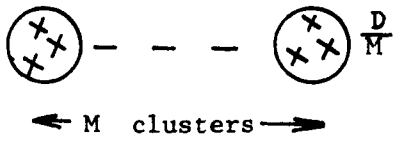


図５－５　クラスタリング後の状況

$$C(M)_{\max\ unify}=\left(\frac{N_e}{M}\right)^2\left(pT_{st}+\frac{p^2}{2}T_c\right)+\frac{N_e}{M}\left(T_d+pT_{sb}-\frac{p^2}{2}T_c\right)+T_{union}$$

（５－１０）

Mを一様分布とすれば、

$$C_{\max\ unify} = \sum_{M=1}^{N_e} \frac{1}{N_e} C(M)_{\max\ unify}$$

$$= \frac{\pi^2}{6} N_e (pT_{st} + \frac{p^2}{2} T_c) + (\ln N_e + \gamma + \frac{1}{2N_e})(T_d + pT_{sb} - \frac{p^2}{2} T_c) + T_{union}$$

(5－11)

式(5－9)と式(5－11)より、C2は、

$$C2 = C_{clus.} + C_{\max\ unify}$$

$$= \frac{N_e(N_e-1)}{2} T_{inner} + \frac{N_e - 1}{2} T_{or} + \frac{\pi^2}{6} N_e (pT_{st} + \frac{p^2}{2} T_c) + (\ln N_e + \gamma + \frac{1}{2N_e})$$

$$\cdot (T_d + pT_{sb} - \frac{p^2}{2} T_c) + T_{union}$$

(但し、γはオイラー定数でγ≒0．577)

(5－12)

の様に表わせる。従って、C1とC2を各基本操作の実行回数で比較すれば、表5－1の様になる(但し、表中の値はオーダである)。

表5－1　　従来の単一化アルゴリズムと本アルゴリズムで
単一化計算を並列処理する場合の時間計算量の比較

基本操作 / アルゴリズム	T_d	T_{sb}	T_{st}	T_c	T_{inner}	T_{or}	T_{union}
従来の単一化アルゴリズム	N_e	N_e	N_e^2	N_e^2	—	—	—
並列処理向き単一化アルゴリズム	$\ell n N_e$	$\ell n N_e$	N_e	N_e	N_e^2	N_e^2	C

表５－１より、クラスタリングの実行コストを除いた単一化の実行コストのみを比較すれば、従来の単一化アルゴリズムの時間計算量が$O(Ne^2)$であるのに対して、本アルゴリズムで単一化計算を並列処理する場合の時間計算量は$O(Ne)$になる事がわかる。　従って、本アルゴリズムを有効なものにするには、クラスタリングの実行コスト(特にTinnerの実行コスト)を小さく押さえるデータ構造の導入が不可欠である(導出操作全体においては、クラスタリング情報の作成コストも小さく押さえる必要がある)。また、本節の考察は、単一化可能な場合に限ってきたが、実際には、不可能な場合も多く発生する。しかし、単一化不可能な場合は、状況が複雑であり、解析は困難である。そこで、次節以下においては、まず本アルゴリズムに適したデータ構造について考察し、次に、ソフトウェア　シミュレーション　によって、本アルゴリズムの有効性を検討する。

５．４　本アルゴリズムに特有な処理の高速化

通常の導出アルゴリズムに見られない本アルゴリズムの特有の処理としては、Wのクラスタリングとクラスタリング情報の生成がある。そこで、本節では、これら２つの処理を効率良く実行する事を検討する。

５．４．１　クラスタリング情報

導出操作は失敗すれば、被導出リテラル候補を次々に変えて実行される。従って導出の度に、被導出リテラルペアの変数を調べて、クラスタリングを行なえば効率が低下する。そこでまず、クラスタリングを効率的に行なうための情報「クラスタリング情報」とそれを蓄えるデータ構造について検討する。

クラスタリングは、変数に基づいて実行されるため、クラスタリング情報としては以下の２つの情報を採用した。

（１）　引数の変数情報

（２）　リテラルのクラスタ情報

（１）は、リテラルの引数が各々どの変数を含んでいるかを示す情報である。クラスタリングは、基本的には、各リテラルの引数の変数情報を基にして実行されるため、これを常に利用できる形で蓄えておく。また、この情報は、内積(論理積)及び論理和の計算が、容易に実行できる形式が望ましい。そこで、データ構造としては、各変数をrenaming(変数名変換操作)によってXn(nは自然数)の形に変換して、自然数と１対１の対応を持たせ、セルのnビット目に１を立てる事により、Xnを含む事を表現する方法を採用する。

（２）は、リテラルが単独でどの様に分割できるかを示す情報であり、図５－６に示すアルゴリズムにより得られる。この情報を基にして、リテラルペアのクラスタリングは、次に述べるクラスタ演算によって効率良く処理できる。データ構造としては、やはり論理積や論理和の計算が容易である事が望ましいため、ある引数が引数リストの先頭からn番目である事をセルのnビット目に１を立てる事により表現する方法を採用する。

以上の２つの情報と変数情報の数(リテラルの引数の数)及びクラスタの数を表わす情報を合わせてクラスタリング情報とし、図５－７の様に、各リテラルに対応付けて格納する。

５．４．２　クラスタリング

本アルゴリズムの前処理であるリテラルペアのクラスタリングは、あらかじめ生成したリテラルのクラスタ情報をクラスタ演算する事によって実現される。このクラスタ演算は、図５－８にそのアルゴリズムが示されるが、一方のクラスタ情報のn番目の成分と他方のクラスタ情報の全要素とのマッチングをとり、マッチングのとれた成分を結合する事により、単一化計算のためのクラスタを順次形成していく操作である。たとえば、図５－９においては、$P(a,g(x_1),x_2,f(g(x_2)))$と$-P(x_4,x_3,f(x_4),f(x_3))$のクラスタ情報にクラスタ演算を施し、順次マッチング(図中の＊)をとりながら、最終的に１つのクラスタになる事が示されている。

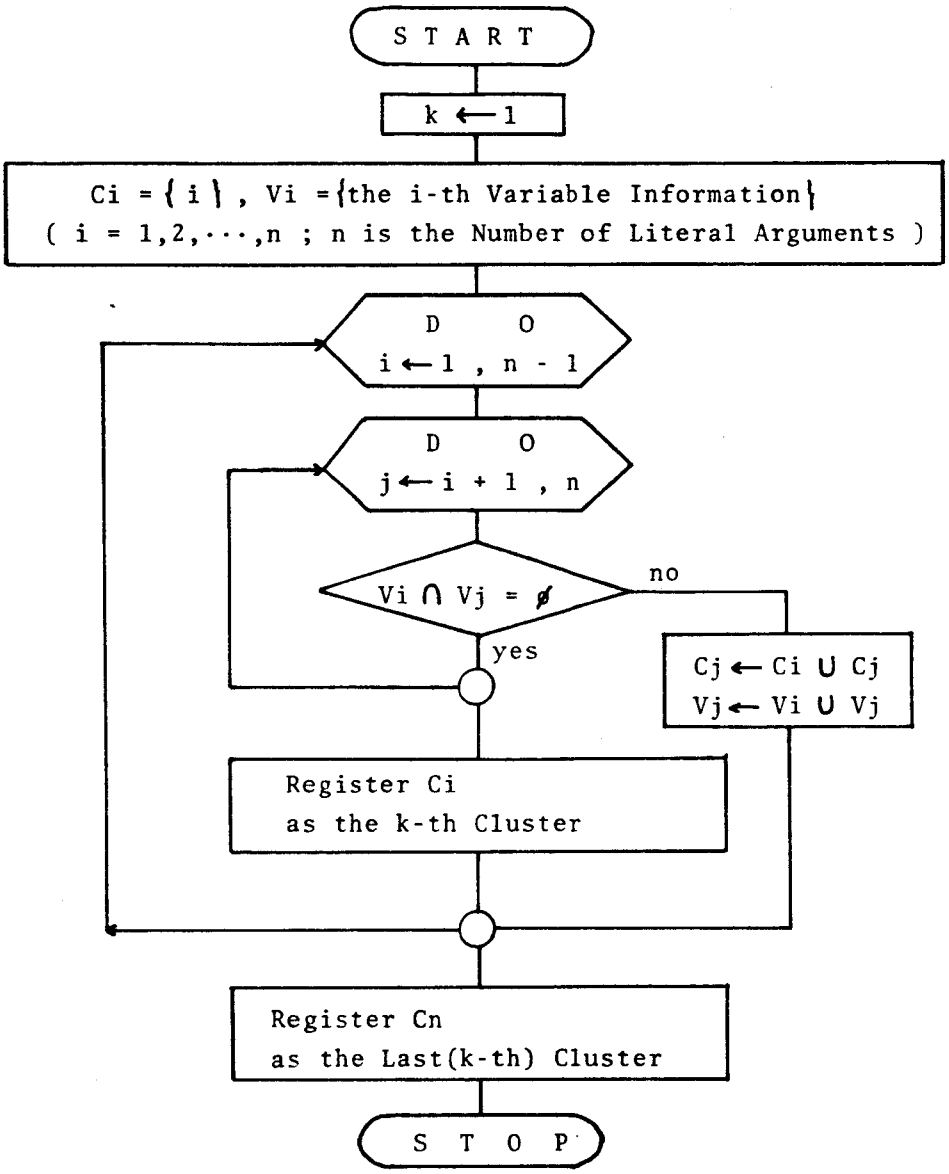


図５－６　リテラルのクラスタ情報生成アルゴリズム

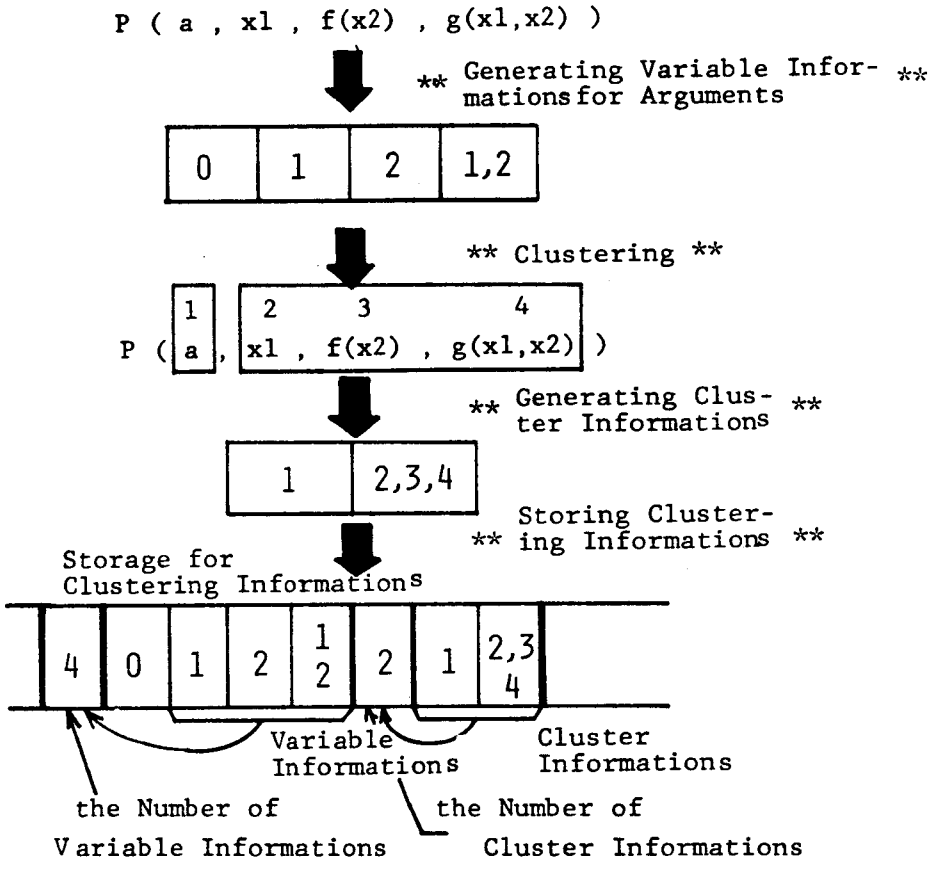


図５－７　クラスタリング情報を蓄えるデータ構造

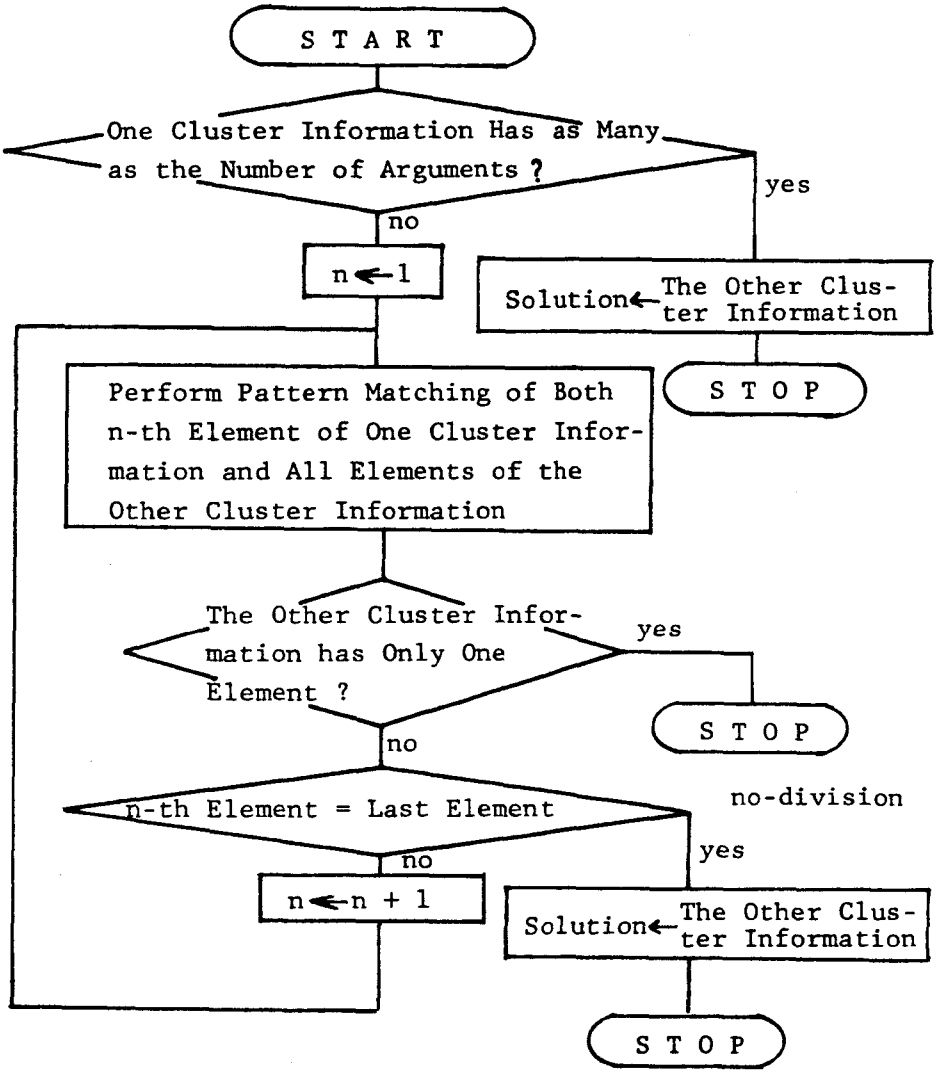


図５－８　クラスタ演算アルゴリズム

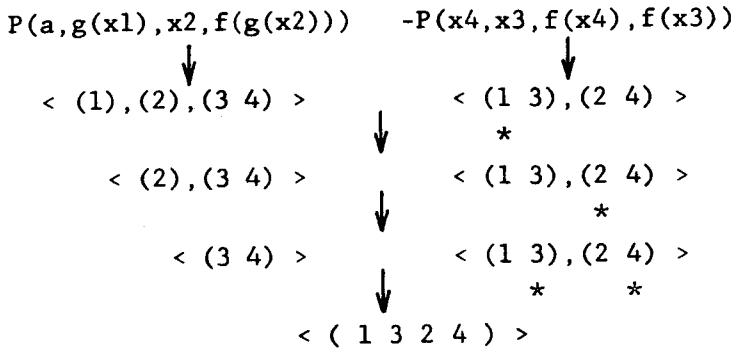


図５－９　クラスタ演算の具体例

５．４．３　クラスタリング情報の作成

５．４．１で述べたクラスタリング情報は、クラスタリングの実行コストを小さくするために導入したものであった。しかしながら、このクラスタリング情報は、導出成功時に毎回作成しなければならないため、実行コストの低減が望まれる。そこで、親節の変数情報を利用した効率的なクラスタリング情報作成手順を示す。図５－１１に、そのゼネラルフローを示し、以下図５－１０の具体例が、各フェーズでどの様に処理されるかを述べる。

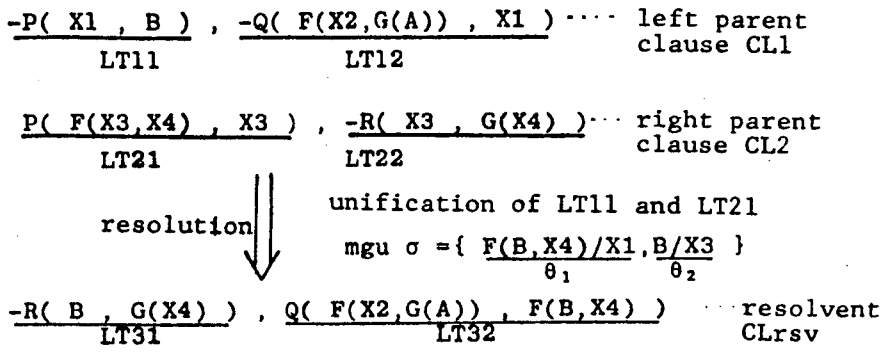


図５－１０　導出過程の一例

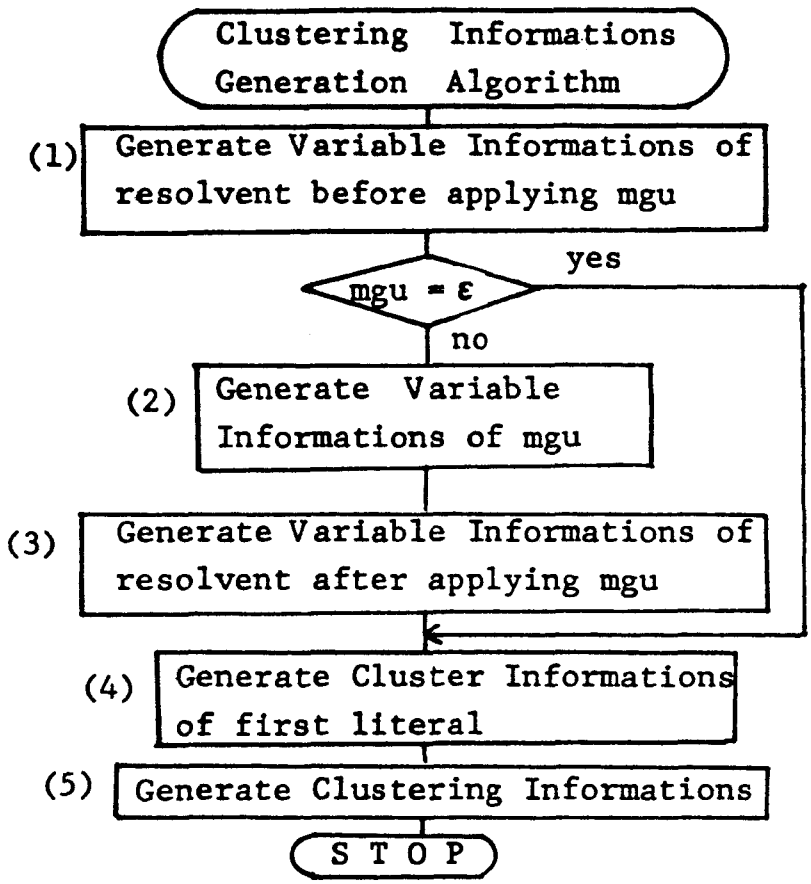


図５－１１　クラスタリング情報作成アルゴリズム

（フェーズ１）　両親節の変数情報から、ＬＴ１２とＬＴ２２の部分を抽出して結合し、まだmguが施されていない導出形ＣＬrsv'の変数情報を作成する(図５－１３)。

（フェーズ２）　mguの代入構成要素に対して、代入する項に含まれる変数番号と代入される変数番号の組みを作成する(図５－１４)。

（フェーズ３）　ＣＬrsv'の変数情報にmguの変数情報を作用させる事により、導出形ＣＬrsvの変数情報を作成する(図５－１５)。

（フェーズ４）　ＣＬrsvの先頭リテラルの変数情報を図５－６の方法でクラスタリングし、クラスタ情報を作成する。

（フェーズ５）　フェーズ３と４で作成した導出形の変数情報と先頭リテラルのクラスタ情報を合わせて、導出形のクラスタリング情報とし記憶領域に格納する。

以上の処理で、フェーズ２だけは、出現する変数を調べる必要があるため、少し処理時間がかかると予想されるが、他は比較的単純な操作であるので、全体としては、実行コストは小さく押さえられると考えられる。

$$CLrsv' = (CL1-LT11) \cup (CL2-LT21)$$
$$= LT22 \cup LT12$$
$$= \underbrace{\sim R(X3, G(X4))}_{LT31'}, \underbrace{\sim Q(F(X2,G(A)), X1)}_{LT32'}$$

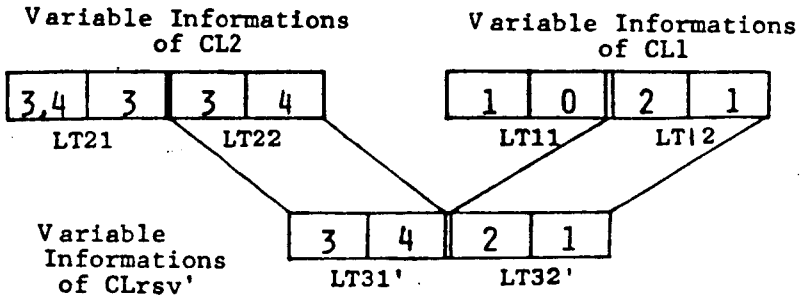


図５－１２　フェーズ１の処理

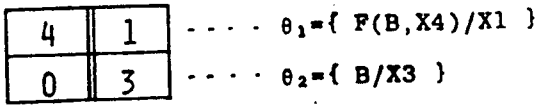


図５－１３　フェーズ２の処理

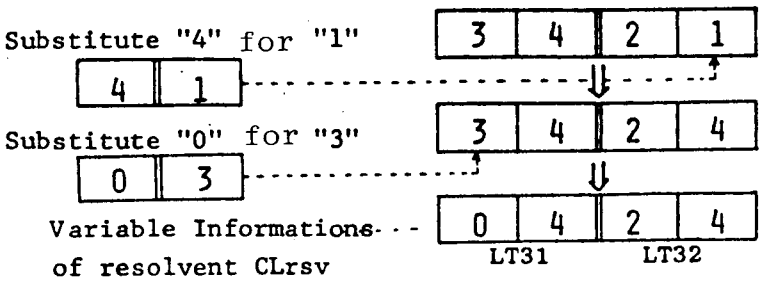


図５－１４　フェーズ３の処理

５．５　効率比較実験による本アルゴリズムの評価

５．３節で本アルゴリズムの時間計算量及び５．４節で本アルゴリズムで特有の処理を高速に実行する手順を示したが、実際に本アルゴリズムがどの程度有効であるのか分からない。　そこで、従来の導出アルゴリズム（Ｊ．Ａ．Robinson，１９６５）[30]と本アルゴリズムを定量的に比較するために、ソフトウェア　シミュレーションによって効率比較実験を行なった。

５．５．１　実験システムの概要

実験方法としては、導出を実行しながら各処理の所要時間を測定し、導出操作全体に要する時間の合計で比較する。

従来のアルゴリズムの実験システムは、リスト構造(置換型)を内部構造とする定理証明システムＳＥＮＲＩ(第３章参照)を基にして作成しており、導出法はＳＮＬ導出法を使用している。また、本アルゴリズムの実験システムにおいて、単一化計算の並列処理時間を測定する部分は、図５－１５に示すゼネラルフローに沿って作成している。このゼネラルフローでは、単一化計算の並列処理を（１)述語記号の比較、（２)クラスタ演算、（３)各クラスタの単一化、

(4)各mguの合併という4つの処理に分けて、処理時間を求めている。

従来のアルゴリズムの総導出処理時間Tsraは、以下の式で求まる。

$$Tsra = Tu + Tr \qquad (5-13)$$

Tu ： 逐次単一化計算の処理時間計

Tr ： 導出形作成時間計

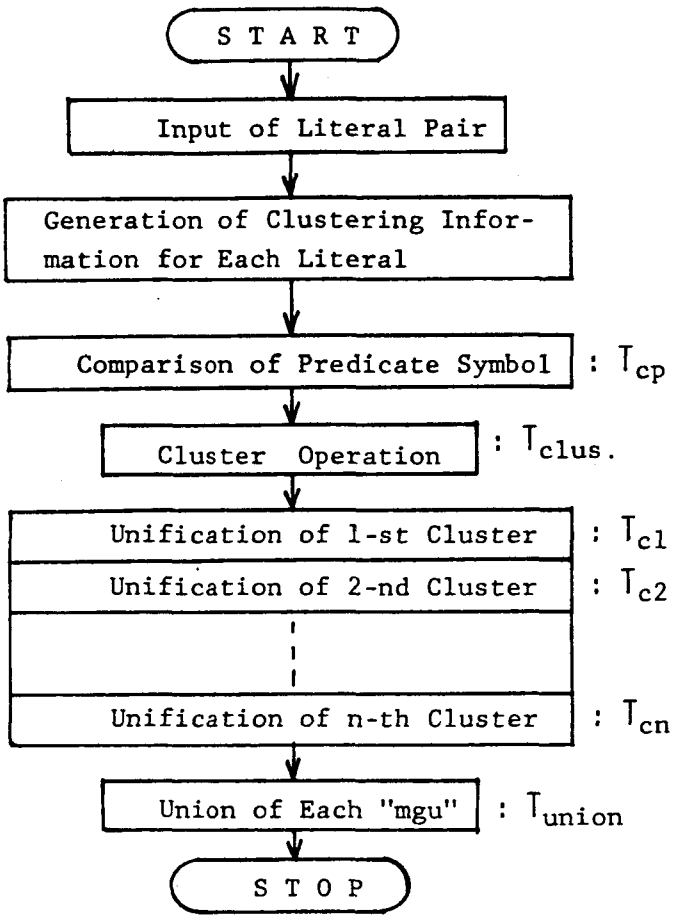


図5－15　単一化計算の並列処理時間を測定するためのゼネラルフロー

また、本アルゴリズムの総導出処理時間Tparaは、以下の式で求まる。

$$Tpara = Tpu + Tr' \qquad (5-14)$$

Tpu ： 単一化計算の並列処理時間計

Tr' ： 導出法及びクラスタリング情報作成時間計

但し、Tpuは以下の式より求まる。

$$Tpu = Tcp + Tclus + \begin{cases} \max(Tc_1, \cdots, Tcn) + Tunion & (5-15) \\ \min(Tc_1', \cdots, Tcm') & (5-16) \end{cases}$$

Tcp	：	述語記号の比較に要する時間
Tclus	：	クラスタ演算に要する時間
$Tc_1 \sim Tcn$	：	各クラスタの単一化に要する時間
$Tc_1' \sim Tcm'$	：	単一化不可能である事を判定する時間
Tunion	：	各mguを合併する時間
n	：	クラスタ数
m	：	単一化不可能なクラスタ数

上記の２式は、以下の事を意味している。

式(５－１５) － リテラルペアが単一化可能な場合の処理時間であり、各クラスタの単一化は、最も処理時間のかかるクラスタが単一化できた時点で完了し、mguの合併を行なう事により、単一化計算は終了する。

式(５－１６) － リテラルペアが単一化不可能な場合の処理時間であり、一つのクラスタが単一化不可能であると判明した時点で、単一化計算は終了する。

実験データとしては、論理プログラムの問題(APPENDとMEMBER)とＤＢ検索問題及びＬＯＯＰ問題(付録参照)を使用した。

５．５．２　実験結果と検討

従来の導出アルゴリズムと本アルゴリズムの効率比較実験結果を表５－２に示す。

表５－２　導出操作の効率比較実験結果

Comparison items \ Problems		MEMBER	APPEND	LOOP	D B
The number of times of resolution trial		6	5	34	57
The number of times of resolution success		4	3	8	4
The number of clusters for pairs of resolved literals		1～2	2	2～3	3～4
S R A	The total time of unification process	6,000	10,245	29,605	11,820
	The total time of generating resolvents	490	485	1,980	1,335
	The total time of resolution process	6,490	10,730	31,585	13,155
P A R A	The total time of unification process	4,220	5,775	12,550	4,560
	The total time of generating resolvents	1,025	1,290	3,830	1,785
	The total time of resolution process	5,245	7,065	16,380	6,345
I R	Improvement rate in unification process	1.42	1.77	2.36	2.59
	Improvement rate in resolution process	1.24	1.52	1.93	2.07

unit time : usec
machine : ACOS system 1000 model 40

S R A : Serial Resolution Algorithm
PARA: Parallel Resolution Algorithm
I R : Improvement Rate

表５－２より、導出形作成の実行コストを比較すると、本アルゴリズムの導出形作成時間計は、従来の導出アルゴリズムのそれと比較して、約２倍程度になっている。従って、クラスタリング情報作成コストは、従来の導出形作成のコストとほぼ同程度であり、全体の導出操作の実行コストに与える影響は小さいと考えられる。

一方、単一化計算における改善率と導出レベルでの改善率を比較すると、導出操作にはクラスタリング情報作成という特別な操作が加わるので、導出レベルでの改善率は悪くなっている。しかしながら、その改善率の悪化は、２割減程度に押さえられており、　導出レベルの改善率は、　発生するクラスタ数の

０．６～０．７倍程度である。これは、クラスタリング情報作成のために加わる時間は、単一化計算を並列処理する事により短縮できる時間に比べれば、十分小さくなっているためだと考えられる。改善率は共に、クラスタ数に比例して向上している事が分かる。

次に、本アルゴリズムが有効になる場合とそうでない場合が、明確に現われたので、各例を以下に示すと共に、それらの処理時間を表５－３に示す。

（例５－１）　単一化可能で本アルゴリズムに有効なペア

```
Literal  Pair  1
MICOM(FM8,FUJITU,6809,218000)
MICOM( X1,   X2   , X3  ,218000)
```

Literal Pair 1 は、前処理のクラスタリングで４つのクラスタに分割でき、各クラスタのmguは、一つの代入構成要素しか持たないため、構造をたどる操作が軽減され、単一化計算の並列処理の効果が大きく現われている。また、単一化可能で本アルゴリズムに有効でないリテラルペアは、例には挙げないが、前処理で、複数のクラスタに分割出来ないペアである。

（例５－２）　単一化不可能で本アルゴリズムに有効なペア

```
Literal  Pair  2
MICOM(MZ80B,SHARP,Z80A,278000)
MICOM(   X1  ,   X2  , X3  ,218000)
```

Literal Pair 2 は、従来のアルゴリズムでは、第４引数で始めて、単一化不可能と判定できるが、本アルゴリズムでは、４つのクラスタに分割でき、第４クラスタが単一化不可能となった時点で、計算が終了するため、Literal Pair 1 に比べて、並列処理の効果が更に大きく現われている。

（例５－３）　単一化不可能で本アルゴリズムに有効でないペア

Literal pair 3

MAKER(OKI ,TOKYO, E)

MAKER(SHARP, X1 ,X2)

単一化不可能で本アルゴリズムに有効でないリテラルペアは、前処理で、複数のクラスタに分割できないペアの他に、Literal Pair 3 の様なペアがある。すなわち、従来のアルゴリズムでは、第１引数の比較で、直ちに単一化不可能と判定できる場合であり、本アルゴリズムでは、前処理の時間だけ余分に必要とする。

表５－３　リテラルペアの処理時間の比較

pairs of resolved literal / Comparison items	Literal Pair 1	Literal Pair 2	Litearl Pair 3
T_{SRA}	1075	1035	60
T_{PARA}	535	195	150

unit time : u sec

machine : ACOS system 1000 model 40

本実験に用いた４つの問題では、並列処理に有効なペアとそうでないペアが共に現われ、偏りが少ないと考えられるが、有効なペアにおけるコスト減が有効でないペアにおけるコスト増に比べてかなり大きいために、全体としては、本アルゴリズムの有効性は十分に現われている。

最後に、クラスタリング情報をクラスタリング以外に導出操作の効率改善に利用する事について考察する。引数の変数情報及びmguの変数情報は、以下の利用法が考えられる。

（１）　代入構成要素作成時のOccur Checkの効率化

Occur Checkとは、単一化アルゴリズムにおいて、不一致集合に項Tkと

変数Xkがある時、Tkの中にXkが現われるかどうかを調べる操作を言う。この時、TkとXkが共にリテラルの引数ならば、それぞれの変数情報の論理積をとる事によって、容易にOccur Checkが実現できる(図5－16参照)。

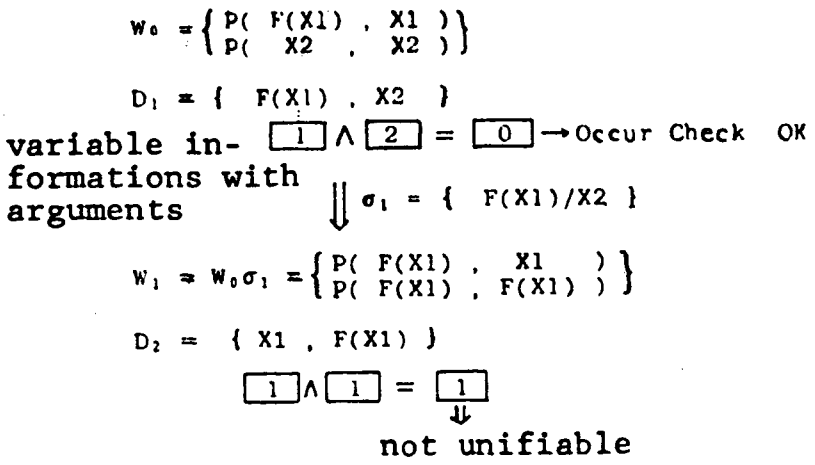


図5－16　Occur　Check　の効率化

(2)　代入の合成の効率化

代入構成要素を作成する度にその変数情報を作成する事にすれば、その変数情報を利用して、ある代入構成要素が合成の影響を受けるかどうかを容易に判定できる(図5－17参照)。

(3)　導出形作成における代入の効率化

引数の変数情報とmguの変数情報を利用して、それらを照合する事により、ある引数に代入が必要かどうかが容易に判定でき、無駄に構造をたどる事が少なくできる(図5－18参照)。

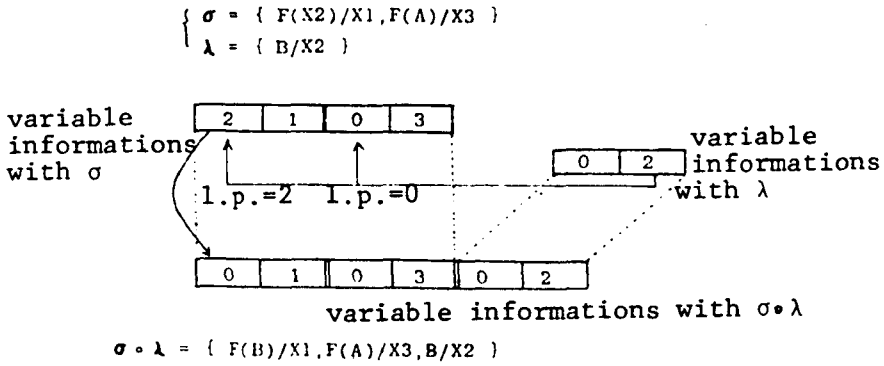


図５－１７　代入の合成の効率化

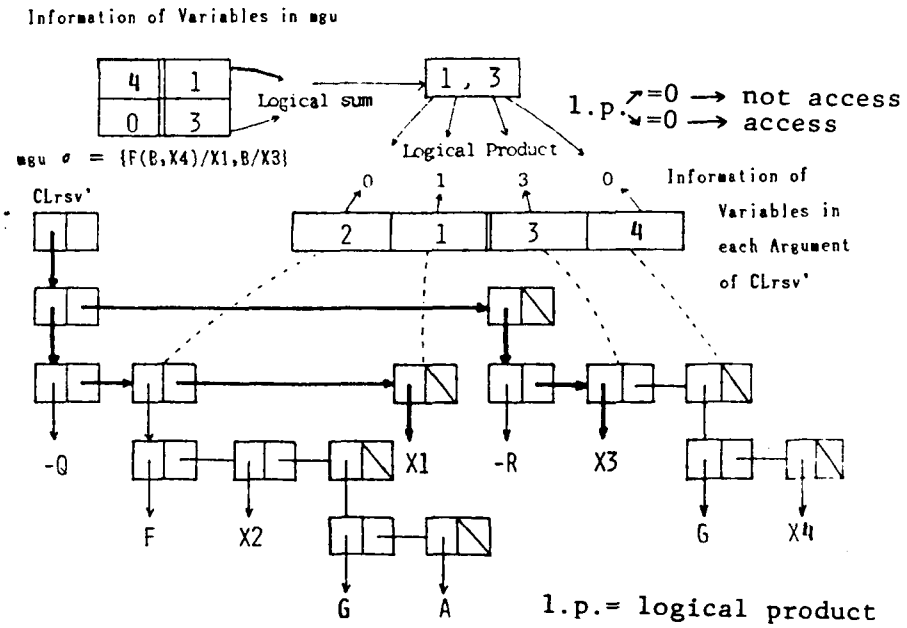


図５－１８　導出形作成における代入の効率化

5．6　結　言

本章で提案した Parallel Resolution Algorithm は、単一化計算を並列処理する事に特徴がある。この単一化アルゴリズムの時間計算量に関して、単一化計算が成功する場合、従来の単一化アルゴリズムが入力系列の要素数nに対して$O(n^2)$の時間計算量になるのに対して、本アルゴリズムの単一化計算の並列処理では$O(n)$の時間計算量になった。しかしながら、この理論値の比較では単一化計算が失敗する場合を考慮しておらず、また本アルゴリズムに特有の処理であるクラスタリング及びクラスタリング情報作成の実行コストが含まれていない。そこで、この２つの特有の処理を高速に実行するアルゴリズムを検討した。更に、効率比較実験により、本アルゴリズムは、リテラルペアが複数のクラスタに分割され、単一化の成否を決定するまでに多くの代入が施される場合、特に有効である事が分かった。また、クラスタ数に比例して効率が改善されるため、引数の数が多く、引数間の独立性が高い定理証明プログラムに本アルゴリズムは有効であると考えられる。

今後の課題としては、本アルゴリズムをLogicマシンの構成法に具体的に応用していく事が残されている。

第６章　結　　論

本論文は、述語論理型処理システムの高速化ににについて、以下の３項目を検討した。

（１）　制限付導出法及び証明戦略を実働化する定理証明システムの構成法

（２）　定理証明プログラムの内部構造の実現方式

（３）　導出操作の並列処理方式

以下、各章において得られた成果について述べる。

第２章においては、定理証明プログラムとその実行理論である導出原理及び今までに提案されている効率改善策について言及した。効率改善策の中で、制限付導出法については、Horn節集合に対して完全性を保持するＳＮＬ導出法及びＳＰＵ導出法等に興味が持たれ、定理証明プログラムの内部構造の実現方式については、導出の実行時に束縛だけを蓄える環境評価型が、効率の良い内部構造の実現方式として現在多く採用されている事を述べた。また、単一化操作については、A. Martelli　及び M. S. Paterson によって提案された逐次処理における改良単一化アルゴリズムについて言及した。

第３章においては、定理証明システムＳＥＮＲＩを構成した。ＳＥＮＲＩの特色としては、ユーザインターフェース的には、従来のシステムが、ひとつの定理証明法を簡潔に構成する事に重点が置かれていたのに対して、オプションの指定を単に変更するだけで、様々な定理証明法を構成できると共に、設定された戦略の不適切さの為反駁が得られない場合、ユーザに助言を与える事が挙げられる。前者の特色からは、定理証明法の変更が容易となり、証明問題に対して実行効率の良い定理証明法を短時間で設定できると共に、異なった定理証明法の比較が容易になる。また、後者の特色からは、適切な戦略設定が容易になる。

次に、システム内部的には、データ構造及びＬＡＶＳの再構成に特色を持たせた。ＳＥＮＲＩのデータ構造は、リスト構造を基礎にして、述語論理式(節)の表現に適する様にデータ圧縮したものを採用したが、リスト構造を基礎にし

たのは、他のデータ構造と比べて、変更・修正等の操作が柔軟にでき、支援システム等の作成が容易になるからである。また、ＬＡＶＳの再構成法は、最初ＧＣを使用していたが、定理証明においては、将来必要となるセルは節集合の表現に関連したものだけである事に着目し、ＬＡＶＳを双方向から利用して、将来必要とならない記憶領域をポインタの再設定だけで再利用するというＬＡＶＳの管理法を新しく提案し、ＧＣと比較してその有効性を確認した。

以上の事から、定理証明システムＳＥＮＲＩは、従来のシステムと比べて、高速で使い易いシステムになったと考えられる。

第４章においては、定理証明プログラムの内部構造の新しい実現方式ＲＳＧ方式を提案した。ＲＳＧ方式は、他の方式と比較して、以下の特徴を持っている。

（１）導出の実行時に、構造をたどる事なく、また束縛環境を参照する事なく、リテラルリスト構造値を利用して、単一化計算を行なうペアを高速に抽出しながら、単一化操作を実行し、一時的な束縛を作成する事により、置換操作を一度だけで済ませているため、実行効率は向上する。

（２）節記録は、束縛環境を蓄える必要がなく、また同型のリテラルリスト構造を蓄える必要がないため、記憶効率は改善される。

（３）節記録は、それだけで節の完全な情報となる直接的な表現であり、導出形の内容を変更する事が容易に実行できる。また、２つの節の同等性が高速に処理できる。

以上の事から、ＲＳＧ方式は、定理証明プログラムにおける新しい置換型の内部構造として位置付けられると考えられる。

第５章においては、Parallel Resolution Algorithmを提案した。本アルゴリズムは、単一化計算を並列処理する事に特徴がある。この単一化アルゴリズムの時間計算量を導出すると、単一化計算が成功する場合、従来の単一化アルゴリズムが、入力系列の要素数nに対して$O(n^2)$の時間計算量になるのに対して、本アルゴリズムの単一化計算の並列処理では、$O(n)$の時間計算量になった。しかしながら、この理論値の比較では、単一化計算が失敗する場合を

考慮しておらず、また本アルゴリズムに特有の処理であるクラスタリング及びクラスタリング情報作成の実行コストが含まれていない。そこで、この2つの特有の処理を高速に実行するアルゴリズムを検討し、効率比較実験により、本アルゴリズムは、リテラルペアが複数のクラスタに分割され、単一化の成否を決定するまでに多くの代入が施される場合、特に有効である事が分かった。また、クラスタ数に比例して効率が改善されるため、引数の数が多く、引数間の独立性が高い定理証明プログラムに本アルゴリズムは有効であると考えられる。

以上、本論文では、述語論理型処理システムの高速化について、定理証明システムの構成法と定理証明プログラムの内部構造及び導出操作の並列処理という3つの事項を検討した。今後の課題としては、本論文の成果を基礎にして、Prologマシンを代表とする知識処理プロセッサの構成法に関与していく事が残されている。

謝　　辞

本研究の全過程を通じ、直接懇切なる御指導、御鞭達を賜わった大阪大学工学部通信工学教室手塚慶一教授に心から御礼申し上げる。

学部および大学院にて御指導、御教示賜わった同通信工学教室熊谷信昭教授、中西義郎教授、滑川敏彦教授、および大阪大学産業科学研究所の角所収教授に対し厚く御礼申し上げる。

筆者の属する手塚研究室の真田英彦助教授、中西暉講師、打浪清一助手、後藤嘉代子技官をはじめ、研究室の諸氏には種々の面で御世話になった。また、山口大学の西岡弘明助手、本学卒業生の手塚正義氏、脇一善氏、石川淳士氏、本学大学院の淡誠一郎氏、学部学生の炭田昌人氏ほかオートマトングループの諸氏には様々な御協力を頂いた。

ここに記して、以上の方々に深く感謝の意を表する。

文　　献

(1) G.Battani, H.Meloni : " Interpreteur du langage de programmation PROLOG ", R,Groupe d'intelligence artificielle, Univ. d'Aix−Marseille (1973).

(2) K.A.Bowen : " PROLOG ", ACM 0−89791−008−7/79/1000/0014 (1979).

(3) R.S.Boyer and J.S.Moore : " The Sharing of Structure in Theorem−Proving Programs ", Machine Intelligence 7, pp. 101−116 (1972).

(4) M.Bruynooghe : "Analysis of Dependencies to Improve the Behaviour of Logic Programs ", Lecture Notes in Computer Science 87, pp.293−305 (1980).

(5) A.Bundy, L.Byrd, G.Luger, C.Mellish and M.Palmer: "Solving Mechanisms Problems Using Meta−Level Inference" Expert System in the Micro−Electronic Age, pp.50−64 (1979).

(6) C.L.Chang : " The Unit Proof and the Input Proof in Theorem Proving ", J.ACM, 17, pp.698−708 (1970).

(7) C.L.Chang : "Resolution Plans in Theorem Proving ", Proc, 6th, IJCAI, pp.143−148 (1979).

(8) C.L.Chang and R.C.T.Lee : " Symbolic Logic and Mechanical Theorem Proving ", Academic Press (1973).

(9) K.L.Clark and F.G.McCabe : " The Control of Facilities IC−PROLOG ", Expert System in the Micro−Electronic Age, pp.122−149 (1979).

(10) J. Doyle : " A Truth Maintenance System ", Artificial

Intelligence 12, pp. 231−272 (1979).

(11) J.Doyle and D.McDermott : " Non−Monotonic Logic I " Artificial Intelligence 13, pp. 41−72 (1980).

(12) M.H.van Emden : " Programming with Resolution Logic ", Machine Intelligence 8, pp.266−299, Ellis Horwood (1977)

(13) H.Gallaire , et al. : " An Overview and Introduction to Logic and Data Bases ", in Logic and Data Bases (1978)

(14) K.T.Harry , Wong , et al. : " Two View of Data Semantics : a Survey of Data Models in Artificial Intelligence and Data Bases Management ", INFOR , 15 , 3 (1977)

(15) L.J.Henschen : " Semantic Resolution for Horn Sets ", Proc, 4th IJCAI, pp. 46−52 (1975).

(16) L.Henschen and L.Wos : "Unit Refutations and Horn Sets ", J. ACM, 21, 4, pp. 590−605 (1974).

(17) L.Henschen, R.Overbeek and L.Wos : " A Theorem−Proving Language for Experimentation ", Commun. ACM, 17, 6, pp. 308−314 (1974).

(18) C.Kellogg, et al. : " Deductive Capability for Data Management ", in Systems for Large Data Bases (1976).

(19) R.Kowalski : " Predicate Logic as Programming Language ", Proc. IFIP 74, pp. 569−574 (1974)

(20) R.Kowalski : " Algorithm = Logic + Control ", Commun. ACM , 22, 7, pp. 424−436 (1972).

(21) R.Kowalski : " Logic for Problem Solving ", NORTH−HOLLAND (1979).

(22) D.Kuehner : " Some Special Purpose Resolution Systems ", Machine Intelligence 7 , pp. 117−128 (1972).

(23) A.Martelli : " Theorem Proving with Structure Shearing and Efficient Unification ", Proc. 5th IJCAI, pp. 543

(1977).

(24) A.Martelli and U.Montanari : " An Efficient Uinfication Algorithm ", ACM Trans, on Programming Lang. and Syst., Vol.4, No.2, pp. 258-282 (1982).

(25) D.McDermott : " The PROLOG Phenomenon ", ACM SIGART, No.72, pp. 16-20 (1980).

(26) J. Minker : " An Exeperimental Relational Data Base System Based on Logic ", in Logic and Data Bases(1978).

(27) N.J.Nilsson : " Problem-Solving Method in Artificial Intelligence ", McGraw-Hill (1971).

(28) M.S.Paterson and M.N.Wegan : " Linear Unification ", Proc. 8th Annual ACM Symp. on Theory of Computing. pp. 181-186 (1976).

(29) F.Pereira and D.Warren : " Definite Clause Grammers for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks ", Artificial Intelligence 13, pp. 231-278 (1980).

(30) J.A.Robinson : " A Machine Oriented Logic Based on the Resolution Principle ", J.ACM 12, pp. 23-41 (1965)

(31) J.A.Robinson : " Computational Logic : The Unification Computation ", Machine Intelligence 6, pp. 63-72 Edinburgh Univ. Press (1971).

(32) S.Sickel : " Variable Range Restrictions in Resolution ", Machine Intelligence 8, pp. 73-85 (1977).

(33) D.Warren : " PROLOG-The Language and its Implemen-tation Compared with LISP ", Proc. of the Symposium on Artificial Intelligence and Programming Language , SIGPLAN/SIGART (1977).

(34) D.Warren : " Implenenting Prolog ", D.A.I. Research

Report No.３９, Univ. of Edinburgh (１９７７).

(35) 浅井 清：" 記号処理言語 ", 総合図書 (１９７１).

(36) 安西 祐一郎, 佐伯 胖, 難波 和明：" ＬＩＳＰで学ぶ認知心理学 ２ 問題解決 ", 東京大学出版会 (１９８２).

(37) 奥乃 博：" 述語型言語の支援ユーティリティの検討 ", 情報処理学会人工知能と対話技法研究会資料, ２６－３ (１９８２).

(38) カルジェニン著, 日野 寛三 訳：" 記号論理学入門 ", 東京図書, 数学新書５１ (１９６６).

(39) クヌース著, 米田 信夫・筧 捷彦 訳：" 基本算法/情報構造 ", サイエンス社 (１９７８).

(40) 後藤 滋樹：" プログラム シンセシス ", 情報処理, Vol.22, No.３ (１９８１).

(41) 中島 秀之：" ＰＲＯＬＯＧという名のプログラム言語 ", bit, Vol.１０, No.１２ (１９７８).

(42) 中島 秀之：" ＰＡＲＡＬＬＥＬ ＰＲＯＬＯＧ ", 東京大学 修士論文 (１９７９).

(43) 長尾 真, 淵 一博：" 論理と意味 ", 岩波講座情報科学－７, 岩波書店 (１９８２).

(44) 西田 富士夫：" 言語情報処理 ", コロナ社 (１９８１).

(45) 淵 一博："述語論理的プログラミング－ＥＰＩＬＯＧの提案 ", 情報処理学会記号処理研究会資料 ,１－２ (１９７７).

(46) 淵 一博：" 問題解決と推論機構 ", 情報処理, Vol.１９,No.１０ (１９７８).

(47) 淵 一博：" 述語論理型言語 ", 情報処理, Vol.２２, No.６ (１９８１).

(48) Ｊ.Ｌ.ファルツ著, 間野 浩太郎 監訳：" 電子計算機データ構造論 ", マグロウヒル好学社 (１９８０).

(49) 古川 康一：" 人工知能とデータベース ", 情報処理, Vol.17,No.10 (１９７６).

(50) 古川 康一：" 評価法に基づく演繹データベースの実現 ",昭５５信学総全大 Ｓ９－８（１９８０).

(51) 山崎 正人，山本 明：" 定理の自動証明のためのプログラミングシステムの構成 "，情報処理，Vol.17，No.5 （１９７６).

(52) 横井 俊夫 他：" ロジック・プログラミングと高機能パーソナル・コンピュータ ",情報処理学会記号処理研究会資料,18－７（１９８２).

(53) 竹内 郁雄：" 第２回ＬＩＳＰコンテスト "，情報処理，Vol.20，No.３（１９７９).

(54) W.Kim 著,植村 俊亮 訳：" 関係モデルデータベースシステム "，inコンピュータ・サイエンス acm computing surveys '７９，bit編集部 （１９８０).

(55) W.D.モウラー 著，佐藤 浩史 訳：" プログラマのためのＬＩＳＰ入門 "，サイエンス社（１９７４).

(56) Ｊ.Ｒ.スレイグル 著，南雲 仁一,野崎 昭弘 共訳：" 人工知能－発見的プログラミング－ "，産業図書（１９７２).

(57) 数理科学「 特集 人工知能 」サイエンス社（１９７６).

(58) 数理科学「 特集 第５世代計算機(知識情報処理システム) 」サイエンス社（１９８２).

(59) 数理科学「 特集 知識構造 」サイエンス社（１９８３).

(60) 情報処理「 特集 人工知能とソフトウェア技術 」 Vol.19， No.10（１９７８).

(61) 情報処理「 特集 非手続き型プログラミングのための計算モデル 」Vol.24，No.２（１９８３).

付録　：　第５章の効率比較実験で使用した実験問題

1. MEMBER

```
1 -MEMBER($X,[A.B.NIL])/-ANS($X)
2 MEMBER($X,[$Y.$Z])/-MEMBER($X,$Z)
3 MEMBER($X,[$X.$Y])
```

2. APPEND

```
1 -APPEND([A.B.NIL],[C.D.NIL],$X)/-ANS($X)
2 APPEND(NIL.$X.$X)
3 APPEND([$X.$Y],$Z,[$X.$U])/-APPEND($Y.$Z.$U)
```

3. LOOP

```
1 MINPATH($X,$Y,$W)/
  -CRSIFT($X,$N,$Y)/-CLSIFT($X,$K,$Y)/-R_OR_L($N,$K,$W);
2 CRSIFT(LOOP($A,$B,$C),0,LOOP($A,$B,$C));
3 CRSIFT(LOOP($A,$B,$C),S($N),LOOP($Z,$X,$Y))/
  -CRSIFT(LOOP($A,$B,$C),$N,LOOP($X,$Y,$Z));
4 CLSIFT(LOOP($A,$B,$C),0,LOOP($A,$B,$B));
5 CLSIFT(LOOP($A,$B,$C),S($N),LOOP($Y,$Z,$X))/
  -CLSIFT(LOOP($A,$B,$C),$N,LOOP($X,$Y,$Z));
6 R_OR_L(S($X),S($Y),$Z)/-R_OR_L($X,$Y,$Z);
7 R_OR_L(0,0,EVEN);
8 R_OR_L(0,S($X),RIGHT);
9 R_OR_L(0,S($X),0,LEFT);
10 -MINPATH(LOOP(OSAKA,NARA,KOBE),
    LOOP(KOBE,OSAKA,NARA),$G)/-ANS($G).
```

4. Data Base 検索

```
1.   -MICOM($X,$Y,$Z,218000)/-MAKER($Y,$W,$U)/-ANS($W)

2    MICOM(APPLE2,APPLE,6502,358000);
3    MICOM(FM8,FUJITSU,6809,218000);
4    MICOM(FP1100,CASIO,Z80A,128000);
5    MICOM(MZ80B,SHARP,Z80A,278000);
6    MICOM(MZ2000,SHARP,Z80A,218000);
7    MICOM(PC6001,NEC,Z80A, 898000);
8    MICOM(PC8001,NEC,Z80A,168000);
9    MICOM(PC8801,NEC,Z80A,228000);
10   MICOM(PASOPIA,TOSHIBA,Z80A,163000);
11   MICOM(IF800M30,OKI,Z80B ,1498000);

12   MAKER(NISSAN,TOKYO,A);
13   MAKER(HONDA,TOKYO,A);
14   MAKER(NEC,TOKYO,E);
15   MAKER(TOSHIBA,TOKYO,E);
16   MAKER(FUJITSU,TOKYO,E);
17   MAKER(OKI,TOKYO,E);
18   MAKER(CASIO,TOKYO,E);
19   MAKER(SHARP,OSAKA,E);
20   MAKER(TOYOTA,AICHI,A).
```