データベース・システム 57－3
（１９８７．１．１９）

集約関数まで拡張した関係論理／関係代数変換法

中野 良平　　斉藤 和巳
（ＮＴＴ　電気通信研究所）

集約関数(aggregate function)が入って来ると、一般にアルファが閉じなくなるので、関係論理から関係代数への変換は簡単でなくなる。本稿は関係論理中の集約関数を最適な関係代数表現に変換する体系を述べたものである。また、集約関数に関するKlugの補正に効率良く対処するため、新らたな集約演算を考案し導入する。新変換法は、代数表現への変換が容易な標準集約形を中継地点とし、それの生成と解決という２フェーズの変換体系を採る。同変換体系は４つの基本変換則と３種の発見的(heuristic)変換則から構成される。新変換法の目的は、集約関数を含んだ関係論理表現を人間が考え出すような最適な関係代数表現に変換することにある。変換プログラムを作成し、考えられる様々な複雑な検索に適用して、極めて満足すべき結果が得られることを確認した。

Reduction of Aggregate Functions in Relational Calculus to Optimal Algebraic Expressions

Ryohei NAKANO　　Kazumi SAITO

NTT Electrical Communications Laboratories
1-2356, Take, Yokosuka-shi, Kanagawa-ken, 238-03 Japan

The advent of aggregate functions makes alphas of relational calculus open; hence reduction from alphas to relational algebraic expressions becomes difficult. This paper describes reduction of aggregate functions in relational calculus to optimal algebraic expressions. We introduce a new aggregate operation to cope with Klug's correction. The present algorithm runs two-phase processing: creation of standard aggregate form and resolution of it. The reduction consists of 4 basic rules and 3 kinds of heuristic rules. Its goal is to produce such optimal algebraic expressions as a man may think of. Reduction program turns out to produce satisfactory results.

1．はじめに

代表的な関係データベース言語ＳＱＬ[１]、ＱＵＥＬ等は関係論理に基づいているが、関係モデルの性能問題克服の期待を担うデータベースマシンのサポート言語は多くの場合関係代数である[２]。従って、関係論理で表現した検索を、データベースマシンでの実行を想定して、最適な関係代数表現に変換する研究が重要になる。

限定した条件（集約関数等の除外）下での関係論理／関係代数変換法については既に報告した[３]。網羅性確保のための基本変換則に、特定のパターンに反応し特に効率的な変換を行う発見的(heuristic)変換則を加え、関係論理の再帰構造に沿って逐次的に変換を進めることにより、記述力の等値性証明アルゴリズム[４，５]ではとても望めない最適な（人間が記述するような）関係代数表現を生成できる。

集約関数(aggregate function)の導入により、変換は著しく難しくなる。それは、今まで全て閉(closed)アルファのみであったのに対し、開(open)アルファが現れるようになるためである。Klug[６]は、関係論理と関係代数に集約関数を導入し、両者の記述力が同等であることを証明した。彼は同論文の中で、基準となるグループ化(group-by)属性値集合と振分けのキーとなるグループ化属性値集合の違いに着目し、前者が後者に含まれない場合には補正が必要であることを指摘した。これを**Klugの補正**と称す。しかし、彼のアルゴリズムは、“太郎の管理者は誰か”といった極めて簡単な検索に対しても、

```
((emp[#3][ ](emp[#1][#1=#1]{太郎}))[ ]emp) [#1=#6,#2=#4][#1]
```

のように煩雑な関係代数表現を生成するので、実行を想定した場合採用できない。また、Kim[７]はＳＱＬを最適な正準形に変換する研究の中で、集約関数も扱っているが、１つのパターンに対し唯１つの変換則を示しているに過ぎない。

以下に述べる新変換法は、筆者達が既に提案した関係論理／関係代数変換法を集約関数まで拡張したものである。まず、関係論理と関係代数に集約関数を導入する。その際、集約関数に関する関係代数の新演算を提案し、Klugの補正に言及する。そして、新変換法の概要と変換プロセスの全貌を述べ、基本変換則、発見的変換則の詳細を例を混じえて説明する。付録に、新変換法による変換例を示す。

2．準備

以下で使用する例題のスキーマを示す。

```
emp    (name,sal,mgr,dept)
sales  (dept,item,vol)
supply(comp,dept,item,vol)
class  (item,type)
loc    (dept,floor)
```

2．1　関係論理の拡張

関係論理への集約関数の導入は以下とする。即ち、次の記法に従う集約関数を項(term)の１つとして追加し、目標リスト(target list)や条件式(qualifier)中に書けるようにする。

```
<aggregate function>::=<function name1> (<alpha>)
                      | <function name2>[<attribute number>] (<alpha>)
<function name1>::=count
<function name2>::=min | max | sum | avg
```

minやmaxに付す属性番号は集約関数を適用する属性を指す。集約関数では重複(duplicates)の扱いが問題になるが、射影（重複除去）は行わず、適用するアルファをいじらずに関数を適用する点に注意が必要である。

2．2　関係代数の拡張

集約関数の代数系演算として、２種用意する。１つはKlugの提案で、他は筆者達の新提案である。前者の例を次に示す。

《例１》管理者と部下の数を対にして求めよ。

```
emp<#3;count>
```

グループ化属性emp[#3]は、グループ化の“基準”であるとともに、“振分けのキー”でもある。両者が一致している。一方、新提案の演算はKlugの補正に対応したものである。例えば、

《例２》現在供給されている商品について、それを販売している店の数を求めよ。

```
(sales[#2,#1])[#1;count](supply[#3])
```

この例で問題になるのは、関係supplyから得られる商品集合と関係salesから得られる商品集合の包含関係である。前者がグループ化の“基準”となり、後者がグループ化の“振分けキー”になっている。普通に

```
sales[#2,#1]<#1;count>  ……（＊）
```

とすると、供給はされているが販売されてない商品が抜け、販売されているが供給されてないものが含まれてしまう。今、関心があるのは“現在供給されている商品”なのである。そのでこぼこ部分の補正がKlugの補正である。前記の新集約演算[#1;count]はその右側に書いたsupply[#3]を基準とし、左側のsales[#2,#1]の第１属性を振分けキーとして、count演算を行う。その結果、供給はされているが販売されてない商品は、count=0となり組として残り、販売されているが供給されてない商品は対象外となる。それ以外は上記(＊)と同じである。

新旧集約演算の構文を以下に定式化する。

```
<aggregate function>::=<<function list>>
                      | <<group-by attributes>;<function list>>
<function list>::=<function>
                  | <function list>,<function>
<function>::=<function name1>
             | <function name2>[#<attribute number>]
<function name1>::=count
<function name2>::=min | max | sum | avg
```

また、集約演算の意味を以下に定式化する。e_1, e_2を関係代数表現、Aをグループ化属性、Ｉをスキーマの任意のインスタンスとするとき、

e_1<A;func>(Ｉ)={t[A]・f:t∈e_1(Ｉ)∧f=func({s:s∈e_1(Ｉ)∧s[A]=t[A]})}

e_1[A;func]e_2(Ｉ)={t・f:t∈e_2(Ｉ)∧f=func({s:s∈e_1(Ｉ)∧s[A]=t})}

ただし、count(φ)=0、min(φ)=max(φ)=sum(φ)=avg(φ)=null値

３．集約関数の新変換法

３．１　新変換法の概要

まず、用語の説明をする。

（ａ）標準集約形(standard aggregate form)

例えば、次のようなアルファのことである。

```
(u[4],avg[2](v
              :emp(v)
              :v[4]=u[4]
             ))
:emp(u)
:()
```

即ち、集約関数を適用するアルファ内の開属性群（u[4]）が、集約関数とともに外側のアルファの目標リストを構成する。目標リストにはそれ以外の属性は入らない。標準集約形は新変換法の重要な中継地点となる。

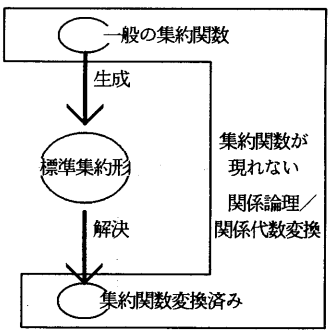

図１　集約関数新変換法の概略

（ｂ）膳の緒(linkage)：新変換法は一般の集約関数から標準集約形を生成するが、新たに生まれた標準集約形と母体のアルファを結ぶものが膳の緒である。膳の緒は具体的には、s[1]=u[4]∧s[2]=u[5]のように、標準集約形の属性と母体の属性の対応を単位式とし、それらを論理積で結合した論理式である。

新変換法は、基本的には２フェーズから構成される（図１）。第１フェーズでは、一般の集約関数に遭遇すると、それをもとに標準集約形と膳の緒を生成して、元の集約関数を解消する。第２フェーズでは、生成した標準集約形を関係代数表現に変換（解決）する。

具体例で説明する。

《例３》店員の氏名、給与、及び所属店の平均給与を求めよ。

```
(u[1],u[2],avg[2](v
                   :emp(v)
                   :v[4]=u[4]
                  ))
:emp(u)
:()
```

⟹

```
(u[1],u[2],s[2])
:emp(u),( (w[4],avg[2](v
                     :emp(v)
                     :v[4]=w[4]
                    ))
          :emp(w)
          :()
        )(s)
: s[1]=u[4]
```

←標準集約形（網掛け部分）

←臍の緒（s[1]=u[4]）

第１フェーズを実行すると、右上となり、

第２フェーズを実行すると、以下となり、集約関数の代数表現が得られる。

```
(u[1],u[2],s[2])
:emp(u),emp<#4;avg[2]>(s)
: s[1]=u[4]
```

新変換法の変換プロセスの全貌を図２に示す。関係論理表現をその再帰構造に沿って変換を進めて行く中で、集約関数に出会ったとする。その場所は、現在変換対象であるアルファの目標リスト内か条件式中である。どちらの場合にも基本的には同じような標準集約形と臍の緒を生成するが、目標リストのときは範囲として追加するのに対し、条件式のときは準結合の形とするのが異なる。標準集約形は、集約関数を含んだ形が特殊なので、新旧の集約演算を用いて簡単に代数表現に変換できる。なお、条件式中のときは、形から明かにKlugの補正が不要なケースがあるので、一気に標準集約形の生成と解決を図るルートがある。以上が大略の流れであるが、その各々について発見的変換則を用意する。発見的変換則のポイントは３点ある。即ち、臍の緒省略、結合省略、Klugの補正不要である。これらは後述する。

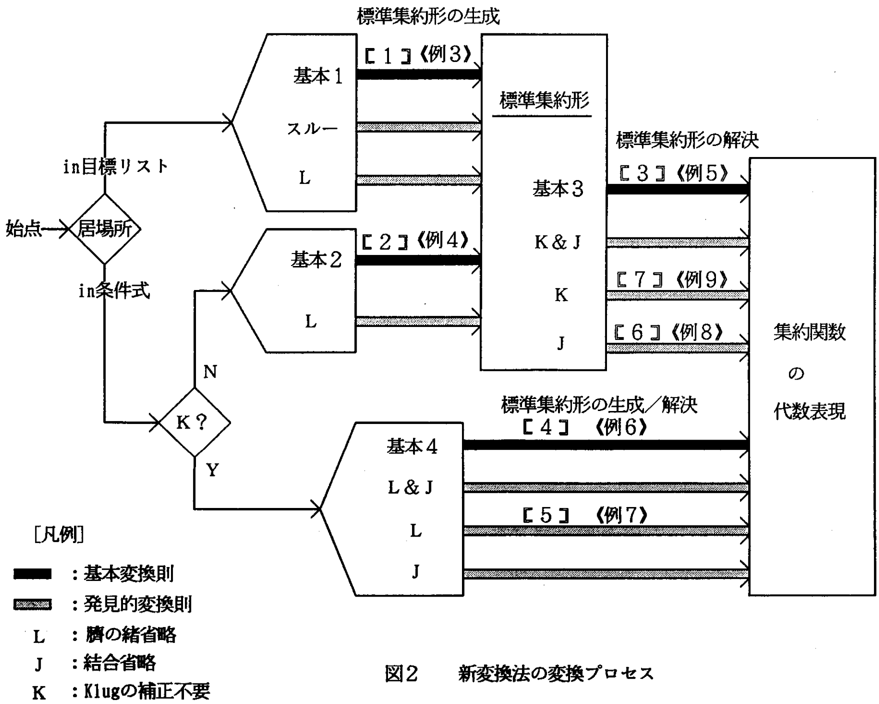

図２　　新変換法の変換プロセス

3．2　集約関数の基本変換

〔１〕目標リスト中の集約関数から標準集約形の生成

前述した例３がこの場合に当たる。対象とするアルファを基に標準集約形と臍の緒を生成し、各々、対象とするアルファの範囲、条件式に追加する。臍の緒は、集約関数中の開属性と、新たに生成した標準集約形の属性とを対応付けるものである。なお、対象とするアルファの条件式は変換によって既に空であるとする。以下に本変換則を定式化する。今後、Ｘ，Ψ等は０個以上の範囲を要素とするベクトル、χ，ϕ等はそれらに対応する組変数ベクトルとする。また、χ_g等は$\alpha(\chi)$中に現れる開属性群を表わす。

［基本変換則１］

$$((t1,func(\alpha(\chi))):X(\chi),\Psi(\phi):())(I)$$
$$=((t1,s[n+1]):X(\chi),\Psi(\phi),\alpha_g(s):f(s,\chi))(I)$$

ただし、・$\alpha_g=(\chi_g,func(\alpha(\chi))):X(\chi):()$

・$f(s,\chi)$はα_gとＸを結ぶχ_gに関する臍の緒

・$n=len(\chi_g)$

〔２〕条件式中の集約関数から標準集約形の生成

《例４》給料が自分が属する店の平均より多い店員を求めよ。

```
(u[1])                                  (u[1])
:emp(u)                                 :emp(u)
:u[2]>avg[2](v                          :∃α_g(s)(s[1]=u[4]∧u[2]>s[2])
              :emp(v)                   ただし、
              :v[4]=u[4]                 α_g=(u[4],avg[2](v
             )                                          :emp(v)
                                                        :v[4]=u[4]
                                                       ))
                                          :emp(u)
                                          :()
```

臍の緒

条件式中にあるときには、上記例のように、標準集約形を準結合の形で組込む。これは、条件式中の集約関数が対象アルファに対し選択的に効く性質を反映したものである。一般に、準結合は結合より負荷が軽いので、処理上有利である。

［基本変換則２］

$$(t1:X(\chi),\Psi(\phi):(func(\alpha(\chi))\theta\ t)\wedge g)(I)$$
$$=(t1:X(\chi),\Psi(\phi):(\exists\alpha_g(s)(f(s,\chi)\wedge s[n+1]\theta\ t))\wedge g)(I)$$

ただし、・$\alpha_g=(\chi_g,func(\alpha(\chi))):X(\chi):()$

・$f(s,\chi)$はα_gとＸを結ぶχ_gに関する臍の緒

・$n=len(\chi_g)$

〔３〕標準集約形の代数表現への変換

《例５》階毎に、その階より上にある店の数を求めよ。

```
(u[2],count(v[1]                        ((u[2],v[1])
            :loc(v)                      :loc(u),loc(v)
            :v[2]>u[2]                   :v[2]>u[2]
           ))                           )[#1;count]((u[2])
:loc(u)                                   新集約演算  :loc(u)
:()                                                   :()
                                                     )
```

ここで対象とするアルファは標準集約形である。集約関数を除いたアルファをグループ化の基準とし、集約関数内のアルファと外のアルファを平坦化して融合したアルファを被グループ化とする新集約演算に変換すればよい。これにより、Klugの補正が対処できるのは前述の通りである。

［基本変換則３］

$((tl_1,func(tl_2:\Omega(\omega):g)):X(\chi):())(I)$

$=(((tl_1,tl_2):X(\chi),\Omega(\omega):g)[\#1,\cdots,\#n;func'](tl_1:X(\chi):()))(I)$

ただし、・$n=len(tl_1)$

・func'はfunc中の属性番号を+nしたもの

・$\chi_g=tl_1$（前提である！）

［４］条件式中でKlugの補正が不要な集約関数の代数表現への変換

《例６》３ケ所以上の階で売られている商品を求めよ。

```
(u[2])                                  (u[2])
:sales(u)                               :sales(u)
:count((v[2])                    ⇒      :∃α_g(s)(s[i]=u[2]∧s[2]>=3)
       :loc(v)                          ただし、        膳の緒
       :∃sales(w)(w[1]=v[1]∧w[2]=u[2])   α_g=((u[2],v[2])
      )>=3                                    :sales(u),loc(v)
                                              :∃sales(w)(w[1]=v[1]∧w[2]=u[2])
                                             )<#1;count>
```

集約関数countが条件式中に現れ、かつ、＞０を満たすとき、Klugの補正は不要である。何故なら、Klugの補正はcount=0になり抜け落ちるケースを補うものであるが、折角補っても所詮、条件を満たさないからである。従って、この場合はKlugの提案した集約演算でよい。集約関数が解決されたアルファ及び膳の緒を生成し、準結合の形で条件式に追加する。

［基本変換則４］

$(tl_1:X(\chi),\Psi(\phi):(count(tl_2:\Omega(\omega):g(\chi,\omega))\theta\ c)\wedge h)(I)$

$=(tl_1:X(\chi),\Psi(\phi):(\exists\alpha_g(s)(f(s,\chi)\wedge s[n+1]\theta\ c))\wedge h)(I)$

ただし、・θ cが＞０を満たすこと（ex. >0, >2, >=4, =3, <>0）

・$\alpha_g=((\chi_g,tl_2):X(\chi),\Omega(\omega):g(\chi,\omega))<\#1,\cdots,\#n;count>$

・$f(s,\chi)$はα_gとXを結ぶχ_gに関する膳の緒

・$n=len(\chi_g)$

３．３　集約関数の発見的変換

発見的変換則のポイントは３点ある。即ち、膳の緒省略、結合省略、Klugの補正不要である。

［５］膳の緒省略

《例７》同一商品を複数の会社から供給され販売している店を求めよ。

```
(u[1])                                  (s[1])
:sales(u)                               :α_g(s)
:count((v[1])                    ⇒      :s[3]>=2
       :supply(v)                       ただし、
       :v[2]=u[1]∧v[3]=u[2]              α_g=((u[1],u[2],v[1])
      )>=2                                    :sales(u),supply(v)
                                              :v[2]=u[1]∧v[3]=u[2]
                                             )<#1,#2;count>
```

膳の緒省略は、対象アルファの目標リスト中の他の属性が集約関数中の開属性群に包まれる場合で、元のアルファの範囲規定を標準集約形ですっかり置換できるので膳の緒を作る必要がない。

［発見的変換則４Ｌ］

$(tl_1:X(\chi),\Psi(\phi):(count(tl_2:\Omega(\omega):g(\chi,\omega))\theta\ c)\wedge h)(I)$

$=(tl_1':\Psi(\phi),\alpha_g(s):s[n+1]\theta\ c\wedge h')(I)$

ただし、・tl_1'、h'は各々tl_1、h中の$\chi(k)$を対応するs[i]に置換したもの

・他の条件は基本変換則４と同じ。

〔６〕結合省略

《例８》現在供給されている商品について、それを販売している店の数を求めよ（例２）。

```
(u[3],count((v[1])                    ((v[2],v[1])
            :sales(v)                  :sales(v)
            :v[2]=u[3]      ⟹         :()
           ))                         )[#1;count]((u[3])
:supply(u)                                        :supply(u)
:()                                               :()
                                                 )
```

結合省略は、集約関数を適用するアルファの条件式が、各開属性が内部の属性と等号で関連付けられ、かつ、その単位論理式が論理積で結ばれている場合に起きるが、実は何等かの形（臍の緒、新集約演算）で外側で結合相当の処理が行われるので内側では省略できるというのが真相である。

［発見的変換則３Ｊ］

$$((tl_1,func(tl_2:\Omega(\omega):g(\chi,\omega)\wedge h)):X(\chi):())(I)$$
$$=(((\chi_g,tl_2):\Omega(\omega):h)[\#1,\cdots,\#n;func'](tl_1:X(\chi):()))(I)$$

ただし、・$g(\chi,\omega)$は単位論理式$\chi[k]=\omega[j]$を論理積で結んだ論理式

・$h=h(\omega)$

・他の条件は基本変換則３と同じ。

〔７〕Klugの補正不要

《例９》階毎に、その階を含めてより上にある店の数を求めよ。

```
(u[2],count((v[1])                    ((u[2],v[1])
            :loc(v)                    :loc(u),loc(v)
            :v[2]>=u[2]     ⟹         :v[2]>=u[2]
           ))                         )<#1;count>
:loc(u)
:()
```

集約関数を適用するアルファの条件式が、各開属性が内部の属性と等号または等号を含む不等号で関連付けられ、その単位論理式が論理積で結ばれており、かつ、外側と内側の範囲が同一または外側が内側の選択型の場合には、空のケースは生じないので、Klugの補正は不要である。

［発見的変換則３Ｋ］

$$((tl_1,func(tl_2:\Omega(\omega):g(\chi,\omega))):X(\chi):())(I)$$
$$=(((tl_1,tl_2):X(\chi),\Omega(\omega):g(\chi,\omega))<\#1,\cdots,\#n;func'>)(I)$$

ただし、・$g(\chi,\omega)$は単位論理式$\chi[k]\theta\,\omega[j]$を論理積で結んだ論理式、θは$=,<=,>=$のいずれか。

・$X=\Omega$，または$X=\Omega[sel]$

・他の条件は基本変換則３と同じ。

４．おわりに

集約関数が入って来ると、一般にアルファが閉じなくなるので、関係論理から関係代数への変換は簡単でなくなる。本稿では、代数表現への変換が容易な標準集約形を中継地点とし、それの生成と解決という２フェーズの変換体系を採った。集約関数の居場所により、標準集約形の生成形も異なる。また、Klugの補正に効率良く対処するため、新らたな集約演算を考案し導入した。標準集約形の生成／解決に関する体系を明かにし、それを構成する基本変換則と発見的変換則の詳細を述べた。新変換法の目的は、集約関数を含んだ関係論理表現から人間が考え出すような最適な関係代数表現を生成することにある。prologを用いて変換プログラムを作成し、考えられる様々な複雑な検索に適用し、極めて満足すべき結果が得られることを確認した。なお、発見的変換則は互いに全く独立ではなく、同時に２つ以上の変換則が適用可能となることもあるので、その優先順位については今後評価する必要がある。

参考文献

[1] ISO/TC 97/SC 21 : Database Language SQL, ISO/TC 97/SC 21/WG 5-15 (1985).

[2] Luo,D.,Xia,D.and Yao,S.B. : Data Language Requirements of Database Machines, Proc. NCC(1982).

[3] 中野良平, 斉藤和巳：ルールに基づいた関係論理／関係代数変換法, 情報処理学会データベースシステム研究会資料, 86-DB-54 (1986).

[4] Codd,E.F. : Relational Completeness of Data Base Sublanguages, in Data Base Systems, Courant Computer Symposium 6, Prentice Hall (1972).

[5] Ullman,J.D. : Principles of Database Systems, Second Edition, Computer Science Press (1982).

[6] Klug,A. : Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions, J.ACM, Vol.29, No.3 (1982).

[7] Kim,W.:On Optimizing an SQL-like Nested Query, ACM Trans.Database Syst. Vol.7, No.3, pp.443-469(1982).

付録　変換例

上段：関係論理表現（変換対象）、下段：関係代数表現（変換結果）

《例３》店員の氏名、給与、及び所属店の平均給与を求めよ。

```
(u[1],u[2],avg[2](v:emp(v):v[4]=u[4])):emp(u):()
(emp[#4=#1](emp<#4;avg[2]>))[#1,#2,#6]
```

《例４》給料が自分が属する店の平均より多い店員を求めよ。

```
(u[1]):emp(u):u[2]>avg[2](v:emp(v):v[4]=u[4])
(emp[∋;#4=#1,#2>#2](emp<#4;avg[2]>))[#1]
```

《例５》階毎に、その階より上にある店の数を求めよ。

```
(u[2],count((v[1]):loc(v):v[2]>u[2])):loc(u):()
((loc[#2>#2]loc)[#4,#1])[#1;count](loc[#2])
```

《例６》３ケ所以上の階で売られている商品を求めよ。

```
(u[2]):sales(u):count((v[2]):loc(v):∋sales(w)(w[1]=v[1]∧w[2]=u[2]))>=3
(sales[#1=#1]loc)[#2,#5]<#1;count>[#2>=3][#1]
```

《例７》同一商品を複数の会社から供給され販売している店を求めよ。…4Lと4Jのコンフリクト

```
(u[1]):sales(u):count((v[1]):supply(v):v[2]=u[1]∧v[3]=u[2])>=2
(supply[#2=#1,#3=#2]sales)[#5,#6,#1]<#1,#2;count>[#3>=2][#1]
```

または　(sales[∋;#1=#1,#2=#2](supply[#2,#3,#1]<#1,#2;count>[#3>=2]))[#1]

《例８》現在供給されている商品について、それを販売している店の数を求めよ。

```
(u[3],count((v[1]):sales(v):v[2]=u[3])):supply(u):()
(sales[#2,#1])[#1;count](supply[#3])
```

《例９》階毎に、その階を含めてより上にある店の数を求めよ。

```
(u[2],count((v[1]):loc(v):v[2]>=u[2])):loc(u):()
(loc[#2>=#2]loc)[#4,#1]<#1;count>
```

《集約関数のネスト例》１００種を越える商品を納める店を２店以上持つ会社を求めよ。

```
(u[1]):supply(u):count((v[2]):(
     (w[1],w[2]):supply(w):count((x[3]):supply(x):x[1]=w[1]∧x[2]=w[2])>100
                                  )(v):v[1]=u[1])>=2
supply[#1,#2,#3]<#1,#2;count>[#3>100][#1,#2]<#1;count>[#2>=2][#1]
```