TR-048

逐次型Prologプログラムの解析

尾内理紀夫、清水　肇
益田嘉直、麻生盛敏

1984. 3

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

逐次型Prologプログラムの解析

尾内理紀夫　清水肇　益田嘉直　麻生盛敏
（財）新世代コンピュータ技術開発機構

1．はじめに

DEC-10 Prolog [18]プログラムを静的，動的に解析するアナライザを開発し，ICOTで開発されたプログラムを解析し，（39個のプログラムを静的に，そのうちの2つを動的に解析。）各種データを収集し，プログラム特性について考察した。

静的アナライザ（DEC-10 Prolog によって記述され，clause数は，約 230）は，単にプログラムを先頭から読み，各種データ（たとえば，OR relation 数，AND リテラル数，述語の参照数，head述語の引数個数，head述語の構造データ引数個数，body側の引数個数，body側構造データ引数個数，組込み述語使用頻度，cut 数，等）を出力する。

動的アナライザ（DEC-10 Prolog によって記述され，clause数は，約 280）は，並列実行可能なPrologプログラムにゴールを与えて実行させ，その過程で各種データ（静的解析時のcut 数を除く収集データ項目，OR fork の際の実際のunification success 数，その他）を収集する。このため，DEC-10 Prolog プログラムを並列実行可能なように書き換える。

この解析の結果，

①DEC-10 Prolog は逐次実行型であり，メモリ空間が十分でない。これに起因して，プログラムの多くは，cut を多用しており，決定的なプログラムとなっている。しかしこれを並列実行可能なプログラムに書き換えることが可能な場合がある。

②AND リテラルの約半分は組込み述語であり，組込み述語の実行速度がプログラム実行速度に影響を及ぼす。

③database clause のOR relation 数はinference clauseのそれの約４倍であり，database clause が大規模化するとこの比は増大するであろう。よって大規模database clause を含むプログラムの場合，database clause に関する unificationの高速化がPrologプログラム実行の高速化に大きな影響を及ぼす。

等が考察された。本稿では，解析結果及びそれに基づく特性の考察，および，逐次型Prologプログラムから並列実行可能なPrologプログラムへの書き換え規則について述べる。

2．静的解析

2.1　収集データ項目

静的アナライザの収集データ項目について述べる。

(1) OR relation数

同一head述語シンボルを持ち，引数個数が同一のclause同志をOR relation にあるという。そのようなclause同志の個数をOR relation 数と呼ぶ。たとえば，次のようなclauseがあった時，この inference clause（後述）のOR relation 数は３である。

```
p([]).
p([X | Y]):-q(Y).
p(1):-r(1).
```

total OR relation 数の定義を次に示す。（Σはプログラム内のclauseについての総和）

Σ OR relation数

静的アナライザでは，cut "!" は，AND リテラルには含めない。よって，たとえば，h :-!. は，ruleではなく，h. というunit clause とみなす。

① inference clause
OR relation にあるclauseの中に少なくとも一つのruleを含むclauseをinference clauseと呼ぶ。

② database clause
OR relation にあるclauseすべてが unit clauseのみから構成される clause を database clauseと呼ぶ。

(2)参照数

あるclauseが参照された回数，即ち，そのclauseのhead述語がbody側に現れた回数である。

(3) ANDリテラル

AND リテラルの定義を次に示す。

①body側の ",” で区切られたリテラル。

②但し，“;”は，“,”と同等とみなす。

③但し，組込み述語 is,>,>=,<,=<, 及び，\+に関しては，その引数の内部構造をも解析する。

例えば，

p(X,Y,Z):- X>Y,Z is Y-X.

のbody側は，三つのリテラル __>__，_is_，__-__から構成されていると見なす。

その他に，head述語引数個数， head 述語の構造データ引数個数(head述語引数の内var,atom,integer以外の引数個数である。var,atom,integer以外の引数を構造データ引数と呼ぶことにする。)，組込み述語(Evaluable predicate)数，body側引数個数（AND リテラルの引数の総和），body側構造データ引数個数（AND リテラルの構造データ引数の総和），cut 数がある。

2.2 静的解析結果と静的特性

ICOTで開発された各種DEC-10 Prolog プログラムの解析結果を示すとともに，それから考察されるDEC-10 Prologプログラムの特性について述べる。

（各プログラムの機能概要については付録３参照，より詳しくは，参考文献参照）

2.2.1 inference clauseの解析結果と特性

(1) 解析結果

プログラム全体（今，プログラム個数は33 付録３参照）についての収集データの平均値の定義と値を次に示す。又，その幾つかに関しては，各プログラムの収集データ値.vs.プログラム分布のグラフを付録２に載せる。（Σは，33個のプログラム全体についての総和）

①平均OR relation 数（av OR ）

Σ Total OR relation数／Σ inference clauseの種類数 =2.7

各プログラムの平均OR relation 数に注目したプログラム分布のグラフを付録 2-1に示す。

②平均参照数

Σ参照数／Σinference clauseの種類数 =3

③平均head述語引数個数

Σ head 述語の引数個数／Σ Total OR relation数=3.2

各プログラムのhead述語引数の平均個数に注目したプログラム分布のグラフを付録 2-2に示す。

④head述語の平均構造データ引数比

Σhead述語の構造データ引数個数／Σ head 述語の引数個数 = 0.2

head述語の構造データ引数比に注目したプログラム分布のグラフを付録 2-3に示す。

⑤平均AND リテラル数

Σ ANDリテラル数／Σ Total OR relation数=3

1 inference clause当たりの平均AND リテラル数に注目したプログラム分布のグラフを付録 2-4に示す。

⑥平均組込み述語数

Σ組込み述語数／Σ Total OR relation数 =1.4

⑦平均組込み述語数比

Σ組込み述語数／Σ ANDリテラル数 =0.5

平均組込み述語数比に注目したプログラム分布のグラフを付録 2-5に示す。

⑧平均body側引数個数

Σ body 側引数個数／Σ Total OR relation数 =6.5

各プログラムのbody側の引数の平均個数に注目したプログラム分布のグラフを付録 2-6に示す。

⑨body側の平均構造データ引数比

Σ Body 側構造データ引数個数／Σ Body 側引数個数 =0.12

body側構造データ引数比に注目したプログラム分布のグラフを付録 2-7に示す。

◎1 clause当りの平均cut 数

Σ cut数／Σ Total OR relation 数 ≒0.65

各プログラムの平均cut 数に注目したプログラム分布のグラフを付録 2-8に示す。

(2) 特性

プログラム全体の平均cut 数は，0.65 であり，付録 2-8より，大部分のプログラムは，決定的に書かれていることがうかがえる。数つかのプログラムにおいて，cut 数が 1 以上になるが，それは，

p(_ ,_):- ……，！；……，！；……！，

の形のclauseが要因となっている。プログラム全体のavORは，2.7であり，平均cut 数0.65と合せ，解析したプログラムの多くは，実行時にAND-OR探索木の幅が，急激には，広がっていかないことが予想される。（なぜならば，OR relation 数は複数あるが，成功するのは，多くの場合一つと考えられるから）もちろん，DCG program （プログラムNo.４）のように高いOR relation 数を持つものもある。

avOR 2.7は，inference clauseの集合から，match する可能性のあるclauseを逐次的にpick up し，それを次々にunification するというpipeline 的な処理をしても，ここ(match&unify) はbottle－neckになりにくいことを示唆している。

付録 2-1において，平均 OR relation数が13～14のプログラムでは，

```
p(_, _) .
   ⋮          154個の並び
p(_, _) .
p(_, _):- !, otherwise(_)
```

というようなclause群があり，これは，定義によりinference clauseとなる。このことが，平均 OR relation数を13～14にも押し上げている理由である。

平均参照数は，約 3であるが，最大参照数が10以上の解析対象プログラムが17と全プログラムの約半分在り，このことは，同一計算の共有（過去の参照結果を記憶しておき，同一の参照は，重複計算を回避して，過去の計算結果を利用する）メカニズム[19]の検討の余地を示唆している。

headの平均引数個数は，約３なので，引数間のunification を並列化しても大きな効果は望めないであろう。

body側のひとつのAND リテラル当たりの平均引数個数は，約 2であり，これは，head述語当たりの平均引数個数の約 3に比べて小さい。これは，body側AND リテラルの約半分が組込み述語であり，組込み述語の引数個数の少なさに影響された結果である。

head述語引数の内の約20% が構造体データであり，又，body側引数の内の約10% が構造体データである。これは，静的な解析結果であり，静的にはvariableでも，実行時には，list data がinstantiate されることは往々にしてあり，動的には，この割合は増加するものと思われる。

2.2.2 inference clause内組込み述語の解析結果と特性

(1) 解析結果

DEC-10 Prolog には広範な組込み述語が用意されており，付録１に，付録３の33個のプログラムを解析して得られた組込み述語の使用頻度ベスト30を示す。

(2) 特性

“mode”と“public”（ 950回出現）を除いた組込み述語全体に対する主な実行機能別[18]組込み述語の割合と個数を示す。

[1] Input / Output	32.4%(1781)
[2] Meta-Logical	17.7%(977)
[3] Arithmetic	11.0%(608)
[4] Comparison of Terms	6.8%(372)
[5] Modification of the Program	5.5%(303)
[6] “＝”（使用頻度第１位）	16.5%(909)

（全組込み述語数（除くmode,public）：5505）

unification の機能を使用する，もしくは使用した方がその実現が容易な組込み述語は，全組込み述語の約41% となるので，unification を必要としない組込み述語の全AND リテラルに占める割合は約25% となる。よって，DEC-10 Prolog プログラム実行を高速化するには，unification を高速化するだけでなく，unification を必要としない処理の負荷を算定し，この処理が全体に占める負荷が軽くないならば，これをも高速化する必要がある。

2.2.3 database clause の解析結果と特性

inference clauseの静的解析に用いた33個のプログラムに，database clause のみからなる以下の 6つのプログラムを付加した。

- No.4のDCG プログラムに対する辞書database
- No.5のBUP プログラムに対する辞書，non-terminalの停止条件等のdatabase
- No.6の形態素解析プログラムの辞書database
- No.19 のプログラムの，fact名のdatabase
- 〃 rule名 〃
- 組込み述語database

新たに入れた 6つのdatabase clause のみからなるプログラムのavORは10.0で最大ORは 655とinference clauseのavOR 2.7，最大OR 155に比べて約４倍程大きく，database clause がさらに大規模化すれば，この倍率はもっと大きくなるであろう。よって，大規模database clause に関するunification の高速化がプログラム実行の高速化に大きな影響を及ぼす。（この６つのプログラムを含むdatabase clause を持つ29個のプログラムのavORは 8.7である。）一方，これら６つのプログラムの平均head述語引数個数は， 2.8となり（29個の場合の平均head述語引数個数は 2.6），inference clauseのそれとあまり変りない。

解析対象プログラムの中で，最大inference clause種類数は 142，また，最大 Total OR relation数 of inference clause は 498，最大database clause 種類数は 153，最大 Total OR relation数of database clauseは1023である．各プログラムは実際にはリンクして使われるが，この数字はPrologマシンの実験機を作成する時の１つの指針となるであろう．

2.2.4 静的解析時の制限

この静的アナライザでは，引数の内部構造は原則として解析されない。

例えば次のような４つのclauseからなるプログラムがあったとする。

```
bc(X,Y):-b__cw(X,Y).
lte(X,Y):-
  apply([tl,bc,tr,ld,tr,lu,tl],X,Y). ── ①
apply([],X,X).
apply([Op| Opseq],X,Y):-
  z=..[Op,X,X1],call(Z),!,apply(Opseq,X1,Y).
```

このプログラムを静的に解析した結果，述語bc（_ ,_）の参照数は 0になる。なぜならば，apply の定義の中まで追っていかないと，clause①内のbcの引数が２だということは分らない。本静的アナライザでは，head述語シンボルと引数個数が一致したとき，その述語が参照されていると判定するので，（head述語シンボルが等しく，引数個数の異なる述語は一つのプログラム内でともに存在することはありうる。）①からだけでは，bc(X,Y) が参照されているとは判定できない。よって静的な解析では，bc(X,Y) が参照されていることを検出することは難しい。

3．動的解析

静的解析に用いた39個のDEC-10 Prolog プログラムから２つのプログラムをえらび，それを並列実行可能なProlog（以下，略して並列Prolog）プログラムに書き換え，動的アナライザで解析し，静的解析結果と比較した。

以下に，並列Prolog，逐次型Prologプログラムから並列Prologプログラムへの書き換え規則，動的解析結果，静的解析結果との比較について述べる。

3.1 並列Prolog

ここで言う並列Prologとは，以下のようなものである。

(1) OR並列で実行する

OR relation にあるclauseは並列に処理される。

(2) AND 並列 "//" を指定できる

AND 関係にあるリテラル間に共有変数がなく，各々のリテラルは高々１個の解しか有さない ── 組込み述語やデータ・フローでいう非ストリーム化述語[20]の場合，AND 並列 "//" を指定することができる。

(3) body側のOR ";" ,cut "!" ,DB機能（assert, retract,…），集合機能（setof,…）は容認しない。

(4) 否定 "↘" の内部はフィルタリング実行される。

即ち，↘P の実行は， Pが１つでも成功した場合，↘P はその時点で失敗し， Pが全部失敗すれば，↘P は成功する。

次の例の場合逐次実行でも "↘" のフィルタリングを行う並列実行でも結果は同じであるが，"↘" のフィルタリングを行わない場合結果が異

なる。ゴールとしてp(X,Y)を与えると，前者は失敗するのに対し，後者はn回成功する。

```
（例）  p(X,Y):-↘q(X),r(Y).
        q(X):-s(X),t(X).
        r(b).        s(1).
        t(1).         ⋮        n+1個の並び
                     s(n+1).
      ?- p(X,Y).
```

3.2 逐次型Prologプログラムから並列Prologプログラムへの書き換え規則

(1) bodyのORの変換

body側にあるOR ';'は，clauseのOR relation に変換する。

(例1)

```
p(X,Y):-( g(X,Z);f(X,Z) ),r(Z,W).
          ⇓
p(X,Y):-g(X,Z) ,r(Z,W).
p(X,Y):-f(X,Z) ,r(Z,W).
```

(例2)

```
p(X,Y):-g(X,Z),
        (r1(Z,W),!,s1(W,Y);
         r2(Z,W),!,s2(W,Y)).
        ⇓
p(X,Y):-g(X,Z) ,body_p(Z,Y).
body_p(Z,Y):- r1(Z,W),!,s1(W,Y).
body_p(Z,Y):- r2(Z,W),!,s2(W,Y).
```

(2) cut "!" の削除

①if…then…else型の"!" の左で条件部分のリテラルが高々1個の成功解しか有さない場合，条件部分の否定を順次追加していくことにより"!" を削除する。特に条件部分が，排他的である場合は，単に"!" を削除する。

```
（例）p(X,Y):-g1(X),!,r1(X,Y).
      p(X,Y):-g2(X),!,r2(X,Y).
        ⋮
      p(X,Y):-gn(X),!,rn(X,Y).
              ⇓
      p(X,Y):-g1(X),r1(X,Y).
      p(X,Y):-↘g1(X)//g2(X),r2(X,Y).
              ⋮
      p(X,Y):-↘g1(X)// ↘g2(X)// ……//
              ↘ gn-1(X)  //gn(X),rn(X,Y).
```

②ただしif…then…else型の"!" の左でhead引数のタイプで場合分けをしている時には，body側で引数のタイプ・チェックを行うよう書き換えた後，①を適用する。

タイプ・チェックが必要な引数Aを変数Vに換えて，以下の規則にのっとった書き換えを行なう。但し，追加はbodyの先頭に行い，最後のclauseはbodyの先頭に"!" があるものとみなす。（例参照）

i) OR relation にある全てのclauseのbody側にvar(V) というリテラルが1つもない場合，各clauseのコピーをそのclauseの前に作成し，その一方にvar(V),V= Aを追加し，他方に，iii) ～vii) の書き換え規則を適用する。

ii) body側にvar(V) というリテラルがある場合は，そのままでよい。

iii) Aは変数であるがbody側にvar(V) というリテラルがない場合，nonvar(V)を追加する。

iv) Aが[]を除くatomicデータの場合，V==Aを追加する。

v) Aが[]である場合，is_nil(V) を追加する。

vi) Aがlistである場合，is_list(V), V= Aを追加する。

vii) Aがlistやatomicではないfunctor の場合，is_functor(V),V= Aを追加する。

ここで，is_nil,is_functor,is_listは，

```
is_nil(X):-    X==[].
is_list(X):-   nonvar(X) , X =[_|_].
is_functor(X):-nonvar(X)// ↘atomic(X)//
               ↘is_list(X).
```

で定義でき，プログラムを修正する時，常にこの3つをinference clauseとして追加する方式も考えられるが，それよりも，組込み述語とした方が良いと思われる（後述する）。

```
(例1)                          ゴールp(A,Y)を
                                  与えた時の解
  p(X,v):-var(X),!.              p(A,v)
  p(a,b).
誤った書き換え
  p(X,v):-var(X).                p(A,v)
  p(a,b).                        p(a,b)
正しい書き換え
  p(V,v):-var(V).                p(A,v)
  p(V,b):-V==a.
```

(例2)

```
f(a,Y):- !,m(Y).
f(b,Y):- !,n(Y).
      ⇓                    ── ②を適用
f(V,Y):-var(V),V=a,!,m(Y).
f(V,Y):-V==a      ,!,m(Y).
f(V,Y):-var(V),V=b,!,n(Y).
f(V,Y):-V==b      ,!,n(Y).
      ⇓                    ── ①を適用
f(V,Y):-var(V),V=a,m(Y).
f(V,Y):-↝( var(V),V=a ) ,
        V==a,m(Y).
f(V,Y):-↝( var(V),V=a ) ,
        ↝  V==a ,
        var(V),V=b,n(Y).
f(V,Y):-↝( var(V),V=a ) ,
        ↝  V==a ,
        ↝( var(V),V=b,n(Y)  ),
        V==b,n(Y).
      ⇓                    ── 人手で
f(a,Y):-     m(Y).           冗長性削除
f(V,Y):-V==b,n(Y).
```

(例3)

```
p(X,Y):-var(X),!,q(Y).
p(a,Z):-      !,r(Z).
p([],Z):-     !,t([],Z).
p([A|B],Z):- !,u(A,B,Z).
              ⇓            ···· ②を適用
p(X,Y):-var(X),!,q(Y).
p(X,Z):-X==a,!,r(Z).
p(X,Z):-is_nil(X),!, t([],Z).
p(X,Z):-is_list(X),!,X=[A|B],u(A,B,Z).
              ⇓            ── ①を適用
p(X,Y):-var(X),q(Y).       ( 排他的である
p(X,Z):-X==a,r(Z).           から単に削除)
p(X,Z):-is_nil(X), t([],Z).
p(X,Z):-is_list(X),X = [A|B],u(A,B,Z).
```

(3) 書き換えがうまくいかない例

①本質的にassert/retractが使われている場合（しかしgensymのように書き換え可能な場合もある。）
②述次制御の必要なメタ述語や，それを使用して記述されている場合(production system等)
③書き換え規則(2) 以外のcut "!" の削除

(例1)cut を削除すると，逐次実行時に得られる解以外の解が返ってくる場合（ゴールはq(X)）

```
q(X):-r(X),!,s(X).
r(a).   r(b).   r(c).
s(a).   s(b).              q(a)
      ⇓
q(X):-r(X),s(X).
r(a).   r(b).   r(c).      q(a)
s(a).   s(b).              q(b)
```

(例2)cut と↝との組合せの場合（ゴールはp(X)）

```
  (factは、r(a).   r(b).   s(b). )
p(X):-↝q(X).
q(X):-r(X),!,s(X).        success
      ⇓
p(X):-↝q(X).
q(X):- r(X),s(X).         fail
```

ただし，factを次のように書き換えれば，このruleに対してのみは，並列実行可能となる。

```
r(a).                  s(b).
r(B):-B==b.
```

3.3 並列Prologプログラムの動的解析の結果

前述した書き換え規則によって，テスト・プログラムのうち，

『論理式の簡単化プログラム』
『形態素解析プログラム』

を書き換え，動的アナライザによって解析した。

3.3.1 動的アナライザ

本アナライザーでは，解析対象プログラムにゴールを与える。そして，そのゴールが実行される過程で，各種データを収集する。その意味で動的である。

(1) 本アナライザーでは，実時間の代りにレベルという考えを導入している。1レベルは，unificationが実行され，次のゴール（AND リテラルに対応するSubgoal 列から形成される）が生成されるまでをいう。後述するAND 並列operator，AND 逐次operatorにより指示されるSubgoal が次のunification の対象となる。変数に関する結合は時間遅延なしに親ゴールに転送される。

(2) reducible なリテラル(Subgoal) は， AND並列operator' // '， AND逐次operator' , ' により指示される。本アナライザーでreducible なSubgoal

全てを同時に（１レベルで）unify する。

(3) LDB clauseを指定できる。

database clause の中で特にOR relation の大きいものをLarge database (LDB) clause と指定することができ，これに関するデータを収集することができる。

(4) レベル毎に静的解析と同様のデータを収集する他，Subgoal のOR並列 unificationの成功個数 (unification success数) が収集される。

(5) 動的アナライザーが読み込めるようにDCG 表記 "-->" を通常の表記":-"に書き換え，callの省略はしない。

(例1)

```
p(A,B)--> q(A),r(B).
        ⇓
p(A,B,S0,S) :-q(A,S0,S1),r(B,S1,S).
```

(例2)

```
p(A,B):-X =.. [A | B] , X.
        ⇓
p(A,B):-X =.. [A | B] ,call(X).
```

3.3.2 解析対象プログラム

(1) 形態素解析プログラム

内部に辞書をもち（LDB clauseとして登録），AND並列にできる部分があまり無いプログラムである。

以下，並列実行可能に書き換えた形態素解析プログラムをP-KEITAIと，書き換える前のプログラムをKEITAIと呼ぶ。

入力データの個数は 5個で，語幹が１意であるものである。

(2) 論理式の簡単化プログラム

"|" が多用されて，divide and conquer 方式のアルゴリズムのため AND並列に実行できる部分が比較的あるプログラムである。

以下，並列実行可能に書き換えた論理式の簡単化プログラムをP-LOGIC と，書き換える前のプログラムをLOGIC と呼ぶ。

入力データの個数は５個で，それらを，AND 逐次及び，AND 並列に実行した。

3.3.3 静的解析結果との比較

(1) Inference clauseに関して

	静的 av-OR rel 数	動的 av-OR rel 数	動的 av-US 数
KEITAI	2.20	——	——
P-KEITAI	2.06	7.13	1.14
LOGIC	3.68	——	——
P-LOGIC	3.49	4.62	3.57

(av-US 数：Subgoal の平均unification success 数
av-OR rel 数：平均OR relation 数)

P-KEITAIプログラムの実行時においては，全inference clauseの参照回数の52% を占める述語のOR relation数が12であるため，動的のav-OR rel 数が静的av-OR rel 数の 2.1から 7.1と増加している。PIM においては，このように，良く参照されるclauseが１つのProccessing Moduleにしか存在しないと，このProcessing Module に参照が集中し，Network のbottleneckとなるであろう。

並列実行可能な論理式の簡単化プログラムでは，av-US数／av-OR rel 数＝ 0.78 であるが，body側のリテラルのunification で大部分failを起こす。

以上のことから，次のことが考察される。

①1 clause当たりの平均cut 数が１に近くても（形態素解析プログラムは 0.50 ，論理式の簡単化プログラムは 0.77)，前述の書き換え規則により並列実行可能なように，書きなおすことが可能な場合がある。

②実行時のOR relation 数（動的OR relation 数）は，実行時に，良く参照されるclauseのOR relation数に依存する。

(2) 構造データについて

	静的 inf clの H-SD/Arg	静的 inf cl B-SD/Arg	動的 DB cl の SD/Arg	動的 Subgoal の SD/Arg
KEITAI	0.22	0.02	0.30	——
P-KEITAI	0.19	0.03	0.30	0.32
LOGIC	0.34	0.12	0.17	——
P-LOGIC	0.28	0.20	0.0	0.50

(inf cl : inference clause
DB cl : database clause
H-SD/Arg : head 述語の平均構造データ引数比
B-SD/Arg : body 側の平均構造データ引数比
SD/Arg : 平均構造データ引数比)

実行時には，実行されるSubgoal の引数に占める構造データの割合は，inference clauseのhead側引数，body側引数あるいはdatabase clause の静的な構造データ引数比に比較して増加する傾向にある。

(3) 組込み述語について

	静的 EV/AND	動的 EV/AND	動的 EV/Subgoal
KEITAI	0.59	——	——
P-KEITAI	0.49	0.21	0.40
LOGIC	0.27	——	——
P-LOGIC	0.51	0.65	0.77

(EV/AND : ANDリテラルに占める組込み述語の比
動的EV/Subgoal： 実行されるSubgoal に占める組込み述語の比)

P-KEITAIにおける，組込み述語の各分類項目別頻度

分類項目	静的頻度(%)	動的頻度(%)
[1] I/O	48	7
[2] Arithmatic	3	26
[3] Comprision terms	13	6
[4] Convenience	6	30
[5] Extra Control	13	11
[7] Meta-Logical	10	21
[14]Environment	6	1

P-LOGIC における，組込み述語の各分類項目別頻度

分類項目	静的頻度(%)	動的頻度(%)
[2]	0.8	0
[3]	6	4
[4]	10	9
[5]	38	19
[7]	33	36
[15]	12	32

([15] とは，is_list,is _functor,is_nil)

前表より，以下のことが考察される。

①実行されるリテラルに占める組込み述語の割合は，動的解析時でも，やはり高く，組込み述語の実行速度が，プログラムの実行速度に影響を及ぼす。

②形態素解析プログラムでは，実行時には，I/O 関連の組込み述語の出現頻度が静的解析時のそれに比べて低くなっている。一般に，I/O 関連の組込み述語の実行時の出現頻度は，静的解析時のそれと比べて低下するであろう。又，論理式の簡単化プログラムでは，実行時には，is_functor,is_list, is_nil の出現の割合が全体の約30% になっており（よって，全実行Subgoal の約24%），これらを組込み述語にし，高速実行をはかるべきことを示唆している。

(4) Large database clause (LDB clause)へのアクセスについて

	LDB clauseの 静的OR rel数	LDB clauseの 動的OR rel数
KEITAI	140	——
P-KEITAI	140	191

動的にも，LDB clauseのOR relation 数は高く，database clause 特に，LDB clauseに関する unification の高速化が，プログラム実行の高速化に影響を及ぼす。

3.3.4 動的解析結果

(1) 並列度

inference clause，組込み述語，database clause，LDB clauseを一緒にした，実行時における平均の並列度（レベル当りの平均reducible Subgoal 数のことで，この場合は AND逐次実行なので，OR並列度）は次の通り。

形態素解析プログラム	8.3
論理式の簡単化プログラム	3.2

ちなみに4Queens プログラムのOR並列度（動的アナライザ使用）は 6.2である。

(2) レベル当たりの最大Subgoal 数

P-KEITAI	レベル数	最大Subgoal 数／レベル
入力1	99	78
入力2	121	63
入力3	186	68
入力4	191	61
入力5	372	83
平均	——	70.6

P-LOGIC	レベル数	最大Subgoal 数／レベル
入力1	116	11
入力2	203	14
入力3	399	11
入力4	617	14
入力5	1563	18
平均	——	13.6

最大のSubgoal 数／レベルは，入力ゴールにそれほど依存しない。

(3) AND 並列実行

論理式の簡単化プログラムをAND 並列に実行し，動的解析を行った。

	OR並列度 (AND逐次，OR並列)	AND-OR並列度 (AND並列，OR並列)
入力1	2.66	3.44
入力2	3.00	4.84
入力3	3.05	7.34
入力4	3.19	8.78
入力5	3.97	14.44

前表のように，AND 逐次実行に比べ，大きく変化したのは，レベル当たりの平均reducible Subgoal 数である。入力ゴールは，入力番号が増えるにしたがって，より複雑な論理式となっている。このように，AND 並列実行による処理の高速化が期待できるプログラムが存在する。

4. おわりに

DEC-10 Prolog プログラムを静的，動的に解析し各種データを収集した結果，

①DEC-10 Prolog は逐次実行型 Prolog であり，メモリ空間が十分でない。プログラマは，そのことを意識してプログラムを記述している。そのためDEC-10 Prolog プログラムは deterministicになる傾向（1に近いcut 数）がある。しかし書き換え規則により，cut を取り，並列実行可能なように書き換えることが可能な場合がある。書き換えて，OR並列実行した時に，静的なOR relation 数と同等あるいは，それ以上のOR relation 数が期待できる。又，AND 並列に実行（但し，AND リテラル間共有変数の解は1つ）することにより，より高い並列度が得られる場合がある。

②実行されるSubgoal の約半分あるいは，それ以上が組込み述語であり，組込み述語の実行速度がプログラム実行速度に影響を及ぼす。

③database clause の静的OR relation 数はinference clauseの静的OR relation 数（約3）の約4倍であり，database clause が大規模化するとこの比は増大する。（動的解析でも，LDB clauseに関しては，これは強まる傾向にある。）よって大規模database clause を含むプログラムの場合，それに関するunification の高速化がプログラム実行の高速化に大きな影響を及ぼす。

等が，考察された。これらは， Prolog マシンを設計する際の一助となるであろう。

最後に，日頃ご指導ご鞭撻をいただく村上国男第一研究室長，解析対象プログラムを提供し，議論して下さったICOT研究員の方々，およびE.Y.Shapiro 氏(Weizmann Institute of Science, Israel)に深謝する。

－参考文献－

[1] 平川秀樹，三吉秀夫，他，“論理プログラミングによるボトムアップパーサ(BUP) トランスレータの開発”，情報処理第26回全国大会．

[2] 三吉秀夫，他，“Prologによる日本語文脈DCG の生成”，情報処理第27回全国大会．

[3] 松本裕治，他，“Prologに埋め込まれたボトムアップパーサ：BUP ”，Logic Programming Conference '83,Tokyo.

[4] 安川秀樹，“LFG in Prolog －Toward a formal system for representing grammatical relations －”，ICOT TR-019 (1983.8) ．

[5] 向井国昭，他，“Prologによる線形順序戦略定理証明プログラム”，情報処理第27回全国大会．

[6] 坂井公，宮地泰造，“Prologへの否定的知識の導入とプログラム検証”，ICOT TR-007 (1983.5).

[7] 古川康一，“Prologによる問題解決”，情報処理第23回全国大会．

[8] 古川康一，近藤浩臣，“Two Dimensional Programming in Prolog ”，Logic Programming Conference '83, Tokyo.

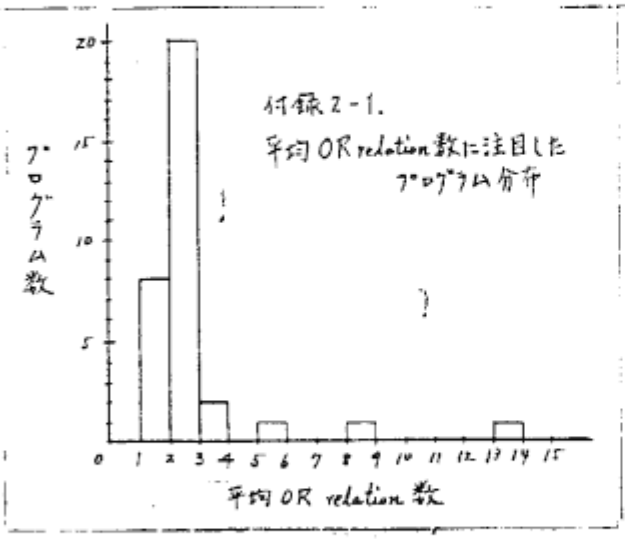
[9] 関藤進，他，“Prologによる対象知識とメタ知識の融合とその応用”，ICOT TR-009 (1983.5).

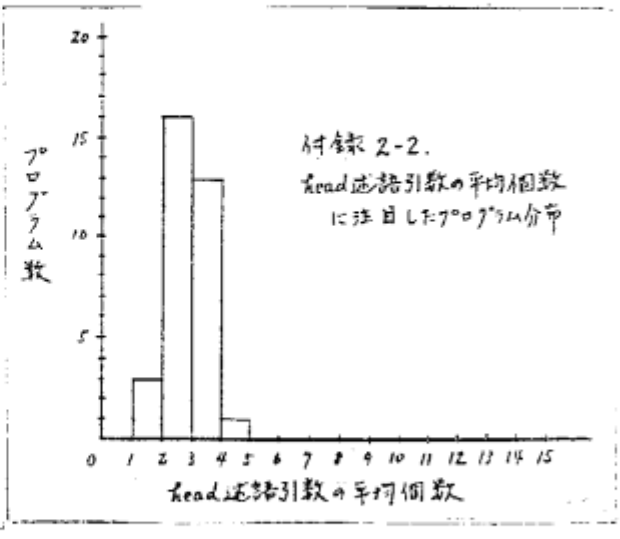
[10] 宮地泰造，他，“論理データベース向きの知識同化方式の一提案”，ICOT TM-0004 (1983.4)
[11] 横田治夫，他，“RDBM Delta (Ⅲ) －コマンド体系について－”，情報処理第26回全国大会．
[12] 横田治夫，他，“An Enhanced Inference Mechanism for Generating Relational Algebra Queries”，ICOT TR-026 (1983.10)．
[13] 古川康一，竹内彰一，“論理型言語におけるtype inference とその type check への応用”，情報処理第26回全国大会．
[14] 近山隆，“パーソナル逐次型マシンψ－その言語システム－”情報処理第27回全国大会．
[15] T.Chikayama, “ESP －Extended Self-contained PROLOG－as a Preliminarg Kernel Language of Fifth Generation Computers”, New Generating Computing,1 (1983).
[16] 高木茂行，“汎用型マイクロプログラム・アセンブラ”，ICOT TR-021 (1983.8)．
[17] E.Y.Shapiro，“A subset of Concurrent Prolog and its Interpreter”，ICOT TR-003 (1983.1)．
[18] D.L.Bowen, “DEC system-10 Prolog User's Manual”，Dept of Artificial Intelligence, University of Edinburgh (1981)．
[19] 平川秀樹，尾内理紀夫，古川康一，“POPS:OR-Parallel Optimizing Prolog System”，情報処理第27回全国大会．
[20] 益田，伊藤，清水，“データフロー方式Prologマシンのシミュレーションによる評価”，同上

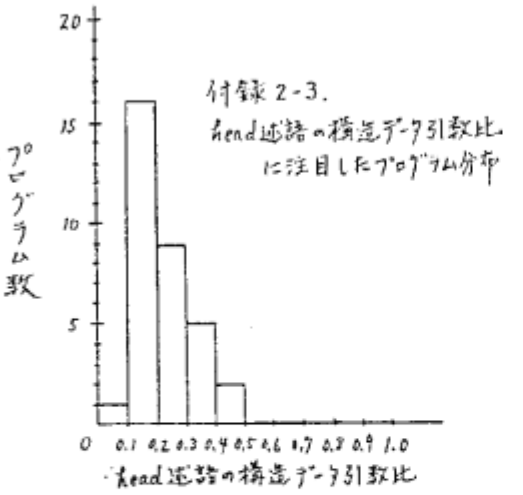
付録 1.　組込み述語の静的出現回数

No	組込み述語	出現回数
1	=	909
2	mode	719
3	nl	487
4	display	428
5	call	354
6	= =	343
7	write	293
8	is	244
9	public	230
10	= . .	222
11	+	154
12	fail	136
13	op	132
14	tab	122
15	ttynl	102
16	var	97
17	arg : 3	85
18	abolish	82
19	asserta : 1	79
20	- : 2	78
21	functor	71
22	name	67
23	true	64
24	put	62
25	retract	59
25	assert : 1	59
27	ttyflush	56
28	tell	52
29	atomic	38
30	>	35

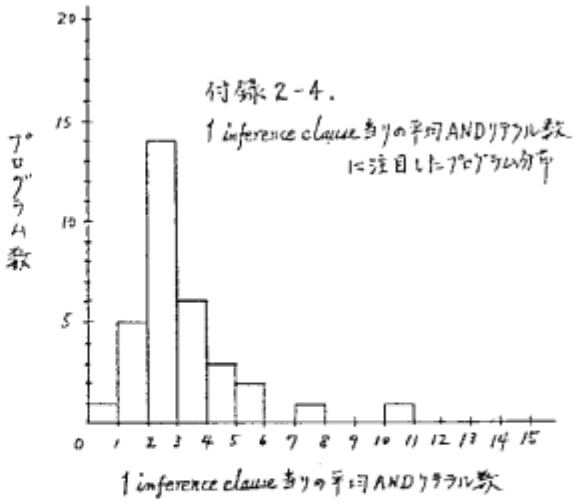
付録 2.



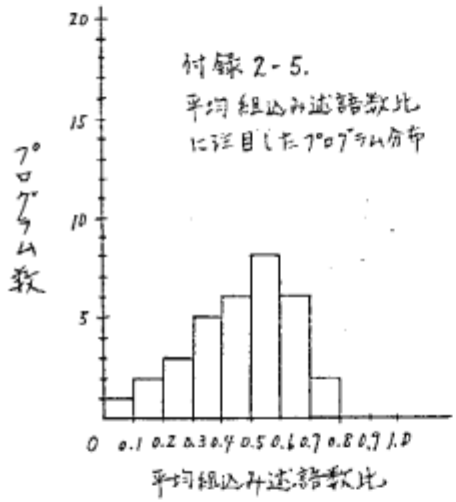




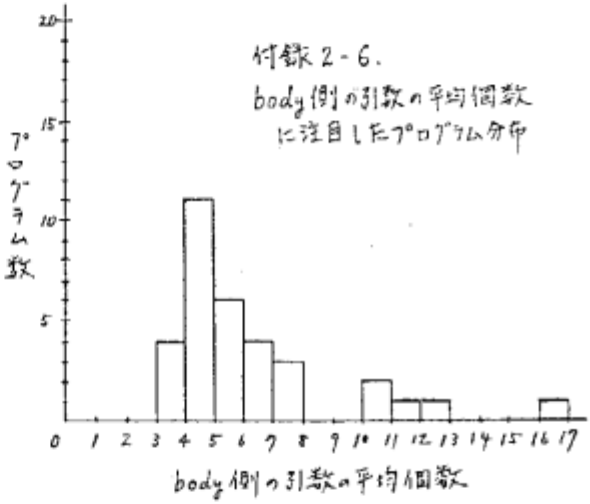
付録 2-3.
head述語の構造データ引数比
に注目したプログラム分布



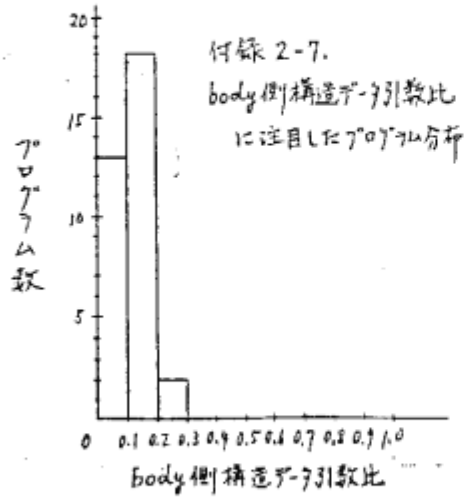
付録 2-4.
1 inference clause 当りの平均ANDリテラル数
に注目したプログラム分布



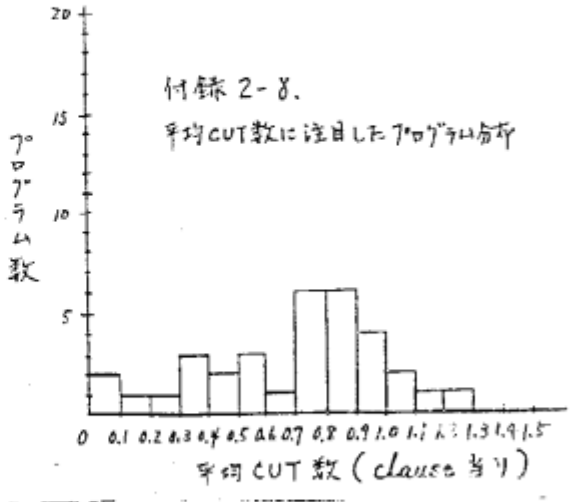
付録 2-5.
平均組込み述語数比
に注目したプログラム分布



付録 2-6.
body側の引数の平均個数
に注目したプログラム分布



付録 2-7.
body側構造データ引数比
に注目したプログラム分布



付録 2-8.
平均CUT数に注目したプログラム分布

付録3．解析対象プログラムの機能概要一覧

No	プログラム名	参考文献	概　要
1 2	BUP トランスレータ BUP トランスレータのユーティリティ	[1],[3]	文法規則の入った DCGファイルを入力とし，BUP 形式のProlog節に変換するプログラム
3	BUP トレーサ	[1],[3]	BUP プログラムのトレーサ
4	文法記述(DCG)	[2]	DCG(Definite Clause Grammar) の記法で書かれた文法規則
5	文法記述(BUP)	[1],[3]	BUP トランスレータにより生成されたBUP 形式の文法規則
6	形態素解析プログラム	[2],[3]	日本語文節の形態素解析（語尾処理，自動分かち書き）を行うプログラム
7 8	LFG システム LFG システムのユーティリティ	[4]	BUP システムを用いて自然言語文の構文解析と，文の機能構造の抽出を行うプログラム (LFG:Lexical Functional Grammar)
9	述語論理式簡単化プログラム		与えられた一階述語論理式を積和標準形に変換するプログラム
10	線形順序戦略定理証明プログラム	[5]	一階述語論理の完全な定理証明プログラムである．証明方法はOrdered Linear法と呼ばれるものである．証明の探索法は下降型混合探索である．
11	無限木Prologインタプリタ		infinite tree も扱うPrologインタプリタ
12	Epilogインタプリタ		Epilogのインタプリタ
13	depth-first PPN インタプリタ	[6]	Prologに普通の意味での否定（cut を使って定義するnegation as failure ではない）を導入した言語(PPN) の逐次型インタプリタ
14	ルービック・キューブ	[7]	simple production systemを用いたルービック・キューブの解法を求めるプログラム
15	ペントミノプログラム	[8]	「ペントミノ」と呼ばれるゲームの解をすべて求めるプログラム
16	知識同化プログラム	[9]	メタ推論用demo述語で実現した、例外値を含む知識同化プログラム
17	知識獲得プログラム	[10]	演繹論理志向の知識の同化を，オブジェクト言語とメタ言語の融合によって定式化した、知識獲得方式のデモプログラム
18	SEQUEL- 関係代数インタプリタ	[11]	関係データベース問合せ言語SEQUELの問合せを関係代数ベースのコマンド列に落とし，関係代数コマンド実行シミュレータにより求める関係を得る
19	Prolog−関係データベース・ インタフェース　プログラム		Prologプログラムのfactの大部分を関係データベースに格納するとした時のPrologと関係データベースのインタフェースを取るプログラム
20	タイプ推論システム	[13]	Prologプログラム中の各述語の各引数のデータ構造を推論するプログラム
21 22 23	ESP クロスシステムのユーティリティ プリプロセッサ・プログラム ランタイム・サポート	[14]， [15]	ESP プログラムを読んでDec-10 Prolog のプログラムに翻訳する pre-compiler.
24 25 26	汎用型アセンブラのメインプログラム 汎用型アセンブラのサブルーチン 汎用型アセンブラの定義プログラム	[16]	アセンブラ・ソースのプログラムの読込み，及び制御を行なうプログラム
27	分岐アドレス編集プログラム		PSI 用μアセンブラの生成する分岐方法を入力し，PSI ハードウェアの内蔵しているμ命令の多方向分岐制御用メモリの内容を生成するプログラム
28	演算子順位構文解析プログラム		演算子順位に従って構文解析を行ない，構文木を出力するプログラム
29	PLA 圧縮プログラム		1ビットデコード方式のPLA の簡単化を行うプログラム
30	DEC-10 Prolog プログラム・ スタティック・アナライザ		DEC-10 Prolog プログラムを静的に解析するプログラム
31	AND-OR探索木特性アナライザ		解析対象プログラムとgoalを入力とし，AND-OR Tree をANDは逐次に，ORは並列に解析するプログラム
32	Concurrent-Prolog アナライザ		Concurrent-Prolog プログラムと，goalを入力とし，sub-goalの状態、および状態に関する統計情報を出力するアナライザ
33	Concurrent-Prolog インタプリタ	[17]	Concurrent-Prolog のインタプリタ