(2) 論理プログラミングの方式
(Logic Programming Schemes)

Keith L.Clark(インペリアルカレッジ・英国)

(抄訳)

概略
Kowalski以来の論理プログラミング言語方式について，意味論と実現方法を中心に概観する。

1．はじめに

Kowalskiが論理プログラミングの方式として，Horn節プログラムにおける後ろ向き推論(backward chaining)に基づく導出推論システムを提唱したのは14年前の1974年で，その方式を反映した具体的言語としてPrologが実現されたのは15年前である(Battani & Meloni 1973)。以来，Kowalskiの方式の拡張が数多く提案され，Prologに次く多くの言語が実現されて来た。本チュートリアルでは，Kowalski以来，(Joxan & Lassez 1987)のCLPに至るまでの論理型言語の方式を紹介する。

これらの方式はColmerauer(1982)による共通の枠組みで表される。意味論についての議論は(Clark 1979, 1982)による。これは，Kowalskiが与えたエルブラン空間(Herbrand universe)上の等式の連言(conjunction)による代入解の意味論についての最初の形式化であった。この形式化によればKowalskiの方式と他のものとが関連付けられ，より一般的な異なる領域上の式の連言で表された関係が計算される。

なお，lambda−Prolog(Milner and Nadathur 1986) のように一階論理に基づいていないものや，ボトムアップ計算に基づいたもの(Ramakrishnan 1988)については本稿では触れない。

2．計算単位

多くの論理プログラミング方式の計算単位は，項(term)の間のユニフィケーションである。項は可算無限個の共通部分を持たないアルファベット集合F（関数），V（変数）の要素から構成される。関数はそれぞれ引数を持つが，特に0－引数関数は定数(constant)である。項は変数，定数，あるいは関数f(t1,..,tk)k≧1で与えられる。また，t(F,V),及びt(F)はそれぞれ全ての項集合，基底項(ground term)集合を表す。

割り当て(assignment)は等式の集合A={X1=t1,...,Xn=tn}で与えられる。ここで，Xiは異なる変数でtiは項である。このとき，tiをXiの束縛(binding)と呼ぶ。
代入(substitution)とはどのXiもtiを含まないような割り当てSである。
項の対t,t'のユニファイアとはt'{φ}=t'{φ}を満たすような代入φである。
ここで，t'{φ}はtにおける変数がSによる束縛値で置き換えられたものである。
以下では，等式の集合として与えられた項の対のユニファイアの計算アルゴリズムを与える。このアルゴリズムは実際，mgu(most general unifier)を生成するが，ユニファイア，mguの形式的な定義については(Robinson 1979)，

(Lassez et al 1978) を参照のこと。

割り当てと代入の違いは重要である。いま, (E)SをSの存在閉包(existential closure)でS中の変数が存在限定化されたものを表すとすると, (E)Sは関数にどのような解釈Iが与えられても真となる。このような解釈は対<D,A>で与えられる. ここで, Dは空でない領域で, Aはk引数関数に対するDkからDへの写像の割り当てである。いま, MをS中の束縛における変数Y1,...,YkをDの要素e1,...,ekへ写す関数, Xi=v(ti)を解釈Aの下でのtiの値とするとMはSを満たすタプルX1=v(t1),...,Xn=v(tn), Y1=e1,...,Yk=ekを与える。

Sが割り当ての場合, (E)Sは全てのIについて常に真であるとは限らない。例えば, 割り当て{X=f(X)}を考えた場合, (EX){X=f(X)}はfが不動点(fixed point)を持つ場合に限って真になる。

エルブラン解釈(Herbrand interpretation)

(EX){X=f(X)}が偽となる解釈は自由またはエルブラン解釈HIである。領域が関数を含む場合, 論理プログラムでは特別な意味を持つ。代入はエルブラン解釈における空でない関係を表すため, 全ての解釈に対しても空でない関係を表す。

ユニフィケーション・アルゴリズム

以下のアルゴリズムは等式集合E={t1=t1',...,tk=tk'}で表された項の対の集合に対して, そのユニファイアを計算する。アルゴリズムは代入S(成功), あるいはfalseを含む代入を生成して停止する。

(a) 等式f(t1,...,tk)=f(t1',...,tk')を全て等式集合t1=t1',...,tk=tk'で置き換える。
(b) X=Xの形式の等式を全て消去。
(c) 等式t=X(tは非変数)をX=tで置き換え。
(d) 等式X=tのうち, Xが他の等式に現れ, X≠tなるものを選択。このとき,
  (i)Xがtに現れる場合(出現検査(occur check)),falseで置き換える。
  (ii)それ以外の場合, 他の等式におけるXを全てtで置き換える。ここで, X=tは消去しないで蓄える。
(e) 異なる関数f, gに対して等式f(..)=g(..)がある場合, falseで置き換えて停止。

上のアルゴリズムは常に停止する。計算が失敗した場合, もとの等式集合はユニファイアを持たない。計算が成功した場合, SはEのmguである。証明は(Martelli and Montanari 1982)参照。このアルゴリズムはエルブランが等式の集合に対して, それがエルブラン解釈の下で解を持つかどうかを調べるために最初に提案したアルゴリズムに本質的に等しい。なお, 上でd(ii)は以下のように変更可能である。他のある等式におけるXをtに置き換える。さらにこの操作を等式中にXが現れなくなるまで繰り返す。これは並列言語において実現されている方式であるが, これについては4節で述べる。d(ii)が適用される等式X=dを束縛等式(binding equation)と呼び, d(ii)の適用を束縛の通信(communicating), あるいは伝播(broadcasting)と呼ぶ。全ての等式に対して代入を適用することを全域(grobal)伝播, Xが現れるある等式に対して適用することを局所(local)伝播という。変数がメモリアドレスへのポインタであるような実現に際しては, 全ての束縛は自動的に検索され全域伝播は単に等式X=tによるtの値をXのアドレスに割り当てる事によって実現される。局所伝播の実現に際してはXに複数のアドレスを割り当てるか, 束縛指定されたXをコピーし, 等式の値で置き換える事によって実現される。

代入の代りに等式による束縛を蓄える実現方法では(d)は以下のようになる。
(d')等式X=tを選択。いま，既に伝播された等式X=t'があればX=tをt't'で置き換え，それ以外の場合，
（ｉ） Xがtに現れる場合，falseで置き換え。
（ii） さもなければ，X=tを伝播。

局所伝播についてもこの規則で伝播する等式を制限する事によって実現される。

また，同時に複数の等式を選択する場合を考えると，全域伝播の場合はある変数に対して唯一つの束縛値が与えられるが，局所伝播の場合は伝播される変数が共通の値を持たない場合，複数の束縛値を選ぶことが出来る。

特殊な等式処理のためのユニフィケーション

X1,...,Xkをもとの等式集合Eに現れる変数とする。ここでEはある解釈Ｉを与える関係RE={<X1,...,Xk>:E}と考えられる。空の等式集合は{<>:true}あるいは単にtrueを定義する。Ｉに対してREが空でないときに限って，等式はＩにおいて解を持つ，ユニフィケーションアルゴリズムで生成されたSは，全ての解釈ＩでRSはREに含まれる。Sが代入の場合，REは空でなく等式は全ての解釈Ｉの下で解を持つ。Sがfalseを含む場合，RSは空の関係falseとなる。このとき，全ての解釈のもとでRE=falseであることはいえないがエルブラン解釈(HI)のもとでのfalseはいえる。これはそれぞれの等式の書き替えはHIに対する等価性を保存するからである。なお，関数に対する解釈の変更は(a),d(i),(e)に反映される。

関数評価におけるユニフィケーション

以下は(Clark 1978）で与えられたエルブラン解釈のための公理である。
(F1）全ての関数に対して，
f(X1,...,Xk)=f(Y1,...,YK)->X1=Y1,...,Xk=Yk
(F2）全ての異なる関数の対f,gについて，
f(X1,...,Xk)≠g(Y1,...,Yn)
(F3）変数Xを含むt(X)について
X≠t(X)

これらの公理は通常の等号に関する反射，対称，推移，代入公理と併せて，エルブランの等号論理(HET)を形成する。ユニフィケーションアルゴリズムのそれぞれのステップは等号論理の公理の応用である。この関係は(Clark 1978）で形成された：
(R2. 1) HET|=RE=RS,or V(E<->S)V:全称閉包

E<-Sの証明では一般等号公理を使って全ての解釈のもとでREがRSを含むことが示される。一方，E->Sの証明には自由公理(freeness axioms)が必要になる。
(R2. 1）の系として，
(R2. 1) S=falseのとき，またそのときに限ってHET|=RE=false
(R2. 3) Sは代入のとき，またそのときに限って全てのＩのもとでRE≠false

このようにHETのもとではSとEは等しくなり，Sのシンタックス上の簡単なテストによってEが解を持つか否かがわかる。SはHETにおける等式Eの解形式j(soluti on form)と呼ばれる。

その他の関数評価における等式処理

ユニフィケーションの過程は，一般に等式集合がある等号論理のもとで解を持つか否かを調べる過程の特殊な場合と考えられる。例えば，(Colmerauer 1982）は無限有理木(infinite f ational tree)における等号論理を論理プログラムに導入している。

一般の等式の処理チェック

関係として見た場合，等式の集合は一階論理式の集合で述語が＝である特殊な場合と考えられる。ユニフィケーションは関数に対するより一般の等式が満たされるかどうかのチェックに置き換えられる。Colmerauer（1986）の方式はこの例である。ここでは，等式と等式の否定の連言(conjunction)が無限有理木の解生成に使われている。

制約処理チェック

最終的な一般化は式集合において一般の述語を許すことである。ここで，必要になるのは式に現れる述語，関数に関する充足性に対する完全(satisfaction complele)な理論Tである。即ち，Tはすべての式に対してその充足性を決定する。式Fに対するこのチェクはユニフィケーションの拡張として制約論理プログラミング(constraint logic programming)において実現されている(Jaffer and Lassez 1987）。

実用からの要求

ユニフィケーションによる等式の充足性のチェックは，アルゴリズミックでそれ自身が解の生成を行い，アルゴリズムもインクリメンタルであるという点で論理プログラミング言語の計算単位として有効である。比較的大きなEUE'を解集合にまで縮約するためにはEの解Sを使って，SUE'にアルゴリズムを適用することができる。その上，SUE'の解は(d)のかわりに(d)'を使うと，Sの拡張のSUS'として表現される。このインクリメンタルな性質はアルゴリズムの効率化の上で有効である。これはユニフィケーションに基づいた論理プログラムの計算は，等式の集合を漸次的に解集合まで縮約する過程が本質だからである。

ユニフィケーションに基づかない方式では，アルゴリズミックでインクリメンタルな解計算がプログラミングの正当化のために必要になる。

3. ユニフィケーションに基づく方式

3. 1 SLD導出−Kowalskiの方式

1974年にKowlskiによって最初に示されたSLD導出は，LUSH導出(Hill 1974)に基づいている。いま，以下の形式のルール

A←A1, . . . , An

を考える。ここで，A,A1, . . Anはアトム(atom)でAをヘッド，A1, . . . , Anをボディと呼ぶ。これらのルールからなるプログラムに対してゴールGが与えられた場合，Gの計算はプログラムのモデルを充足するGの解釈の計算に相当する。導出の計算過程はゴールGと代入Sの対<G,S>で表される。いま，与えられたゴールGに対して計算規則(computation rule)に従って導出を繰り返すことによって計算木(computation tree)を展開する。この結果，終端ノードの状態が<true,S>である場合は計算は成功(success)したといい，<G,S>である場合は失敗(fail)したという。なお，計算木は常に有限とは限らない。また，計算木を展開する過程における探索戦略(search strategy)のうち，全ての枝をくまなく探索する戦略は均等(fair)であるという。この結果，Gの解は成功した計算木における代入Sのうち，Gに現れる束縛値から計算される。

ここで，理論的に重要な結果として以下のものがある：

（R3. 1. 1） 健全性(soundness)
計算が成功したゴールはプログラムから証明される。

（R3. 1. 2） 計算規則に対する独立性
計算結果は規則に依存しない。

（R3. 1. 3） 強完全性
(strong completeness)
プログラムから証明されるゴー

ルは全て計算可能である。

なお, Kowalskiの方式ではor−並列性のみが許されるが，コンカレントな実行を許したand−並列への拡張も考えられる。

計算規則のタイプ

ゴールの呼び出しにおいて常に一つのアトムを選択するものを深さ優先探索(depth first search)と呼び, 特に常に最も左のアトムを選択するものを最左優先規則(left most call rule)と呼ぶ。これに対してコルーチン規則(coroutining rule)ではそれぞれの呼び出しが交互に計算可能である。

Kowalskiの方式の実現

PrologはKowalskiの方式より一年早く世に出たが，最左，深さ優先バックトラック探索を実現している。IC−Prolog(Clark et al 1979)ではプログラムに注釈(annotation)記述を導入することにより，さらに一般的な計算規則の記述を可能にした。ここでは, Prolog同様深さ優先バックトラック探索が採用されているが，アトムの選択に関してはモードの指定によって最左以外のものが選択できる仕組みになっている。具体的には変数の入出力指定は変数の束縛に対する制約が記述される。

ColmerauerらによるPrologIIはIC−Prologの注釈記述の代わりに，変数の凍結(freeze)機能を導入した。ここではfreeze(V,B)によって，アトムBにおける変数Vの値が具体化されるまでBの評価はサスペンドされる。変数が具体化されない場合，その凍結されたアトムの連言が解として返される点がIC−Prologと異なる点である。

MU−Prolog(Naish 1985)でもコルーチン機能はwait記述によって達成されている。いま，呼び出しゴールがWAIT指定されている場合その評価はサスペンドされ，次のゴールが評価される。Naishはこのwait生成のアルゴリズムを与えているが，深さ優先探索の場合サスペンドによって無限木が生成される場合がある。

NU−Prolog(Thom et al 1987)ではwaitの代りにwhenが導入されたが，これはゴールのサスペンドではなく，選択条件の記述を可能にしている。

その他, SLD導出にor並列を導入したもの(Ciepielewski et al 1984), 制限付でand並列を導入したもの(Conery 1987)などがある。

3．2　Heterogeneous SLD−Naishの方式

SLDのバリエーションとして面白いのがHeterogeneous SLDと呼ばれるものでコルーチンの実現としてNaish（1984）によって提案された。ここでは，計算過程において単一のアトム呼び出しが行なわれ，ボディによって置き換えられる事に変わりはないが，以下の計算では必ずしもこの同じ呼び出しを選ぶ必要は無い。中には別の呼び出しが選ばれ，節の一部を使って計算される場合もある。ここでは計算過程で<call,clause>の対の列が生成されるが，列における対の順序は探索戦略には影響を及ぼさない。

Naishは計算過程で生成される列は呼び出しに使われた全ての節を含むことを示し, SLD導出との等価性を示した。操作的には<G,S>において節Cを使ってGにおけるBiの呼び出しを解こうとして失敗した場合，バックトラックしてGの他の呼び出しBjを解くが，以後の計算では余分なBiの計算は繰り返さない。Naishはこれが以下の知的バックトラッキング(intelligent backtracking)を正当化していることも指摘している。即ち，ある呼び出しが失敗してバックトラックの後別の呼び出しを行う場合，最近呼び出したゴールを優先的に選択する。この操作は呼び出しが成功するか，バックトラックするゴールがなくなるまで繰り返される。

3．3　SLDNF−Clarkの方式

(Clark 1978)では問い合せ，ルールのボディに否定(negation)がある場合のSLD導出の拡張が提案された。ここで，プログラム中のルールは以下の形式

A←L1,...,Lk

で表され，Liは正または負のリテラルである。ゴールの計算規則は負のリテラルの選択に関しては変数を含まないものに限る。このような計算規則を安全(safe)と呼ぶ。正リテラルの選択に関してはSLDと同様である。

いま<G,S>における負のゴール～Bの呼び出しに対して，変数を含まない問い合せB{S}が評価され，その結果が全て失敗した場合もとの～Bは成功しGから除去される。逆に，B{S}が成功した場合～Bは失敗し<G,S>は有限失敗する。これが失敗による否定(negation as failure)と呼ばれる規則である。なお，負のゴール呼び出しが変数を含む場合，その計算木flounderと呼ばれる。

PrologはSLDNF規則に基づいているが，安全性はプログラマの注意に委ねられている。一方，MU−Prolog，NU−Prologでは安全性が保証されており，またIC−Prologは安全性は保証されていないものの，負のゴールが変数をばらまこうとするとエラーメッセージを出力する，SLDNFによる計算結果はプログラムの論理的帰結ではない。第1に失敗を否定の証明と解釈するのはエルブラン解釈においてのみ有効であり，第2にプログラムの節は述語の完全な定義を与えているものと暗黙に仮定されているからである。

プログラムの完備化(completion)

プログラム節

(a) r(t1,...,tn)←L1,...,Lkに対して，

(b) r(X1,...,Xn)←X1=t1,...,Xn=tn:L1,...,Lkをガード形(guarded form)といい，

(c) r(X1,...,Xn)←(EY1,...,Yj)(x1=T1,...,xN=TN:l1,...,lK)を一般形(general form)という。ここで，X1,...,Xnは(a)に現れない新しい変数で(EY1,...,Yj)は(a)中の全ての存在限定された変数である。また，':'は理論積','に等しい。

いま，

r(X1,...,Xn)←E1

:

r(X1,...,Xn)←Em

をrの一般形とするとｒの完備化は以下のようになる。

r(X1,...,Xn)<=>E1 v..v Em

いま，プログラムPに対してcompleted(P)を

（ｉ）Pにおいて節のヘッドに現れる全ての述語の完備化

（ii）それ以外の述語Q(x1,...,Xk)について

Q(x1,...,Xk)<=>false

とすると，プログラムPの完備化Comp(P)は

Comp(P)=completed(P)＋エルブランの等号公理(HET)となる。

SLDNFの健全性

(R. 3. 3. 1)計算が成功したゴールは完備化プログラムから証明される。

(R. 3. 3. 2)安全な計算規則のもとで失敗したゴールは完備化プログラムからその否定が証明される。

(R. 3. 3. 3)ゴールGに対する解は完備化プログラムから証明される値と正確に一致する。

否定ゴールにおける局所変数

(R. 3. 3. 2)でSLDNFプログラムが否定の連言(conjunction)を考える。この場合～Cにおける安全性はC{S}がG{S}の他のゴールと

変数をシェアしないという条件にまで弛められる。これらの局所変数は否定の内側で存在限定されており, PrologのNegati on as Failureの動作と一致する。しかしながら，こうした緩和は節中の変数は全称限定されているという規則に影響を及ぼすため, NU−Prologのように否定における存在限定は陽に記述する方法のほうが良い。NU−Prologでは否定において存在限定されていない変数に対しては安全性のチェックを行っている。

否定ゴールのサスペンド

失敗による否定の他の拡張として，束縛されていない全称限定の変数を含む否定ゴールが選ばれた場合，そのゴールをサスペンドしておく方法がある。このゴールは変数が束縛され評価可能となった段階で再び選択される。

全域変数の値生成

（R. 3. 3. 3）によれば失敗による否定から解を計算することが可能となる。いま，〜Cが選択された場合, C{S}の計算を考える。いま, C{S}の変数束縛を行わない成功枝がある場合, C{S}はfalseで置き換えられる。全ての枝が失敗した場合, C{S}は消去される。また, C{S}の変数の全ての束縛値がS1, . . . , Snで与えられた場合，〜Cは〜S1^..^〜Snで置き換えられる。このように否定を分配することによって生じる不等式を扱うためには，ユニフィケーションの過程で不等式との一貫性をチェックするためのSLDNFの拡張が必要になるが，詳しくは6節で触れる。

構成的証明

(Clark 1978)でも指摘したようにSLDNFはそれぞれの否定アトムの構成的証明を必要とするため，場合分けによる証明は使われない。例えば，プログラム

p←p　q←p　q←〜p

を完備化したものからはqが導かれるにも関わらず問い合わせqはいかなる計算規則の下でも停止しない。これはqの評価過程で排中律pV〜pが使われないため, pの真偽値に関わらずqが真である事が証明できない。

完全性

残念ながら完全性は簡単にはいえない。先ず，一般のプログラムにおいて問い合せがflounderを起こさないと言う保証はない。2番目にcomp(P)から得られる結果は全て排中律を使うことなく，構成的に証明されねばならない。3番目にcomp(P)から構成的に〜B{S}が証明される場合，それを導出する有限失敗木の存在を保証する必要がある。最初の2つの条件はPのシンタックスに対する制約を課し，3番目の条件は計算規則が安全である上に均等(fair)であることを要求する。

均等な計算規則という概念はLassezとMaherによって論理プログラムに導入されたもので，どの述語の呼び出しも不確定に延期しないと言う規則である。深さ優先探索は均等では無い，均等であるためにはコルーチン機能が必要になる。以下のプログラムは均等な計算規則必要とする例である：

p(X)←q(Y),r(Y)
q(h(Y))←q(Y)
r(g(Y))

Prologの最左呼び出しでは, p(a)の呼び出しに対して無限木が生成される。均等な計算規則の元では有限になる。

階層プロフラム

Pの述語間の関係を表す有向グラフを考える。述語PとQの間のエッジに対して, QがPの定義節に正のアトムとして現れている場合は＋を負のアトムとして現れている場合は－を不加する。

エッジは＋と－の両方のラベルを持つ場合もある。階層プログラムとはこのようなグラフがサイクルを持たないプログラムである。即ち，階層プログラムは再帰節を含まない。(Clark 1978)では階層プログラムに対する完全性の議論がインフォーマルに示されている。階層プログラムに対する重要な結果は，後に Sherpherdson（1985）によって与えられた：

（R. 3. 3. 4）階層プログラムにおいて，節中の全ての変数はボディの正リテラルに出現し，ゴールGにおいて負リテラルに現れる変数は全て正リテラルにも現れる場合，Comp(P)から証明される結果はSLDNFによって計算される。

上で変数に対する条件は問い合せ評価がflounderを起こさないことを保証し，階層条件は計算木の有限性を保証している。従って，comp(P)からは全てが構成的に証明される。(Llyd and Topor 1986)が示したように，プログラムのシンタスク上の制限は否定の呼び出しのみに使われる述語に対して緩和される。これらの述語において節のボディにおける局所変数は正のアトムにも現れねばならないが，ヘッドに現れる変数に対してはこの限りでは無い。この条件を満たすプログラムとゴールは容認される(allowed)と呼ばれる。容認されたプログラムとゴールに対しては，プログラムの計算結果はゴールの変数束縛値を与える。

否定がないプログラムの安全性

Jaffer et al（1983）による完全性に関する定理は以下の通り。
(R. 3. 3. 5）確定プログラム(definite clause program)においては基底アトムBに対してcomo(P)|=～Bならば，全ての均等な計算規則においてはBは有限失敗する。

容認されたプログラムにおいて，否定ゴールの呼び出しが確定プログラムで定義された場合に上の定理を応用しようとすると再びcomp(P)における構成的でない証明の問題が生じる，先の例

p←p　q←p　q←～p

で，否定の呼び出し～pは確定プログラムで定義されている。

厳密プログラムに対する完全性

comp(P)から構成的でない証明が存在しないような厳密プログラム(strict program)に対する完全性がKunen(1988b)に議論されている。プログラムが厳密(apt et al.1988)でないとは，述語間の依存グラフにおいてPからQへ偶数個の－ラベルのエッジを含むパスと奇数個の－ラベルのエッジを含むパスが存在する場合である。厳密なプログラムでは，述語pは直接的にも間接的にもある述語qの肯定と否定の両方によって定義される事はない。Kunenの結果は以下の通り。
(R3. 3. 7)PとGが許容されていて(allowed)，Pが厳密である(strict)場合，
基底代入(ground substitution)Sに対して，comp(P)|=GSならばSはSLDNFで計算可能な解である。また，comp(P)|=~EGならばGの有限失敗木が存在する。

なお，Sherperdson（1988），Kunen（1988）の否定に関するサーベイは失敗による否定の意味論，健全性と安全性について議論しており，大変興味深い。

4．スキームに基づく並列ユニフィケーション（単一化）

4．1　GLD導出－Wolfram,Maher,Lassezスキーム

unrestrictred and並列評価－すなわち共有変数をもつ2つ以上の（ゴールの）呼び出しでの並列ユニフィケーション－を許す図式のうちで最初のものは，GLD図式(Wolfram et al 1984)である。プログラムとゴールはSLDの場合と同様であるが，計算の状態は<G,S>という対である。ここで，Gは（ゴールの）呼び出しのmultisetである。sLdの場合と同様に，Sは代入か又はfalse（失敗）を含む。この計算規則ではGからN>=1個以上の呼び出し{B1,...Bn}を選びだす。{A1<-G1,...An<-Gn}がn個のプログラム節のvariant（変数名とつけかえたもの）で（しかもGとの間に共有変数を含まないもので），かつBiとAiが同じ述語名をもつならば，計算の次の状態は

<G U G1 U...U Gn（G,G1,..,GnNO和集合），S'>

となる。ここで，S'は次の形の単一化の解である。

SU{B1=A1,...Bn=An}

（ここでは単一化のアルゴリズムはrが述語である場合についてr(t1,..,tk)=r(t'1,....,t'k)を規則(a)を使って書き換えられるように自明な拡張がなされているものとする。）

非同期GLD

GLDスキームでは，並列性のscopeには制限がある。なぜならば，次のステップのためのゴールが選ばれる前に単一化全体が停止しなければならないからである。選択されたゴールの節のボディへのreductuinは，同期して行なわれる。次の示すGLDの一般化は，Wolframらによってimplicitlyに記述されたものであるが，異なるプロセッサでの非同期なreductionを許すものである。非同期GLD(AGLD)では状態は<G,E>という形で待つ。ここで，Gはゴール呼び出しのmultisetであり，Eは等式のmultisetでありfalseを含みうる。この計算規則では，k個の(k>=0)ゴールの呼び出しをGから，またn個(n>=0)の等式をEから選び出し，（ここでk+n/=0である。）ここで各等式には単一化の規則のいずれかが適用される。選ばれた等式は，適当な規則で処理される。E'を書き換えの済んだ等式の集合（この場合はmultiset?）とする。
また，A1<-G1,...Ak<-Gkを選ばれた呼び出しに出現する述語を定義している節のk個のvariantとする。SはE'の部分集合で，このstep以前にEのすべての等式にglobalに伝達されたbindingであるものとする。計算の次のstepでの状態は次の様になる。

<G U G1 U...U Gk,E'U{B1{S}=A1,...,Bk{S}=Ak}>

成功と失敗および計算結果の定義はSLDの場合と同様であり，Wolframらの論文によれば計算結果は，いかなる計算規則のもとでもSLDの場合と同じである。

計算規則の様々なタイプ

GLDは次の様な計算規則を持つAGLD（の特殊な場合）である。すなわちE成分が解の形式にまでreduceされるまで等式の方だけを常に選ぶようにしている。反対に，状態のゴール成分がtrueであるときだけ等式を選び，すべての単一化を計算の最後まで遅らせるような計算規則をも考えることができる。
Wolframらはこのような規則を計算規則の独立性の証明に使っている。これは（which is.. whichが何をさすのか，わかりません。）現在のところ，解の形にreduceされる等式の最終的な集合を集める唯一の方法である。

中間的な計算規則を考えよう。すなわち，等式と一緒にアトムを選ぶことをも許し，また，Bi{S}=Aiがbinding式の集合Siにreduceされるまでi番目の節のボディ部分から出てきたアトム（の列）Giのうちのいずれも選ぶことは遅らせ，これらのbindingはBi{S}Aiの子孫の等式にのみローカルに伝搬する。すなわち，Biの共有変数へのbindingは，最初は他に選ばれた呼び出しに伝えられることはない。さらに，Giの中のボディの呼び出しはローカルに拡張されたSUSiに依存して単一化される。これは，各呼び出しをSLDと同様に評価し，同時に選ばれた呼び出しの間の共有変数への計算済みのbindingは各呼び出しがtrueになった時のみに（bindingをglobalに伝搬することによって）比較するような実現のしかたに対応している。Epilog system(Wise 1986)やPrism system (Kasif 1983)はこの計算規則によるAGLDのOR−並列版である。

逐次AGLD規則は各ステップで一つのアトムか一つの等式を選ぶ規則である。Eが解の形になるまで常に等式だけを選ぶ場合はSLDと同じである。中間的な規則はSLDの拡張となる。なぜならば，これはundoneでない部分単一化を他のアトムに切り換える前に行なう様なコルーチン的な実現を許すからである。例えば，Xについて変数でないbindingを生成することを許さない様な呼び出しBiが呼び出され，X=tというbindingが生成されるまで，B1{S}=Aiの書き換えがつきまとわれる。このbindingは保持されるがbroadcastはされない。次の呼び出し，又はXのプロデューサがX=t'というbindingを作るために選ばれ，これがbroadcastされる。B1{S}=Aiの書き換えはX=tのところをt=t'に置き換えて続けられる。

Absys(Foster & Elcock 1969)は，これは最初の論理型プログラミング言語であるが，本質的には逐次規則のAGLDの実現である。項は定数か変数か又はリストであり，プログラムは3．3での定義の構文的な変形である。この計算規則は規則d(ii)の適用のためにX=Y(XとYは異なる)という形の等式を選ぶことはない。このような変数対変数の等式は，他の等式がX又はYに変数でないbindingを生成しなければ，計算済みの答えについての制約として残る。Absysはnagation as failureの規則も実現したが，safety checkは含まなかった。

データフロー並列規則

逐次型コルーチン規則の次の様な一般化，すなわち，ある呼び出しの入力変数へのbindingの式が生成されたとき，他のアトムを選ぶ規則は，節Ai<−Giのボディ部分Gi=に含まれる呼び出しをBi{S}=Aiがallowed binding,すなわちgrobalにbroadcastしてよいbanding,の集合にreduceされるまで遅らせるような並列規則となる。呼び出しの入力変数Xのためのいかなるbinding X=tも，allowed bindingではない。このようなbindingはbroadcastできない。X=tの処理はbinding式X=t'が別に生成されて，X=tに伝わるまでsuspendされる。

If need...（この1文，よくわかりません。）

このような変数の出現を，呼び出しBiの入力変数出現と呼ぶ。Biの入力でない変数について生成されたbinding式だけがallowedであり，globalにbroadcastできる。

Allowed bindingのbroadcastを遅らすことについて。

Bi{S}=Aiの書き換えによって生成されたallowed bindingは生成後，すみやかにbroadcastされる必要はない。データフロー規

則が，呼び出しの評価をあるbindingがそこに伝搬するまで遅らせるために用いられる。（何故ならば呼び出しを評価するのに適当な節は，受け取ったbindingから決まるからである。）これが，呼び出しGiの選択を，Bi{S}=Aiが余分にbroadcastされたbindingをも含めたコンテクストの元で成功するまで，遅らせる理由である。もし，すべてのallowed bindingをbroadcastするのをBi{S}=Aiがallowed bindingに完全にreduceされるまで遅らせるとしたら，失敗する単一化は他の呼び出しに何の影響も及ぼさない。ある節Ai<-Giに他の節A'i<-G'iを，Bi{S}=Aiがallowed bindingの集合にreduceするように代入することは，何の問題もなく可能である。通常は，allowed bindingをBi{S}=Aiの子孫にローカルにbroadcastすることが，必要となる。例えば，節のヘッド部分に出現する変数への整合性のないbindingのチェック等のためにである。しかしながら，環境Eの中の他のすべての式へのglobal broadcastingについては遅らせることができる。ある変数へのallowed binding Y=tをglobal broadcastするのを遅らせることは，もしもYへのある（別の）allowed bindingが別の呼び出しから生成されたら，Y=tがt'=tに変換されることを意味する。

Atomicな（割り込みのない）単一化

Bi{S}=Aiがallowed bindingの集合Siに完全にreduceされている様なAGLD計算の状態が存在するものとしよう。（他の単一化によって生成されたBi{S}中のある変数へのglobally broadcastされたbindingを受け取った後であるかもしれない。）もし，計算規則がSiからglobal broadcastingのためにあるbindingを選んだら，それらをすべて選ぶこと，という制約を持つ場合，その規則はatomicな単一化を実現している。（2節で既に述べたように，いかなる変数についても，global broadcastingするbindingは一つだけを選ぶように，計算規則を制約しなければならない。）加えて，もし，Bi{S}=Aiから生成されるすべてのallowed bindingがglobal broadcastingのために選ばれる前に，Giのいかなる呼び出しも選ばれなかったとしたら，この計算規則はatomicなtest unificationを実現している。マルチプロセッサによる実現では，atomicな単一化は，変数のbindingのglobal broadcastingの同期を必要とする。どのプロセッサも，bindingをbroadcastする前に，Siでbindされる各変数にallowed bindingの別解を作りうる場合は，他のすべてのプロセッサにbindingの許可を求めなければならない。許可をすべてのプロセッサから得られない場合は，（あるプロセッサの許可がおりない場合は，）許可を撤回するような用意も必要である。マルチプロセッサでatomicな単一化を実現するためには，とても複雑なプロセス間のプロトコルが必要である。

Specifying the allowed....

Prolog IIのfreeze callに似たものが，呼び出しBiの入力変数出現のannotationの一種である。Concurrent Prolog(Shapiro 1983)では，これを入力を？で記号付けすることによって実現している。freeze callを使う場合と同様に，この制限は（代入した結果には）遺伝されない。呼び出しBiの？のついた変数に受けとられたbinding t'での変数の出現は，それ自身が？がついていないかぎり，Biの入力変数ではない。もし，入力変数である，という性質が自動的に遺伝するものならば，そしてさらに，Biの子孫の呼び出しに適用されるならば，これはIC-Prologのeager consumerと似たものになる。

呼び出しBi{S}とある節のヘッド部Aiの単一化のallow bindingを決めるもうひとつの方法

は，MU−Prologの場合と同じようにuseのallowed mode(??)をその節と結びつけることである。

Relational Languageおよび，Parlog (Clark & Gregory)では，これはk−引数の各述語rについて，各引数を入力(?)および出力(^)でモード付けすることによって実現している。r(t'1,...t'k)を今（マッチしようと）試している節のヘッドと，またr(t1,...tk)を呼び出しとする。i番目が入力引数であるならば，この変数に出現する項t'iのために生成されたbindingfだけが，ti{S}=t'iの単一化allowed bindingである。この単一化の書き換えが他の変数についてV=tという等式を生成したら，これは入力出現とみなされ，このbindingはbroadcastされない。もしj番目の引数が出力であった場合，単一化tj{S}=tjで生成されたすべてのbindingがallowedである。出力引数の項の等式の書き換えは，すべての入力引数の等式がallowed bindingにreduceされた後になって，はじめて開始される。このことは，入力変数Vへのsuspendしている等式V=tへのすべてのbinding broadcastは，他の呼び出しの単一化によって生成されなければならないことを，意味する。さらにこのことは，入力引数項ti{S}に出現するすべての変数，及びt'i={S}がallowed bindingだけにreduceされる前にglobalにbroadcastされるbinding(for these variables:these variablesが何を示すかは，はっきりしません)に含まれる変数は，すべて呼び出しの入力変数であることを意味する。入力変数であるという性質は遺伝される。GHC (Ueda 1985) においては，単一化全体Bi{S}=Aiにおいて節に現われる変数へのbindingのみがallowedである。Parlogの言葉で言うなら，すべての引数が入力である。GHCにおいては，すべての出力は節のボディ部分で陽に記述された等式呼び出しによって実現される。

ガードの呼び出し

単一化Bi{S}=Aiでのallowed bindingのglobalなbroadcastingは，常にdisallowed bindingがなくなるまで遅らせられる。さらに，ボディ呼び出しGiのガード部分G'iがtrueにreduceされるまで，さらにこれらのglobal broadcastingを遅らせることも可能であった。ガード部分を評価している間は，ガード呼び出しとBi{S}=Aiの書き換えで生成された等式へのローカルなbroadcastingだけを持って，allowed bindingのgiobal broadcastingをも防がなければならないことに，注意されたい。GHCにおいては，入力変数としての制限を遺伝することによって実現している。すなわち，呼び出しのすべての入力変数はG'iのすべての呼び出し及びそれらの子孫の入力変数となる。Concurrent Prologでは，ガードが成功するまでは，呼び出しの変数ローカルなbroadcastingのみが許される。Parlogでは，呼び出しの入力変数へのallowed bindingを生成できない呼び出しのみが，ガードに出現できる。このようなガードは安全であるといわれる。これら3つの言語のflat版においては，primitive（システムで用意された述語）だけが，ガード部分G'iに出現できる。これらの呼び出しを評価は，節のヘッド部の単一化の拡張として実現される。

呼び出し変数へのallowed bindingをbroadcastすることをこのように抑制することは，（呼び出しの）節のヘッドとの単一化，及びガード呼び出しの評価を，あるBiについてすべての節で並列に行うことを可能にすることに注意されたい。このような並列な評価はbindingのbroadcastと競合することはない。さらにこのことは，Bi{S}=Aiの書き換えと同

時にガードの呼び出しを開始できることを意味する。

Committed choice
Bi{S}=Aiですべてのガード部分の評価がallowed bindingの集合Siを出力して成功している状態を考えよう。Relationa language（これは最初のcommitted choice言語であるが）及び，Parlog,GHCにおいては，（このような場合）呼び出しBiについて節Ai<−Giを使うことにcommitmentできる。他の節を用いた並列単一化，ガード部の評価については，すみやかに放棄されるか，停止してしまう。Gi−G'iに含まれる呼び出しが選ばれRelational language及びParlogについては，出力引数項のための等式の書き換えt'j{S}=tj{S}が開始される。この書き換えによって生成されたallowed bindingのbroadcastingはatomicではない。これらは，いつ生成されたかとは独立にbroadcastされ得る。GHCにおいてはGi−G'iでの等式の生成は呼び出し変数へのallowed bindingを生成し，これらはやはりatomicにはbroadcastされない。

Concurrent Prologにおいては，Bi{S}=AIの書き換え及びガードの評価をしている間に，すべてのallowed bindingが生成されるまではcommitすることはなく，ガードの評価はatomic nibroadcastされる。

ParlogとGHCにおいては，共有変数に対してはただひとつの呼び出しがbindingを生成するようにし，他のすべての（その変数を共有する）呼び出しは，そのbindingがbroadcastされるまでsuspendするように，プログラムを組まなければならない。Concurrent Prologにおいては，atomic test unificationを使って唯ひとつのbindingがglobalにbroadcastされることを確認し，呼び出しは節にcommitする前にbroadcastされた値をtestするように強制することによって，競合するようにプログラムを組むことが許される。この場合の不利な点はatomicな単一化の実現の困難さである。

（Burt & Ringwood 1988）は近年，Flat Parlogの拡張の中で，単一のallowed bindingのatomic test broadcastingについての，より簡単な概念を提案した。呼び出し変数の単一のallowed bindingはtest bindingとして指定される。計算規則はこのような指定されたallowed broadcastを選び，他のbindingをbroadcastしようとする前にgobalにbroadcastするか又は，Gi−G'i呼び出しを選ぶ。最近提案されたConcurrent Prologの改良版であるFCP(|,:,?)(Klinger 1988)では(Saraswt 1988)のアイデアを借りて，ガード部分をask−成分とtell−成分に分けている。Tell成分だけが，節に含まれない変数についてのbindingを生成することができる。呼び出しと節のヘッド部の単一化とask成分の評価については，節の変数へのbindingだけが許される。tell成分のはたらきは，Parlogの出力変数項への単一化と同様である。tell成分によって生成されたいかなるallowed bindingも，ヘッド部の単一化やask成分の評価のためにbroadcastされることはない。違いは，FCP(|,:,?)においては，ヘッドの単一化とガードが成功しtell成分のlallowed bindingがatomicにbroadcastされるまでcommittmentが起きないことである。

（Takeuchi & Furukawa 1986），(Shapiro 1988)はいずれもAGLDスキームに基づくcommitted choice型並列論理型言語の族についてのサーベイであり，プログラミング技法の例を含む。

Suspending.....

もうひとつの計算規則は, Bi{S}と単一化可能なヘッドAiを持つ節が唯一つであるような呼び出しBiを選ぶものである。これを実現するには, bindingのlocal broadcastingについて異なる節を試さなければならない。もし2つ以上の呼び出しとヘッドの単一化がbindingの集合にreduceされた場合は, 1つを残してすべてfailされるようなbindingが他のどこかからbroadcastされてくるまで, この呼び出しはsuspendする。このような単一化によって生成されたbindingは, すべてbroadcastされる。P-Prolog(Yang-Aiso 1986)はこのようなsuspension規則を持つ。

Paralell selection...

lbroadcastされる変数へのbindingを待ちながら呼び出しをsuspensionさせるような, すべての言語において, deadlockが生じうる。deadlockはsuspensionの規則を無視して, ひとつの呼び出しを選ぶことによって打ち破られる。

Andorra Prolog (Brand1988) において, deadlockは正にこの方法で打ち破られている。計算規則は唯一つの候補節をもつような呼び出しを任意個選び, 2つ以上の節があるときはsuspendする。呼び出しの候補節とは, allowed bindingだけを生成してヘッド部の単一化が成功し, かつガード部のprimitiveへの呼び出しのある集合が成功のうちに終了したものである。呼び出しへのallowed bindingはNU-Prologのwhen文と同様にwait宣言によって, 記述される。commit operatorはcommitted choice言語のそれと同様に節を候補節にする。唯一つの節が候補節として残るまで, bindingはglobalにはbroadastされることはない。この言語はatomicなtest unificationを持つ。すべての呼び出しがwait宣言によってか各呼び出しについて候補節が二つ以上存在してsuspendした場合, 一つの呼び出しが選ばれて各候補節について複数の新たな選択可能な状態が生成され, OR並列計算における場合と同様につきまとう。この言語は, SLDスキームの探索能力にcommitted choice付AGLDの並列性を組み合わせている。ペナルティは実現が必要以上に複雑になることである。

P-PrologはOR並列とAND並列評価を両方備えているが, OR分岐はすべてのAND並列呼び出しがsuspendするまで, 遅れることはない。これは選択可能な計算pathの並列評価を持つuncommitted AND並列を備えている。

Saraswat (1987) のCP言語は, call blockの概念を持つcommittedかつuncommittedなAND並列性を持つ。call blockはbindingのbroadcastを, 呼び出しとその子孫に制限する。bindingは兄弟blockの間では, そのblockの各呼び出しが成功した時のみ, broadcastされる。各呼び出しをblockに置くことは, EpilogやPrismで使われたtermination computation規則での通信を与える。Saraswatの言語は, altanativeな評価pathの並列及び逐次(backtrackingによる) 検索をも許す。

Parallelised NU-Prolog(Naish 1988) やANDOR-II(Takeuchi 1987)も, committed choice AND並列とaltanative pathのuncommittedな探索の組み合せについて, 最近提案されたものである。(Clark & Gregory 1987) はPrologとParlogの組み合せを可能とする方法について議論したものである。