ソフトウェア工学 111－1
（1996．9．11）

Prologプログラムの属性グラフ文法に基づいた可視化

安達 由洋 †，　今木 孝哲 †

† 東洋大学　〒350 埼玉県川越市鯨井2100

Prologは，ユニフィケーションやバックトラックなどのメカニズムによりプログラムを簡潔にかつ柔軟に記述できる強力なプログラミング言語である．しかしながら，CやPASCALなどの手続き型言語にはないこれらの機能がProlog言語の学習を難しくしている．また，大規模で複雑なプログラムを理解することは非常に難しく，現在のProlog処理系が備えているデバッグツールでは，全体構造の把握や実行過程の解析には機能が不十分である．

そこで本論文では，Prologプログラムを木構造図として表示し，このプログラム図上で実行過程を可視化する研究について報告する．まず，Prologの構文規則をもとにプログラム図を生成する属性グラフ文法を定義する．次に，この文法に基づいてプログラム図をX Window上に描画するシステムを実現する．これにより，本システムが任意の正しいプログラムを表示できること(完全性)と表示したプログラム図が正しいこと(健全性)を保証できる．

このシステムはプログラムの実行過程をリアルタイムにプログラム図上に表示する機能を持つので，プログラム動作の解析やデバッグの道具として役立つ．

キーワード：Prolog，視覚化，属性グラフ文法，デバッグ，プログラム図

Prolog Visualization Based on Attribute Graph Grammar

Yoshihiro ADACHI †，　Takanori IMAKI †

† Faculty of Engineering, Toyo University, 2100, Kujirai, Kawagoe, Saitama, 350, Japan

We describe a method to draw a Prolog program as a tree-structured program diagram,and our system which visualizes Prolog program execution by using the diagram.

First we define an attribute graph grammar corresponding to a Prolog syntax, and then implement a system which displays a Prolog program diagram on X Window based on the grammar. Therefore, it is guaranteed that the system can draw a diagram of any correct Prolog program(completeness)and any program diagram displayed by the system is always correct(soundness).

Because it can also display realtime process of Prolog execution, our system is very useful as a Prolog programming and debugging tool.

key words：Prolog, visualization, attribute graph grammar, debug, program diagram

1 はじめに

Prologは，ユニフィケーションやバックトラックなどのメカニズムによりプログラムを簡潔にかつ柔軟に記述できる強力なプログラミング言語である．しかしながら，CやPASCALなどの手続き型言語にはないこれらの機能が，Prolog言語の学習や，大規模で複雑なプログラムの実行過程の解析，プログラム内容の理解などを難しくしている．また，現在のProlog処理系が備えているデバッグツールは一般にボックスモデルに基づくテキストベースのトレース機能が主で，プログラムの全体構造の把握や実行過程の解析を支援する道具として機能が不十分である．

そこで我々は，Prologプログラムを木構造図として表示し，このプログラム図上で実行過程を可視化するための研究を行っている．

ところで，プログラム図表示システムを実現するとき，そのプログラム言語を定義した文法に基づいて任意の正しいプログラムが表示できること（完全性）と表示したプログラム図が正しいこと（健全性）を保証できることが望ましい．また，プログラム図をディスプレイ上に表示するためには図のレイアウトが不可欠である[1]．これらの観点から，まずPrologの構文規則をもとにプログラム図の生成を形式的に定義するプロダクションとレイアウト情報を属性として計算する意味規則からなるPrologプログラム図の属性グラフ文法を定義する．

次に，この文法に基づいてユーザが指定したゴールに対するPrologプログラム図をX Window上に描画するProlog可視化システムを実現する．このシステムは，Prologプログラムからプログラム図の内部表現へ変換するトランスレータと，この内部表現をもとにX Window上に描画するビューアにより成っている．さらに，このシステムはプログラムの実行過程をリアルタイムにプログラム図上に表示する機能も持っており，プログラムの動作の解析やデバッグを容易にする．

2 Prologプログラム図の属性グラフ文法

まず，Prologの可視化システムの理論的基礎となる文脈自由グラフ文法，属性グラフ文法を定義する．これらの詳細は文献[2][3]を参照されたい．そして次に，Prologプログラム図の生成規則について述べる．

2.1 グラフ

Σ（ノードアルファベット）上のグラフとは3項組$H=(V,E,\varphi)$である．ただし，

1. Vはノードの空でない集合
2. $E \subseteq V \times V$はエッジの集合
3. $\varphi : V \rightarrow \Sigma$はノード・ラベリング関数という．$v \in V, x \in \Sigma$で$x=\varphi(v)$となるとき，$v$は$x$とラベル付けされたといい，$x$を$v$のラベルという．

2.2 文脈自由グラフ文法

文脈自由グラフ文法は，5つ組$GG=(\Sigma_N,\Sigma_T, S, D_0, P)$で表される．ただし，

1. Σ_Nは非終端ノードアルファベットである．
2. Σ_Tは終端ノードアルファベットである．
3. $S \in \Sigma_N$は開始ラベルである．
4. $D_0=(V_0,E_0,\varphi_0)$は開始グラフである．ただし，$V_0=v_0, E_0=\phi, \varphi_0(v_0)=S$である．
5. Pはプロダクションの集合である．

   なお，プロダクション$p \in P$は$p=(X,D,I,O)$であり，以下の条件を満たしている．

   (a) $X \in \Sigma_N$
   (b) $D=(V,E,\varphi)$は$\Sigma=\Sigma_N \sqcup \Sigma_T$上の連結グラフである．
   (c) $I \in V$を入力ノードという．
   (d) $O \in V$を出力ノードという．

プロダクション$p=(X,D,I,O)$は$X \rightarrow D^{I,O}$とも書く．

ここで，書き換えられる前のグラフをD_1，プロダクションpを$X \rightarrow D_2^{I,O}$とし，D_1にXとラベル付けされたノードxがあるとき以下の順序で新しいグラフが導出される．

1. Xとラベル付けされたノードxをD_2と同型なグラフで置き換え，グラフDとする．
2. もしxの入力ノードがあれば，Iに対応するノード$I' \in D$の入力ノードとする．
3. もしxの出力ノードがあれば，Oに対応するノード$O' \in D$の出力ノードとする．

このとき，D_1とDとの関係を$D_1 \Rightarrow D$と書き，D_1からDが直接導出されたという．

上記の操作を繰り返すことにより，Sとラベル付けされた1つのノードのみからなるグラフD_0(開始グラフ)から始めて，いくつかのプロダクションを適用してΣ_Tのみによってラベル付けされたグラフが導出される．

2.3 属性グラフ文法

属性グラフ文法は3つ組 $AGG = (GG_u, A, F)$ である．

1. $GG_u = (\Sigma_n, \Sigma_t, S, D_0, P)$ は，AGG の基底文脈自由グラフ文法である．また，$\Sigma = \Sigma_n \sqcup \Sigma_t$ 上のグラフ $D = (V, E, \varphi)$ について，$Lab(D) = \{\varphi(v) \mid v \in V\}$ とする．
2. GG_u の各ノードアルファベット $X \in \Sigma$ に対し，互いに素な2つの有限集合，継承属性の集合 $I(X)$ と合成属性の集合 $S(X)$ が付随している．X の属性全体の集合を $A(X) = I(X) \sqcup S(X)$ で表す．$A = \sqcup_{X \in \Sigma} A(X)$ を AGG の属性集合という．
3. P の各プロダクション $p = (X, D, I, O)$ に対し，$S(X) \sqcup_{X \in Lab(D)} I(X)$ の属性のみをすべて定義する意味規則の集合 F_p が付随している．集合 $F = \sqcup_{p \in P} F_p$ を AGG の意味規則集合という．

2.4 Prologプログラム図対応属性グラフ文法

Prologプログラム図を生成するために，その生成規則を属性グラフ文法を用いて定式化した．この文法は，Prologプログラム図生成の規則を文脈自由グラフ文法として定義し，また生成された図の配置情報を意味規則によって定義している．

Prologプログラム図の配置情報を導くための継承属性および合成属性の一部を以下に示す．

継承属性

- `x` ： 配置されるセルのX座標
- `RootX` ： ルート（一番左上の）セルのX座標
- `RootY` ： ルート（一番左上の）セルのY座標
- `MinW,MinH` ： セルの最小幅，最小の高さ
- `GapX,GapY` ： 隣接セルの間隔
- `id` ： セルの識別番号

合成属性

- `y` ： 配置されるセルのY座標
- `w,h` ： セルの幅，高さ
- `string` ： セル内部の文字列
- `width` ： 部分木内の最大X座標
- `bottom` ： 部分木内の最下辺のY座標
- `nc` ： セルの数
- `line` ： セルとセルを結ぶ制御線

また，この属性グラフ文法の終端アルファベットおよび非終端アルファベットの一部をFig.1に示す．

ここで，定式化した属性グラフ文法のプロダクションと意味規則について述べる．この属性グラフ文法は，Sicstus Prolog[4]およびIF/Prolog[5]の構文規則を参照して定式化した．定式化したプロダクションは17あり，その一部を示し説明する．

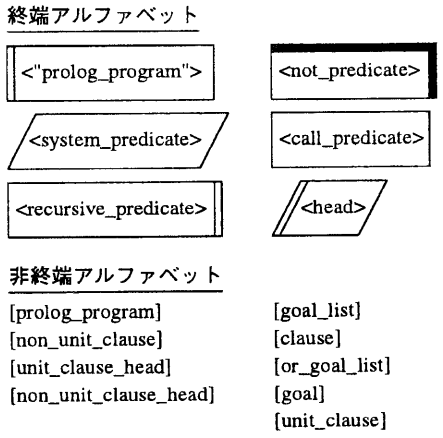

Fig. 1. 生成規則のアルファベット

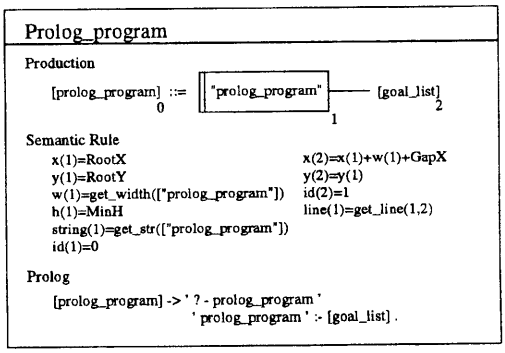

Fig. 2. [prolog_program]

◇ [*prolog_program*]

開始ノードアルファベットである[prolog_program]を定義する属性グラフ文法の生成規則および意味規則をFig.2に示す．

開始ノードアルファベット $[\text{prolog\_program}]_0$ が導出するルートセル（Fig.2の1のラベルをもつノードで，Prologプログラム図中の最も左上に位置するセル）のX座標はRootX(定数)であり，Y座標はRootY(定数)である．

ユーザが作成したプログラム内で可視化するゴールを必ず"prolog_program:-[goal_list]"という節を追加し呼び出している．Prologの文法では質問(query)に対応している．

◇ [*goal_list*]

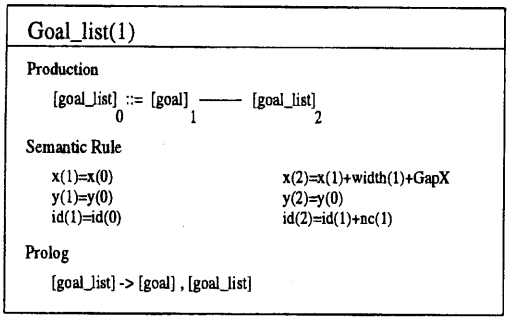

Fig. 3. [goal_list]

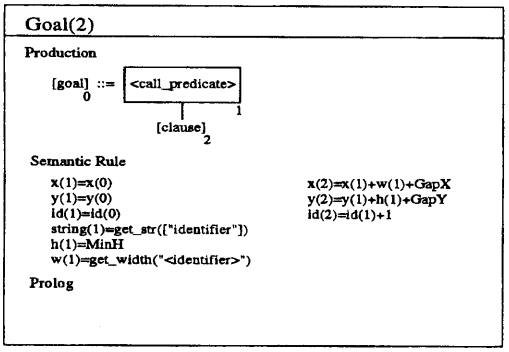

Fig. 4. [call_goal]

Fig.3 の文法は Prolog の AND に相当するもので，非終端アルファベット $[\text{goal\_list}]_0$ を制御線で接続されている $[\text{goal}]_1$ と $[\text{goal\_list}]_2$ に書き換えている．$[\text{goal}]_1$ と $[\text{goal\_list}]_2$ は親子の関係である．

◇ [*call_goal*]

Fig.4 の文法は Prolog の節の呼び出しに相当する部分である．Prolog の文法では，ゴールを詳細に分類せず一つのゴールとして定義されているが Prolog プログラムを木構造図で表示するために，節を呼び出す規則を加えた．呼び出しを表すセルの下に，呼び出しの述語と関係のあるプログラム構造 (部分木) が $[\text{clause}]_2$ の非終端アルファベットより導出され制御線で接続される．

◇ [*unit_clause_head*]

非終端アルファベットである [unit_clause_head] を定義する文法の一つを Fig.5 に示す．これは単位節の書き換え規則であり，同じ節が他にある場合 $[\text{clause}]_2$ の非終端アルファベットから導出される．本可視化システムでは，Prolog インタープリタと同様に "述語 (項):- true" も単位節とみなして表示している．

◇ [*not_goal*]

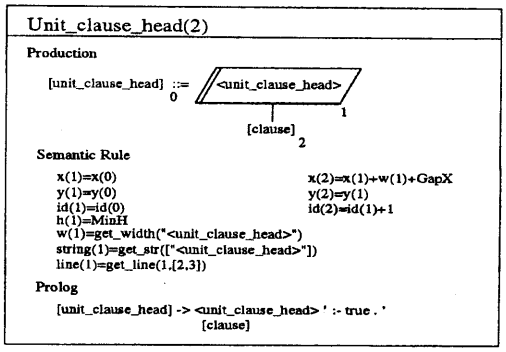

Fig. 5. [unit_clause_head]

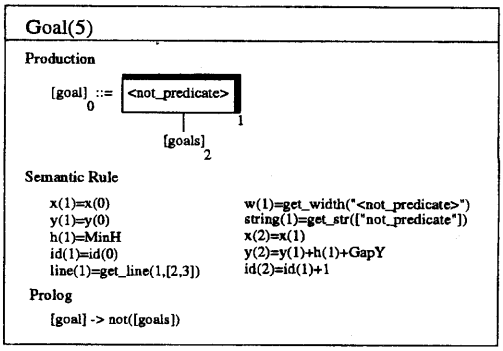

Fig. 6. [not_goal]

非終端アルファベットである [goal] を定義する文法の一つを Fig.6 に示す．この規則はオペレータの否定を表現したものである．否定のオペレータは，括弧内のゴールの結果の否定をとるものであり，ゴールが複数存在してもよい．そこで否定をとるゴールの構造を表示するために，否定のセルの下にある $[\text{goal\_list}]_2$ で否定をとるゴール全体の構造も木構造図で表示される．

◇ [*or_goal_list*]

非終端アルファベットである [or_goal_list] を定義する生成規則と意味規則を Fig.7 に示す．この規則はセミコロンを用いた OR を表す規則である．$[\text{goal\_list}]_1$ を始めに導出し，次に $[\text{or\_goal\_list}]_2$ を導出する．ここで $[\text{goal\_list}]_1$ の部分木の幅と $[\text{or\_goal\_list}]_2$ の部分木の幅の最大値を取り，その最大値にあわせて辺の形を整え接続線を閉じている．

◇ [*recursive_goal*]

非終端アルファベットである [goal] を定義する文法の一つを Fig.8 に示す．これは再帰呼び出しをするゴールを書き換える規則である．Prolog を木構造図で表示す

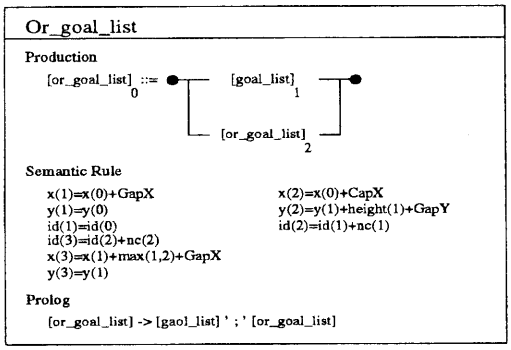

Fig. 7. [or_goal_list]

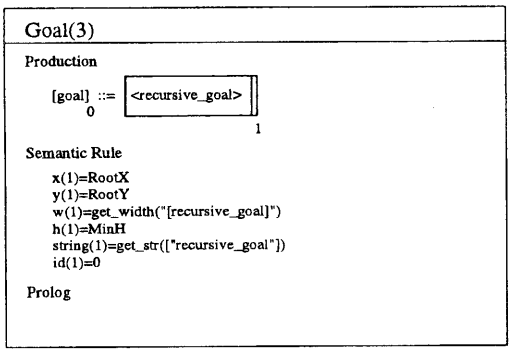

Fig. 8. [recursive_goal]

ると再帰ループの部分で無限の木構造図になる．そこで，再帰のゴールは再帰を示すセルで置き換えている．

以上で述べたPrologプログラム図表示のための文法のプロダクションおよび意味規則中の記号，関数は以下のように定義している．

- 開始記号 = [prolog_program]
- 〈 X 〉：文脈自由文法における非終端アルファベット X
- "X"：文脈自由文法における終端アルファベット X
- "●"：辺の形を整えるためのダミー接点
- GapX,GapY：セルとセルの最小幅，最小高
- RootX,RootY：ルートセルの X 座標，Y 座標
- max(X,Y)：X と Y の最大値を求める関数
- get_str(X)：セル内の文字列を求める関数
- get_width(X)：文字列のリストからセルの幅を求める関数
- get_line(S,E)：始点セル S，終点セル E を結ぶ線の始点と終点の座標を求める関数

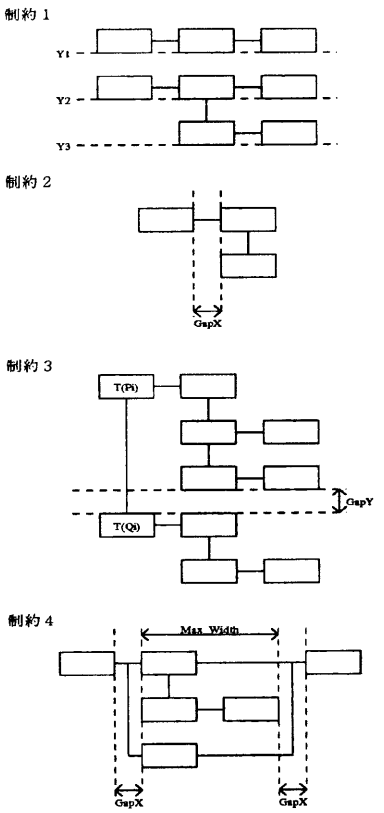

Fig. 9. 制約条件

意味規則は，プログラム図のセル配置に関する制約式を含んでいる．この制約式は，プログラム図全体の処理の流れが見やすくなり，しかも簡単であるという理由より以下の 4 つの条件に基づいている．

1. 同レベルのセルの Y 座標はすべて等しい．
2. Y 軸に関した接続線でつながれているセルは X 軸に関して GapX(定数) だけ離れている．
3. 部分木 $T(P_i)$ と部分木 $T(Q_i)(y(P_i)(y(Q_i))$ は Y 軸に関して GapY(定数) だけ離れている．
4. 1 本の接続線でつながれている隣り合うセル (部分木) は X 軸に関して GapX(定数) だけ離れている．

3 属性グラフ文法に基づく可視化システム

前節の文法に基づいて Prolog からプログラム図の内部表現に変換するトランスレータと，内部表現から対応するプログラム図を描画するビューアで Prolog 可視化システムは構成されている (Fig.10)．

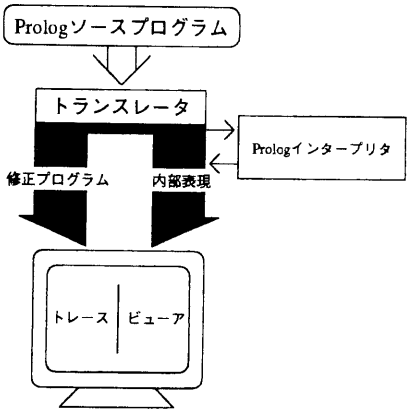

Fig. 10. Prolog 可視化システムの構成

なお，本システムは Prolog の構文規則と完全に対応するように属性グラフ文法を定義し，この定義に基づいて実現されているので任意の正しい Prolog プログラムを表示できる（完全性）ことと，表示したプログラム図は正しい（健全性）ことが保証される．

以下にそれぞれの処理系について説明する．

3.1 トランスレータ

トランスレータは，Prolog プログラムを入力としユーザが指定したゴールに対応する内部表現を生成する．ここでは Prolog プログラム図の内部表現と，その生成過程を説明する．

● Prolog プログラム図の内部表現

Prolog プログラム図の内部表現は，各々のセルを Prolog の項として以下のように表現した．

```
prolog_cell(ID,Text,
  link(Parent,Upper,Children,Lower),
    Type,Graphic).
```

各々の引数の意味は以下の通りである．

`ID`	:	セルの識別番号
`Text`	:	ソースプログラムの行番号
`link`	:	他セルとの接続関係
`Type`	:	セルの種類
`Graphic`	:	セル内の文字列やセルの座標

Prolog インタープリタは，変数を内部で無名変数に置き換えてしまうのでセル内の文字列に変数名を情報として残している．

セルの識別番号は，Prolog プログラム図の記号表現においてユニークであり，他のセルとの接続関係は，プログラム図の生成において不可欠な要素である．接続関係を示す属性 link は次の通りである．

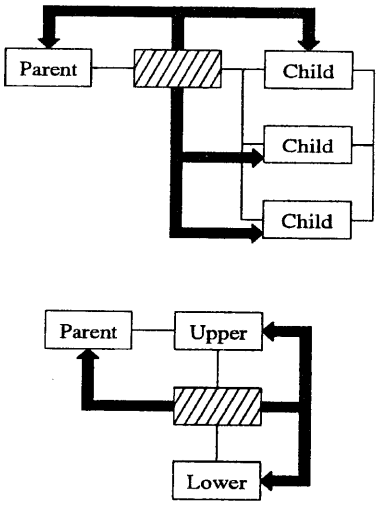

Fig. 11. セルの接続関係

`Parent`	:	親セルへのポインタ
`Children`	:	子セルへのポインタリスト
`Upper`	:	同じ親セルを持ち，自分より 1 つ上に位置するセル
`Lower`	:	同じ親セルを持ち，自分より 1 つ下に位置するセル

これらの 4 つの属性値はそれらが指すセルの ID である．セルに接続する各線は以下のように表現している．

```
connect_andLine(Cell1,Cell2,X1,Y1,X2,Y2).
connect_orLine(Cell1,Cell2,X1,Y1,X2,Y2).
connect_neckLine(Cell1,Cell2,X1,Y1,X2,Y2).
connect_callLine(Cell1,Cell2,X1,Y1,X2,Y2).
```

接続線は，X 軸方向に AND とネック記号，Y 軸方向に OR と述語の呼び出しとにわかれている．ここで，Cell1 と Cell2 は接続するセルの識別番号であり，X と Y は接続線の両端の座標を表している．

● 内部表現の生成手順

内部表現の生成処理は以下の手順で行われている．

1. 入力されたソースプログラムを一行ごとに抽出して，行番号を付加する．これにより，プログラム図のセルとソースプログラムを対応させる．
2. Prolog インタープリタで字句解析と構文解析をする．このとき，コメント文は削除される．
3. 指定したゴールに対して文法に基づき属性評価を行い内部表現を生成する．

本トランスレータは再帰ループのチェックを行っている．すなわち，内部表現を生成するとき今まで訪れたゴールの頭部情報をチェックして，再帰であれば再帰セルの内部表現を生成する．したがって，再帰プログラムでも無限ループには陥らない．

● メタプログラムの生成

トランスレータで内部表現を生成するのと同時に，トレースに必要なPrologプログラムをメタプログラミング技法を用いて生成している．生成されるメタプログラムの構成をFig.12に示す．

```
[prolog_program] :- prolog2D_startClause(Head,ID1,Parent),
                    [goal_list],
                    prolog2D_endClause(Head,ID1).
[goal_list] :- prolog2D_callGoal(Goal,ID2),
               [goal],
               prolog2D_exitGoal(Goal,ID2).
[goal_list] :- prolog2D_callGoal(Goal,ID2),
               [goal_list],
               prolog2D_exitGoal(Goal,ID2).
```

Fig. 12. メタプログラムの構成

Prologプログラムのゴールすべてにトレースに必要な述語を付加している．それぞれの述語は以下のとおりである．

prolog2D_startClause	: 節の始まり
prolog2D_endClause	: 節の終わり
prolog2D_callGoal	: ゴール呼び出し
prolog2D_exitGoal	: ゴール終了

トレースのアルゴリズムは，節のはじまりとゴール呼び出しで，セルを失敗の色で着色しソースプログラムのゴールが成功すると節の終わりとゴール終了で，そのセルを成功の色で着色している．

内部表現とメタプログラムを生成するシステムのプログラムをFig.13に示す．これは，Prologの頭部セルの内部表現を生成する部分である．

3.2 ビューア

トランスレータで生成した内部表現を入力し，プログラム図を描画する．

始めにPrologプログラムをPrologインタープリタに読み込み，Prologソースプログラム内の頭部の情報をすべて取り出すことで以下に示すポップアップウィンドウ上に選択できるゴールを表示する(Fig.14)．このとき，ゴールを選択して引数に値を与えると，それに対応したプログラム図が表示される．

```
generate_Prolog_goalTree_head(ID0,ID2,CallGoal,Line,Upper,Lower,
  X,Y,OldW,OldH,Width,Height,Visited) :-
    current_consultedClause((Goal :- _),VarList,File,Line),

/* 再帰ループのチェック */
    check_recursiveGoal_ID(File,LN1,RID,Visited),
    !,
    groundInstantiate_prologGoal(Goal,VarList,Grounded),
    transform_groundedGoal_Atom(Grounded,GoalLabel),
    get_prologCell_width_height(GoalLabel,LW,LH),
    ID1 is ID0 + 1,
    prolog2D_cell_gapY(GapY),
    max(OldW,LW,NewW),
    NewH is OldH + GapY + LH,
    Y1 is Y + LH + GapY,
    generate_Prolog_goalTree_head(ID1,ID2,CallGoal,Line,
      Upper,Lower,X,Y1,NewW,NewH,Width,Height,Visited),

/* セルの内部表現の生成 */
    asserta(prolog_cell(id(File,ID0),text(Line),link([],Upper,[],Lower),
      [call_type(recursive),recursive_goal(RID),calling_goal(CallGoal)],
        graphicData(GoalLabel,X,Y,LW,LH,_))),

/* メタプログラムの生成 */
    asserta(temporary_modifyingClause_for_trace(File,Line,
      (Goal :- prolog2D_startClause(Goal,id(File,ID0),CallGoal),
        prolog2D_recursiveCall(Goal,RID),
          prolog2D_endClause(Goal,id(File,ID0))))),
    HeadBottom is Y + LH,
    prolog2D_default_cellSize(SizeX,_),
    HeadXCenter is X + (SizeX - 1) // 2,

/* 制御線の生成 */
    assertz(connect_orLine(ID0,ID1,HeadXCenter,HeadBottom,
      HeadXCenter,Y1)).
```

Fig. 13. プログラム

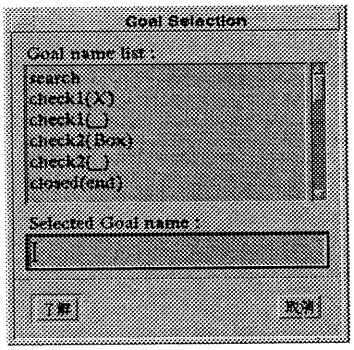

Fig. 14. Prologのゴール選択

Prologプログラムの AND/OR の木構造をプログラム図では，X軸方向を AND，Y軸方向を OR に対応させて表示している．表示画面をFig.15に示す．

次に，ビューアの機能について説明する．

● トレース機能

メタプログラムにより生成したトレース用プログラムをで実行することで，ビューア上にゴールの実行過程をリアルタイムに表示することができる．Prologは，ゴールが真であるか偽であるかを判断しながら実行が進んでいく．そこで，最初はセルの色は白であるが実行を行ったゴールに対応するセルに対して，その結果が真であるとき青色に，偽であるとき黄色に変化し，実行していないゴールは色の変化をしないので，即座にユーザは

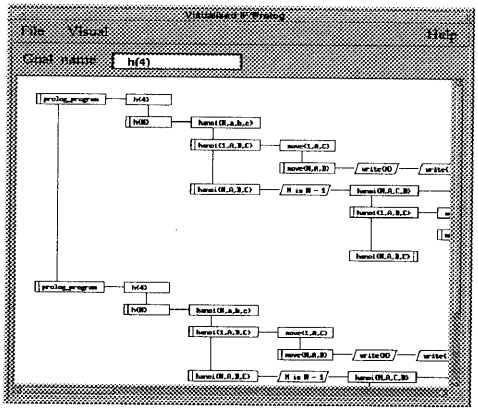

Fig. 15. Prologの図表示

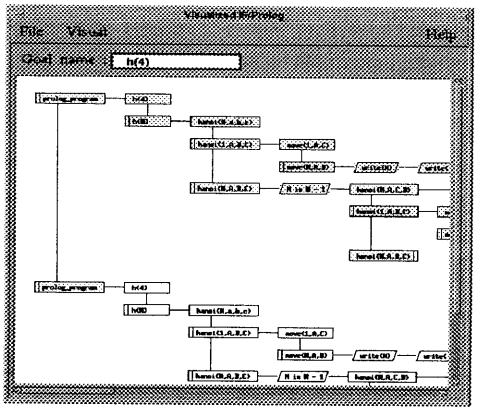

Fig. 16. トレースの実行画面

実行過程を理解することができる．

トレース機能を行った実行画面をFig.16に示す．トレースの際，高速に実行過程を処理するので1ステップごとにトレースを一時中断し，ユーザがマウスまたキーボードを押すと次のステップに進むモードも備えている．

● プログラム図とソースプログラムとのリンク

ソースプログラムを表示するコマンドを選択すると，ソースプログラムがポップアップウィンドウ上に表示される．このとき，セルをクリックするとそれに対応するソースプログラムの1行がハイライトされる．また，ソースプログラムの1行をマウスでクリックすると対応したセルがハイライトされる．この機能により，プログラム図とソースプログラムとの間の対応関係が容易に認識できる．Fig.17にその実行画面を示す．

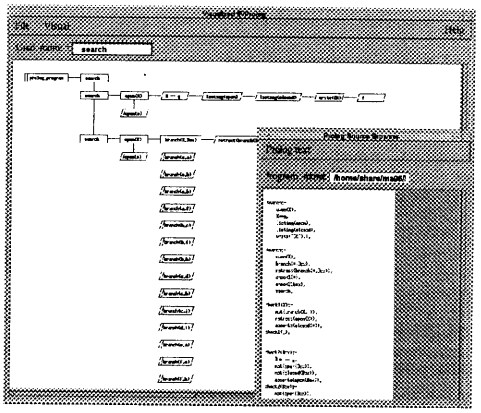

Fig. 17. ソースプログラムとの対応

4 おわりに

Prologプログラムを図表示するための属性グラフ文法を定式化し，それに基づいたProlog可視化システムを実現した．

この可視化システムは，ユーザが任意に選択したゴールに対して内部表現を自動生成して対応するプログラム図を表示する．また，本システムにはレイアウトのアルゴリズムも組み込まれており，属性グラフ文法に基づいて実装されている．このレイアウトルーチンは他のレイアウト条件に基づくルーチンによって容易に置き換えること可能である．トレース機能は，メタプログラミング技法を利用しており，リアルタイムに実行過程を表示することが出来る．

なお，本システムはIF/Prolog Ver5.0AとOSF/Motifで記述されており，SUN OS 4.1.3及びSoralis 2.4上で稼働している．

今後，実行時にassert，retractなどの述語呼出しによるプログラムの動的な変化を表示する機能や，変数への代入過程を表示する機能などを付加してより高度なPrologプログラミング支援環境へと発展させる．

参考文献

[1] Y.Adachi, K.Anzai, K.Tsuchuda, T.Yaku : Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, IEEE COMPSAC(1996).

[2] Vigna,P.D. and Ghezzi, C. : Context-Free Graph Grammars, Inf.Control,Vol.37,pp.207-233(1978).

[3] 西野 ：属性グラフ文法とそのHichart型プログラム図式に対するエディタへの応用，コンピュータソフトウェア,Vol.5,pp.81-92(1986).

[4] Sicstus Syntax, http://www.sics.se/ps/sicstus.html

[5] IF/Prolog Manual, IF Computer Japan Limited