データベース・システム 65－7
（１９８８．５．１９）

関係演算の効率的プログラムへの変換技法
― 証明図を用いる変換技法

中山秀明
(株) リコー　ソフトウェア研究所

この論文では、集合演算によって表現されたデータベースの問い合わせを、レコードごとの操作を行なうループプログラムに変換する手法を提案する。

この手法において、変換操作は次のように進む:

1. データベースに対する問い合わせを、ある構成的論理体系における証明図に変換する。
2. 証明図を別の証明図に変換する。
3. 変換操作後の証明図からプログラムを抽出する。

我々の手法では、証明図の変換操作をどのように進めるべきかを容易に決定でき、unfold/fold 変換、項書き換え規則などのプログラム変換技法を用いる従来からの手法よりも、効率よく変換操作が行なえる。

Transformation of relational operations into efficient programs
— an application of proof-transformation

Hideaki Nakayama

Software Research Center
Ricoh Co., Ltd.

1-1-17 Koishikawa, Bunkyo-ku, Tokyo 112, Japan

This paper proposes a method by which a database query expressed by set-oriented operations is transformed into an iterative program that accesses databases in record-wise manner. In our method, the transformation process proceeds as follows:

1. **A query is transformed into a proof in a constructive logic system.**
2. **We transform the proof into another proof.**
3. **An iterative program is extracted from the proof obtained by step 2.**

Since we can deterministically transform proofs, the method described here transforms a query into a program more efficiently than the existing method utilizing the program transformation techniques such as unfold/fold rules and term rewriting rules.

1 はじめに

関係代数によって表現された問い合わせを効率よく処理するために、集合演算式に対する代数的な操作規則の研究が行なわれてきた[9]。しかし、現実の計算機において、集合演算を直接計算することができないので、集合演算をデータベースのレコードごとの演算に還元する必要がある。代数的操作は、集合演算式を集合演算式に置き換えるだけであるので、集合演算とレコードごとの演算とのギャップをどうするかについては答えてはくれない。

この集合演算とレコードごとの演算とのギャップを埋める技法として、本論文では証明図の変換に基づく技法を提案する。提案する技法の概略を説明すると次のようになる。ある構成的論理体系において、個々の集合演算子を表現する証明を用意しておき、集合演算式が与えられた時、式の構成に従って証明を組み合わせて、集合演算式を表現する証明をつくる。こうして得られた証明に変換操作を行ない、変換後の証明から、目的とするプログラムを抽出することを行なう。この技法の利点は、組み合わせて得られる証明図の変換をどう進めるかを、個々の演算子の証明を調べることにより、あらかじめ知ることができるので、効率良く変換を進めることができる点にある。

このギャップを埋める他の技法としては、**Freytag** [2] の成果を挙げることができる。**Freytag** は、**unfold/fold** 変換 [1]、項書き換え規則 [6] を用いることにより、和、積、選択、射影からなる集合演算式を、レコードごとの演算を行なう効率のよいプログラム（ループプログラム）に変換できることを示した。しかし、[2] のアプローチでは、項書き換え規則が適用できなくなるまで適用する操作を行なう必要があり、効率のよい変換操作であるとは言えない。また、変換操作の正しさを示すために、項書き換え規則の停止性と合流性 [6] の証明にかなりのページ数を費やす必要がある。一方、我々が提案する技法では、証明図の **path** を調べることにより、どのように変換操作を進めるべきかを決定できるので効率良く変換を行なえる。また、変換操作の正しさも証明図の大きさの帰納法で簡単に示すことができ、この２点において、我々の技法の方が **Freytag** の技法より優れている。

この論文では、今述べた変換操作のうち集合演算式に対する証明の構成と証明の変換操作に焦点を当てることにする。証明からプログラムを抽出する技法については、例えば、[5] を参照されたい。次のセクション２と３において論理体系について触れ、その論理体系の性質、特に、証明と計算の関係について簡単に述べる。セクション４で、集合演算に対する証明について説明する。セクション５と６において、証明の変換操作について説明する。セクション７では、セクション５で説明した変換操作を集合演算に対する証明に適用した場合について述べる。

2 論理体系

集合演算を表現する証明を作る論理体系として、[4] における **ID** のサブセットを用いる。**ID** は直観主義論理体系 **NJ** [8] に基づく体系である。

論理記号としては、$\wedge, \vee, \supset, \forall, \exists$ を用いる。論理記号の結合の強さは、ここに示した順に減少するものとし、結合の順序を変えるために '(' と ')' を使う。A を式として、x を変数、t を項とするとき、$A_x(t)$ により、A の中の自由変数 x を t に置き換えた結果を表わすことにする。ただし、t の中の自由変数が、束縛されないようにするため、A の中の束縛変数の名前の付け換えがなされてから、置き換えがなされるとする。

推論規則は、$\dfrac{F_1\ F_2\ \cdots\ F_n}{F}$ のような形をしたものである。ただし、$F_1, F_2, \ldots, F_n, F$ は式である。この推論規則において、$F_1, F_2, \ldots, F_n$ を前提といい、F を結論と呼ぶ。

演繹とは、推論規則を１回以上繰り返して適用したものであり、木構造をとる。演繹の根に位置する式を、**end formula**、葉に位置する式を仮定と呼ぶ。以下の推論規則の説明において、'[' と ']' で囲まれた式を、推論規則により消去される仮定という。演繹の仮定のうち、消去されていないものを **open assumption** といい、**open assumption** のない演繹を、証明と呼ぶ。演繹、証明のことを証明図ともいう。F が、ある証明図の **end formula** であるとき、$\vdash F$ と記す。また、$\begin{array}{c}A\\ \downarrow\\ B\end{array}$ により、B を **end formula** とし、A を仮定の１つとする演繹を表わすことにする。

推論規則は大きく分けて、論理記号に関するものと、プロダクションというものに関する規則とから成る。論理記号に関する推論規則は各論理記号の導入と除去の規則からなり、次の１２個のものがある:

$$(\wedge\text{-}I)\ \frac{A\quad B}{A\wedge B} \qquad (\wedge\text{-}E)\ \frac{A\wedge B}{A}\quad\frac{A\wedge B}{B}$$

$$(\vee\text{-}I)\ \frac{A}{A\vee B}\quad\frac{B}{A\vee B} \qquad (\vee\text{-}E)\ \frac{A\vee B\quad \begin{array}{c}[A]\\ \downarrow\\ C\end{array}\quad \begin{array}{c}[B]\\ \downarrow\\ C\end{array}}{C}$$

$$(\supset\text{-}I)\ \frac{\begin{array}{c}[A]\\ \downarrow\\ B\end{array}}{A \supset B} \qquad (\supset\text{-}E)\ \frac{A \supset B \quad A}{B}$$

$$(\forall\text{-}I)\ \frac{F_x(a)}{\forall x F} \qquad (\forall\text{-}E)\ \frac{\forall x F}{F_x(t)}$$

$$(\exists\text{-}I)\ \frac{F_x(t)}{\exists x F} \qquad (\exists\text{-}E)\ \frac{\exists x F \qquad \begin{array}{c}[F_x(a)]\\ \downarrow\\ H\end{array}}{H}$$

ただし、($\forall$-I),($\exists$-E) においては、固有変数条件が満たされていなければならない。すなわち、($\forall$-I) において、変数 a は F が依存している仮定の中の自由変数ではなく、また、($\exists$-E) において、a は H が依存している $F_x(a)$ 以外の仮定、および、$\exists x F$ の中に自由変数として現れてはならない。

プロダクションとは、次の形をしたものである。ただし、$P_1, P_2, \ldots, P_n, P$ は述語記号である:

$$\frac{P_1(p_1)\ P_2(p_2)\ \cdots\ P_n(p_n)}{P(p)}$$

プロダクションに関する推論規則には、プロダクション導入とプロダクション除去とがある。プロダクション導入とは、プロダクションの中の変数を適当な項によって置き換えたものである。

プロダクション除去を適用するためには、述語に対して 式と変数の列を対応づける 関数を決めておく必要がある。ただし、述語記号に対応づけられる変数の列は、相異なる変数からなり、長さは述語記号のとる引き数の数とする。今、そのような関数を Υ とし、述語記号 P に対し、式 F と、変数の列 $\boldsymbol{x}$ が対応づけられているとき、$\Upsilon_P(t)$ によって $F_{\boldsymbol{x}}(t)$ を表わすとする。プロダクション除去とは、R を述語記号とするとき、次のような推論規則である:

$$\frac{R(s) \quad \textit{minor deductions}}{\Upsilon_R(s)}$$

ただし、*minor deductions* は、各プロダクション $\frac{Q_1(t_1)\cdots Q_m(t_m)}{Q(t)}$ に対する次のような演繹からなるとする:

$$\begin{array}{c}[\Upsilon_{Q_1}(t_1')]\cdots[\Upsilon_{Q_m}(t_m')]\\ \downarrow\\ \Upsilon_Q(t')\end{array}$$

ここに、$'$ は、$t_1, \ldots, t_m, t$ の中の変数を、固有変数条件が成り立つように付け換えることを表わす。

3 論理体系の性質

前のセクションで説明した論理体系の性質として、次のことが知られている[7]。

- 素論理式 $P(t)$ が証明されたとすると、その証明図を正規化することにより、プロダクション導入だけからなる $P(t)$ の証明図を得ることができる。
- $F \vee G$ が証明されたとすると、正規化により、F か G の証明図を得ることができる。
- $\exists x F(x)$ が証明されたとすると、その証明図を正規化することにより、$F(s)$ が証明できるような項 s を得ることができる。

証明図の正規化とは、ある論理記号の導入規則の次に同じ論理記号の除去規則が並ぶ場合に、この2つの推論規則を消し、また、プロダクション導入の次にプロダクション除去が並ぶ場合にはこれらの推論規則をプロダクション導入に対応する **minor deduction** に置き換える、という手続きである。

これらの性質により $\forall x(F(x) \supset \exists y G(x,y))$ という式の証明図を、入力が x で、出力が y であるようなプログラムとして、正規化により解釈実行することができる[3]。また、[5] におけるように、**realizability interpretation** を用いて直接実行可能なプログラムを得ることもできる。したがって、証明図の正規化過程は、計算過程とみなすことができる。

4 集合演算の論理体系内での表現

ここでは、レコードの集合としてのデータベース、集合演算子に対応する述語をプロダクションによって定義し、集合演算に対して、どのような証明図を作るべきかを説明する。

$\mathbf{db}(s)$ により、s がレコードの集合であることを表わす。集合をリストによって表現することにすれば、

$$\frac{}{\mathbf{db}([\,])} \qquad \frac{\mathbf{db}([s]) \quad \mathbf{record}(a)}{\mathbf{db}([a.s])}$$

となる。厳密には、db, record という述語記号は、データベースごとに用意しなければならないが、どのデータベースに対しても同じ述語記号を使うことにする（以下に示す述語についても同様である。）。

$\mathbf{union}(s,t,u)$ により、s と t の和が u であることを表す:

$$\frac{}{\mathbf{union}([\,],[\,],[\,])} \qquad \frac{\mathbf{union}([\,],t,u)}{\mathbf{union}([\,],[b.t],[b.u])} \qquad \frac{\mathbf{union}(s,t,u)}{\mathbf{union}([a.s],t,[a.u])}$$

$\mathbf{join}(s,t,u)$ により、s と t の積が u であることを示す:

$$\frac{}{\mathbf{join}([\,],t,[\,])} \qquad \frac{\mathbf{join}(s,t,v) \quad \mathbf{join2}(a,t,v,u)}{\mathbf{join}([a.s],t,u)}$$

$$\frac{}{\mathbf{join2}(a,[\,],v,v)} \qquad \frac{\mathbf{join2}(a,t,v,u) \quad \mathbf{app}(a,b,c)}{\mathbf{join2}(a,[b.t],v,[c.u])}$$

ただし、$\mathbf{app}(a,b,c)$ は、レコード a と b の連接が c になることを表わすものとし、

$$\vdash \forall a \forall b(\,\mathbf{record}(a) \wedge \mathbf{record}(b) \supset \exists c(\,\mathbf{app}(a,b,c) \wedge \mathbf{record}(c)\,)\,)$$

が成り立つものとする。

$\mathbf{select}(s,u)$ により、s のレコードのうち、ある条件を満たすものを集めたものが u であることを表す:

$$\frac{}{\mathbf{select}([\,],[\,])} \qquad \frac{\mathrm{p}(a) \quad \mathbf{select}(s,u)}{\mathbf{select}([a.s],[a.u])} \qquad \frac{\mathrm{q}(a) \quad \mathbf{select}(s,u)}{\mathbf{select}([a.s],u)}$$

ただし、次のことが成り立つものとし、

$$\vdash \forall a(\,\mathbf{record}(a) \supset \mathrm{p}(a) \vee \mathrm{q}(a)\,)$$

さらに、$\mathbf{record}(a)$ を満たす任意の a に対し、$\vdash \mathrm{p}(a)$ と $\vdash \mathrm{q}(a)$ との両方が成り立つことはないとする。

$\mathbf{project}(s,u)$ により、s の各レコードの射影を取ってできる集合が u であることを示す:

$$\frac{}{\mathbf{project}([\,],[\,])} \qquad \frac{\mathbf{project}(s,u) \quad \mathbf{prj}(a,b)}{\mathbf{project}([a.s],[b.u])}$$

ただし、$\mathbf{prj}(a,b)$ は、レコード a の射影が b であることを表わし、

$$\vdash \forall a(\,\mathbf{record}(a) \supset \exists b(\,\mathbf{prj}(a,b) \wedge \mathbf{record}(b)\,)\,)$$

が成り立つものとする。

以上の様に定義されたプロダクションを持つ論理体系において、述語記号db にそれぞれ、

$$\begin{aligned}
\Phi^1_{\mathbf{db}}(x) &\equiv \exists u(\,\mathbf{union}(x,t,u) \wedge \mathbf{db}(u)\,)\\
\Phi^2_{\mathbf{db}}(x) &\equiv \exists u(\,\mathbf{join}(x,t,u) \wedge \mathbf{db}(u)\,)\\
\Phi^3_{\mathbf{db}}(x) &\equiv \exists u(\,\mathbf{select}(x,u) \wedge \mathbf{db}(u)\,)\\
\Phi^4_{\mathbf{db}}(x) &\equiv \exists u(\,\mathbf{project}(x,u) \wedge \mathbf{db}(u)\,)
\end{aligned}$$

を対応付けてプロダクション除去を適用して証明を行うと、それぞれ、

$$\left.\begin{array}{l}
\forall s \forall t(\,\mathbf{db}(s) \supset \mathbf{db}(t) \supset \exists u(\,\mathbf{union}(s,t,u) \wedge \mathbf{db}(u)\,)\,)\\
\forall s \forall t(\,\mathbf{db}(s) \supset \mathbf{db}(t) \supset \exists u(\,\mathbf{join}(s,t,u) \wedge \mathbf{db}(u)\,)\,)\\
\forall s(\,\mathbf{db}(s) \supset \exists u(\,\mathbf{select}(s,u) \wedge \mathbf{db}(u)\,)\,)\\
\forall s(\,\mathbf{db}(s) \supset \exists u(\,\mathbf{project}(s,u) \wedge \mathbf{db}(u)\,)\,)
\end{array}\right\} (\#)$$

を証明することができる。なお、union に関する式を証明する場合には、プロダクション除去のminor deduction の中に、

$$\Phi^5_{\mathrm{db}}(x) \equiv \exists u(\mathrm{union}([\,], x, u) \wedge \mathrm{db}(u))$$

という対応づけを用いたプロダクション除去が現れる。また、join に関する式の証明にも、プロダクション除去の中に、

$$\Phi^6_{\mathrm{db}}(x) \equiv \exists v(\mathrm{join2}(a, x, u, v) \wedge \mathrm{db}(v))$$

という対応づけを用いたプロダクション除去が現れる。このようなプロダクション除去のネストにより、証明図からプログラムを構成すると、ネストしたループプログラムができることになる。

(#) で印を付けた式の証明図を組み合わせることによって、集合演算子を組み合わせて得られる集合演算式を表わす証明図が得られる。例えば、積を求めて射影を取るという集合演算式に対しては(#)における join と project に関する証明を組み合わせて、

$$\dfrac{\dfrac{\mathrm{db}(s)\quad \mathrm{db}(t)}{\exists u(\mathrm{join}(s,t,u) \wedge \mathrm{db}(u))}\downarrow_1 \qquad \dfrac{\dfrac{[\mathrm{join}(s,t,a) \wedge \mathrm{db}(a)]}{\mathrm{join}(s,t,a)} \qquad \dfrac{\dfrac{[\mathrm{join}(s,t,a) \wedge \mathrm{db}(a)]}{\mathrm{db}(a)}\downarrow_2}{\exists v(\mathrm{project}(a,v) \wedge \mathrm{db}(v))}}{\mathrm{join}(s,t,a) \wedge \exists v(\mathrm{project}(a,v) \wedge \mathrm{db}(v))}}{\exists u(\mathrm{join}(s,t,u) \wedge \exists v(\mathrm{project}(u,v) \wedge \mathrm{db}(v)))}$$

$$\exists u(\mathrm{join}(s,t,u) \wedge \exists v(\mathrm{project}(u,v) \wedge \mathrm{db}(v)))$$

という演繹が得られる。ただし、$\downarrow_1$, $\downarrow_2$ は、それぞれ、(#)における join と project に関する式の証明に現れる演繹である。上の演繹の open assumptioon である $\mathrm{db}(s)$, $\mathrm{db}(t)$ を end formula とする証明図が得られれば、前のセクションで述べた論理体系の性質により、s と t の積を求め、射影を施して得られるレコードの集合が計算できることになる。

5 証明図の変換

前のセクションにおいて、集合演算式に対してそれを論理体系内で表現する証明図について述べた。集合演算子を２つ組み合わせてできる集合演算式を表現する証明図には次のような部分がある:

$$\dfrac{\dfrac{\overset{\downarrow_1}{P(p)}\quad \textit{minor deductions}_1}{\Phi_P(p)}\,(\dagger)}{\Psi_Q(q)}\,(\ddagger)\qquad \dfrac{\overset{\downarrow_2}{Q(q)}\qquad \textit{minor deductions}_2}{\Psi_Q(q)}\,(\ddagger)$$

ただし、P, Q は述語記号であり、(†) 、(‡) の印が付いている推論規則は、プロダクション除去とし、$\downarrow_2$ は、プロダクションに関する推論規則が用いられていない演繹を表わしているとする。例えば、前セクションの join と project を組み合わせた証明図の例における $\downarrow_1$, $\downarrow_2$ の中のプロダクション除去が上に示す証明図の (†), (‡) に対応しており、(∃-E) (の一部) が上に示す証明図の $\downarrow_2$ に対応している。

$\overset{\downarrow_1}{P(p)}$ で表わされている証明図が正規化されているとすると、上の証明図の正規化は、

(1) $Q(q)$ を end formula とする証明図を正規化すること、

(2) $\Psi_Q(q)$ を end formula とする証明図を正規化すること、

という２つの操作を、この順に行うか、(1)、(2) の中の正規化の手続きを適当に組み合わせて行うことになる。

しかし、(1) の操作により、$Q(q)$ の正規化された証明図を得ても、結局は(2) の操作により別の正規化された証明図に変換されてしまうので、上で述べた正規の過程には無駄がある。この様子を詳しく調べてみると、$Q(q)$ の正規化された証明図の中のプロダクション導入は、*minor deductions*$_1$ の中に現れるプロダクション導入により生じたものであり、そのプロダクション導入は、(‡) のプロダクション除去の minor deduction に置き換えられて $\Psi_Q(q)$ の正規化された証明図に変換されることがわかる。したがって、*minor deductions*$_1$ の中のプロダクション導入を

***minor deductions*$_2$** の中の対応する演繹に置き換えれば、(†) と (‡) のプロダクション除去を１つにまとめることができ、証明図の正規化の効率をあげることができる。以下では、上に示したような２つのプロダクション除去を１つにまとめる手続きについて述べる。

定義 式 F に対し、F^{Ψ} を次のように定義する:

- P が述語記号のとき、$P(p)^{\Psi}$ を $P(p) \wedge \Psi_P(p)$ とする。
- $\diamond$ を、$\wedge, \vee$ とするとき、$(F \diamond G)^{\Psi}$ を $(F^{\Psi}) \diamond (G^{\Psi})$ とする。
- $(F \supset G)^{\Psi}$ を $F \supset (G^{\Psi})$ とする。
- Q を、$\forall, \exists$ とするとき、$(Qx F)^{\Psi}$ を $Qx(F^{\Psi})$ とする。

式の構造の関する帰納法を用いれば、今定義した F^{Ψ} に対し、$\vdash F \supset F^{\Psi}$, $\vdash F^{\Psi} \supset F$ が成り立つことを示すことができる。ここで、$\vdash F^{\Psi} \supset F$ は論理記号に関する推論規則だけを用いて示すことができるが、$\vdash F \supset F^{\Psi}$ はプロダクション除去を用いないと示すことができないことを注意しておく。

定義 式 F, G に対し、$F \rightharpoonup G$ を以下のように定義する:

- 推論規則のうち、$(\wedge\text{-}I)$, $(\wedge\text{-}E)$, $(\vee\text{-}I)$, $(\supset\text{-}I)$, $(\forall\text{-}I)$, $(\forall\text{-}E)$, $(\exists\text{-}I)$, プロダクション導入において、F が推論規則の前提であり、G が結論である時、$G \rightharpoonup F$ とする。
- $(\vee\text{-}E)$ $\dfrac{\begin{array}{ccc} A \vee B & \overset{1}{\overset{\downarrow}{C}} & \overset{2}{\overset{\downarrow}{C}} \end{array}}{\overset{3}{C}}$ において、$\overset{3}{C} \rightharpoonup \overset{1}{C}$, $\overset{3}{C} \rightharpoonup \overset{2}{C}$, $A \rightharpoonup A \vee B$, $B \rightharpoonup A \vee B$ とする。
- $(\supset\text{-}E)$ $\dfrac{A \supset B \quad A}{B}$ において、$B \rightharpoonup A \supset B$ とする。
- $(\exists\text{-}E)$ $\dfrac{\exists x F \quad \overset{1}{\overset{\downarrow}{H}}}{\overset{2}{H}}$ において、$\overset{2}{H} \rightharpoonup \overset{1}{H}$, $F_x(a) \rightharpoonup \exists x F$ とする。
- プロダクション除去 $\dfrac{R(s) \quad \textit{minor deductions}}{F}$ において、G を ***minor deductions*** の中の任意の演繹の **end formula** とする時、$F \rightharpoonup G$ とする。

定義 ある演繹の中の式 F, G に対し、$F \rightharpoonup^{*} G$ が成り立つとき、G は F を出発点とする **path** 上にあるという。ただし、$\rightharpoonup^{*}$ は $\rightharpoonup$ の反射推移閉包を表わす。

このセクションの始めに示した証明図は、以下に述べる操作により、次の証明図に変換することができる:

$$\dfrac{\dfrac{\begin{array}{cc} \overset{1}{\downarrow} & \\ P(p) & \textit{minor deductions}_3 \end{array}}{\Phi_P(p)^{\Psi}}}{\dfrac{\overset{3}{\downarrow}}{\dfrac{Q(q) \wedge \Psi_Q(q)}{\Psi_Q(q)}}}$$

ただし、$\downarrow_3$ は、プロダクションに関する推論規則がない演繹を表わしているとする。このような証明図を作るためには、***minor deductions***$_3$ と $\downarrow_3$ を構成すればよい。

minor deductions$_3$**の構成法** ***minor deductions***$_1$ の中の演繹の１つを

$$\dfrac{[\Phi_{A_1}(a_1)] \cdots [\Phi_{A_n}(a_n)]}{\Phi_A(a)}$$

とするとき、**end formula** $\Phi_A(a)$ を出発点とする **path** 上の全ての式 F を F^{Ψ} によって置き換えることを行なう。すると、次の２つの場合を除き、正しく推論規則が適用された演繹を得る:

1. プロダクション導入$\frac{B_1(b_1)\cdots B_m(b_m)}{B(b)}$ において、結論の $B(b)$ が置き換えられた場合。
2. プロダクション除去により消去される仮定以外の仮定が置き換えられた場合。

1 の場合を解決するためには、

$$\dfrac{\dfrac{B_1(b_1)^{\Psi}}{B_1(b_1)}\cdots\dfrac{B_m(b_m)^{\Psi}}{B_m(b_m)}}{B(b)}\qquad\dfrac{\dfrac{B_1(b_1)^{\Psi}}{\Psi_{B_1}(b_1)}\cdots\dfrac{B_m(b_m)^{\Psi}}{\Psi_{B_m}(b_m)}}{\Psi_B(b)}\ \downarrow_4$$

$$B(b)^{\Psi}$$

とすればよい。ただし、$\downarrow_4$ は、プロダクション導入$\frac{B_1(b_1)\cdots B_m(b_m)}{B(b)}$ に対応する ***minor deductions*$_2$** の中の演繹である。

2 の場合を解決するためには、$\vdash F \supset F^{\Psi}$ を用いればよい。

以上の操作を行なった後で、$\Phi_{A_i}(a_i)$ $(1 \leq i \leq n)$ の中に置き換えられていないものがあれば、$\vdash \Phi_{A_i}(a_i)^{\Psi} \supset \Phi_{A_i}(a_i)$ により、$\Phi_{A_i}(a_i)^{\Psi}$ が演繹の仮定になるようにする。

このような操作を行なうことにより、***minor deductions*$_3$** が構成できる。

$\downarrow_3$の構成法 これは、***minor deductions*$_3$** を構成するときの操作を、$\downarrow_2$ で表される演繹に対して行なえばよい。

6 証明図の変換操作の性質

前のセクションで述べた変換操作の性質としては、次のことが挙げられる:

1. 2つのプロダクション除去が1つにまとめられるように見えても、実は、プロダクション除去が減らない場合がある。このような場合は、$\vdash F \supset F^{\Psi}$ を用いた場合である。しかし、逆のことを言えば、$\vdash F \supset F^{\Psi}$ を用いずに変換ができる場合には、2つのプロダクション除去を1つにすることができる。
2. 変換操作は、演繹の中の **path** 上に式、推論規則を置き換えることにより行なわれるので、証明図が与えられれば、証明図のどこを置き換えればよいかを簡単に知ることができる。

以上のことにより、組み合わせる前の個々の証明図がわかっていれば、前セクションの変換操作により、正規化の過程の効率が上がるかどうかを判定することができ、また、変換操作をどのように進めるべきかも決定できることがわかる。

7 集合演算を表す証明図の変換

セクション4において、和、積、選択、射影に関する論理式 (#) が証明できることを述べた。それらの式の証明図を調べると、プロダクション除去の **minor deduction** の中の演繹においては、**end formula** を出発点とする **path** は、全て、プロダクション除去によって消去される仮定で終わることがわかる。また、それらのプロダクション除去を組み合わせるために必要な推論規則は、(∃-*E*), (∧-*E*) である。したがって、それらの証明図を組み合わせたものに変換操作を行なうと、$\vdash F \supset F^{\Psi}$ を用いる必要がないので、プロダクション除去の数が減ることになり、正規化の効率が上がる。たとえば、セクション 4 の終りに示した積と射影との組み合わせに対応する証明図を変換すると、$\downarrow_2$ の部分のプロダクション除去がなくなり、次のような証明図が得られる:

$$\dfrac{\dfrac{\mathrm{db}(s)\quad \mathrm{db}(t)}{\exists u(\mathrm{join}(s,t,u)\wedge \mathrm{db}(u))^{\Phi^4_{\mathrm{db}}}}\downarrow_{1'}\qquad\dfrac{\dfrac{[(\mathrm{join}(s,t,a)\wedge \mathrm{db}(a))^{\Phi^4_{\mathrm{db}}}]}{\mathrm{join}(s,t,a)}\quad\dfrac{[(\mathrm{join}(s,t,a)\wedge \mathrm{db}(a))^{\Phi^4_{\mathrm{db}}}]}{\exists v(\mathrm{project}(a,v)\wedge \mathrm{db}(v))}}{\mathrm{join}(s,t,a)\wedge\exists v(\mathrm{project}(a,v)\wedge \mathrm{db}(v))}}{\exists u(\mathrm{join}(s,t,u)\wedge\exists v(\mathrm{project}(u,v)\wedge \mathrm{db}(v)))}$$

$$\exists u(\mathrm{join}(s,t,u)\wedge\exists v(\mathrm{project}(u,v)\wedge \mathrm{db}(v)))$$

ただし、

$$(\mathrm{join}(s,t,u)\wedge \mathrm{db}(u))^{\Phi^4_{\mathrm{db}}} \equiv \mathrm{join}(s,t,u)\wedge \mathrm{db}(u)\wedge\exists v(\mathrm{project}(u,v)\wedge \mathrm{db}(v))$$

である。

一般に、２つの集合演算に対する証明図を組み合わせたものの変換後の形は、次の形になる:

$$\dfrac{\dfrac{\mathrm{db}(s) \qquad \textit{minor deductions}}{\exists u(op_1(s,u) \wedge \mathrm{db}(u) \wedge \exists v(op_2(u,v) \wedge \mathrm{db}(v)))}}{\exists u(op_1(s,u) \wedge \exists v(op_2(u,v) \wedge \mathrm{db}(v)))}$$

ただし、$\downarrow$ は、論理記号に関する推論規則のみからなる演繹である。これを見ると、$\downarrow$ の上下の論理式はほとんど同じであり、正規化の過程を考えてみると、$\downarrow$ の部分は計算に寄与しないのでこの部分は不要である。すなわち、変換後の証明図を得るためには、各集合演算の証明図を組み合わせてから変換を行なう必要はなく、集合演算の証明の中のプロダクション除去の minor deduction 中の path に沿って置き換えを行なって証明図を構成すればよいことになる。したがって、各集合演算の証明図中の path をあらかじめ調べておけば、集合演算式から(変換後の)証明図を構成する作業を効率良く進めることができる。

minor deductions の中で証明図の正規化に関係する演繹は、次の通りである:

$$\dfrac{\dfrac{F(t,x) \quad \mathrm{record}(a)}{\exists x F([a.t],x)}}{\exists x F([\,],x)} \qquad \dfrac{\exists x F(t,x) \qquad \exists x F([a.t],x)}{\exists x F([a.t],x)}$$

[5] におけるような方法で証明図からプログラムを構成すると、($\vee$-*E*) は、if 文、($\exists$-*I*) は関数のリターン値の指定、($\exists$-*E*) は関数を実行し値を得ること、プロダクション除去は再帰呼び出しをする関数の定義に対応する。したがって、上に示した証明図からプログラムを構成すると、自分自身の再帰呼び出しをする関数が得られる。このような関数は、ループプログラムに変換することができるので、和、積、選択、射影からなる集合演算式は、ループプログラムによって計算することができることがわかる。

8 おわりに

この論文では、個々の集合演算子に対する証明図を組み合わせることにより、集合演算式に対する証明図を構成し、それを変換して得られる証明図から、集合演算式を計算する効率のよいプログラムが得られることを示した。ここに示した変換方法は、path を調べることにより変換操作をどう進めればよいかを前もって知ることができる特長を持っていることも説明した。

9 謝辞

本研究の機会を与えて下さった（株）リコー ソフトウェア研究所の國井秀子所長に感謝致します。

References

[1] Burstall, R. M., and Darlington, J.: A transformation system for developing recursive programs, *J. ACM*, vol. 24, No. 1, 1977, pp. 44-67.

[2] Freytag, J. C.: *Translating Relational Queries into Iterative Programs*, Lecture Notes in Computer Science, vol. 261, Springer-Verlag, 1987.

[3] Hagiya, M: *A Proof Description Language and Its Reduction System*, Technical Report 82-03, Department of Information Science, University of Tokyo, 1982.

[4] Hagiya, M., and Sakurai, T.: Foundation of Logic Programming Based on Inductive Definition, *New Generation Computing*, vol. 2, 1984, pp. 59-77.

[5] Hayashi, S.: Extracting Lisp Programs from Constructive Proofs: A Formal Theory of Constructive Mathematics Based on Lisp, *Publ. RIMS, Kyoto Univ.*, vol. 19, 1983, pp. 169-191.

[6] Huet, G.: Confluent Reductions: Abstract Properties and Applications of Term Rewriting Systems, *J. ACM*, vol. 27, No. 4, 1980, pp. 797-821.

[7] Martin-Löf, P.: Hauptsatz for the intuitionistic theory of iterated inductive definitions, *Proc. 2nd Scandinavian Logic Symposium*, North-Holland, Amsterdam, 1970, pp. 179-216.

[8] Prawitz, D.: *Natural Deduction*, Almqvist and Wiksell, Stockholm, 1965.

[9] Ullman, J. D.: *Principles of Database Systems*, Computer Science Press, San Francisco, 1982.