第６章

論理型プログラミング

6.1 宣言的表現

★

その記述順によらず，それぞれ独立に閉じた意味を持つような
宣言の集まりとして記述されたプログラムは，わかりやすい。

従来のプログラミングパラダイムとして位置付けられる手続き型言語は，フォン・ノイマン型アーキテクチャを基本としている。4 章で述べた N. Wirth と R. A. Kowalski の図式

プログラム＝アルゴリズム＋データ構造
アルゴリズム＝論理＋制御

に従えば，手続き型パラダイムに基づくプログラムでは，データとはメモリ（記憶装置）を意味し，制御は明示的に記述される。データがメモリである故にプログラムの実行に副作用が伴い，プログラムの意味がわかりにくくなる。さらに，制御構造を自由に記述できる故にプログラムの実行順序が複雑になり，プログラムの意味がわかりにくい。

宣言的パラダイムはこのような欠点がなく，以下の特徴を有する。

①プログラムが副作用を伴わない。

②プログラムの構成要素がそれぞれ独立に閉じた意味を持ち，その構成要素の記述順序がプログラムの意味を変えない。

この①と②の特徴は互いに独立ではないが，本書では，プログラミングパラダイムにおける「宣言的」という言葉をこのような意味で用いる。宣言的パラダイムに基づくプログラミングの例として，本章で論理型プログラミング，次章で関数型プログラミングについて述べる。

6.2 Prolog

★ *データと制御によるコンピュータのふるまいの記述を避け，論理だけを記述したプログラムは，わかりやすい。*

6.2.1 Prologの概要

論理型プログラミング言語 Prolog は，1970 年頃にフランスのマルセーユ大学の A. Colmerauer によって考案された。当初は，構文解析ツールや定理証明ツールとして用いられた。その後，1974 年にロンドン大学インペリアルカレッジの R. Kowalski が，これにプログラミング言語としての解釈を与え，1977 年にはエジンバラ大学の D. Warren が実用的な性能の処理系を開発した。さらに，1980 年代に入って，日本の第 5 世代コンピュータ (ICOT) の基本言語として採用されたことから，人工知能の研究分野を中心に広く使われ始めた。現在では，プロトタイピング用の言語としても利用されている。

Prolog の一般的な特徴は次のようなものである。

①プログラムは，述語論理の一形式であるホーン節で記述され，導出原理に基づいて実行される。
②ユニフィケーションと呼ばれる強力なパターンマッチング機能がある。
③自動的なバックトラッキングによる探索機能がある。

以下ではこれらの特徴をはじめとする Prolog のプログラミング機能について，

例を用いて具体的に説明する。言語仕様は，広く用いられているエジンバラ大学の DEC-10 Prolog に従うが，記号として日本語も使用する。

6.2.2 節形式によるプログラム表現

Prolog でのプログラムの表現形式の基本となる「事実」，「規則」，「質問」について，まず図 6.1 の家系図を例にとって説明する。図 6.2 の具体的なプログラム例の中で，事実の記述は，この家系図で示される親子関係を記述したものである。“父(a, b)”の形式の表現は，「a は b の父である」という事実を表している。同じく，“母(a, b)”は，「a は b の母である」という事実を表している。なお，左端の番号は本書での説明用に便宜的に付けたものである。

1 番目の規則 r1 の“祖父(X, Z) : －父(X, Y), 父(Y, Z).”は，「X が Z の祖父であるためには，X が Y の父であり，Y が Z の父であればよい」ことを表現している。同じく 2 番目の規則 r2 は，「X が Z の祖父であるためには，X が Y の父であり，Y が Z の母であればよい」ことを表現している。最後の“？－祖父(X, 公暁).”の形式の表現は，「公暁の祖父は誰ですか？」という質問を意味する。このプログラムでは，“X＝頼朝”という回答が出力される。このプログ

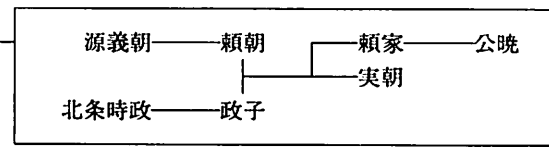

図 6.1　家系図の例

```
[事実]
(f1)  父(義朝, 頼朝).
(f2)  父(時政, 政子).
(f3)  父(頼朝, 頼家).
(f4)  父(頼朝, 実朝).
(f5)  父(頼家, 公暁).
(f6)  母(政子, 頼家).
(f7)  母(政子, 実朝).

[規則]
(r1)  祖父 (X, Z) : －父 (X, Y), 父 (Y, Z).
(r2)  祖父 (X, Z) : －父 (X, Y), 母 (Y, Z).

[質問]
 ? －祖父 (X, 公暁).
```

図 6.2　Prolog のプログラム例

ラムの実行過程は後で詳しく説明する。

ここでは，まずPrologの基本構文について述べる。Prologのプログラムは，図6.3に示すようなホーン節(Horn clause)と呼ばれる表現の集合である。その一般形“A：－B_1，B_2，…，Bm.”は，

結論：－条件1，条件2，…，条件m.

を意味している。すなわち，「左辺の結論Aが成立するためには，右辺のすべての条件Biが成立すればよい」あるいは「右辺のすべての条件Biが成立すれば，左辺の結論Aが成立する」ことを示している。ここでAおよびBiは，素論理式（atomic formula）といわれ，

述語名（項，項，…，項）

の形をしている。項（term）は次のいずれかである。

①定数：先頭が英小文字の名前，日本語の名前，整数など
　（例）path，頼朝，123
②変数：先頭が英大文字の名前など
　（例）X，Y，Route
③構造：名前（項，項，…，項）の形式
　（例）父（頼朝，頼家），path（X，Y，[X，Y]）

基本構文の2番目の右辺が空のホーン節は，一般形の特殊な場合で，左辺の結論が成立するための条件が右辺に記述されていない。したがって，「結論Aが無条件に成立する」あるいは「Aは事実である」ことを示している。基本構文の3番目の左辺が空のホーン節は，「右辺の条件が成立するか否かを調べよ」を

① A：－B_1，B_2，…，Bm.　[ホーン節の一般形（規則に対応）]
② A.　[ホーン節の右辺が空（事実に対応）]
③ ？－B_1，B_2，…，Bm.　[ホーン節の左辺が空（質問に対応）]

図6.3　Prologの基本構文

意味する。条件の中に変数が含まれる場合，条件を成立させるために関連付けた値がその変数の求める解となる。3番目の基本構文の理論的な意味は6.3節で述べる。

6.2.3 プログラム実行方式

図6.2の例題プログラムを用いて，Prologプログラムの実行過程を説明する。例題は，7つの事実と2つの規則を用いて，「公暁の祖父は誰ですか？」という質問に答えるものである。この質問を意味する"祖父(X, 公暁)"は，Prologでは，ゴールまたは目標と呼ばれ，このゴールすなわち「公暁の祖父はXである」という述語が正しいことを証明する過程が，プログラムの実行過程である。その過程で変数Xに結合した値がこの質問の答となる。

その実行過程を**図6.4**に示す。

① まずゴールが与えられると，図6.2のプログラムを上から順に調べて，そのゴールと対応のとれる節を探す。ゴール"祖父 (X, 公暁)"と規則r1の左辺"祖父 (X, Z)"は，変数Zを公暁に対応させると一致することがわかる。このような操作をユニフィケーション (unification) という。

② これは，ホーン節の本来の意味から，ゴールが成立するためには，規則r1の右辺が成立すればよいことを意味する。そこで，ゴールを証明する代わりに右辺の2つの素論理式をサブゴールと考えて，同じ操作を繰り返す。この時，サブゴールの中の変数で特定の値と結合したものはその値で置換しておく。この例では，2番目のサブゴールのZは公暁で置き換えられる。

③ 1番目のサブゴール"父 (X, Y)"は，①と同様に図6.2のプログラムを上から順に調べ，まず事実f1の"父 (義朝, 頼朝)"とのユニフィケーションに成功する。この時，Xは義朝，Yは頼朝に結合する。

④ 次に2番目のサブゴールの変数Yを頼朝に置き換えた"父 (頼朝, 公暁)"に対してユニフィケーションを試みるが，図6.2のプログラムの中には一致する節がないので，ユニフィケーションは失敗する。

⑤ そこで，もう一度，1番目のサブゴールに戻って，事実f1以外で一致する節をf1のすぐ下から順に探すと，事実f2の"父 (時政, 政子)"とのユニフィケーションに成功する。この時，Xは時政，Yは政子に結合する。このよ

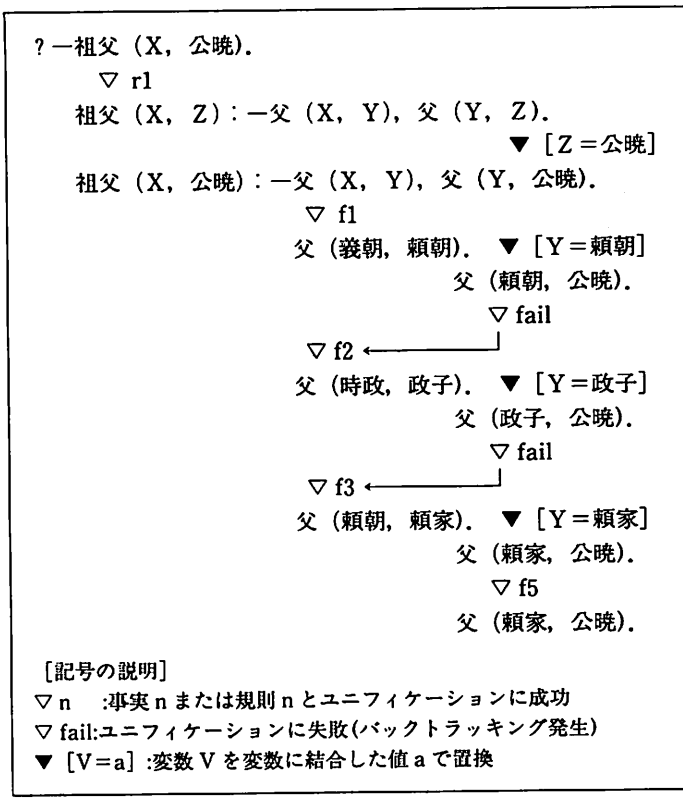

図 6.4 Prolog プログラムの実行過程

うにユニフィケーションに失敗したときに，それまでのゴールの証明過程を後戻りしてやりなおすことをバックトラッキング（backtracking）という。

⑥ しかし，この場合も④と同じく，2番目のサブゴールで失敗して，バックトラッキングが発生する。

⑦ 再び1番目のサブゴールに戻り，事実 f3 の“父（頼朝，頼家）”とのユニフィケーションに成功する。この時，X は頼朝，Y は頼家に結合する。

⑧ 今度は2番目のサブゴール“父（頼家，公暁）”が事実 f5 とのユニフィケーションに成功する。

⑨ 規則 r1 の右辺の2つのサブゴールが成功したので，その左辺に対応する最初のゴールが成功したことになる。この時，変数 X は頼朝と結合しているので，“X＝頼朝”すなわち「公暁の祖父は頼朝である」という答が得られたこ

とになる。

このようにPrologプログラムの実行では，ユニフィケーションとバックトラッキングが基本的な役割を果たしている。なお，この例からわかるように，Prologでのユニフィケーションは節集合のプログラムを上から順に調べて実行するために，節の記述順序によって実行結果が異なる場合がある。したがって，6.1節で述べた宣言的パラダイムの「記述順序がプログラムの意味を変えない」という特徴は満たさなくなっている。

6.2.4　節形式の手続き的解釈

このようなPrologの実行過程を従来のPascalやFORTRANなどの手続き型言語に対応させると，

A：－B_1，B_2，…，Bm.

という節は，

[Pascalの場合]

```
procedure A;
  begin
     B1;
     B2;
     …
     Bm
  end
```

[FORTRANの場合]

```
SUBROUTINE A
    CALL  B1
    CALL  B2
          …
    CALL  Bm
RETURN
END
```

と解釈できる。すなわち，節の左辺が手続き部の頭部，節の右辺が手続き呼び出しの列からなる手続き本体に対応する。

一方，手続き型言語との大きな違いは，ユニフィケーションとバックトラッキングの機能があることである。そのため，前述の家族関係のプログラムでいえば，次の4種類の質問は1つのプログラムで処理できる。

①	?－祖父（頼朝，公暁）.	——「頼朝は公暁の祖父ですか？」
②	?－祖父（頼朝，X）.	——「頼朝は誰の祖父ですか？」
③	?－祖父（X，公暁）.	——「公暁の祖父は誰ですか？」
④	?－祖父（X，Y）.	——「誰が誰の祖父ですか？」

これは，Prologのゴールの引数が定数でも変数でもよく，定数のときは入力パラメータ，変数のときは出力パラメータとみなされるためである。手続き型言語では，一般にこれらの4種類の質問に対し，別々のプログラムが必要になる。

6.2.5 プログラミング機能

次に，Prologのプログラミング機能を説明する。そのために2つの例題を考える。

[例題1：平均点を求める]

学生の点数のリストを与えられて，その平均点を計算する。たとえば，入力が[70，90，60，80]のとき，75が出力される

プログラム例を図6.5に示す。

① 平均点を求める述語averageの第1引数は入力パラメータとなる学生の点数リスト，第2引数は出力パラメータとなる平均点の計算結果である。1番目の節に示すように，入力が空リストのときは0を返す。

② 述語sumnumは，第1引数で点数リストを入力として受け取り，その合計点および合計人数を第2，第3引数として出力する。3番目の節に示すよう

```
average( [ ] , 0).
average(Marklist, Ans) :- sumnum(Marklist, Sum, Num),
                          Ans is Sum / Num.
sumnum( [ ] , 0, 0).
sumnum( [Data | Rest] , S, N) :- sumnum(Rest, S1, N1),
                                 S is S1 + Data, N is N1 + 1.
```

図6.5 例題1：平均点を求めるPrologプログラム

に，入力が空リストのときは共に0を返す。

③　節の右辺で使用されているisはその右側の算術式の値を左側の変数に代入する演算子である。＋と／はそれぞれ加算と除算の演算子である。これらの演算子は一般になじみのあるインフィックス形式で示しているが，基本形にしたがって，is(Ans,/(Sum, Num))のように書いてもよい。

④　リストは，[70, 90, 60, 80]のように表され，[ ]は空リストを意味する。[Data | Rest]は，Dataがリストの第1要素に対応し，Restが残りのリストに対応する。たとえば，[70, 90, 60, 80]と結合したときはDataが70，Restが[90, 60, 80]となる。

ここで，述語sumnumは2つの節を用いて定義されている。最後の節の右辺でsumnumの呼び出しが使用されているが，このように左辺の述語が右辺でも使用されることを再帰（recursion）という。この例では，再帰的計算方法により，[70, 90, 60, 80]の合計点と合計人数の計算手順は，

```
(70＋(90＋(60＋(80＋(0)))))
( 1＋( 1＋( 1＋( 1＋(0)))))
```

のように右から順に和をとるアルゴリズムになっている。

すなわち，ゴールとして，

```
?- average( [70,90,60,80], A ).
```

が与えられると，プログラムの2番目の節のaverageの左辺と一致し，その右辺でサブゴールとしてsumnum([70, 90, 60, 80], Sum, Num)が実行される。この時，sumnumの2番目の節が再帰的に4回呼び出され，その都度，入力リストの要素数が1つずつ減っていく。最後にその右辺のサブゴールがsumnum([ ], S1, N1)となり，入力リストが空になったとき，sumnumの1番目の節が呼びだされる。この後，これまで呼び出された順番とは逆の順に呼び出されたところへ戻りながら，点数と人数の合計をしていくため，右から順に和をとる計算となる。

[例題2：成績順にソートする]

名前と点数を対にしたリストを要素とするリストを与えられて，点数の高い順にリスト要素を並べ替える。たとえば，

[[abe, 70], [baba, 90], [chiba, 60], [dan, 80]]

が入力されると

[[baba, 90], [dan, 80], [abe, 70], [chiba, 60]]

を出力する。

プログラム例を図6.6に示す。

① 組み込み述語readは，項の読み込みに使用される。

② 組み込み述語writeは，項の書き出しに使用される。

③ 点数順にソートする述語nmsortの第1引数は入力パラメータとなる学生の名前と点数の対のリスト，第2引数は出力パラメータとなるソートされた結果である。入力が空リストのときは空リストを返す。

④ 述語insertは，第1引数として学生の名前と点数の対のリスト，第2引数としてすでにソート済みのリストを入力として受け取り，前者を後者の適切な場所に挿入し，第3引数として出力する。ここでは簡単な挿入ソート法(insertion sort)を用いている。

```
listout :- write('Input a list of  [name, marks]  :'), read(Inlist),
           nmsort(Inlist, Outlist),
           write('A sorted list of students : '), write(Outlist).

nmsort( [] , [] ).
nmsort( [Student | Rest] , Slist) :- nmsort(Rest, Sublist),
                                    insert(Student, Sublist, Slist).

insert(S, [] , [S] ).
insert( [N, M] , [[N1, M1] | Rest] , [[N1, M1] | Ilist] ) :-
                                  M < M1, insert( [N, M] , Rest, Ilist).
insert(S, Slist, [S | Slist] ).
```

図6.6 例題2：点数順にソートするPrologプログラム

先の例では，以下の順にソートされていく。

```
[ ]
[[dan, 80]]
[[dan, 80], [chiba, 60]]
[[baba, 90], [dan, 80], [chiba, 60]]
[[baba, 90], [dan, 80], [abe, 70], [chiba, 60]]
```

6.2.6　人工知能言語としての探索機能

Prologの実行過程を先の6.2.3項ではゴールの証明過程として説明したが，問題解決のための探索機能としてみることもできる。その説明のために次のような問題を考える。

[例題3：出発地から目的地までの飛行ルートの選択]

図6.7の右上に示したように，5つの都市を結ぶ8つの飛行ルートがあるとき，2つの都市を指定して，その間を結ぶ飛行ルートを選択する。

この問題のプログラム例を図6.7に示す。述語flight(a, b)は，aからbへの航空路があるという事実を示す。述語pathは，第1引数の都市から第2引数の都市へ行くために，第3引数のようなルートがあることを示している。規則r1

```
(f1)  flight(札幌, 東京).
(f2)  flight(札幌, 大阪).
(f3)  flight(札幌, 福岡).
(f4)  flight(東京, 大阪).
(f5)  flight(大阪, 福岡).
(f6)  flight(東京, 沖縄).
(f7)  flight(大阪, 沖縄).
(f8)  flight(福岡, 沖縄).

(r1)  path(X, Y, [X, Y] ) :- flight(X, Y).
(r2)  path(X, Y, [X | Route] ) :- flight(X, V), path(V, Y, Route).
```

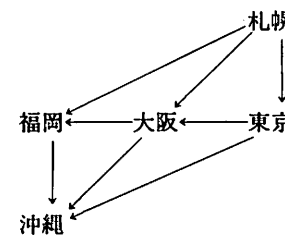

図6.7　例題3：飛行ルート選択のPrologプログラム

は直行便を示す。規則 r2 は乗り継ぎ地 V を経由して目的地 Y へ行くルートを再帰的に定義している。

このプログラムを用いて札幌から沖縄へ行くルートを求めてみよう。ゴールは次のように与えられる。

```
?- path(札幌, 沖縄, Route)
```

この実行過程は，図 6.8 のようになり，[札幌，東京，沖縄]というルートが求まる。

ここで，このプログラムと選択されたルートの搭乗券を予約する述語 reserve とを組み合わせて用いた場合を考えてみると，ゴールは次のようになる。

```
?- path(札幌, 沖縄, Route), reserve(Route).
```

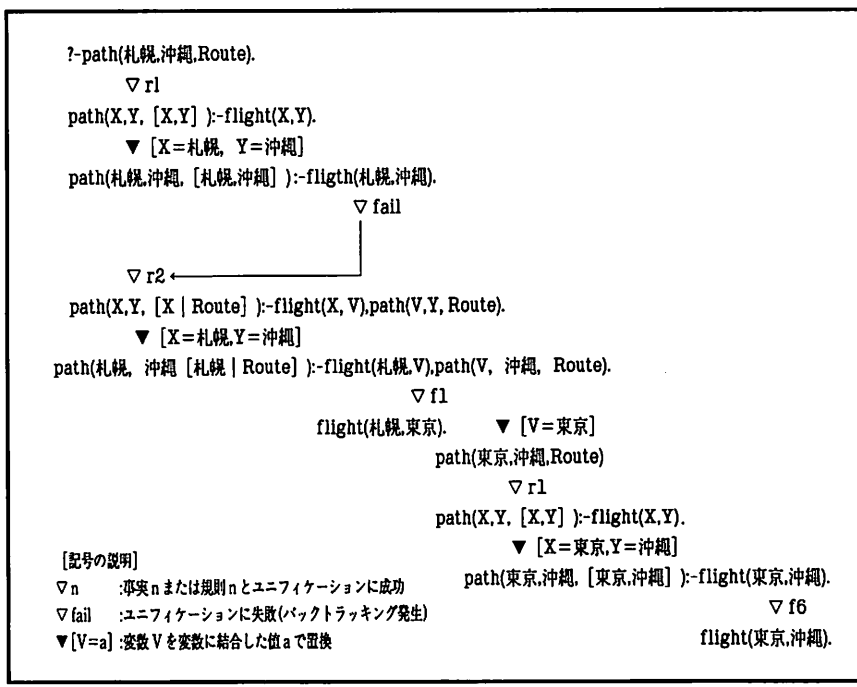

図 6.8 Prolog プログラムの実行過程

このゴールはreserveのところで搭乗券の予約に失敗する度にpathに戻り，次のルートを別解として再び搭乗券の予約を試みる。この別解の探索は，直前の解を求めたときの最後のユニフィケーションが失敗した場合と同様に実行される。したがって，2番目の解は，図6.8の右下のf6の事実とのユニフィケーションに成功したところを失敗させ，1つ手前のr1のユニフィケーションのところに戻り，r2とのユニフィケーションを試みる。以下同様にして，第2のルート[札幌，東京，大阪，沖縄]が求まる。このpathとreserveの組み合わせのように，最初のゴールで可能な解の候補を求め，2番目のゴールでその解の候補が解としての条件を満たすか否かを検査するような問題解決方法を生成テスト(generate and test)法という。

結局，この問題では，6つの解が次の順序で見つけられる。

①[札幌，東京，沖縄]
②[札幌，東京，大阪，沖縄]
③[札幌，東京，大阪，福岡，沖縄]
④[札幌，大阪，沖縄]
⑤[札幌，大阪，福岡，沖縄]
⑥[札幌，福岡，沖縄]

このように経路探索の途中で複数の選択が可能なとき，その1つを選んで先へ進み，行きづまれば1つ手前に後戻りして別の選択をするようなバックトラッキングによる解法を縦型探索または深さ優先探索(depth-first-search)という。

これに対し，複数の選択が可能なものはすべて同時に調べながら進む方法を横型探索または広さ優先探索(breadth-first-search)という。この場合は乗り換え回数の少ない飛行ルートが優先的に選ばれるため，解の求まる順序は①④⑥②⑤③となる。

6.2.7 カット

Prologでは，この縦型探索機能を制御するためのカットという特殊な機能がある。プログラムのわかりやすさや宣言的表現という観点からは好ましくないが，プログラミング機能としては有用であるので簡単に説明しておく。カット

は節の右辺で'!'という記号を用いて記述するが，その意味は次のようなものである。

[カットの意味]

プログラムの実行過程でこのカットが実行されると，親の節が呼び出されてから実行されたすべての選択を最終的なものと決定し，他の未実行の選択を放棄する。

この定義ではわかりにくいので，カットの代表的な3つの用法を以下で例を用いて説明する。例題としては，結婚相手の選択アルゴリズムを考える。

(a) 用法1：別解の抑止

先に述べたように，Prologではどこかでユニフィケーションに失敗するとバックトラッキングが生じて自動的に別解を求めるが，この別解が必要でないときにそれを抑止するのに用いる。たとえば，以下の結婚相手の選択アルゴリズムにおいて，「本命の相手にプロポーズをして断られたら結婚を諦める」ためにカットを用いている。

```
結婚(W) :- 条件(W),!, 結婚申込(W).
条件(X) :- 女性(X), テニス(X), ゴルフ(X).
条件(X) :- 女性(X), テニス(X).
条件(X) :- 女性(X), ゴルフ(X).
条件(X) :- 女性(X).
```

これは，ある男性が，まずテニスとゴルフの両方できる女性を最良とし，もしそういう人がいなければ次にテニスだけできる女性，それもいなければゴルフだけできる女性，いずれもいなければ女性であればよい，という条件で結婚相手を探すプログラムである。もし1行目にカットがなければ，条件を満たす女性に結婚を申込み，断られたら次の条件を満たす女性（別解）を探してまた申し込む。しかし，この場合は，カットが結婚申込みの直前に置かれているので，条件を満たす最初の女性に結婚を申込み，失敗したら'条件(W)'に戻って別解を求めることはせず，1行目の節全体が失敗したことになる。

(b)　用法2：無駄な探索の回避

プログラム実行中に，あるところで失敗すれば，その後，探索を続けても必ず失敗することがわかっているような場合に，無駄な探索をしないでプログラムの実行効率をよくするためにカットを用いることができる。たとえば，以下の結婚相手の選択アルゴリズムにおいて，「相手の年齢がわかったときに，この人がもっと若ければとは考えない」ためにカットを用いている。

```
結婚(M) :- 男性(M), 条件(M), 結婚申込(M).
条件(X) :- 年齢(X, A), A>35, !, fail.
条件(X) :- 年齢(X, A), A>30, !, 車(X).
条件(X) :- 年齢(X, A), A>25, not(煙草(X)).
```

これは，ある女性が結婚相手の条件を年齢によって分けている。男性Mの年齢が35歳より上ならば，組み込み述語failを用いて無条件に失敗させている。35歳以下で30歳より上ならば車を条件とし，30歳以下で25歳より上ならば煙草を吸わないことを条件にしている。ここで2，3行目にカットがなければ，男性Mの年齢が35歳より上のときまたは30歳より上でかつ車を持たないとき，その下の条件を探索に行くが，年齢のチェックで必ず失敗する。カットはこの無駄を避けるため，男性Mがある条件の年齢チェックを満たした後で失敗したときは，他の条件を見にいかないようにしている。

(c)　用法3：否定の実現

否定の述語notは，カットを用いて以下のように定義できる。

```
not(P) :- call(P), !, fail.
not(P).
```

1行目は，引数の述語Pが，call(P)で実行され，もし成功すれば，failで必ず失敗させ，しかもカットによって2行目の節を実行しないようにしている。2行目の節は，1行目のcall(P)で失敗し，カットを実行しなかったときだけ実行され，かつ必ず成功するようになっている。用法2のプログラム例では，「煙草はダメよ」という結婚の条件に用いられている。

6.3 論理型プログラミングの特徴

★

論理で表現されたプログラムの意味は，機械独立かつ人間志向の用語で定義できるので，作りやすく，わかりやすく，改良しやすく，再利用しやすい。(Robert Kowalski, 1979, 文献(40))

Prologと関連の深い論理型プログラミングの概念である節形式とホーン節の関係および導出原理について説明しておく。

6.3.1 節形式とホーン節

論理を仮定と結論の関係とみて，次のように表現したものを節形式と呼ぶ。

$A_1, \cdots, An \leftarrow B_1, \cdots, Bm$

ここで，AiとBjは，6.2.2項で説明した素論理式である。この節形式の意味は，この節が含むすべての変数$x_1, \cdots, xk$について以下のようなものである。

① $n \geqq 1$, $m \geqq 1$のとき,
A_1 or ⋯ or An if B_1 and ⋯ and Bm

② $n \geqq 1$, $m=0$のとき,
A_1 or ⋯ or An

③ $n=0$, $m \geqq 1$のとき,
$\neg$ (B_1 and ⋯ and Bm)

④ $n=0$, $m=0$のとき,
□ (常に偽)

この節形式の結論部をたかだか1つの素論理式，すなわち，$n \leqq 1$と限定したものが，すでに前節で述べたようにPrologで用いられているホーン節である。

6.3.2 導出原理

前節でPrologの実行過程は，ゴールの正しさの証明過程であると説明した

が，その基礎となる考えは，J. A. Robinson によって提案された導出原理(resolution principle) である。これは公理から定理を導く過程を自動的（機械的）に行う方法である。すなわち，2つの節があり，一方の節の左辺ともう一方の節の右辺に同一の素論理式があれば，これらの節を結合し，この素論理式を取り除いて，新しい節を導くことができる，というものである。たとえば，

$$\begin{aligned} A &\leftarrow P, Q \\ Q, R &\leftarrow X, Y \end{aligned}$$

という2つの節から右辺と左辺のQに着目して，

$$A, R \leftarrow P, X, Y$$

という節を導ける。

Prolog のゴールは，6.2.2 項の図 6.3 で述べたように，節の左辺がない形で与えられたが，これは上記③に対応し，「ゴールが成立しない」ということを意味している。そして，Prolog の実行過程では，事実と規則から成る Prolog プログラムの節集合を用いて，質問に対応するゴールを否定した節に対して導出原理を何度も適用し，左辺も右辺も空の節を導いている。すなわち，

$$\leftarrow B_1, \cdots, Bm$$

から，右辺の置き換えで，

$$\leftarrow$$

を導く。この空節は上記④に対応し，常に偽，すなわち矛盾を意味する。結局，「ゴールが成立しない」という仮定から出発して矛盾を導くことにより，最初の仮定が誤りであったこと，すなわち「ゴールが成立する」ことを示すという方法でゴールの正しさを証明している。

図 6.4 の例では，「X は公暁の祖父である」という証明すべきゴールを否定した，「X は公暁の祖父ではない」という左辺が空の節から出発して，その右辺に節 r1，f3，f5 を順次適用して両辺が空の節を導いている。この矛盾を導くことに成功した証明過程で変数 X と結合した「頼朝」が X の1つの解となっている。