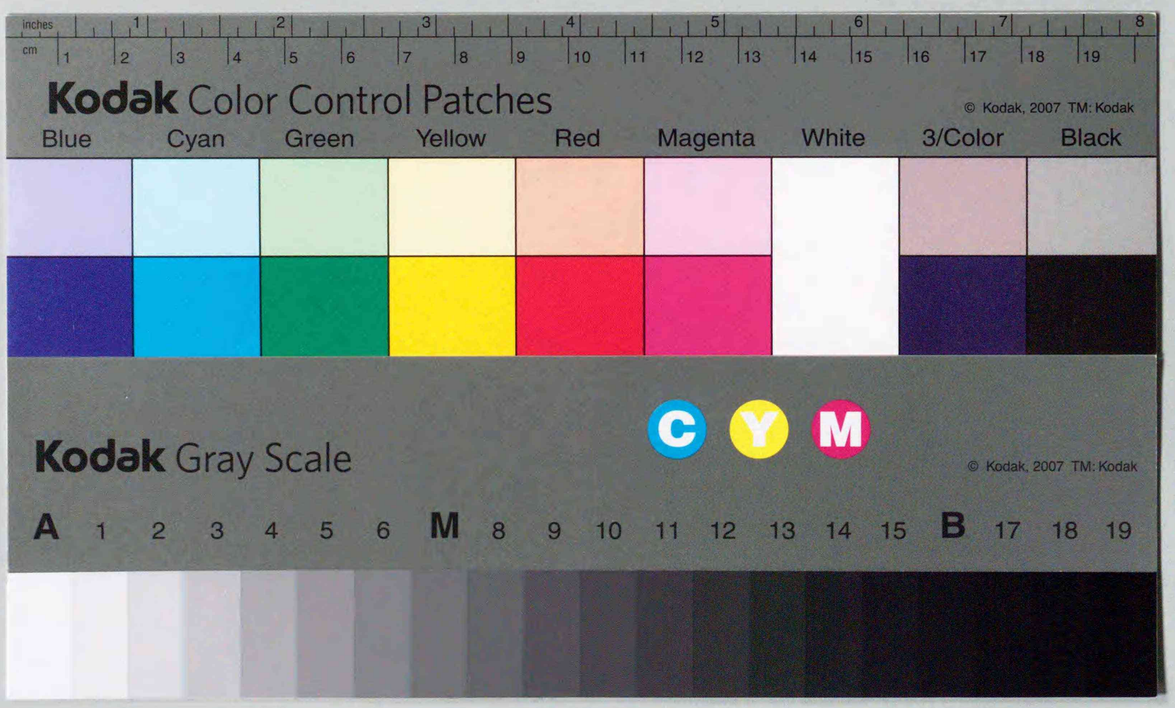
HOKKAIDO UNIVERSITY

Title	論理プログラミングへの語彙と語彙構築機構の導入に関する研究
Author(s)	後藤, 文太朗
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第3488号
Issue Date	1994-09-30
DOI	https://doi.org/10.11501/3097891
Doc URL	https://hdl.handle.net/2115/50125
Type	doctoral thesis
File Information	000000278127.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

論理プログラミングへの
記事と記事構造探索機構の導入に関する研究

学位申請者
岩 間 一 雄



論理プログラミングへの
語彙と語彙構築機構の導入に関する研究

後藤　文太朗

目次

第1章　序論 1
1.1　自然言語における語彙の役割 1
1.2　論理プログラミング 6
1.3　知識表現言語としてのPrologの問題 10
1.4　計算機への語彙の導入 12
1.5　論理プログラミングにおける視覚化 14
1.6　シンセティック・メディア・アーキテクチャと語彙 17
1.7　本論文の位置づけ 19

第2章　Prologへの語彙の導入 21
2.1　論理型プログラミング言語Prolog 22
2.1.1　項 22
2.1.2　述語と事実 23
2.1.3　ルール 25
2.1.4　Prologプログラムと質問 25
2.1.5　リスト 29
2.1.6　カット 30
2.2　名詞 33
2.3　形容詞 34
2.4　語の合成 36
2.5　語彙構築 36
2.6　まとめ 37

第3章　拡張論理型プログラミング言語Vocalog 39
3.1　Vocalogのシンタックス 39
3.2　基本名詞 41
3.3　名詞句 47
3.4　基本形容詞 58

3.5　形容詞句 62
3.6　派生名詞と派生形容詞 70
3.7　実現手法 72
3.7.1　Prolog in Prolog 72
3.7.2　Vocalogインタープリタ 76
3.8　まとめ 78

第4章　Vocalogによるプログラミング例 81
4.1　家系に関する語彙の構築例 81
4.2　文法規則の表現例 96
4.3　リスト処理 108
4.4　まとめ 117

第5章　Vocalogプログラムの可視化 119
5.1　概略 119
5.2　論理プログラミングにおける可視化 120
5.3　VPEシステムの概略 121
5.4　まとめ 137

第6章　VPEシステムの実現 139
6.1　概要 139
6.2　IntelligentPad 139
6.3　実現手法 143
6.4　まとめ 147

第7章　VPEシステムの使用例 149

第8章　結論 155

謝辞 157

参考文献 159

第1章　序論

1.1　自然言語における語彙の役割

我々人間は，言語を用いて個々人の内部に蓄積された情報のやり取りを行い，より高次なレベルの思考を行うことができる．自然言語では，‘語彙’と‘構文’の二つをその道具として用いている．なかでも語彙は非常に強力な道具である．自然言語における語彙には，

(1)　対象の概念や対象間の関係概念を語で表す機能，
(2)　語を合成して複雑な対象の概念や複雑な対象間の関係概念を表現する機能，
(3)　合成語に対して名前付けを行い，この名前を新たに語として用いることを許す語彙構築機能

がある．これらの機能は，人と人との間における高度なコミュニケーションや，概念モデルの構築を可能にしている．

“対象の概念や対象間の関係概念を語で表す機能”により，実際にその対象を示さなくても，その対象の概念を伝えることが可能である．すなわち，コミュニケーションをとっている人々の共通語彙の中に，その対象を表現する語が存在すれば，その語を使うだけで目的とする対象を示すことができる．さらに，その語が表す概念がそれぞれの人の頭の中で，どのように構成されているかを気にする必要もない．

例えば，‘人’というプロパティを持つ対象を表したいとき，‘人’を表す単語がなければ実際に‘人’とはどういうものであるかを示さねばならない．すなわち，実際に‘人’というプロパティを持つ対象を集めて，そのような対象一般が‘人’であるとい

うことをコミュニケーションをとっている人の間で合意しなくてはならない．しかし，会話をする人々の共通語彙の中に‘人’を表す単語「人」が含まれていれば，その対象を「人」という単語により指示できる．すなわち，コミュニケーションをとっている人との間で，実際に‘人’というプロパティを持つ対象を集めて，そのような対象一般が‘人’であるということをコミュニケーションをとっている人の間で合意しなくても，単に「人」という語を使うだけで，‘人’というプロパティを持つ対象を指示できる．このとき，コミュニケーションをはかっている人それぞれの頭の中で，どのように‘人’という概念が構成されているかを気にする必要はない（図1.1）．

“語を合成して複雑な対象の概念や複雑な対象間の関係概念を表現する機能”により，語彙の中に含まれる語を用いてそれぞれの語が表す概念を個々独立に表現するだけではなく，それらの語を組み合わせて句を構成し，それぞれの語が表していた概念を組み合わせた複合概念を表現することができる．語を組み合わせる際に，コミュニケーションをはかっている人それぞれの頭の中で，それらの語が表す概念がどのように構成されているかを気にする必要はない．単に，語のレベルで組み合わせを行うだけで，それぞれの語が表していた概念を組み合わせた複合概念を表現することができる．

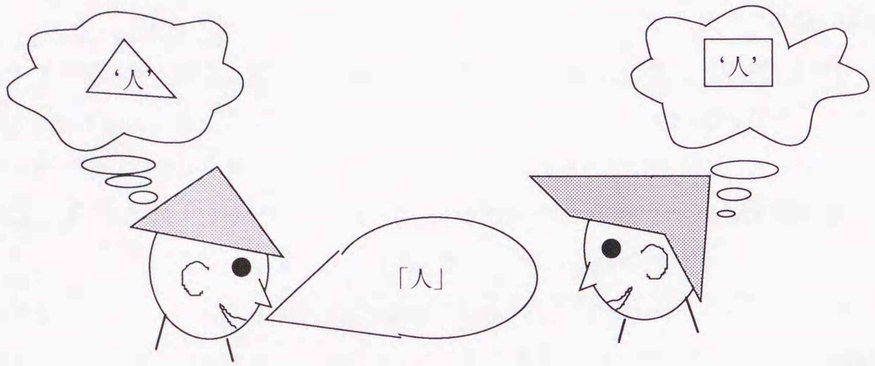


図1.1　語による対象概念の表現

例えば，共通語彙の中に，「親」，「男」という単語が含まれている場合を考えてみる．それらの語を元にして『男親』という句を合成することができる．『男親』という句により，それぞれの語が表していた概念を合成した複合概念‘親であり，かつ男であ

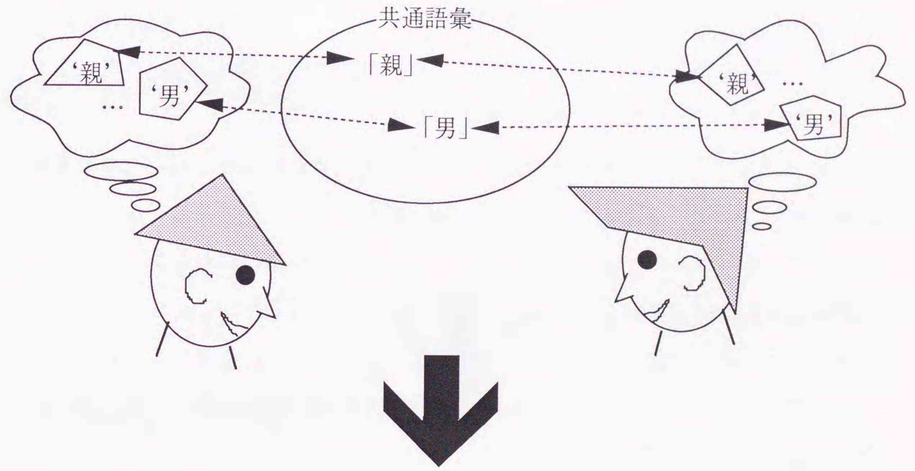


図1.2　合成語による複合概念の表現

る’を表現することができる．したがって，‘親であり，かつ男である’という概念を伝えるために，‘親であり，かつ男である’というプロパティを持つ対象を実際に集めて示さなくてもよい．さらに，‘親であり，かつ男である’という概念を直接表現する語がなくてもよい．すなわち，‘親’と‘男’という概念を表す語として「男」と「親」が，共通語彙の中に存在すれば，それらの語を組み合わせた『男親』という句を使うだけで，‘親であり，かつ男である’という概念を表現できる（図1.2）．この際に，「親」と「男」という語が表す‘親’とか‘男’という概念が，コミュニケーションをとっているそれぞれの人の頭の中でどのように構成されているかを気にする必要はなく，単にそれらの語を組み合わせるだけでよい．

“合成語に対して名前付けを行い，この名前を新たに語として用いることを許す語彙構築機能”により，コミュニケーションをとる人々の間の共通語彙を増やしていくことが可能である．句に対して名前付けを行うことで，よく使われる合成概念をコンパクトに表現することができる．さらに，一度その新たな名前を使うことを決めてしまえば，そのあとは，その名前がどのような句に対して付けられたかを気にする必要はない．また，その名前がどのような句に対して付けられたものであるかを伝えることで，その概念を伝えることが可能である．

例えば，前述のように，‘親’と‘男’という概念を表す語として「男」と「親」が，共通語彙の中に存在したとする．それらの語を組み合わせた『男親』という句に対して「父親」という名前付けを行い，「父親」という語で『男親』という句が表していた概念を表現することに決めれば，次回からは「父親」という語で‘親であり，かつ男である’という概念を表現できる（図1.3）．

基本的な概念を表現する基本語彙が人々の間に共通に存在すれば，語彙構築機能により共通語彙を増やしていくことが可能となる．個々人の頭の中においても，基本概念を表す語を組み合わせたものの中で有用なものに対して名前付けを行っていくことで，複雑な概念を構成する方法の一つにもなっていると考えられる．そして，そのように構成された概念を別の人に伝えるには，基本概念を表す語に対するコンセンサスを得たあとで，どのようにして複雑な概念を表す語を構成していったかを順次示していけばよい．

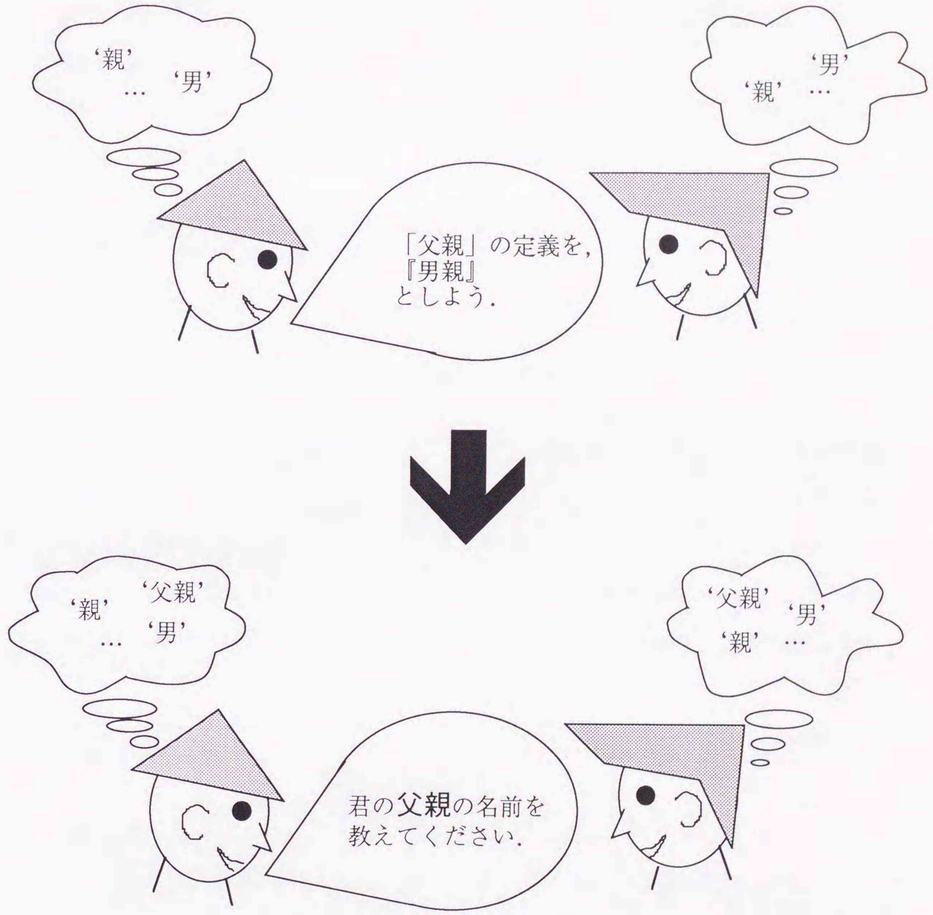


図1.3　語彙構築機能

1.2　論理プログラミング

論理プログラミングは，「論理をプログラミング言語として使うことができる」という考えに基づくものである．KowalskiとColmerauerにより，この考え方がもたらされた[1][2]．そして，一階述語論理の一部分であるホーン節に基づいた論理型プログラミング言語Prolog(Programming in logic)が作られた．

Kowalskiは，彼の論文のタイトルにもあるように，論理プログラミングの中心的な考え方の一つは，

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

であると述べている[3]．すなわち，アルゴリズムは問題が何(What)であるかを記述している論理(Logic)の部分と，どのように(How)問題を解くかを記述している制御(Control)の部分からなるという考え方である．論理プログラミングでは，Whatの部分をプログラマーが記述し，Howの部分は論理プログラミングシステムが自動的に実行してくれるというのが理想である．Prologは，完全ではないがこの理想を実現している．

論理プログラミングでは，ふつう，プログラムは確定節と呼ばれる論理式の集合である．確定節とは，

$$P \vee \neg Q_1 \vee \ldots \vee \neg Q_n \qquad (n \geq 0)$$

という形をした論理式である．ここで，$P, Q_i (1 \leq i \leq n)$は素論理式である．これを次のように表す．

$$P \leftarrow Q_1, \ldots, Q_n \qquad (n \geq 0)$$

n=0のときは，特に次のように表す．

$$P$$

論理プログラミングにおけるプログラムのことを論理プログラムと呼ぶ．論理プログラム（すなわち，論理式）はプログラム（手続き）として解釈され，計算機上で実行される．プログラムをある解釈に従って評価し計算する機構は，一般的にインタープリタと呼ばれる．論理プログラミングにおけるプログラムの実行は，ゴール節と呼ばれる論理式を論理プログラムのインタープリタに与えることにより行われる．ゴール節とは，

$$\neg Q_1 \vee \ldots \vee \neg Q_n \qquad (n \geq 1)$$

という形をした論理式である．確定節の場合と同様に，これを

$$\leftarrow Q_1, \ldots, Q_n \qquad (n \geq 1)$$

と表す．

インタープリタにより，論理式はプログラムとしての‘手続き的な意味’を持つ．この意味論は論理プログラムの手続き的意味論と呼ばれる．一方，論理プログラムをそのままの形の論理式とみなして解釈する意味論である宣言的意味論がある．

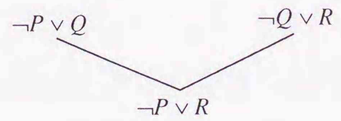
命題論理の場合，三段論法は，

$$P \rightarrow Q$$
$$Q \rightarrow R$$

という二つの論理式から

$$P \rightarrow R$$

という論理式を論理的帰結として導き出すものである．ここで，P, Q, Rは素論理式である．これを書きなおすと次のようになる．



すなわち，二つの論理式$\neg P \vee Q$と$\neg Q \vee R$とからそれぞれQと$\neg Q$を取り除いて，新しい論理式$\neg P \vee R$を導き出している操作とみることができる．この操作は導出原理と呼ばれる．$\neg P \vee R$は$\neg P \vee Q$と$\neg Q \vee R$からの導出節と呼ばれる．導出原理は一般的には，

$$P_1 \vee P_2 \vee \ldots \vee P_m \vee Q \qquad \neg Q \vee R_1 \vee R_2 \vee \ldots \vee R_n$$

$$P_1 \vee P_2 \vee \ldots \vee P_m \vee R_1 \vee R_2 \vee \ldots \vee R_n$$

となる．Qは素論理式，P_iとR_jはリテラルである．リテラルとは素論理式または素論理式の否定である．素論理式とその否定の対は相補リテラルと呼ばれる．命題論理の導出原理は，二つの節の中に相補リテラルを見つけ出し，それらを取り除いて選言をとった節を導き出す操作である．

一階述語論理の導出原理では，さらに変数と項との対応を考える．すなわち，変数を項に置き換える代入を適用してから相補リテラルを見つけ出す．例えば，$p(a,X)$と$\neg p(Y,b)$というリテラルは，そのままでは相補リテラルではないが，変数Xをbに置き換え，変数Yをaに置き換える代入$\{X/a, Y/b\}$を適用することで，相補リテラルとなる．変数を別の項で置き換えることにより，表現を一致させることを単一化と呼ぶ．表現$E_1, \ldots, E_n$が代入θにより一致するとき，代入θは$\{E_i\}$の単一化代入と呼ばれる．このような単一化代入が存在するとき，$\{E_i\}$は単一化可能であるといい，そうでないときは単一化不可能であるという．一階述語論理における導出原理を図示すると次のようになる．

$$P_1 \vee P_2 \vee \ldots \vee P_m \vee Q \qquad \neg Q' \vee R_1 \vee R_2 \vee \ldots \vee R_n$$

QとQ'は代入θにより単一化可能

$$(P_1 \vee P_2 \vee \ldots \vee P_m \vee R_1 \vee R_2 \vee \ldots \vee R_n)\theta$$

ただし，$E\theta$は表現Eに代入θを適用した結果得られる表現を表している．

論理式をプログラム（手続き）と解釈することにより，導出原理に基づく証明を計算とみなすことができる．論理プログラムPとゴール$\leftarrow G$が与えられたとする．このとき，$P \cup \{\leftarrow G\}$の節に対して導出原理を適用していき，空節□が得られれば，$P \cup \{\leftarrow G\}$が充足

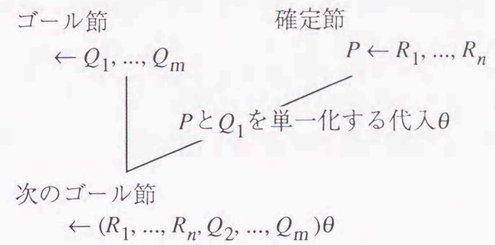


図1.4　SLD導出

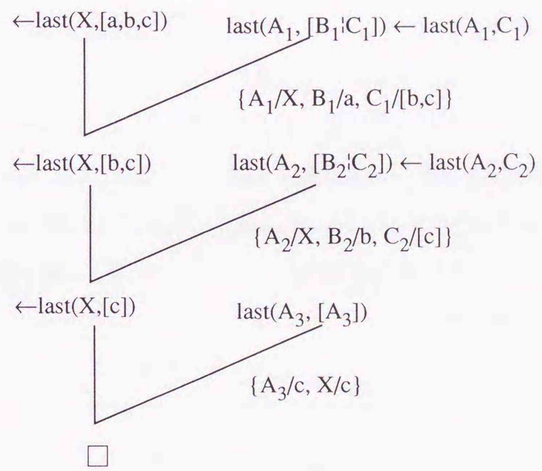


図1.5　プログラム実行例

不可能であることを示す．$P \cup \{\leftarrow G\}$が充足不可能であるということは，GがPからの論理的帰結であることと同値である．$P \cup \{\leftarrow G\}$の節に対して導出原理を適用していき，空節□が得られる過程で求められる代入は，$P \cup \{\leftarrow G\}$が充足不可能である反例を構成するのに用いられる．この代入におけるG中の変数に対する代入は手続きを実行した結果の出力であるとみなすことができる．

ゴール節$\leftarrow G$と確定節の集合P中の節とから新たなゴール節$\leftarrow G'$を導出する図1.4の形

の導出は，一般にSLD導出と呼ばれる．プログラムの実行は，この過程を空節□が導出されるまで繰り返す．そのような例を図1.5に示す．図1.5は，←last(X, [a,b,c])の反駁を，X=cという反例を示すことにより行っている．手続き的にいうと，リスト[a,b,c]の最終要素を変数Xに求めている．

SLD導出を用いたインタープリタでは，手続き的意味と宣言的意味が一致する[35]．すなわち，計算結果はプログラムのもとで論理的に正しく，かつプログラムのもとで論理的に正しいものは，いつでもプログラムで計算して得られる．

1.3 知識表現言語としてのPrologの問題

親子関係を調べるアルゴリズムを記述する場合を考えてみる．Prologでは，誰が誰と親子関係にあるかを記述するだけでよい．すなわち，「‘太郎’は‘次郎’の親である」，「‘花子’は‘恵子’の親である」，....といったような事実を次のような述語表現を使った文で表現するだけでよい．

```
is_parent_of(‘太郎’, ‘次郎’).
is_parent_of(‘花子’, ‘恵子’).
    ...
```

ここで，is_parent_of(‘太郎’, ‘次郎’)が「‘太郎’は‘次郎’の親である」という関係を表現する述語表現である．最初に関係を表現する述語名が記述され，その引数としてその関係にあるオブジェクトが記述される．引数位置はプログラマーが責任を持って決めておく必要がある．is_parent_of(‘太郎’, ‘次郎’)とis_parent_of(‘次郎’, ‘太郎’)のように引数位置を変えた述語表現は，その表している関係は一般的には同じではない．

上述のように誰が誰と親子関係にあるかを記述しておいて，親子関係を調べるには，

次のような質問をPrologシステムに行えばよい．

```
?- is_parent_of(‘花子’, ‘恵子’).
```

これは，「‘花子’は‘恵子’の親ですか？」という質問を表している．すると，Prologシステムは記述されている親子関係を調べて，その関係があれば‘yes’を，なければ‘no’を返す．プログラマーは，どのようにして親子関係を調べるかは気にしなくてよい．Prologシステムが自動的に記述された親子関係を調べる．しかし，プログラマーまたはユーザーは述語is_parent_ofがいくつの引数をとるのかを知っている必要がある．述語is_parent_ofが2引数だと知った上でもさらに，引数のどちらが親でどちらがその子供なのかを把握している必要がある．

前述のように，Prologにおいては，プログラマーは何が問題であるかというWhatの部分の記述を行えばよいという点は，従来の手続き型プログラミング言語にないよい性質である．これは，知識表現言語としてPrologが優れている理由の一つである．しかし，Prologで用いている述語表現は，非常に一般的な表現形式であり，その表現が何を表しているかということは，プログラマーの責任において決めることである．知識表現言語には，さらに高レベルな記述が望まれる．すなわち，問題とする対象領域のオブジェクトをより直接的に表現できることが望まれる．

そのような目的を持った研究の中に，レコード構造に着目した‘レコードプログラミング’の研究がある[4]．レコードプログラミングは，論理型プログラミング言語における項を拡張して，レコード的な構造をもつオブジェクトを扱えるようにしたものであり，KBL[5]，LOGIN[6]，CIL[7]，CRL[8]などが提案されている．例えば，KBLでは次のような表現形式が用いられる．

```
person( id⇒name;
        domicile⇒address(city⇒cityname);
        father⇒person   ).
```

ここで，personはオブジェクトのタイプを表す．id,domicile,fatherはpersonタイ

プのオブジェクトが持つ属性を表す．name， address(city⇒cityname)，personは，それぞれ属性id，domicile，fatherがその値として持つオブジェクトのタイプを表している．このように，オブジェクトを表現する際に，Prologでは述語表現を使って，プログラマーの責任において引数位置を決めていたが，レコードプログラミングでは引数の属性とその値を使ったレコード表現が可能である．Prologの述語表現による一般的な記述に比べて，より直接的にオブジェクトを記述することが可能である．

1.4 計算機への語彙の導入

1.1節で述べたように自然言語において語彙は，人と人との間における高度なコミュニケーションや，概念モデルの構築を可能にしているという点で重要な役割を担っている．その点に着目し，自然言語における語彙が持つ特徴を計算機の世界に導入する研究がTanakaにより行われてきた[9][10][11][12]．論文[9]と[10]では，データベースへの問い合わせに語彙を導入している．そして，論文[11]と[12]において，一階述語論理で記述された知識ベースへ語彙を導入することを行っている．論文[11]において，述語の各引数位置に対応させて属性を導入している．そして，属性とその値を使った*predicative list*という考えを導入している．例えば，

$$\mathtt{person}(t_1,t_2,t_3)$$

という述語の第1，第2，第3引数の位置にName， Sex， Birthdayという属性を定義する．そして，述語名personのかわりにこれらの属性を使った

$$\mathtt{<Name,\ Sex,\ Birthday>}$$

という表現を使うことを許し，この表現をpredicative listと呼んだ．

$$\mathtt{person}(t_1,t_2,t_3)$$

という述語表現のかわりに，

$$\mathtt{<Name,\ Sex,\ Birthday>}(t_1,t_2,t_3)$$

というpredicative list表現を用いた．predicative list表現は，前述のレコードプログラミングと同様に，述語表現による一般的な記述に比べて，より直接的にオブジェクトを記述することが可能である．属性と対応させて基本名詞という概念を定義している．

[11]ではさらに，導入した属性の拡張を行っている．そして，それらの拡張された属性をpredicative listにおいて用いることを許している．FatherとMotherという属性が定義されていたとすると，

$$\mathtt{Father\ +\ Mother}$$

が拡張属性の例である．

Tanakaによるアプローチが前述のレコード・プログラミングによるアプローチと異なる大きな点は，次に述べる修飾詞の概念の導入による．修飾詞は自然言語の形容詞に相当する概念であり，属性を修飾してオブジェクトに制約を加えたり，オブジェクト間の関係を表現するものである．parentという修飾詞を用いたpredicative listを用いたpredicative list表現の述語の例をあげると次のようになる．

$$\mathtt{<parent(Person)>}(t_1)$$

$$\mathtt{<Person,\ parent(Person)>}(t_2,\ t_3)$$

ここで，Personは属性である．最初の表現では属性値t_1をもつオブジェクトは親であるという制約を表している．2番目の表現では，属性値t_3を持つオブジェクトは属性値t_2をもつオブジェクトの親であるという関係を表している．修飾詞と対応させて基本形容詞という概念を定義している．さらに，修飾詞も属性と同様に拡張を行い，合成修飾詞などを導入している．そして，属性もさらに修飾詞を含む形に拡張を行っている．

基本名詞を属性と対応させて定義を行い，基本形容詞を修飾詞と対応させて定義を行っているが，属性や修飾詞を拡張したのと同様に，基本名詞と基本形容詞をもとにし

て，合成語を構成し，それらの合成語に名前付けを行う関数をステップワイズに構成していくプロセスとして語彙構築を定義している．そして，そのようにして導入された名詞や形容詞のセマンティックスを，それらの名詞や形容詞を属性や修飾詞に変換する関数として定義を行っている．

1.5 論理プログラミングにおける視覚化

オブジェクトをテキスト表現だけではなく視覚的に表現することは，ユーザーインターフェースやCAIシステムなどに有用である．例えば，BarwiseとEthemendyによるTariski's World[19]というシステムは，一階述語論理における視覚的CAIシステムである．Tariski's Worldでは，一階述語論理におけるオブジェクトとそれらの関係を視覚化することで，論理式の理解などを視覚的に支援している．また，Prologシステムに関しても，プログラム中でウィンドウなどの視覚オブジェクトを扱えるようにした，SICStus Prolog上のGraphics Manager[18]やQuintus Prolog上のProXTやProXL[28]などが開発されている．

プログラミングの視覚的支援は，プログラムの効率的な作成・デバッグ，プログラムの効率的な利用に有効である．プログラミングにおける視覚化には，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化がある．テキストのみを用いたプログラミングに比べて，プログラム実行の流れを視覚化することでその流れが把握しやすくなったり，テキストだけを用いたユーザーインターフェースよりも，よりユーザーフレンドリーなインターフェースを与えることができる．論理プログラミングの視覚化に関する研究も，この3つのうちのどこをターゲットにしているかにより分類することができる．

Prologプログラムの作成に関しては，プログラムを可視化することで，プログラム全体の構造を把握しやすくしたり，データの表現をわかりやすいものにすることが可能となる．Prologプログラムの作成を視覚的に支援する研究として，LadretとRueherのVLP[24]，PauとOlasonのVPP[26]，YeungのMPL[32]，渋谷と田中によるVisiLog[33]などが行われ

表1.1 論理プログラミングにおける視覚化システムの比較

1. プログラムの作成環境の視覚化
2. プログラムのデバッグ環境の視覚化
3. プログラムの利用環境の視覚化

システム	1	2	3
VLP	○	○	×
TPM	×	○	×
PROEDIT2	×	○	×
VPP	○	×	×
DEWLAP	×	○	×
Polakらによるシステム	×	○	×
MPL	○	○	×
Emdenらによるシステム	×	×	○
LOGIFORM	×	×	○
VisiLog	○	×	○
VPE	○	○	○

てきた．

Prologにおけるプログラミングでは，プログラマーは主に問題が何かというWhatの部分を記述する．どのように問題を解くかというHowの部分はPrologシステムが自動的に行ってくれる．これは，プログラムを記述するという面では利点ではあるが，プログラムをデバッグするという面では問題となる．すなわち，プログラムをデバッグする際には，プログラマーが記述した‘問題が何かというWhatの部分’だけではなく，Prologシステムが自動的に実行する‘どのようにして問題を解くかというHowの部分’もトレー

スしていく必要があるということである．Prologプログラムの実行の流れを可視化することで，デバッグを支援する研究として，LadretとRueherのVLP[24]，EisenstadtとBrayshawのTPM[23]，NumaoらによるPROEDIT2[25]，DewarとClearyのDEWLAP[22]，Polakらによるシステム[27]，YeungのMPL[32]などが行われてきた．

Prologプログラムの利用も，テキストを使って質問をするのではなく，視覚的インターフェースを与えることで，ユーザーがより利用しやすい環境を提供することが可能となる．Prologプログラムに対して，スプレッドシートをインターフェースとして用いる研究として，Emdenらによるもの[31]，BonsignoriらによるLOGIFORM[5]，渋谷と田中によるVisiLog[35]などがおこなわれてきた．

以上の論理プログラミングにおける視覚化システムがそれぞれ，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化のうちのどの部分を行っているかをまとめてみると，表1.1のようになる．これらのシステムは，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化のうちの一部分を行っているにすぎない．しかし，プログラムの作成，デバッグ，そしてプログラムの利用はそれぞれ独立したものではなく相互に関連している．これらの環境を個々独立に視覚化したのでは，プログラム利用のための視覚オブジェクトの動きを見ながら視覚的なデバッグを行うといった，機能連携が行えない．したがって，これらの環境を一つのシステムに統合して視覚化すべきである．これらの環境を一つのシステムに統合する際には，プログラムの作成モード，プログラムのデバッグモード，プログラムの利用モードとして個々独立に視覚化するのではなく，これらのモードが一つの環境に混在しうる統合システムにすべきである．これにより，視覚化された個々のモードを独立に利用するのみならず，モード間をわたった機能連携を行うことが可能となる．

エンドユーザーレベルで3つのモード間の自在な機能連携を行うためには，3つのモードをつなぎめなく統合することが必要となる．すなわちモード間の差異にとらわれずに視覚オブジェクトを統一的に操作することが可能となるように，どのモードにおいても視覚オブジェクトの表現形態が同一であるようにすべきである．それぞれのモード用の

視覚オブジェクトの操作も含んだ見せ方を用意しておくことで，それぞれのモードに適した視覚オブジェクトの操作及び表示が可能となる．これにより，3つのモード間の自在な機能連携を行うことができ，かつ各モードに特有の視覚オブジェクトの操作及び表示が可能となる．

1.6 シンセティック・メディア・アーキテクチャと語彙

我々人間の知的活動のための解放型包括的統合環境としてのシンセティック・メディア・アーキテクチャと，それを実現したシステムIntelligentPadが提案されている[29][30]．シンセティック・メディア・アーキテクチャとは，

(1) 各種メディアのツールキット化，
(2) 部品の組合せによる合成プログラミング，
(3) 新たな部品の追加登録機能，
(4) メディアオブジェクトの統一的操作，
(5) メディアオブジェクトの流通機構

を実現したメディア・アーキテクチャである．「メディア」という枠組みで，計算機上にのせられるあらゆる知識資産を統一的に取り扱う．

IntelligentPadでは，すべてのメディアを機能を持った「紙」状のオブジェクトとして可視化を行っている．この機能を持った「紙」状のオブジェクトはパッドと呼ばれる．パッドという表現形態により，各種メディアが実現される．すなわち，パッドという表現形態により，各種メディアが部品化されている．そして，メディアオブジェクトに対する操作には，「紙」のメタファーが用いられている．すなわち，紙は移動させたり，コピーしたり，紙の上に紙をはったり，逆に剥がしたりするといったことができる．同様にパッドは自在に移動させたり，複製を作ったりできる．パッドどおしをはりあわせたり，逆に剥がしたりするといったことができる．パッドどおしのはりあわせにより，

各パッドが表現するメディアの機能を合成して新たな機能を持ったメディアオブジェクトが作られる．

自然言語における語彙と「メディア」という概念を対応させることで，上記のシンセティック・メディア・アーキテクチャと自然言語の間には緊密な対応関係をみることができる．前述の5個の項目を自然言語における語彙という観点からみてみると次のように言い換えることができる．

(1) 各種概念の語による表現
(2) 語を組み合わせた合成語による複合概念の表現
(3) 新たな語の追加機能
(4) 語レベルでの概念の操作
(5) 辞書による概念の流通機構

(1), (2), (4)は1.1節で述べたことから明らかである．(3)は，新たにある概念を表す語を追加したとしても，その語もそれまでの語と組み合わせて用いることが可能であるということである．(5)は，語の定義を流通することで，さまざまな概念の流通が行えるということである．

「メディア」を知識処理を行う上での情報処理媒体ととらえると，語彙も一種の「メディア」である．本論文では，論理型プログラミング言語Prologに語彙を導入したVocalogに関して述べるが，VocalogとIntelligentPadを融合することは，上記の議論より非常に自然であると考えられる．論理プログラミングへの語彙の導入は，従来Prologでは不可能であったシンセティックプログラミングを可能にすると考えられる．シンセティックプログラミングとは，部品の合成により，新たな部品を構成していくことでプログラミングを行っていくことである．語を合成し，その合成語に対して名前付けを行い語彙を増やしていく過程はシンセティックプログラミングとみなすことができる．IntelligentPadを用いてVocalogプログラムを可視化することで，IntelligentPadにおけるシンセティックプログラミングとVocalogにおけるシンセティックプログラミングを自然に対応させることができる．

1.7 本論文の位置づけ

自然言語において，語彙は人と人との間における高度なコミュニケーションや，概念モデルの構築を可能にするうえでの重要な機能を備えている．拡張論理型プログラミング言語Vocalog(Vocabulary based logic programming language)は，論理型プログラミング言語Prologに自然言語の語彙が持つ特徴を導入したものである．Prologに自然言語の語彙が持つ特徴を導入することで，知識表現言語としての記述力を高めることが可能である[16][17]．

プログラムの可視化は，プログラムの効率的な作成・デバッグ，プログラムの効率的な利用に有効である．シンセティック・メディア・アーキテクチャを実現したIntelligentPadにおけるシンセティックプログラミングと拡張論理型プログラミング言語Vocalogにおけるシンセティックプログラミングとの間には緊密な対応関係をとることができる．IntelligentPadを用いてVocalogプログラムを可視化することが自然に行える．VPE(Vocalog programming environment)は，IntelligentPadを用いてVocalogプログラムを可視化したVocalogの視覚的統合プログラミング環境である．プログラミングにおける視覚化には，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化がある．プログラムの作成，デバッグ，そしてプログラムの利用はそれぞれ独立したものではなく相互に関連している．VPEは，プログラムの作成モード，プログラムのデバッグモード，プログラムの利用モードが一つの環境に混在しうる統合システムである．VPEでは，視覚化された個々のモードを独立に利用するのみならず，モード間をわたった機能連携を行うことが可能である[34]．

本論文はこれらの研究成果を述べたものである．以下に本論文の概要を示す．

第2章では，まず，論理型プログラミング言語Prologについて説明する．そのあとで，自然言語の語彙が持つ特徴として，どのようなものをPrologに導入し，その結果どのような利点が得られるかを明らかにする．

第3章では，第2章で明らかにした特徴を持つ拡張論理型プログラミング言語Vocalog(Vocabulary based logic programming language)について説明する．

第4章では，Vocalogによるプログラミング例に関して説明を行う．

第5章では，Vocalogプログラムをどのようにして可視化を行ったかについて示す．

第6章では，IntelligentPadを用いてVocalogプログラムの可視化を行っているVocalogの視覚的統合プログラミング環境VPEがどのようにして実現されているかを説明する．

第7章では，VPEシステムの使用例として，世界の都市に関するデータベースの例を示す．

第8章では，本研究で得られた結論を示す．

第2章　Prologへの語彙の導入

論理プログラミングは，「論理をプログラミング言語として使うことができる」という考えに基づくものである．KowalskiとColmerauerにより，この考え方がもたらされた．そして，一階述語論理の一部分であるホーン節に基づいた論理型プログラミング言語Prolog(Programming in logic)が作られた．

Kowalskiは，彼の論文のタイトルにもあるように，論理プログラミングの中心的な考え方の一つは，

Algorithm = Logic + Control

であると述べている[3]．すなわち，アルゴリズムは問題が何(What)であるかを記述している論理(Logic)の部分と，どのように(How)問題を解くかを記述している制御(Control)の部分からなるという考え方である．論理プログラミングでは，Whatの部分をプログラマーが記述し，Howの部分は論理プログラミングシステムが自動的に実行してくれるというのが理想である．Prologは，完全ではないがこの理想を実現している．

拡張論理型プログラミング言語Vocalog(Vocabulary based logic programming language)は，Prologに語彙を導入し利用できるようにしたものである．語彙をPrologに導入する目的は，知識表現言語としての記述力を高めることである．特に，基本的な概念をコンテキストから取り出して名詞や形容詞としてジェネリックに定義し，これらの合成によりより高次の概念をコンテキストから独立に定義することを目的とする．

Vocalogでは，名詞と形容詞を定義でき，それらを用いることでそれぞれ対象の概念や対象間の関係概念をジェネリックに記述できる．語構成子を用いて名詞や形容詞を組み合わせて名詞句や形容詞句を作ることができる．名詞句や形容詞句を用いることで，そ

れぞれ対象の概念や対象関の関係概念を表現できる．語彙構築機能を用いることで，基本的な語彙をもとに，派生語彙を増やしてくことができる．導入した名詞や形容詞を述語定義や質問で利用する機構としてリスト述語を導入している．

本章では，論理型プログラミング言語Prologに語彙を導入することでどのような利点が得られるかについて説明を行う．本章の構成は次のとおりである．

2.1節では，論理型プログラミング言語Prologについて説明を行う．2.2節では，名詞概念をPrologに導入すること，その利点について説明する．2.3節ではVocalogにおける形容詞に関して述べる．2.4節では，語の合成に関して説明を行う．2.5節では，語彙構築機能に関して説明する．最後に関連研究について述べたあと，まとめを述べる．

2.1 論理型プログラミング言語Prolog

論理型プログラミング言語Prologは一階述語論理に基礎を置くプログラミング言語である．まず，Prologにおいてオブジェクトを表現する項について説明する．次に，述語による事実の表現を説明する．次いで，ルールの記述に関して説明する．Prologにおけるプログラミングは，事実およびルールをデータベースに定義し，そのデータベースに対して質問を発することである．最後に，このことに関して説明を行う．

2.1.1 項

項は，定数，変数，複合項（構造体）にわけられる．定数は，数字または文字列である．数字でない定数をアトムと呼ぶ．ただし，本論文では文字列を表す定数は小文字から始まるとする．大文字で始まる文字列は次に述べる変数とする．大文字から始まる文字列を定数とする場合には，その文字列をシングルクウォートでくくればよい．定数の例を次にあげる．


```
taro
jiro
7
1994
'Birthday'
```

変数は，その値として項をとる．本論文では，変数は大文字またはアンダースコア‘_’から始まる文字列で表すことにする．変数の例を次にあげる．

```
X
Y
Name
Z1
_birthday
```

同一の節中に出現する同じ字面の変数は，それが束縛される際には同一の値を取る．ただし，特別な変数名として‘_’がある．‘_’は無名変数といい，同一の節中に複数個‘_’が出現したとしてもPrologシステムでそれらを別々のユニークな変数に置き換える．

複合項（構造体）は，`f(a,X)`や`father(john)`のような形をしている．`f`や`father`のことを関数記号またはファンクタと呼ぶ．関数記号は，小文字で始まる文字列で表す．`a`，`X`，`john`のことを引数と呼ぶ．引数は項である．

2.1.2 述語と事実

Prologでは，オブジェクト間に成り立っている関係を述語表現を用いて記述する．例えば，「‘Taro’は‘Jiro’の親である」という関係は，Prologでは，

```
is_parent_of('Taro' , 'Jiro')
```

という述語で表現される．ここで，is_parent_ofが述語名である．この場合，述語is_parent_of(X, Y)は2つの引数を持ち，2項関係を表している．述語の第1引数が親の名前，第2引数が子供の名前を表している．一般的には，n項関係として，

$$p(t_1, \ldots, t_n)$$

と表現される．ここで，pは述語名，t_iは項である．$p(t_1, \ldots, t_n)$を素論理式と呼ぶ(述語，原子論理式，句とも呼ぶ)．

実際に成り立っている関係は，上述の述語表現を用いて，事実として記述する．例えば，「'Taro'は'Jiro'の親である」という事実は，

```
is_parent_of('Taro' , 'Jiro') .
```

のように，述語表現の後にピリオドをつけた文で記述する．一般的には，n項関係の事実は

$$p(t_1, \ldots, t_n).$$

と表現される．ここで，t_iは変数を含まない項である．一般にt_iが項のときは，

$$p(t_1, \ldots, t_n).$$

を単位節と呼ぶ．すなわち，事実は単位節で変数を含まないものである．この表現は，第1章の論理プログラミングの節でのべた，

$$p(t_1, \ldots, t_n) \leftarrow$$

という表現に相当する．

このような事実の集合により，Prologのプログラムを構成できる．上述の親子関係に関する事実の集合により，親子データベースが構成される．

2.1.3 ルール

事実はオブジェクト間の関係を表すものであるが，Prologでは事実に加えて，一般的なルールにより関係を表現することができる．例えば，「親の親は祖父母である」という関係は次のように表現できる．

```
is_grandparent_of(X,Y) :-
        is_parent_of(X,Z),
        is_parent_of(Z,Y).
```

ここで，':-'の右側が':-'の左側が成り立つための条件を表している．':-'の右側の','は'論理積'を表している．一般的には，

$$P :- Q_1, \ldots, Q_n .$$

という形でルールは表現される．これをプログラム節（確定節）と呼ぶ．ここで，P，Q_iは素論理式（述語）である．単位節はプログラム節の特殊な形(n=0のとき)である．':-'の左側をプログラム節の頭部，':-'の右側をプログラム節の本体と呼ぶ．これは，第1章の論理プログラミングの節でのべた，

$$P \leftarrow Q_1, \ldots, Q_n$$

という表現に相当する．

2.1.4 Prologプログラムと質問

Prologのプログラムは，前述の事実やルールの集合である．Prologにおけるプログラミングは，事実やルールをデータベースに定義し，Prologシステムに質問を行うことであ

る．

質問も述語表現を用いる．例えば，「‘Taro’は‘Jiro’の親ですか？」という質問は，

```
?- is_parent_of('Taro', 'Jiro').
```

のように表現される．一般的には，

$$?- Q_1, \ldots, Q_n .$$

という形で質問は表現される．これをゴール節と呼ぶ．ここで，Q_iは素論理式（述語）である．これは，第1章の論理プログラミングの節でのべた，

$$\leftarrow Q_1, \ldots, Q_n$$

という表現に相当する．プログラム節とゴール節をまとめてホーン節と呼ぶ．

Prologシステムは，このような質問が入力されると，データベース中の事実とルールを使って，この関係が成り立っているかどうかを調べる．成り立っている場合は‘yes’という返事をし，成り立っていない場合は，‘no’という返事をする．

いま，図2.1.1に示されているようなプログラムがデータベース中に定義されているとする．上記の質問がPrologシステムに与えられたときの，Prologシステムの動きを図2.1.2に示す．図2.1.2では，最初に与えられたゴールがデータベース中の最初の事実と一致するかどうかを調べている．そして，図2.1.2では，一致していないので次の事実を調べにいく．この場合，ゴールとして与えらた述語の部分と，事実の述語の部分が一致しているので‘yes’を返している．このように，とりあえず複数の候補から一つを選んで前に進んでみて，その先の段階のどこかでそれ以上動作が進まなくなったら，後戻りして別の候補を選んで処理を進める方法はバックトラックと呼ばれる．バックトラック機能は，Prologにおいて宣言的プログラミングを可能にしている機能の一つである．

変数を用いた質問を行うことで，その関係が成り立っているオブジェクトをデータベース中から取り出すことができる．例えば，「‘Jiro’の親Xを求めなさい」という


```
is_parent_of('Ken', 'Taro').
is_parent_of('Taro', 'Jiro').
is_parent_of('Hanako','Jiro').
....
```

図2.1.1　Prologプログラム例

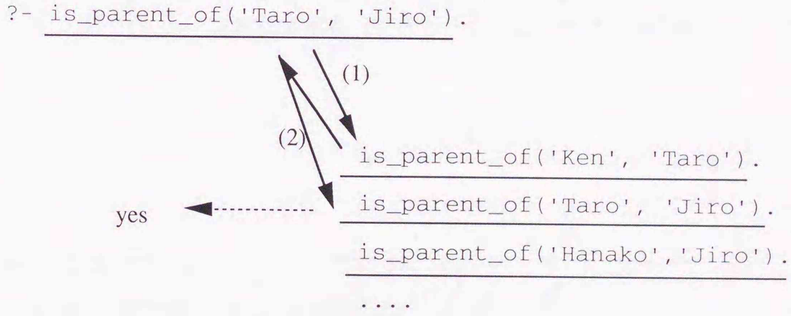


図2.1.2　質問実行の様子(バックトラック)

質問は，

```
?- is_parent_of(X, 'Jiro').
```

のように表現される．Prologシステムはデータベース中に，この関係が成り立つようなXの値をさがしにいく．そのようなXの値があれば，PrologはXにその値を返し，無い場合は‘no’という返事をする．Prologのデータベース中に図2.1.1のプログラムが定義されているとする．すると，上述の質問の実行過程は，図2.1.3のようになる．図2.1.3では，まず最初に質問中に与えられた述語is_parent_of(X, 'Jiro')という表現が，データベース中の最初の事実の述語is_parent_of('Ken', 'Taro')という表現とパターンマッチングが可能かどうかを調べる．パターンマッチングが可能であるとは，それらの表現が同一であるか，または，両者の表現中の変数に項を代入した後に，それらの項が同一になるように変数を具体化できるときである．この場合は，うまくいかないので，後戻りにより，別の候補をさがしにいく．次の候補では，パターンマッチングが成功し，答としてX='Taro'を返している．図2.1.3では，さらにユーザーが‘;’をタイプインして，強制的に後戻りを起こさせることで，別解を求めにいっている．

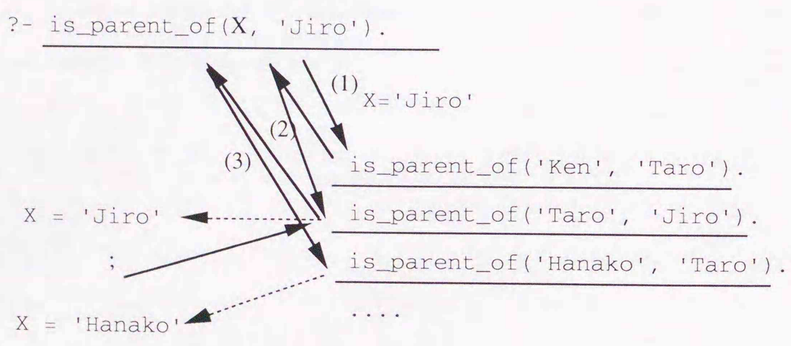


図2.1.3　変数を含んだ質問の実行

2.1.5　リスト

Prologプログラムにおいて，よく使われるデータ構造としてリストがある．リストは項を任意個並べた系列であり，

[] または，　[t_1, t_2, …, t_n]

という形をしている．ここでt_iは項である．リスト自身も項である．‘[]’は空リストを表すアトムであり，‘[t_1, t_2, …, t_n]’は‘.’を関数記号として持つ複合項である．‘[t_1, t_2, …, t_n]’は実際には，

.(t_1, .(t_2, .(…, .(t_n, [])…))

を表している．2引数の複合項とみたとき，第1引数のことをリストの先頭(head)，第2引数をリストの尾部(tail)と呼ぶ．また，‘[t_1, t_2, …, t_n]’を

[t_1,|[t_2, …, t_n]]

もしくは

[t_1, t_2, …|[t_1, …, t_n]]

とも記述することができる．

リストを用いたプログラムとして，memberという述語を示す．述語member(X,List)はXがリストListの要素であるという関係を表す．「XがリストListの要素である」ということは，

(1) XがリストListの先頭であるか，または，
(2) XがリストListの尾部の要素である

ときである．したがって，これをPrologプログラムで記述すると次のようになる．

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

最初の節が(1)の条件を表し，2番目の節が(2)の条件を表している．述語memberを使った例を以下に示す．

```
?- member(c,[a,b,c,d]).
yes
?- member(X,[a,b,c,d]).
X=a;
X=b;
X=c;
X=d;
no
```

最初の質問の実行は，定数cがリスト[a,b,c,d]の要素であるかという質問を行い，答としてyesが返ってきたものである．2番目の質問では，リスト[a,b,c,d]の要素xを求めるものである．リスト[a,b,c,d]の先頭のaが最初に求められている．‘;’を入力することで強制的にバックトラックをおこして別解が次々に求められている．最終的にもうこれ以上別解がなくなって，noが返ってきている．

2.1.6 カット

Prologには，実行の流れを制御する機構としてカットがある．カットはバックトラックを禁止する機構である．カットを用いることで，不効率なデータベースの探索や，論理的には正しい節でもPrologシステムの実行メカニズムのために無限ループに陥ってしまうような場合を防ぐことが可能になる．

たとえば，次のように定義される述語を考えてみる．


```
pass(Person,yes) :- test(Person,Score),Score>=80.
pass(Person,no)  :- test(Person,Score),Score<80.
```

述語pass(Person,Answer)はPersonがテストの成績が80点以上のときは，Answerにyesを，80点未満のときはAnswerにnoを返すものである．テストの成績が述語test(Person,Score)によりデータベース中に格納されているとしている．この述語の使い方として，「かならず第1引数には定数をいれて使う」と仮定する．すると，80点未満の人に関しては，最初の節の適用は必ず失敗し，2番目の節の適用が成功する．実行時には，同じ引数に対して述語test(Person,Score)による探索が2度行われる．問題の領域によっては，このような不効率な探索が生じる場合がある．

上記の例題に関しての不効率な探索を避けるために，最初の節の適用が失敗した人は，無条件に述語pass(Person,Answer)の第2引数のAnswerにnoを返せばよいと考えて，プログラムを次のように変更したとする．

```
pass(Person,yes) :- test(Person,Score),Score>=80.
pass(Person,no).
```

すると，80点以上をとっている人に関してバックトラックが生じた場合，次のようなおかしな答えが返ってくる．

```
?- pass(taro,X).
X=yes;
X=no;
no
```

これは，最初の節の適用により最初はテストが合格したというX=yesの答えが返ってきたにもかかわらず，強制的にバックトラックを行うと不合格であるというX=noの答えが返ってきてしまっているのが問題である．この原因は最初の節の適用が成功した場合でも，バックトラックが生じると2番目の節の適用が無条件に成功してしまう点にある．

カットを用いることで，このようなバックトラックを禁止することができる．カットを用いてプログラムを直したものは次のようになる．

```
pass(Person,yes) :- test(Person,Score),Score>=80,!.
pass(Person,no).
```

上記のプログラム中の‘!’がカットである．1番目の節の本体の部分を左から実行していき，それらが成功して‘!’の前まできたとする．実行は‘!’を越えて右側に進む．バックトラック時には，カットの機構により‘!’の左側を越えてまではバックトラックがおこらない．すなわち，上記の新しいプログラムに関しては，1番目の節の適用が成功した場合には，バックトラックが生じても2番目の節の適用はなされずにすむ．

カットを用いることで，Prologにおいて否定を扱うことができる．否定を扱う述語‘\+’は次のように定義できる．

```
\+ P :- call(P),!,fail.
\+ P.
```

ここで，述語callはその引数をゴールとして実行する述語である．述語failは必ず失敗する述語である．したがって，最初の節により成功するゴールに関しては‘no’を返し，失敗するゴールに関しては2番目の節により‘yes’を返す．ただし，この否定は論理的な否定表現とは一致しない．例えば，

```
?- \+ member(X,[a,b,c]).
```

は上述の定義によりmember(X,[a,b,c])を満足するようなXが存在すれば失敗する．それに対して，論理的な否定では「全てのXに対してmember(X,[a,b,c])は成り立たないか」ということをあらわす．論理的な否定と一致させるためには，\+ Pの実行はその中の変数が具体化されたあとに行う必要がある．そのような機構としては，Prolog-IIにおける述語freezeなどがある．

2.2 名詞

自然言語における名詞は，事物の名を表したり，事物の一般的な概念を表したりする．例えば，「太郎」という名詞は‘太郎という人’をあらわす名であるし，「人」という名詞は‘人’という一般的な概念を表す．

論理型プログラミング言語Prologにおける述語表現では，述語へ引数を与えるということは，決まった引数位置にその項を置くことである．すなわち，ユーザにとって，与える引数がどのようなものであるかということとの直接的な関連付けの根拠が乏しい位置情報により引数を指定する．

Prologで述語表現を使ってプログラムを行う際にユーザの頭の中では，各引数位置とそこに与えられる引数の種類，役割といった類のものとの暗黙的なマッピングを行っていると考えられる．拡張論理型プログラミング言語Vocalogでは，引数を位置情報と対応させて与えるのではなく，引数の種類，役割といった類のものを表すようなラベルを導入し，そのラベルと対応させて引数を与えることが可能である．このラベルのことをVocalogにおける名詞と呼ぶ．

「名前が‘Taro’という人の血液型Xを求めよ」という質問をPrologとVocalogで記述した例はそれぞれ次のようになる．

```
Prolog  ?-  person('Taro',_,_,_,X).
Vocalog ?-  [first_name:='Taro',blood_type:=X].
```

Prologの利用者は述語personが5引数であり，第1引数から第5引数がそれぞれ，名前，姓，性別，誕生日，血液型であることを知っている必要がある．そして，第1引数に定項'Taro'，第5引数に変数Xを与え，他の引数には無名変数‘_’を与えなくてはならない．無名変数は，システムが与えるユニークな変数であり，同じ字面であっても別々の位置に現れれば，システムにより別々のユニークな変数が割り当てられる．それに対して，Vocalogの利用者は，リスト中で，first_nameとblood_typeという名詞と対応さ

せて，それぞれ定項'Taro'と変数Xを与えるだけでよい．

Prologでは，$p(t_1,\ldots,t_m)$のように述語を記述するが，述語名pにより，引数間の関係は示せるが，各引数の役割は明示されない．Prologでは，固定された順番で，引数を与えなくてはならない．述語がいくつの引数を取り，各引数位置にどのような引数がくるかは，利用者の責任において管理しなくてはならない．

Vocalogでは，述語の引数の役割を表すラベルとして，基本名詞というものを導入している．m引数の述語pの第1引数から第m引数の役割を表すものとして，基本名詞$n_1,\ldots,n_m$を導入することにより，述語$p(t_1,\ldots,t_m)$を$[n_1:=t_1,\ldots,n_m:=t_m]$と記述できる．$[n_1:=t_1,\ldots,n_m:=t_m]$をリスト述語と呼ぶ．リスト述語において名詞を利用することにより，

(1) 基本名詞と対応づけて引数を与えるので，リスト中の順番は任意でよい．
(2) 省略された引数には，システムが自動的にユニークな変数を与えるので，指定したい引数だけを与えればよい．

というように表現が簡便になり，かつPrologにおいて利用者が述語の引数と引数位置の役割を覚えておかなくてはいけないという欠点が改善されている．

2.3 形容詞

自然言語における形容詞は事物の性質，様子，状態を表す．たとえば，「白い」という形容詞は白色であることをあらわし，「白い花」のように名詞「花」を修飾することで‘白い色の花’を示すことができる．また，格助詞の「の」は上の語を下の名詞（体言）に結びつけて連体修飾語（形容詞的修飾語）として，所有，所属，関係の起点などを示す．例えば，「父の靴」では‘父の所有物であるところの靴’を表している．

論理型プログラミング言語Prologでは，このような事物の性質，様子，状態を表すものもまた述語として表現される．事物を表す述語の引数と，事物の性質，様子，状態を表す述語の引数との間で共有変数を用いることで，どの事物がどのような性質，様子，状態なのかを示すことが可能である．しかし，ここでも前節で説明したのと同様に述語の引数位置により引数として与えられる項の役割が暗黙的に示されるだけであるという問題点がある．複数の述語を共有変数で関係づけて利用する際に，各引数位置にどのような引数が与えられるかをユーザが把握している必要がある．拡張論理型プログラミング言語Vocalogでは，対象の性質，対象間の関係概念をジェネリックに定義できるものとして形容詞を導入している．形容詞で前節で示した名詞を修飾することで，対象の性質，対象間の関係をその名詞で表される対象に適用することが可能となる．

「名前が'Taro'という人の，親の血液型を求めよ」という質問をPrologとVocalogで記述した例はそれぞれ次のようになり，答えは変数Zに求めることができる．

```
Prolog   ?-  person('Taro',U,_,_,_),
             is_parent_of([X,Y],['Taro',U]),
             person(X,Y,_,_,Z).
Vocalog  ?-  [first_name:='Taro', parent@blood_type:=Z].
```

Prologの利用者は，2引数の述語is_parent_ofが親子関係を表していて，第1引数が親の名前と姓のリストで，第2引数が子供の名前と姓のリストであることを知っている必要がある．さらに共有変数を用いて，述語personに与える引数と関係づけを行わなければならない．Vocalogでは，リスト中で，名詞first_nameと対応させて引数'Taro'を与え，形容詞parentで名詞blood_typeをparent@blood_typeのように修飾したものと引数Zを対応づけて与えるだけでよい．

Prologでは，共有変数を用いた複数の述語の連言により，複雑な関係を表現できるが，利用者はどの述語のどの引数と，どの述語のどの引数とを共有変数を用いて関係づけるかを知っている必要がある．

Vocalogでは，対象間の関係概念がジェネリックに形容詞を用いて定義される．リスト述語中で，名詞を形容詞で修飾するだけで対象間に成り立つべき条件を付加できるので，表現が簡便になるという利点がある．

2.4 語の合成

2.2節と2.3節では，語彙を導入することによる利点として，リスト述語で名詞や形容詞という語彙を用いることで表現が簡便になることを述べた．語彙を導入することによる別の利点として，語彙レベルで複雑な概念や関係概念をジェネリックに記述できるというものがある．語構成子というものを使って名詞や形容詞を合成した名詞句や形容詞句を用いて，複雑な概念や関係概念をジェネリックに記述できる．名詞句や形容詞句はリスト述語で利用することができる．

例えば，親であるという関係を表す形容詞parent，子供であるという関係を表す形容詞child，自分自身であるという関係を表す形容詞selfがあれば，兄弟であるという関係は，「親の子供であって，自分自身ではない」というものであるので，‘(child:parent)&(-self)’と記述できる．そして，この形容詞句を用いて，「'Taro'という名前の人の兄弟の名前Xを求めよ」という質問を次のように記述できる．

```
?- [first_name:='Taro',
        ((child:parent)&(-self))@first_name:=X].
```

2.5 語彙構築

Vocalogでは，基本名詞と基本形容詞を述語と対応させて導入する．さらにこれらの基本語彙を組み合わせた名詞句や形容詞句に対して名前付けを行い，この名前を新たに名詞や形容詞として用いることができる．これを語彙構築と呼び，語彙構築で導入された名詞と形容詞をそれぞれ派生名詞と派生形容詞と呼ぶ．

Vocalogにおいて，導入された語彙は基本語彙と派生語彙に分けられる．基本語彙はプログラムと対応させて導入されるのに対し，派生語彙は基本語彙をもとにレキシカルに

定義されるのでプログラムとは独立である．　語彙構築により派生語彙を導入する利点として，

(1)　複雑な概念や関係概念のジェネリックな定義が可能
(2)　複数プログラムでの同一語彙の利用が容易

というものがある．(1)は語彙レベルで複雑な概念や関係概念を定義できるからである．(2)は基本語彙とプログラムとの対応だけをとりさえすれば派生語彙を利用できるからである．

2.6 まとめ

自然言語における語彙は，人と人との間のコミュニケーションや，概念構築において重要な役割を担っている．本章では，自然言語における語彙の持つ特徴をPrologに導入したシステムVocalogの概略を示した．

Vocalogでは，名詞と形容詞というものを使うことができる．名詞を使えることにより，Prologにおける述語表現の問題点が解決される．すなわち，第一に，述語の引数の数を覚えておく必要がなくなる．第二に，各引数位置がどのような役割をもっているかということを利用者が覚えておく必要がなくなる．形容詞が使えることにより，どのようにして述語を組み合わせて使うのかということを覚えておく必要がなくなる．言い換えると，どのようにして述語を組み合わせるのかという使い方をプログラムの部品とすることが可能となっている．

さらに，Vocalogでは，語構成子というものを使って語を合成するだけで，複雑な概念や対象間の関係概念を記述することができる．Prologにおいて，連言や共有変数を用いて表していた複雑な概念や対象間の関係概念を，述語表現という構造とは独立に記述することが可能となっている．

論理型プログラミング言語における項を拡張して，レコード的な構造をもつオブジェクトを扱えるようにしたものとして，KBL[5]，LOGIN[6]，CIL[7]，CRL[8]などが提案されている．データベースの分野では，コンプレックス・オブジェクトを扱うための論理体系として，O-logic[15]，F-logic[14]，C-logic[13]などが提案されている．例えば，KBLでは次のような表現形式が用いられる．

```
person( id⇒name;
        domicle⇒address(city⇒cityname);
        father⇒person    ).
```

ここで，personはオブジェクトのタイプを表す．id,domicle,fatherはpersonタイプのオブジェクトが持つ属性を表す．name,address(city⇒cityname),personは，それぞれ属性id, domicle, fatherがその値として持つオブジェクトのタイプを表している．このように，オブジェクトを表現する際に，Prologでは述語表現を使って，プログラマーの責任において引数位置を決めていたが，上述のシステムでは引数の属性とその値を使ったレコード表現が可能である．

これらのシステムには，実際のコンプレックス・オブジェクト中で，オブジェクト間の制約を記述する機能がある．それに対して，Vocalogでは，オブジェクト間の制約を，実際のコンプレックス・オブジェクトとは独立に記述できる形容詞を導入している．形容詞を組み合わせた形容詞句という形で，複雑なオブジェクト間の制約をジェネリックに記述できる．

第3章　拡張論理型プログラミング言語 Vocalog

この章では，前節で示した語彙のPrologへの導入方法について説明する．導入された語彙は，リスト述語において利用することができる．以下の節では，Vocalogにおける語彙に含まれる語句のシンタックスとリスト述語のシンタックスを示し，語彙の定義方法とリスト述語の評価規則を示す．そして，Vocalogインタープリタを実現している方法に関して説明を行う．最後にまとめを述べる．

3.1　Vocalogのシンタックス

Vocalogにおける語彙には名詞，形容詞，名詞句，形容詞句がある．名詞句と形容詞句は語構成子を用いて名詞と形容詞を組み合わせたものである．名詞句を作るための語構成子としては，代入(':=')，論理演算('+', '&', '-')，グループ・バイ演算('/')，射影演算('<<')，関数，リスト化，形容詞句がある．形容詞句を作るための語構成子としては，逆演算('~')，論理演算('+', '&', '-')，合成演算(':')，推移閉包演算('*')がある．名詞句と形容詞句のシンタックスを図3.1のように定義する．

名詞と形容詞は，さらに基本名詞と派生名詞，基本形容詞と派生形容詞に分けられる．これらについては次節以降で説明する．

リスト述語は，語彙を述語定義や質問で利用するためのものである．リスト述語は，名詞句からなるリストであり，$N_1,\ldots,N_k$を名詞句とすると，リスト述語は$[N_1,\ldots,N_k]$という形になる．

```
<名詞句> ::=    <名詞>
                ¦<名詞句>:=<項>
                ¦<名詞句>+<名詞句>
                ¦<名詞句>&<名詞句>
                ¦ -<名詞句>
                ¦<名詞句> / <名詞句>
                ¦<名詞句> << <名詞句>
                ¦<関数名>(<名詞句>,…, <名詞句>)
                ¦ [] ¦ [<名詞句>,…,<名詞句>]
                ¦<形容詞句>@<名詞句>
<形容詞句> ::= <形容詞>
                ¦<形容詞句>~
                ¦<形容詞句>+<形容詞句>
                ¦<形容詞句>&<形容詞句>
                ¦ -<形容詞句>
                ¦<形容詞句> : <形容詞句>
                ¦<形容詞句>*
```

図3.1　名詞句と形容詞句のシンタックス

3.2　基本名詞

基本名詞は，述語の引数の役割を表すラベルとして導入している．そして，述語$p(t_1,\ldots,t_m)$のかわりに，$[n_1:=t_1,\ldots,n_m:=t_m]$という形の表現を用いることができる．$n_i$が述語の各引数位置の役割を表すラベル，すなわち基本名詞である．各基本名詞を属性と考え，基本名詞の集合は対象の概念（クラス，タイプ）を表しているとみなすことができる．

m引数の述語pの第1引数から第m引数の役割を表すラベルとして基本名詞$n_1,\ldots,n_m$を導入するには次の形の式を用いる．

$$\text{basic\_predicate}(p)::=[\ n_1,\ldots,\ n_m\ ]. \tag{3.2.1}$$

これにより，述語$p(t_1,\ldots,t_m)$を$[n_1:=t_1,\ldots,\ n_m:=t_m]$と記述できる．あるいは，表現$[n_1:=t_1,\ldots,\ n_m:=t_m]$を直接節の頭部で用いることで，基本名詞$n_1,\ldots,n_m$を導入することも可能である．

$$\begin{aligned} [n_1:=t_1,\cdots,n_m:=t_m] \ &:-\ Q_1,\cdots,Q_k \\ &\cdots \\ [n_1:=s_1,\cdots,n_m:=s_m] \ &:-\ R_1,\cdots,R_l \end{aligned} \tag{3.2.2}$$

(3.2.2)式は，次のような述語定義と，(3.2.1)式の基本名詞の定義が組み合わされたものである．

$$\begin{aligned} p(t_1,\cdots,t_m) \ &:-\ Q_1,\cdots,Q_k \\ &\cdots \\ p(s_1,\cdots,s_m) \ &:-\ R_1,\cdots,R_l \end{aligned} \tag{3.2.3}$$

Vocalogシステムでは，(3.2.1)式を次の形の事実に変換してシステムに格納する．

varg(n_1 , x_1, $p(x_1,\ldots, x_m)$).

…　　(3.2.4)

varg(n_m , x_m , $p(x_1,\ldots, x_m)$).

ただし，$x_1,\ldots,x_m$は互いに異なる変数である．

このようにして定義された基本名詞を用いたリスト述語では，引数は基本名詞と対応させて与えればよいので，リスト中の順番は任意でよい．リスト中で，m個の基本名詞のうちi個だけが使われ，i個の引数だけが指定されたときは，残りの$(m - i)$個の引数には，自動的にユニークな変数が割り当てられる．基本名詞を用いたリスト述語の評価規則は次のようになる．

$[N_1:=X_1,\ldots,N_k:=X_k]$

⇒　　varg(N_1,X_1,P),…,varg(N_k,X_k,P), call(P).　　(3.2.5)

[]

⇒　　true.　　(3.2.6)

ここで，N_i (i =1,…, k)は基本名詞，X_i (i =1,…, k)は変数である．call(P)は，Pを述語として評価する述語である．trueは常に成功する述語である．(3.2.5)式では，述語varg(N_i,X_i,P)により，基本名詞N_iと対応づけて与えられた項X_iを述語表現Pの適当な位置に割り当てている．これにより，各引数位置に適当な引数が与えられた述語表現Pが得られる．そして，call(P)により得られた述語表現をPrologの述語として評価する．(3.2.6)式は，要素を含まないリスト[]のリスト述語としての評価は必ず成功することを示している．

例　次のような述語が定義されていたとする．

```
person(taro, yamada, male, "1940.7.14", tokyo).
person(ken, yamada, male, "1965.9.21", tokyo).
person(hanako, sato, female, "1970.3.10", sapporo).
```

この述語の各引数位置に対応させて，次のような基本名詞を定義する．


```
basic_predicate(person) ::=
    [ first_name, last_name, sex, birthday, birthplace ].
```

これにより，Vocalogシステムには次のような情報が格納される．

```
varg(first_name,X1,person(X1,_,_,_,_)).
varg(last_name,X2,person(_,X2,_,_,_)).
varg(sex,X3,person(_,_,X3,_,_)).
varg(birthday,X4,person(_,_,_,X4,_)).
varg(birthplace,X5,person(_,_,_,_,X5)).
```

このとき，これらの基本名詞を用いて，「taroという名前の人の誕生日はいつですか？」という質問は，

```
?- [first_nmae:=taro, birthday:=X].
```

または，

```
?- [birthday:=X, first_nmae:=taro].
```

のように記述できる．これらの質問の評価過程をそれぞれ，図3.2.1，図3.2.2に示す．これらの場合は，同一の述語の各引数位置に対応づけられた基本名詞をリスト述語で用いている．基本名詞の表す引数位置に，その基本名詞と対で与えられた引数が与えられている．これらの質問の実行は成功して，X="1940.7.14"を答えとして返す．

また，ここで定義した以外の基本名詞foodを基本名詞first_nameと一緒に用いた質問:

```
?- [first_nmae:=taro, food:=Y].
```

の評価過程を図3.2.3に示す．この場合，基本名詞foodと基本名詞first_nameは同一述語の引数位置に対応づけられていない．したがって，この質問の実行は失敗し，noを答えとして返す．

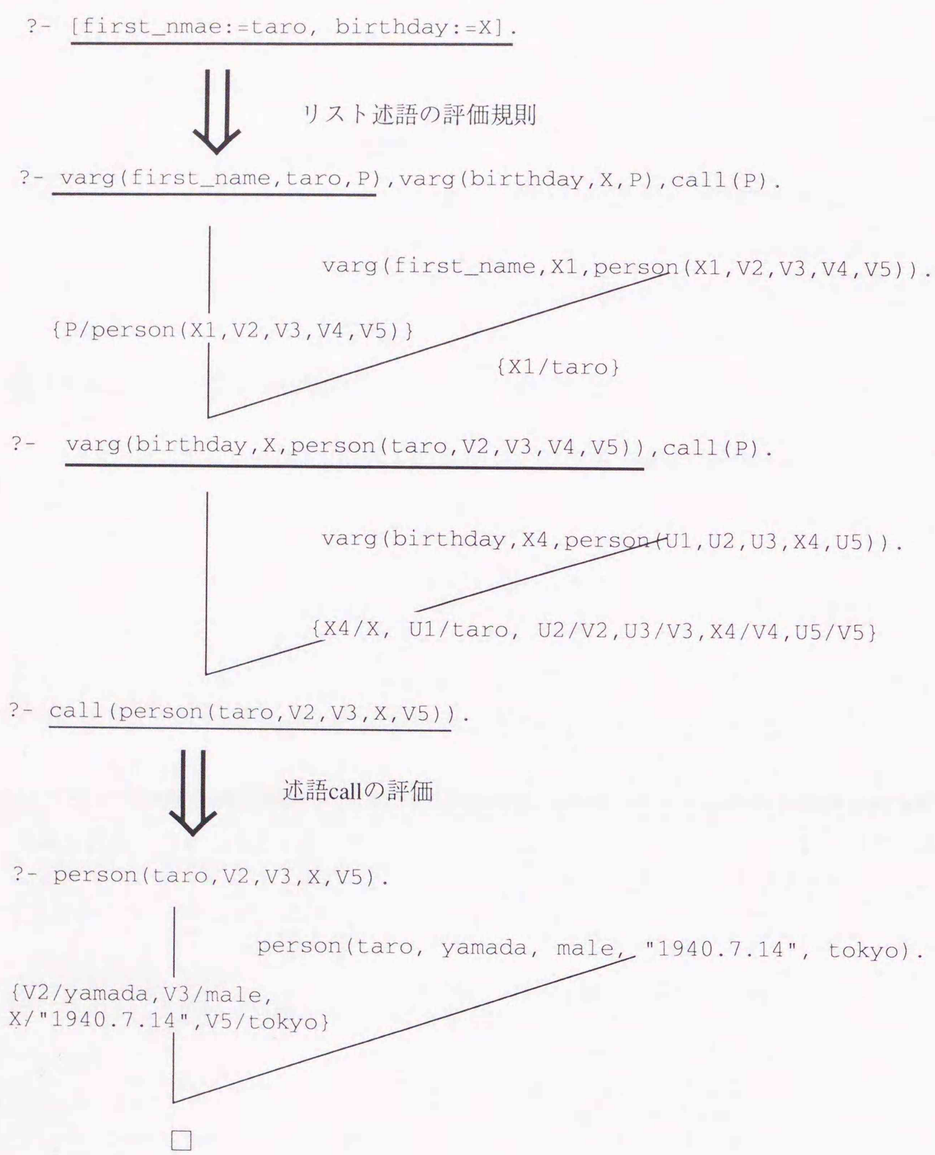


図3.2.1　基本名詞を用いたリスト述語の評価過程の例1

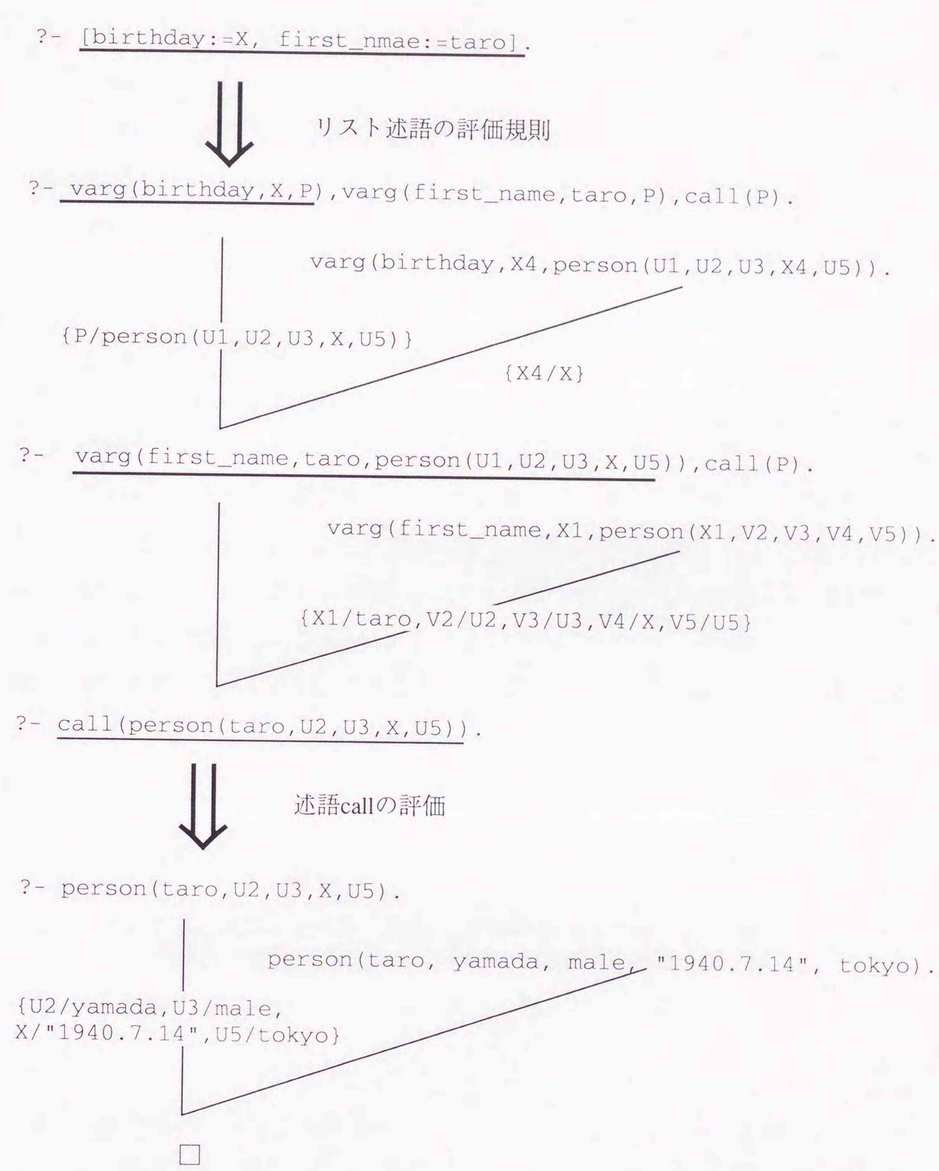


図3.2.2　基本名詞を用いたリスト述語の評価過程の例2

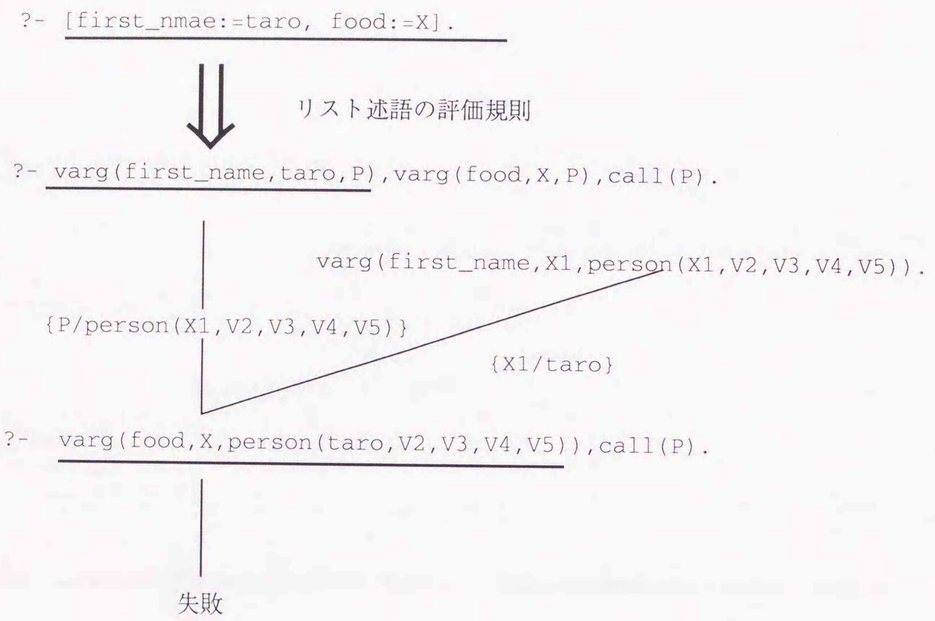


図3.2.3　基本名詞を用いたリスト述語の評価過程の例3

3.3　名詞句

この節では，形容詞句以外の語構成子について説明し，それらを用いた名詞句を含むリスト述語がどのようにして評価されるかを示す．

●代入

代入は，名詞句の取る値を指定するのに用いる．前節では基本名詞の取る値を指定するのに用いていた．すなわち，前節ではリスト述語中で，基本名詞と対応させて必ず値を与えていた．代入を語構成子とみなし，代入を用いない基本名詞または名詞句がリスト述語中に現れることも許す．この場合，基本名詞または名詞句の取る値は特に指定されていないとみなし，システムがユニークな変数を与えて評価を行う．これにより，単に基本名詞が表す属性を対象が持つことをより直接的に表現できる．すなわち，基本名詞と対応させてユニークな変数を与えなくても，ただ基本名詞をリスト述語中に記述するだけでよい．たとえば，対象がmiddle_nameという属性を持ってさえいればよいことは，

```
[ middle_name:=_, last_name:=X, first_name:=ken]
```

のように基本名詞と対応させて無名変数‘_’を与えなくても，ただ単に，

```
[ middle_name, last_name:=X, first_name:=ken]
```

とすればよい．

また，基本名詞の取る値を特定の値にしたものを名詞句とすることができる．さらに，代入を複数回重ねて用いることも許す．これにより，基本名詞の取る値を部分的に決めていくことができる．たとえば，

```
[ (n1:=f(a,X)):=f(Y,b),n2:=c]
```

というリスト述語では，基本名詞n1のとる値は，n1:=f(a,X)により，まずf(a,X)である．さらに，(n1:=f(a,X)):=f(Y,b)により，f(Y,b)でもある．すなわち，それぞれの代入において，基本名詞n1のとる値が部分的に決定され，全体としてはf(a,b)という値を持つ．

代入をに関する評価規則を以下に示す．

(1) $[N_1,\ldots, N_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, N_i :=t_i ,\ldots, N_k]$.

(2) $[N_1,\ldots, (N_i :=t):=t_i ,\ldots, N_k]$

⇒ $t=t_i , [N_1,\ldots, N_i :=t_i ,\ldots, N_k]$.

(1)式は，代入を用いない名詞句 N_i を含むリスト述語は，N_i を $N_i:=t_i$ で置き換えたリスト述語を評価することを示す．ただし，t_i はシステムが与えるユニークな変数である．(2)式は，代入を二重に用いた名詞句 $(N_i:=t):=t_i$ を含むリスト述語は，$t=t_i$ と，$(N_i:=t):=t_i$ を $N_i:=t_i$ で置き換えたリスト述語 $[N_1,\ldots, N_i :=t_i ,\ldots, N_k]$ を評価することを示す．

例

代入を用いない場合の評価例をあげる．

```
[ middle_name, last_name:=X, first_name:=ken]
```

⇒ `[ middle_name:=_, last_name:=X, first_name:=ken]`

ここでは，名詞middle_nameが代入により値を指示されていない．値として，無名変数が与えられて評価が続けられる．

代入を二重に用いた場合の評価例をあげる．

```
[ (n1:=f(a,X)):=f(Y,b),n2:=c]
```

⇒ `f(a,X)=f(Y,b),[ n1:=f(a,X),n2:=c]`

ここでは，(n1:=f(a,X)):=f(Y,b)の部分で代入が二重に用いられている．値として与えられているものが等しいものであるという条件を付加し，外側の代入による値の指定を取り除いて評価が続けられる．

●論理演算

論理演算としては，論理和（＋），論理積（＆），否定（-）がある．

N_1とN_2を名詞句とすると，論理和を用いた名詞句N_1+N_2は名詞句N_1または名詞句N_2を表す．これにより，対象が属性N_1と属性N_2のうちのどちらかをもてばよいということを示すことができる．すなわち，リスト述語中で名詞句N_1+N_2の取る値は，それをN_1またはN_2に置き換えたリスト述語でそれらの名詞句が取る値である．

N_1とN_2を名詞句とすると，論理積を用いた名詞句N_1&N_2は名詞句N_1でありかつ名詞句N_2であることを表す．すなわち，リスト述語中で名詞句N_1&N_2が取る値は，N_1&N_2をN_1に置き換えたリスト述語中でそのN_1が取る値であり，かつN_1&N_2をN_2に置き換えたリスト述語中でそのN_2が取る値でもある．

Nを名詞句とすると，否定を用いた名詞句-Nは，属性Nがないということを表す．すなわち，-Nを取り除いたリスト述語が成り立ち，-NをNに置き換えたリスト述語が成り立たない．

論理演算を用いた名詞句を含むリスト述語の評価規則を以下に示す．

(1) $[N_1,\ldots, (N_{i1}+N_{i2}):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots,N_{i1}:=t_i ,\ldots, N_k]$.

(2) $[N_1,\ldots, (N_{i1}+N_{i2}):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, N_{i2}:=t_i ,\ldots, N_k]$.

(3) $[N_1,\ldots, (N_{i1}\&N_{i2}):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, N_{i1}:=t_i ,\ldots, N_k] , [N_1,\ldots, N_{i2}:=t_i ,\ldots, N_k]$.

(4) $[N_1,\ldots, (-N_i):=t_i ,\ldots, N_k]$

$$\Rightarrow \quad [N_1,\ldots, N_{i-1}, N_{i+1},\ldots, N_k],$$
$$\text{not}([N_1,\ldots, N_{i-1}, N_i := t_i, N_{i+1},\ldots, N_k]).$$

(1)式と(2)式は，和を用いた名詞句$(N_{i1}+N_{i2}):=t_i$を含むリスト述語は，$(N_{i1}+N_{i2}):=t_i$を$N_{i1}:=t_i$または$N_{i2}:=t_i$で置き換えたリスト述語として評価されることを示している．

(3)式は，積を用いた名詞句$(N_{i1}\&N_{i2}):=t_i$を含むリスト述語は，$(N_{i1}\&N_{i2}):=t_i$を$N_{i1}:=t_i$に置き換えたリスト述語と，$(N_{i1}\&N_{i2}):=t_i$を$N_{i2}:=t_i$に置き換えたリスト述語の連言として評価されることを示している．

否定を用いた名詞句$(-N_i):=t_i$を含むリスト述語は，$(-N_i):=t_i$を取り除いたリスト述語と$(-N_i):=t_i$を$N_i:=t_i$で置き換えたリスト述語の否定の連言として評価されることが(4)式により示されている．

例

(first_name:=taro)+(first_name:=jiro)という名詞句は，名前がtaroまたはjiroということを表す．この名詞句を用いたリスト述語を使った質問:

?- [(first_name:=taro)+(first_name:=jiro), last_name:=X].

が成功した場合は，変数Xに求められるのは，名前がtaroまたはjiroである人のlast_nameである．

birthplace&present_addressという名詞句は，出生地と現在すんでいるところが同じであるということをあらわす．この名詞句を用いたリスト述語を使った質問:

?-[(birthplace&present_address), first_name:=ken,birthday:=Y].

が成功した場合，名前がkenという人で出生地と現在すんでいるところが同じ人の誕生日が変数Yに求められる．

-middle_nameという名詞句は，対象がmiddle_nameという属性を持たないことを表す．この名詞句を用いたリスト述語を使った質問:

?- [-middle_name, first_name:=ken, birthday:=Z].

が成功した場合，ミドルネームを持たない，kenという名前の人の誕生日が変数Zに求められる．

それに対して，-(middle_name:=fitzgerald)という名詞句は，対象がmiddle_name:=fitzgeraldという属性，すなわち，ミドルネームがfitzgeraldではないということを表している．この場合，対象は，ミドルネームがfitzgerald以外であれば，もっていてもかまわない．

●グループ・バイ演算

グループ・バイ演算（/）により，属性のとる値の集合をその値としてとるような属性を記述できる．N_1, N_2を名詞句とすると，グループ・バイ演算を用いた名詞句N_1/N_2は，名詞句N_2の取る値が同じであるような，名詞句N_1の値の集合である．そして，その時の名詞句N_2の値は，リスト述語中で決定される他の属性との関係で決まる．

例えば，first_name/last_nameという名詞句の取る値は，last_nameの値が同じである対象のfirst_nameの値の集合である．そして，[birthplace:=sapporo, (first_name/last_name):=X]というリスト述語は，出生地がsapporoであるような人で，last_nameの値が同であるような人のfirst_nameの集合がXであるということを表している．

グループ・バイ演算を用いた名詞句を含むリスト述語の評価規則を以下に示す．

$$[N_1,\ldots, (N_{i1}/N_{i2}):=t_i,\ldots, N_k]$$
$$\Rightarrow \quad [N_1,\ldots, N_{i2}:=t_{i2},\ldots, N_k],$$
$$\text{setof}(t_{i1}, [N_{i1}:=t_{i1}, N_{i2}:=t_{i2}], t_i).$$

この評価規則は，グループ・バイ演算を用いた名詞句$(N_{i1}/N_{i2}):=t_i$を含むリスト述語は，$(N_{i1}/N_{i2}):=t_i$を$N_{i2}:=t_{i2}$で置き換えたリスト述語と，$t_i=\{t_{i1}|[N_{i1}:=t_{i1}, N_{i2}:=t_{i2}]\}$の連言であることを示している．ただし，$t_{i1}, t_{i2}$はシステムが与えるユニークな変数である．実際には，

集合はリストを用いて表現されている．setof(t_{i1},[N_{i1}:=t_{i1},N_{i2}:=t_{i2}],t_i)は，t_i={t_{i1}|[N_{i1}:=t_{i1},N_{i2}:=t_{i2}]}なるt_iを求める述語である．

例

child/personという名詞句は，personの値が同じである人のchildの値の集合を値に取る．すなわち，ある人の子供の集合を値に取る．この名詞句を含んだリスト述語を用いた質問:

?- [(child/person):=X, first_name:=taro, birthplace:=Y].

の評価過程を図3.3.1に示す．

図3.3.1では，まずリスト述語からグループ・バイ演算を用いた名詞句(child/person):=Xを取り除き，そのかわり，person:=Uという名詞句を加えている．ここで，personは名詞句(child/person):=Xの‘/’の右側の名詞句であり，変数Uはシステムが自動的に与えるユニークな変数である．つぎに，[child:=V,person:=U]なるリスト述語を構成している．ここで，childとpersonはそれぞれグループ・バイ演算の演算子‘/’の左側と右側に与えられた名詞句である．そして，述語setofの第2引数としてリスト述語[child:=V,person:=U]を与え，第1引数で与えた変数Vが取る値の集合を，第3引数の変数Xに求める．

●射影演算

射影演算（<<）は，属性N_1に対して他の属性N_2を付加するものである．これにより，対象が単にその属性N_1を持つだけではなく，同時に属性N_2をも持つことを示すことができる．N_1,N_2を名詞句とすると，N_1<<N_2という射影演算を用いた名詞句は，対象がN_1という属性に加えてN_2という属性を持つことを示している．ただし，この名詞句の値としてはN_1の取る値である．

射影演算を用いた名詞句を含むリスト述語の評価規則を以下に示す．

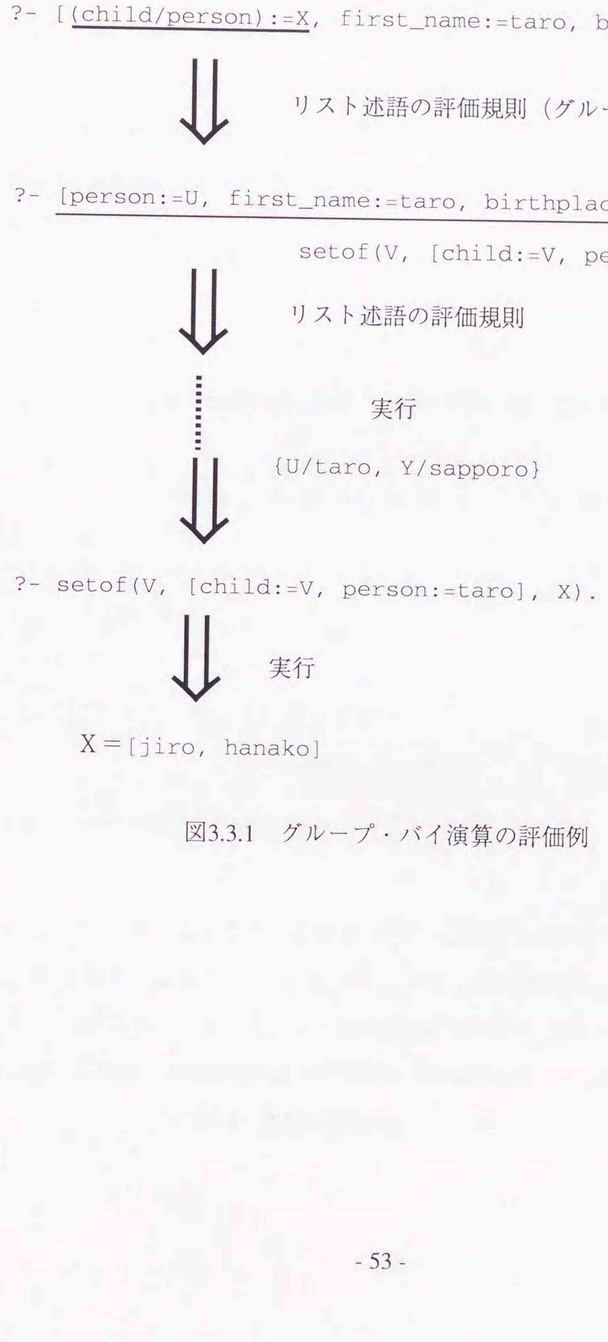


図3.3.1　グループ・バイ演算の評価例

$[N_1,\ldots, (N_{i1}<<N_{i2}):=t_i ,\ldots, N_k]$

$\Rightarrow$ $[N_1,\ldots, N_{i1}:=t_i , N_{i2},\ldots, N_k]$.

射影演算を用いた名詞句$(N_{i1}<<N_{i2}):=t_i$を含むリスト述語は，$(N_{i1}<<N_{i2}):=t_i$を取り除き，$N_{i1}:=t_i$とN_{i2}を加えたリスト述語として評価される．

例

first_name<<(last_name:='Yamada')という名詞句は，last_nameが'Yamada'である人のfirst_nameの値だけを取ることができる．この名詞句を含んだリスト述語を用いた質問:

?- [(first_name<<(last_name:='Yamada')):=X, birthplace:=sapporo].

の評価過程は次のようになる．

?- [(first_name<<(last_name:='Yamada')):=X, birthplace:=sapporo].

⇓

?- [first_name:=X, last_name:='Yamada',birthplace:=sapporo].

リスト述語中から射影演算を含んだ名詞句(first_name<<(last_name:='Yamada')):=Xが取り除かれている．そのかわりに，'<<'の左側の名詞句first_nameに射影演算を用いた名詞句と対応づけられていた変数Xとを結びつけた名詞句first_name:=Xと'<<'の右側の名詞句がリスト述語の並びに追加されている．この質問が成功した場合，変数Xには出生地がsapporoでlast_nameが'Yamada'である人のfirst_nameが得られる．

●関数

名詞句に関数を適用することができ，それもまた名詞句である．名詞句$N_{i1},\ldots,N_{im}$への関数fの適用を$f(N_{i1},\ldots,N_{im})$と記述する．$f(N_{i1},\ldots,N_{im})$も名詞句である．名詞句$f(N_{i1},\ldots,N_{im})$の取る値は，名詞句$N_{i1},\ldots,N_{im}$が取る値に関数fを適用した値である．名詞句$N_{i1},\ldots,N_{im}$が取る値は，名詞句$f(N_{i1},\ldots,N_{im})$が使われるリスト述語中で決定される．

関数を用いた名詞句を含むリスト述語の評価規則を以下に示す．

$[N_1,\ldots, f(N_{i1},\ldots, N_{im}):=t_i ,\ldots, N_k]$

$\Rightarrow$ $[N_1,\ldots, N_{i1}:=t_{i1},\ldots, N_{im}:=t_{im} ,\ldots, N_k]$,

evaluate_function($f(t_{i1},\ldots,t_{im}),t_i$).

名詞句$f(N_{i1},\ldots,N_{im}):=t_i$を含むリスト述語は，$f(N_{i1},\ldots,N_{im}):=t_i$を取り除き，$N_{i1}:=t_{i1}$,…, $N_{im}:=t_{im}$を加えたリスト述語と，t_{i1} , …, t_{im}を関数fに適用した値$f(t_{i1}$, …, $t_{im})$がt_iであることとの連言として評価される．ただし，t_{i1}, …, t_{im}はシステムが与えるユニークな変数である．また，evaluate_function($f(t_{i1}, \ldots, t_{im}), t_i$)は，$f(t_{i1}, \ldots, t_{im})$を評価してその値を$t_i$に求める述語である．

例

count(X)という関数が，リストX の要素数を返す関数だとする．count(child/person)の値は子供の数になる．この名詞句を含んだリスト述語を用いた質問:

?- [first_name:=taro, count(child/person):=X].

の評価過程を図3.3.2に示す．まず，リスト述語から，関数を含んだ名詞句が取り除かれ，そのかわり，関数が適用されていた名詞句child/personが追加されている．名詞句の値としては，システムがユニークな変数Uを与えている．このリスト述語が評価されると，変数Uにtaroがfirst_nameである人の子供の名前からなるリスト[ken,jiro]が得られる．述語evaluate_functionは，第1引数に与えられた関数名countと，第2引数に与えられたリスト[ken,jiro]から，リスト[ken,jiro]に関数countを適用した値を第3引数の変数Xに求める．リストの長さとして2が変数Xの値として求められる．

```
?- [(first_name:=taro, count(child/person):=X].
```

⇓ リスト述語の評価規則（関数）

```
?- [first_name:=taro, (child/person):=U],
                    evaluate_function(count,U,X).
```

⇓ リスト述語の評価規則(グループ・バイ演算)

実行

{U/[ken,jiro]}

⇓

```
?- evaluate_function(count,[ken,jiro],X).
```

⇓ count([ken,jiro]) =2
リスト[ken,jiro]の長さを求める.

```
X=2
```

図3.3.2　関数を含んだリスト述語の評価例

●リスト化

リスト化により複数の名詞句の並びが一つの名詞句となる.

リスト化を用いた名詞句を含むリスト述語の評価規則を以下に示す.

(1) $[N_1,\ldots,[N_{i1},\ldots,N_{im}]:=t_i,\ldots,N_k]$

⇒ $[N_1,\ldots,N_{i1}:=t_{i1},\ldots,N_{im}:=t_{im},\ldots,N_k]$, $t_i=[t_{i1},\ldots,t_{im}]$.

(2) $[N_1,\ldots,[]:=t_i,\ldots,N_k]$

⇒ $[N_1,\ldots,N_{i-1}:=t_{i-1},N_{i+1}:=t_{i+1},\ldots,N_k]$, $t_i=[]$.

(1)式は名詞句$[N_{i1},\ldots,N_{im}]:=t_i$を含むリスト述語は，$[N_{i1},\ldots,N_{im}]:=t_i$を取り除いて，$N_{i1}:=t_{i1},\ldots,N_{im}:=t_{im}$を加えたリスト述語と$t_i=[t_{i1},\ldots,t_{im}]$の連言として評価されることを示す．ただし，$t_{i1},\ldots,t_{im}$はシステムが与えるユニークな変数である.

(2)式は名詞句$[]:=t_i$を含むリスト述語は，$[]:=t_i$を取り除いたリスト述語と$t_i=[]$の連言として評価されることを示している.

例えば，[first_name,last_name]という名詞句の値は，first_nameの値とlast_nameの値からなるリストである.

3.4　基本形容詞

Wを定義済みの名詞と形容詞から作られるすべての名詞句の集合とする．基本形容詞Aは

basic_adjective(A) ::=

[source(N_s):=X_s, destination(N_d):=Y_d, p(X_s,Y_d)]. (3.4.1)

のような形の式で定義する．W_1とW_2をWのコピーとする．図3.4.1に示すように，基本形容詞Aは，W_1とW_2との間に，W_1の名詞句N_sの値X_sとW_2の名詞句N_dの値Y_dとの間に述語で表される関係p(X_s , Y_d)が成り立っているという制約を表している．W_2の名詞句Nに対して，A@Nのようにラベルをつけて表すことにより，W_1とW_2の語彙を区別でき，WからW´(=W∪A@W, ただし A@W = {A@N | N∈W})へと使用可能な語彙が増加する．ラベルAは名詞句Nを修飾することで制約を加えており，自然言語における形容詞に相当する．‘@’は基本形容詞とそれが修飾している名詞句とをVocalogの構文において分離するための区切り子である．W_1の側をソース側の世界あるいはオブジェクトと呼ぶ．W_2の側をデスティネーション側の世界あるいはオブジェクトと呼ぶ．

Vocalogシステムは，(3.4.1)式のような基本形容詞の定義を次のような事実に変換してシステムに格納する．

adj_sdc(A, N_s :=X_s , N_d :=Y_d , p(X_s , Y_d)) .

このように定義された基本形容詞を用いた名詞句(A@N_i):=t_iを含むリスト述語は，(A@N_i):=t_iをN_s:=Xで置き換えたリスト述語と，述語p(X,Y)と，リスト述語[N_d:=Y , N_i :=t_i]の三つの連言として評価される．評価規則を以下に示す．

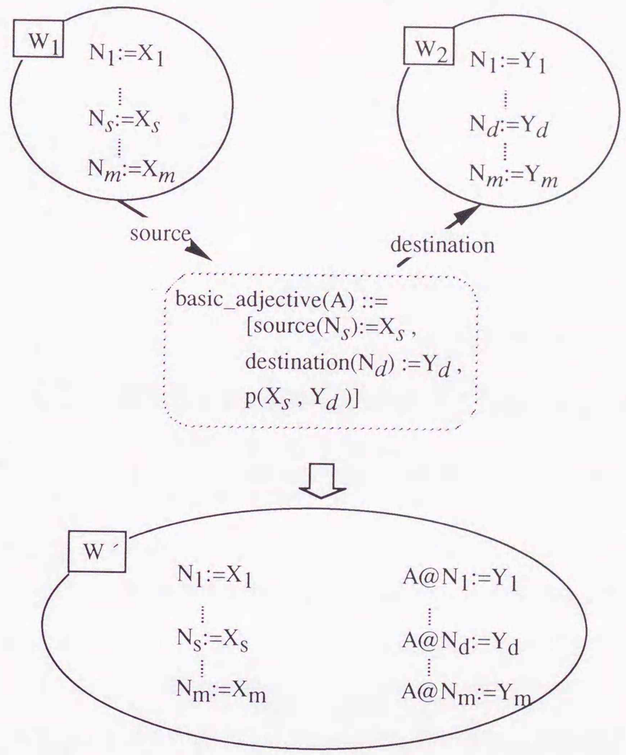


図3.4.1　基本形容詞による制約の付加

$[N_1,\ldots,(A@N_i):=t_i,\ldots,N_k]$

$\Rightarrow$ adj_sdc(A, N_s :=X, N_d :=Y, p(X,Y)),

$[N_1,\ldots, N_s$:=X,$\ldots, N_k]$,

p(X,Y),

$[N_d$:=Y , N_i :=$t_i]$

例

基本形容詞of_parentが次のように定義されていたとする．

```
basic_adjective(of_parent) ::=
    [   source([first_name,last_name]):=X,
        destinatioin([first_name,last_name]):=Y,
        is_parent_of(Y,X)      ].
```

このとき，この基本形容詞を含んだリスト述語を用いた質問:

```
?- [first_name:=taro, of_parent@first_name:=Z].
```

の評価過程を示すと図3.4.2のようになる．図3.4.2では，まず基本形容詞of_parentの定義に基づき，形容詞により修飾されているほうの名詞句を取り除き，そのかわりに，定義中のsourceでラベル付けされている名詞句を加える．そして，述語is_parent_of(Y,X)と，もう一つのリスト述語との連言をとっている．もう一つのリスト述語は，形容詞により修飾されたほうの名詞句と，定義中のdestinationでラベル付けされている名詞句を並べたものである．最初の部分のリスト述語により，質問として与えられたリスト述語から親子関係の制約を付加するためのデータを取り出している．すなわち，述語is_parent_of(Y,X)の第2引数に与えるべきデータを名詞を使って，質問として与えられたリスト述語から取り出している．こうして得られたXの値から，述語is_parent_of(Y,X)を評価することで，tarоの親であるという制約を付加するためのデータYが得られる．そして，そのデータを形容詞of_parentで修飾された名詞句に渡すために，最後のリスト述語が評価されている．


```
?- [first_name:=taro, of_parent@first_name:=Z].
```

⇓ リスト述語の評価規則（基本形容詞）

```
?- [first_name:=taro, [first_name,last_name]:=X],
        is_parent_of(Y,X),
            [first_name:=Z, [first_name,last_name]:=Y].
```

⋮ 実行 {X/[taro,yamada]}

```
?-  is_parent_of(Y,[taro,yamada]),
        [first_name:=Z, [first_name,last_name]:=Y].
```

⋮ 実行 {Y/[ichiro,yamada]}

taroの親としてichiroが見つかる．

```
?-  [first_name:=Z, [first_name,last_name]:=[ichiro,yamada]].
```

⇓

```
Z=ichiro
```

図3.4.2　基本形容詞を用いたリスト述語の評価過程の例

3.5　形容詞句

この節では，形容詞句を作るための語構成子について説明し，それらの語構成子を用いた形容詞句が使われたリスト述語がどのように評価されるかを示す．

●逆演算

逆演算を用いることで，形容詞が表す関係の逆の関係を表すことができる．基本形容詞Aに逆演算を適用した形容詞句A~が使われたときは，次に示す評価規則を用いて評価される．

$[N_1,\ldots,(A^{\sim}@N_i):=t_i,\ldots,N_k]$

⇒　　adj_sdc(A, N_s :=X, N_d :=Y, p(X,Y)),

$[N_1,\ldots, N_d$:=Y,…, $N_k]$,

p(X,Y),

[N_s :=X, N_i :=t_i].

この評価規則は，図3.5.1に示すように，前節で示した基本形容詞での関係付けの方向とは逆である．すなわち，図3.5.1では，W_1の語彙にラベルA~をつけている．逆演算を用いることで，形容詞が表す関係の逆の関係を表すことができる．

語構成子を用いて構成された形容詞句への逆演算の適用は次のように定義する．

$(A^{\sim})^{\sim} = A$

$(A+B)^{\sim} = (A^{\sim})+(B^{\sim})$

$(A\&B)^{\sim} = (A^{\sim})\&(B^{\sim})$

$(-A)^{\sim} = -(A^{\sim})$

$(A:B)^{\sim} = (B^{\sim}):(A^{\sim})$

$(A^*)^{\sim} = (A^{\sim})^*$

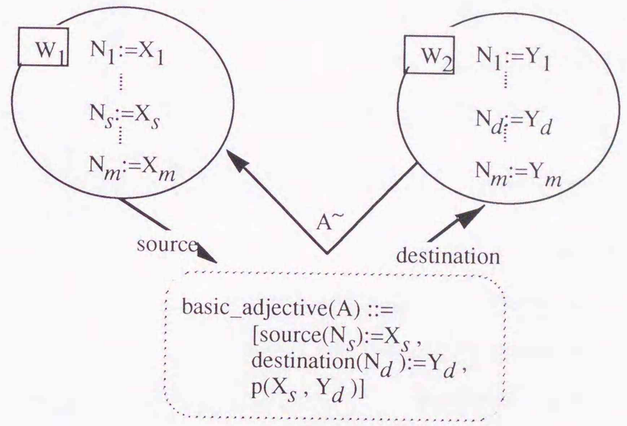


図3.5.1　基本形容詞への逆演算の適用による制約の付加

ここで，A,Bは任意の形容詞句であり，‘+’，‘&’，‘-’，‘:’，‘*’は後述する語構成子である．逆演算を用いた形容詞句が使われたリスト述語は，次に示す評価規則を用いて評価される．ただし，述語inverse(X,Y)は，形容詞句Xに逆演算を適用した結果の形容詞句Yを求める述語である．

$[N_1,\ldots, (A^{\sim}@N_i):=t_i ,\ldots, N_k]$

⇒　　inverse(A, B), $[N_1,\ldots, (B@N_i):=t_i ,\ldots, N_k]$.

例　基本形容詞of_parentが次のように定義されていたとする．

```
basic_adjective(of_parent) ::=
    [   source([first_name,last_name]):=X,
        destinatioin([first_name,last_name]):=Y,
        is_parent_of(Y,X)       ].
```

このとき，この基本形容詞に逆演算を適用したものを含んだリスト述語を用いた質問:

```
?- [first_name:=taro, (of_parent~)@first_name:=Z].
```

の評価過程を示すと図3.5.2のようになる．図3.5.2は，先ほどの図3.4.2とほとんど同じである．違いは図中にも示してあるが，定義中のsourceでラベル付けされている名詞句と，destinationでラベル付けされている名詞句を加えるリスト述語が図3.4.2の場合と逆になっている．最初の部分のリスト述語により，質問として与えられたリスト述語から親子関係の制約を付加するためのデータを取り出している．すなわち，述語is_parent_of(Y,X)の第1引数に与えるべきデータを名詞を使って，質問として与えられたリスト述語から取り出している．こうして得られたYの値から，述語is_parent_of(Y,X)を評価することで，taroの子供であるという制約を付加するためのデータXが得られる．そして，そのデータを形容詞of_parentで修飾された名詞句に渡すために，最後のリスト述語が評価されている．

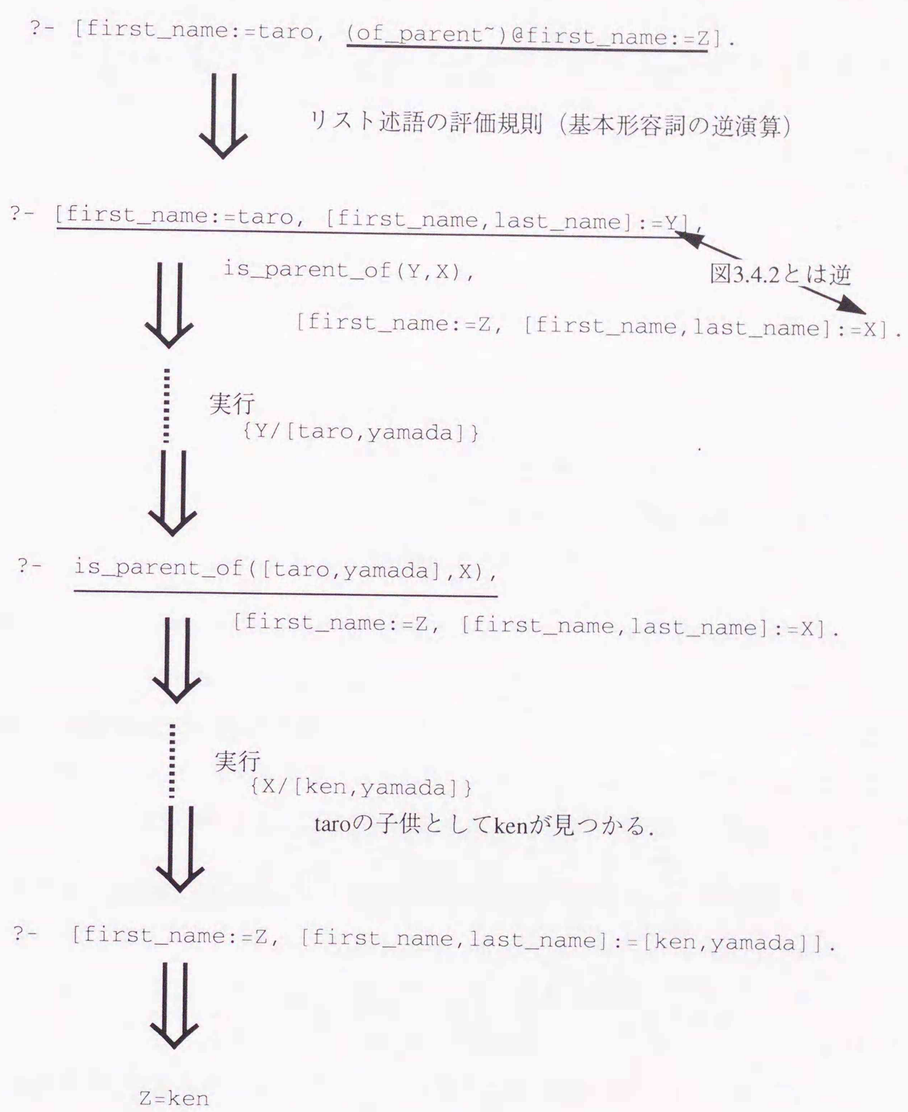


図3.5.2　基本形容詞へ逆演算を適用した場合の評価過程の例

●論理演算

AとBが形容詞句のとき，形容詞句A+B, A&B, -Aは，それぞれAかBによって表される関係，AとBの両方で表される関係，Aによって表される関係がないという制約を表す．それぞれの評価規則を以下に示す．

(1) $[N_1,\ldots, ((A+B)@N_i):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, (A@N_i):=t_i ,\ldots, N_k]$.

(2) $[N_1,\ldots, ((A+B)@N_i):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, (B@N_i):=t_i ,\ldots, N_k]$.

(3) $[N_1,\ldots, ((A\&B)@N_i):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, (A@N_i):=t_i ,\ldots, N_k]$, $[N_1,\ldots, (B@N_i):=t_i ,\ldots, N_k]$.

(4) $[N_1,\ldots, ((-A)@N_i):=t_i ,\ldots, N_k]$

⇒ $[N_1,\ldots, (-(A@N_i)):=t_i ,\ldots, N_k]$.

(1)と(2)が形容詞句A+Bを含んだリスト述語の評価規則である．この場合，A+BをAで置き換えたリスト述語とA+BをBで置き換えたリスト述語の選言として評価される．(3)が形容詞句A&Bを含んだリスト述語の評価規則である．この場合A&BをAで置き換えたリスト述語とA&BをBで置き換えたリスト述語の連言として評価される．(4)が形容詞句-Aを含んだリスト述語の評価規則である．これは名詞句の否定に変換されて評価される．

例　リスト述語[N1:=t1, (A+B)@N2:=t2]があらわすオブジェクト間の関係を図示すると図3.5.3のようになる．オブジェクトO1は，属性N1の値としてt1を持つ．そして，オブジェクトO1には，Aという関係にあるオブジェクトO2またはBという関係にあるオブジェクトO3が存在する．ただし，O2，O3は属性N2の値としてt2を持つ．

リスト述語[N1:=t1, (A&B)@N2:=t2]があらわすオブジェクト間の関係を図示すると図3.5.4のようになる．オブジェクトO1は，属性N1の値としてt1を持つ．そして，オブジェクトO1には，Aという関係にあるオブジェクトO2とBという関係にあるオブジェクト

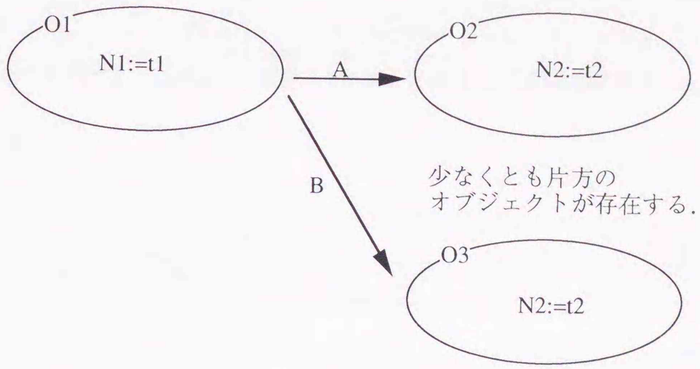


図3.5.3　形容詞句(論理和)によるオブジェクト間の関係

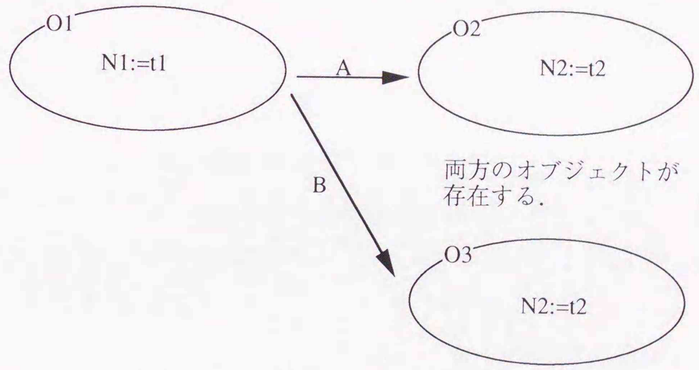


図3.5.4　形容詞句(論理積)によるオブジェクト間の関係

O3が存在する．ただし，O2，O3はともに属性N2の値としてt2を持つ．

リスト述語[N1:=t1,(-A)@N2:=t2]があらわすオブジェクト間の関係を図示すると図3.5.5のようになる．オブジェクトO1は，属性N1の値としてt1を持つ．そして，オブジェクトO1には，Aという関係にあるオブジェクトO2で属性N2の値としてt2を持つようなものが存在しない．

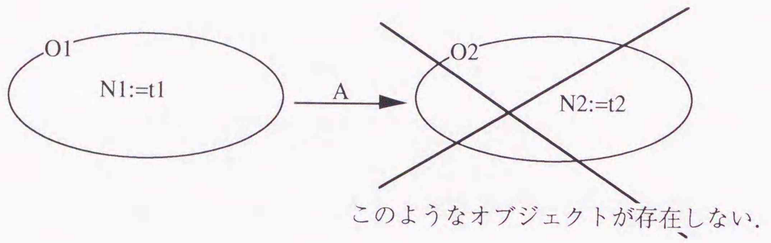


図3.5.5　形容詞句(否定)によるオブジェクト間の関係

●合成演算

基本形容詞は，二つのオブジェクト間の直接的な関係を表していた．合成演算により，あるオブジェクトを介した間接的な関係を表すことができる．すなわち，図3.5.6にあるように，オブジェクトO1とオブジェクトO2の間に関係があり，オブジェクトO2とオブジェクトO3の間に関係があるとき，オブジェクトO1とオブジェクトO3の間にオブジェクトO2を介した関係が表現される．

オブジェクトO1とオブジェクトO2の間にBという形容詞句で表される関係があり，オブジェクトO2とオブジェクトO3の間にAという形容詞句で表される関係があるときに，オブジェクトO1とオブジェクトO2の間にA:Bという形容詞句で表される関係がある．AとBが形容詞句のとき，形容詞句A:Bは，関係BとA をこの順番で合成した関係を表す．すなわち，Nを任意の名詞句とするとき，(A:B)@Nは，B@(A@N)のように定義される．評

価規則を以下に示す．

$[N_1,\ldots,((A:B)@N_i):=t_i,\ldots,N_k]$

$\Rightarrow \quad [N_1,\ldots,(B@(A@N_i)):=t_i,\ldots,N_k].$

●推移閉包演算

推移閉包演算により，形容詞が表す関係の推移閉包を表すことができる．すなわち，X*という形容詞句は，X+X:X+X:X:X+...という形容詞句に等しい．評価規則を以下に示す．

$[N_1,\ldots,((A^*)@N_i):=t_i,\ldots,N_k]$

$\Rightarrow \quad [N_1,\ldots,(A@N_i):=t_i,\ldots,N_k].$

$[N_1,\ldots,((A^*)@N_i):=t_i,\ldots,N_k]$

$\Rightarrow \quad [N_1,\ldots,((A^*:A)@N_i):=t_i,\ldots,N_k].$

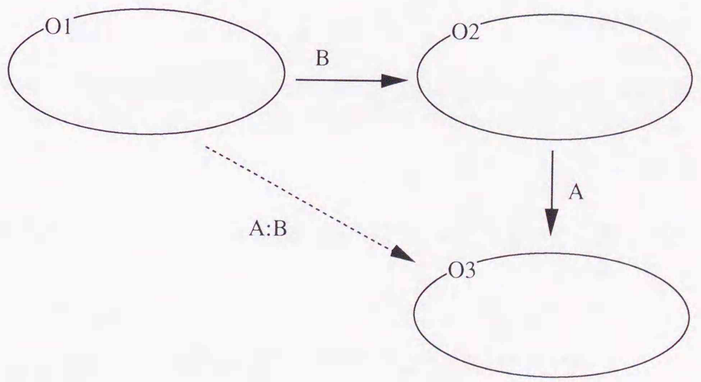


図3.5.6　オブジェクト間の間接的な関係

3.6　派生名詞と派生形容詞

名詞句Nに対して新たにN´という名詞としての名前付けを行い，派生名詞N´を定義するには，

noun(N´)::= N.

という形の式を用いる．Vocalogシステムは，これを次の形の事実に変換して格納する．

dicNL(N´, N) .

このように定義された派生名詞を用いた名詞句N´:=t_iを含むリスト述語は，

$[N_1,\ldots,N´:=t_i,\ldots,N_k]$

⇒　　dicNL(N´, N), $[N_1,\ldots,N:=t_i,\ldots,N_k]$.

のようにN´をNで置き換えたリスト述語として評価される．

形容詞句Aに対して新たにA´という形容詞としての名前付けを行い，派生形容詞A´を定義するには，

adjective(A´) ::=A.

という形の式を用いる．Vocalogシステムは，これを次の形の事実に変換して格納する．

dicAL(A´, A) .

このように定義された派生形容詞を用いた名詞句$(A´@N_i):=t_i$を含むリスト述語は，

$[N_1,\ldots, (A´@N_i):=t_i,\ldots,N_k]$

⇒　　dicAL(A´, A), $[N_1,\ldots, (A@N_i):=t_i,\ldots,N_k]$.

のように，A´をAで置き換えたリスト述語として評価される．

例　派生形容詞として，siblingを次のように定義したとする．

adjective(sibling) ::= (child:parent) & (-self).

この派生形容詞を含んだリスト述語を用いた質問:

?- [first_name:=taro, sibling@first_name:=X].

の評価は，次のように行われる．

?- [first_name:=taro, sibling@first_name:=X].

⇓　派生形容詞siblingの定義の展開

?- [first_name:=taro, ((child:parent) & (-self))@first_name:=X].

⇓　さらに，評価を続けていく．

.....

3.7 実現手法

Vocalogインタープリタは，メタインタープリタの手法により実現されている．そこで，Prologにおけるメタインタープリタの手法を説明してから，Vocalogインタープリタの実現手法を説明する．

3.7.1 Prolog in Prolog

Prologでは，Prologインタープリタを Prolog自身で容易に記述することができる．カットを扱わないPrologインタープリタの例は，次のようになる．

```
solve(true).
solve((A,B)) :- solve(A), solve(B).
solve(A) :- clause(A, B), solve(B).
```

この述語solveの引数は，インタープリタに与えられるゴールである．最初の文では，trueというゴールは必ず成り立つことを表している．2番目の文は(A,B)というゴールの実行は，Aというゴールと，Bというゴールを実行することを表している．3番目の文は，Aというゴールを解くための手続き‘A :- B’をデータベース中から取りだし，Aのかわりに Bというゴールを実行することを表している．clause(A,B)という述語は，Prologの組み込み述語であり，データベース中から‘A :- B’という形の節を取り出すものである．述語clause(A,B)は，データベース中の事実に関しては第2引数にtrueを返す．

例えば，親子関係を表す2引数の述語is_parent_ofと，祖父母との関係を表す2引数の述語is_grand_parentが以下のようにPrologデータベース中に定義されているとする．

```
is_parent_of(taro,  jiro).
is_parent_of(jiro,  hanako).
```



```
is_parent_of(yuki, sakura).
is_parent_of(sakura, hanako).
is_grandparent_of(X,Y) :-
        is_panrent_of(X,Z), is_parent_of(Z,Y).
```

上述のPrologインタープリタを使った質問実行過程を図3.7.1に示す．図3.7.1は，質問として，‘?-solve(is_grandparent_of(X,hanako)).’が与えられたときの証明木である．述語solveの実行はPrologシステムにより行われるが，solveの引数として与えられたゴールの実行は述語solveにより制御されている．

さらに，トップレベルの述語toplevelを次のように定義しておくことで，このメタインタープリタ用のインターフェースを構築できる．

```
toplevel :-
        read_goal(Goal, Var),
        (   solve(Goal), write_answer(Var),next_answer
        ;   write(no),nl
        ),
        toplevel.
```

述語toplevelでは，read_goal(Goal, Var)によりユーザがゴールを入力することを促し，ゴールを表す項Goalと，ゴール中で使われた変数を表す文字列と変数との対応を格納するリストVarを得る．次に，述語solveにより与えられたゴールを実行し，得られた変数の値を述語write_answer(Var)によりユーザに出力する．そして，述語next_answerでユーザが別解を求めているかどうかを判断し，別解を求めている場合はバックトラックにより別解を求める．述語solveによりゴールを実行した結果，答えが見つからなかった場合及び別解を求めようとしたが見つからなかった場合は，ユーザに‘no’を返す．述語‘;’は選言を表す述語である．そして，再び述語toplevelを呼び出して，ユーザが新たなゴールを入力するのを待つ．

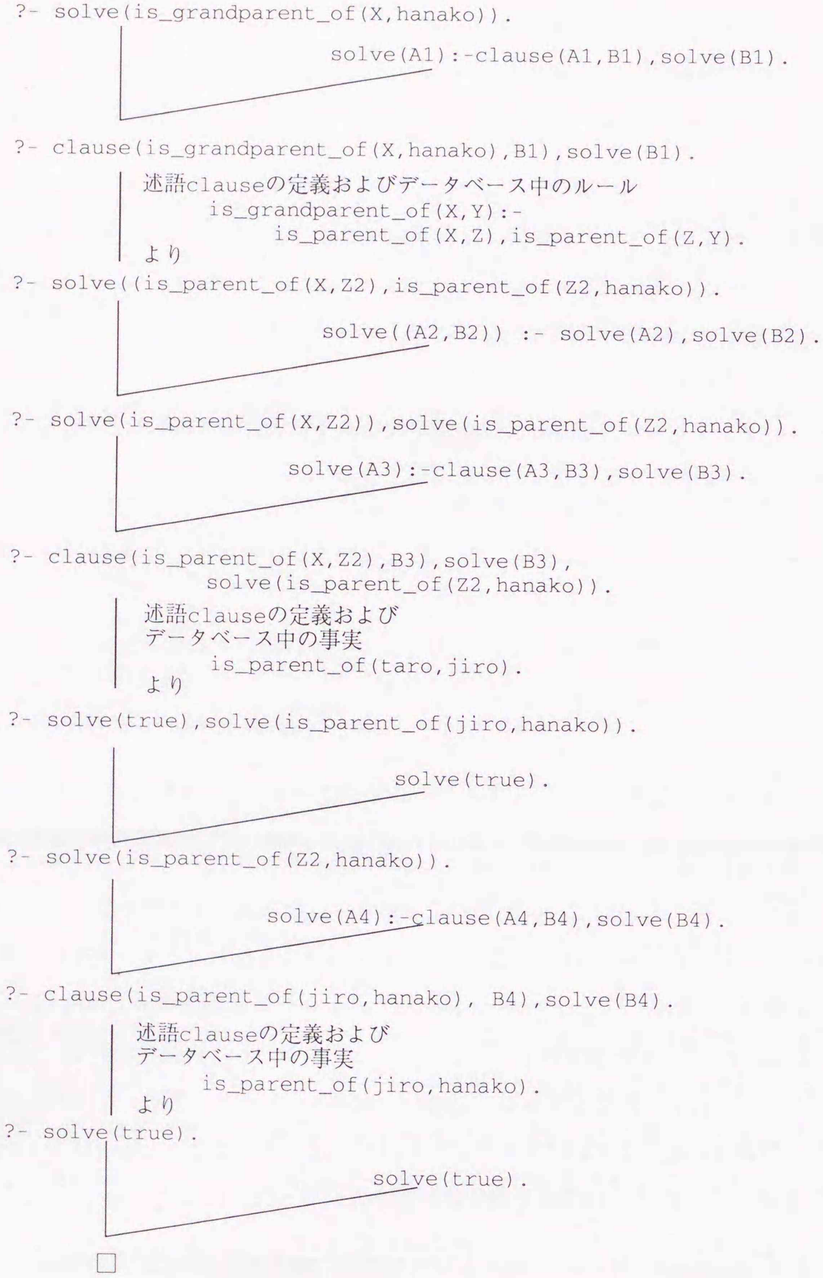


図3.7.1　Prologメタインタープリンタによる質問実行例

このプログラムを改良することで，

システム述語を扱えるPrologインタープリタ，
証明木を出力するPrologインタープリタ，
カットを扱うPrologインタープリタ

を作成することができる．

システム述語を扱えるようにするためには，述語solveの定義にシステム述語を扱うための節を追加すればよい．例えば，データベース中に定義されている，ユーザ述語のリストを表示するためのlistingというシステム述語を扱えるようにするためには，次の節を述語solveの定義に追加すればよい．

```
solve(listing) :-   listing_user_predicate.
```

ここで，述語listing_user_predicateはユーザ述語のリストを表示するための述語である．このほかにも，システム述語として定義しておきたい述語がある場合は，同様に述語solveを拡張すればよい．

3.7.2　Vocalogインタープリタ

前節で示したPrologのメタインタープリタを改良することで，Vocalogシステムを実現している．Vocalogシステムの概略を図3.7.2に示す．VocalogシステムはSICStus Prolog上に実現されている．

図3.7.2のインタープリタ部がVocalogインタープリタである．Vocalogインタープリタはさらに，語変換部，メインインタープリタ部，Prologインターフェース部からなっている．

Vocalogシステムのトップレベルは述語query_expansion(X,Y)を定義することで，SICStus Prolog上のトップレベルにおいてVocalogのリスト述語及び語彙管理部のコマンドを扱えるように拡張している．query_expansion(X,Y)が定義されていると，与えられたゴールが第1引数として与えられ，それが変換されて第2引数に出力されれば，それがゴールとして実行される．Vocalogシステムでは述語query_expansion(X,Y)は次のように定義されている．

```
query_expansion(X,Y) :- vocalog_query_expansion(X,Y).
```

述語vocalog_query_expansion(X,Y)は次のように定義されている．

```
vocalog_query_expansion((X,Y),(X1,Y1)) :-
            !,
            vocalog_query_expansion(X,X1),
            vocalog_query_expansion(Y,Y1).
vocalog_query_expansion([],vocalog([])) :- !.
vocalog_query_expansion([H|T],vocalog([H|T])) :- !.
vocalog_query_expansion(X,X).
```

ここで，最初の節は連言を扱うための変換規則である．2番目と3番目の節がリスト述語を扱うための変換規則である．述語vocalogがVocalogインタープリタを実現した述語で

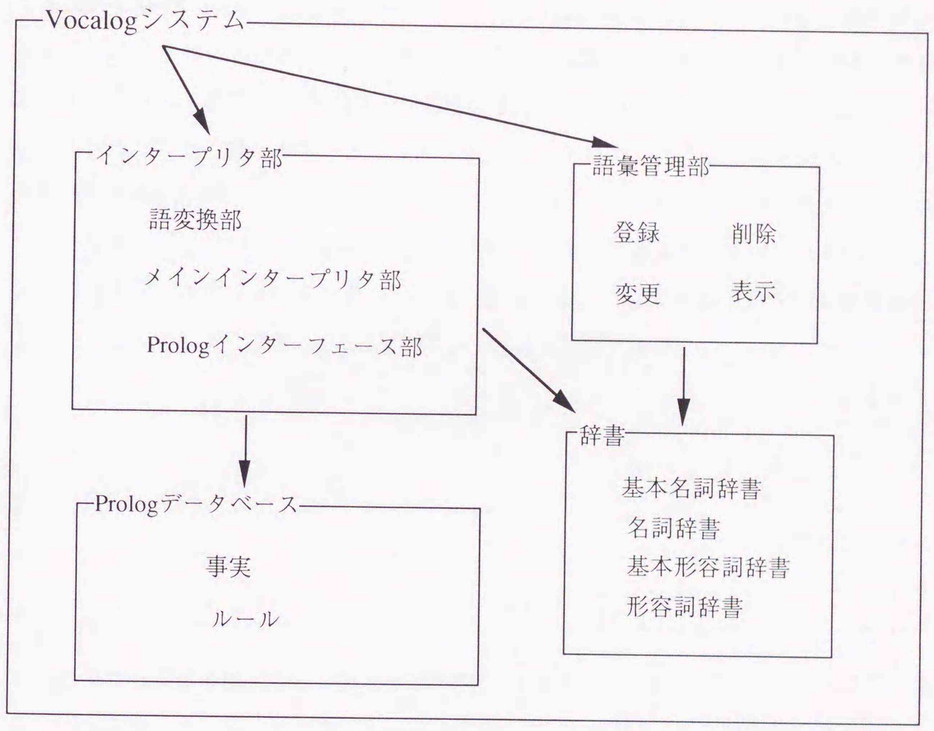


図3.7.2　Vocalogシステムの概略

ある．最後の節は与えられたゴールをそのまま実行するものであり，これはPrologシステムの述語及び語彙管理部のコマンドとして実現された述語を実行するためのものである．

Vocalogインタープリタは，語変換部，メインインタープリタ部，Prologインターフェース部からなっている．語変換部では，派生語の定義を求める．与えられたリスト述語中に出現する派生名詞，派生形容詞を辞書中の定義を参照して基本名詞と基本形容詞からのみ構成されるものに変換を行う．メインインタープリタ部では，語構成子を含んだリスト述語の解釈実行を行う．Prologインターフェース部では，Prologデータベース上に定義されている述語と基本名詞のみを含んだリスト述語との対応をとっている．

語彙管理部では，語の登録，削除，変更，および定義ずみの派生語の一覧を表示する機能を持っている．この語彙管理部を通してのみ，Vocalogの辞書を変更することが可能である．

3.8 まとめ

本章では，拡張論理型プログラミング言語Vocalogについて説明を行った．Vocalogでは，述語の引数の役割を表すラベルとして基本名詞を導入した．基本名詞を用いることで，述語の引数を直接的に指定することができるようになった．さらに，名詞に対する語構成子を導入し，語構成子を用いて名詞を組み合わせた名詞句で，複雑な概念を表すことができた．名詞に対する語構成子として，代入(‘:=’)，論理演算(‘+’，‘&’，‘-’)，グループ・バイ演算(‘/’)，射影演算(‘<<’)，関数，リスト化，形容詞句を導入した．

名詞句と述語との間の関係として基本形容詞を導入した．基本形容詞により，名詞間に新しい関係を定義でき，扱える概念の幅を増やすことができた．形容詞に対する語構成子を導入し，これを用いて形容詞を組み合わせた形容詞句で，複雑な関係概念をジェネリックに表現できた．形容詞に対する語構成子としては，逆演算(‘~’)，論理演算(‘+’，‘&’，‘-’)，合成演算(‘:’)，推移閉包演算(‘*’)を導入した．

定義された語やそれらを組み合わせた句に対して名前付けを行い，この名前を新たに語として用いることを許す機能として，語彙構築機能を導入した．語彙構築機能により，基本語彙から，使える語彙を増やしていくことが可能となった．さらに，基本語彙は，プログラムと対応させて導入したが，語彙構築により導入された派生語彙は，基本語彙をもとにレキシカルに定義することでプログラムとは独立に導入することができた．

最後にVocalogシステムの実現方法について説明を行った．VocalogシステムはPrologのメタインタープリタを拡張することで実現した．

第4章　Vocalogによるプログラミング例

Vocalogにおけるプログラミング例として，家系に関する語彙の構築例と文法規則の表現例，およびリスト処理に関する語彙の構築例を示す．

4.1　家系に関する語彙の構築例

個人情報を管理している5引数の述語personと，親子関係を表している2引数の述語is_parent_ofが定義されているとする．各人の名前，姓，性別，誕生日，血液型がそれぞれ述語personの第1引数から第5引数に格納されている．各親子の組に対して，述語is_parent_ofの第1引数には，親の名前と姓のリストが，第2引数には，子供の名前と姓のリストが格納されている．

最初に，述語personの第1引数から第5引数の役割を表すラベルとして，基本名詞first_name, last_name, sex, birthday, blood_typeを導入する．

```
basic_predicate(person) ::=
        [first_name,last_name,sex,birthday,blood_type].
```

次に，基本名詞と述語を用いて，基本形容詞を定義する．親子関係を表す基本形容詞parent，修飾された名詞が男の人の属性を表す基本形容詞male，修飾された名詞が女の人の属性を表す基本形容詞female，修飾された名詞が自分自身の属性を表す基本形容詞selfを次のように定義する．

●基本形容詞parentの定義

デスティネーション側のオブジェクトが，ソース側のオブジェクトの親であるという関係を表す基本形容詞は次のように定義できる．

```
basic_adjective(parent)::=
        [source([first_name,last_name]):=X,
        destination([first_name,last_name]):=Y,
        is_parent_of(Y,X) ].
```

図4.1.1に示すように，first_nameがX1でlast_nameがX2であるようなオブジェクトと，first_nameがY1でlast_nameがY2であるようなオブジェクトのうち，

述語is_parent_of([Y1,Y2],[X1,X2])

が成り立っているオブジェクト間に関係parentが存在する．

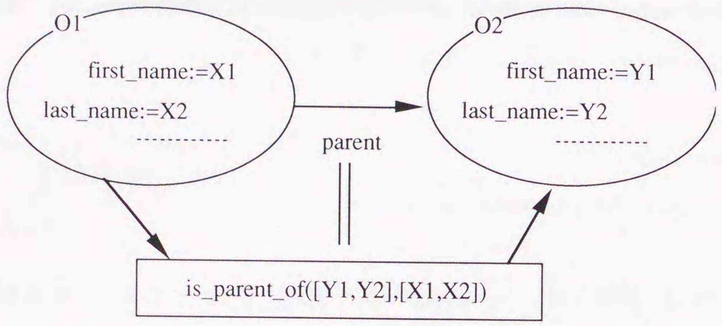


図4.1.1　基本形容詞parent

●基本形容詞maleの定義

デスティネーション側のオブジェクトの性別が男であることを表す基本形容詞maleは次のように定義できる．

```
basic_adjective(male)::=
        [destination(sex):=X, X=male].
```

図4.1.2に示すように，これはデスティネーション側のオブジェクトの属性sexの値がmaleであるという制約を付加している．この定義では，ソース側のオブジェクトに対しては制約を何も付加していない．

他の形容詞との論理積をとることで，デスティネーション側のオブジェクトに対してのみ，性別が男であるという制約を付加することができる．

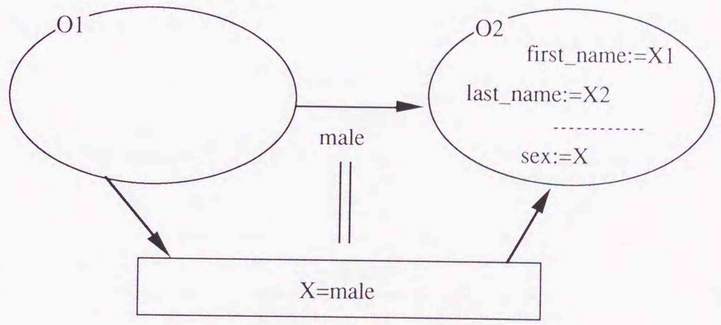


図4.1.2　基本形容詞male

●基本形容詞femaleの定義

デスティネーション側のオブジェクトの性別が女であることを表す基本形容詞femaleは次のように定義できる.

```
basic_adjective(female)::=
        [destination(sex):=X,X=female].
```

図4.1.3に示すように，これはデスティネーション側のオブジェクトの属性sexの値がfemaleであるという制約を付加している．この定義では，ソース側のオブジェクトに対しては制約を何も付加していない.

他の形容詞との論理積をとることで，デスティネーション側のオブジェクトに対してのみ，性別が女であるという制約を付加することができる.

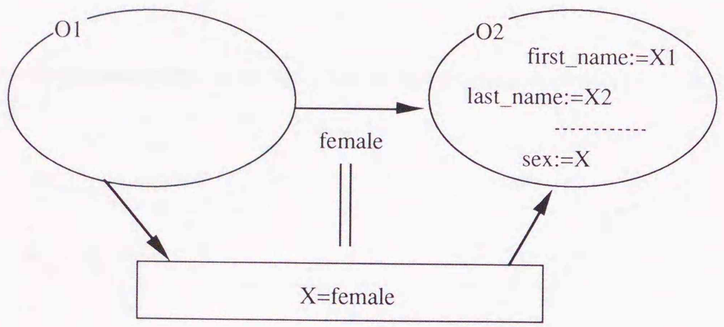


図4.1.3　基本形容詞female

●基本形容詞selfの定義

ソース側のオブジェクトがデスティネーション側のオブジェクトと同一であること，すなわち自分自身との関係を表す基本形容詞selfは次のように定義できる.

```
basic_adjective(self) ::=
        [source([first_name,last_name]):=X,
        destination([first_name,last_name]):=Y,
        X=Y].
```

図4.1.4に示すように，ソース側のオブジェクトのfirst_nameとlast_nameの値とデスティネーション側のオブジェクトのfirst_nameとlast_nameの値が同じであるという制約が付加されている．すなわち，ここでは，first_nameとlast_nameの値が同一であるということでオブジェクトが同じであるということを表現している.

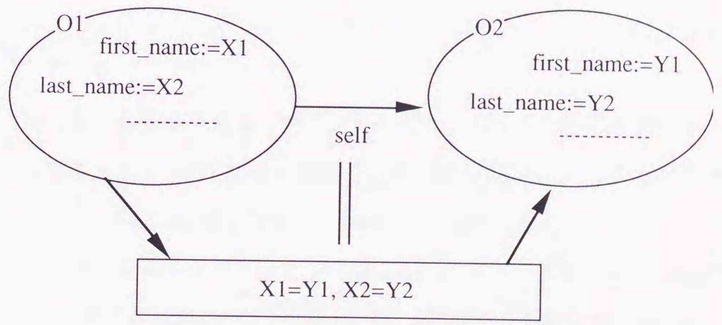


図4.1.4　基本形容詞self

以上のような基本概念を表す基本語彙をもとに，次に示すような家系に関する複合概念をジェネリックに定義することができる．語彙という形で家系に関する概念を定義しておくことにより，語彙中の語を組み合わせ，複雑な文を容易に表現することができる．

●派生形容詞child

デスティネーション側のオブジェクトがソース側のオブジェクトの子供であるという関係を表す形容詞childを定義する．これは，基本形容詞parentが表していた関係において，ソース側のオブジェクトとデスティネーション側のオブジェクトとが入れ替わったものである．すなわち，親という関係の逆方向であるから，逆演算を用いて次のように定義できる．

adjective(child)::=parent~.

●派生形容詞father

●派生形容詞father

デスティネーション側のオブジェクトがソース側のオブジェクトの父親であるということを表す形容詞fatherを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの親であるということを表す基本形容詞parentはすでに定義ずみである．父親はさらにデスティネーション側のオブジェクトの性別が男であればよい．したがって，基本形容詞parentとデスティネーション側のオブジェクトが男であるということを表す基本形容詞maleとの論理積をとれば，デスティネーション側のオブジェクトがソース側のオブジェクトの父親であるという関係を記述できる．よって，形容詞fatherは次のように定義できる．

adjective(father)::=parent&male.

●派生形容詞mother

デスティネーション側のオブジェクトがソース側のオブジェクトの母親であるということを表す形容詞motherを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの親であるということを表す基本形容詞parentはすでに定義ずみである．母親はさらにデスティネーション側のオブジェクトの性別が女であればよい．したがって，基本形容詞parentとデスティネーション側のオブジェクトが女であるということを表す基本形容詞femaleとの論理積をとれば，デスティネーション側のオブジェクトがソース側のオブジェクトの母親であるという関係を記述できる．よって，形容詞motherは次のように定義できる．

adjective(mother)::=parent&female.

●派生形容詞son

●派生形容詞son

デスティネーション側のオブジェクトがソース側のオブジェクトの息子であるということを表す形容詞sonを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの子供であるということを表す形容詞childはすでに定義ずみである．息子はさらにデスティネーション側のオブジェクトの性別が男であればよい．したがって，形容詞childとデスティネーション側のオブジェクトが男であるということを表す基本形容詞maleとの論理積をとれば，デスティネーション側のオブジェクトがソース側のオブジェクトの息子であるという関係を記述できる．よって，形容詞sonは次のように定義できる．

adjective(son)::=child&male.

●派生形容詞daughter

デスティネーション側のオブジェクトがソース側のオブジェクトの娘であるということを表す形容詞daughterを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの子供であるということを表す形容詞childはすでに定義ずみである．娘はさらにデスティネーション側のオブジェクトの性別が女であればよい．したがって，形容詞childとデスティネーション側のオブジェクトが女であるということを表す基本形容詞femaleとの論理積をとれば，デスティネーション側のオブジェクトがソース側のオブジェクトの娘であるという関係を記述できる．よって，形容詞daughterは次のように定義できる．

adjective(daughter)::=child&female.

●派生形容詞sibling

デスティネーション側のオブジェクトがソース側のオブジェクトの兄弟であるということを表す形容詞siblingを定義する．兄弟というのは，自分自身からみると親の子供であって自分自身ではない人である．デスティネーション側のオブジェクトがソース側のオブジェクトの親であるということを表す形容詞parent，デスティネーション側のオブジェクトがソース側のオブジェクトの子供であるということを表す形容詞child，デスティネーション側のオブジェクトがソース側のオブジェクトと同一であるということを表す形容詞selfはすでに定義ずみである．これらの形容詞を使って，兄弟であるという関係を図示すると図4.1.5のようになる．したがって，形容詞siblingは次のように定義できる．

adjective(sibling) ::= (child:parent)&(-self).

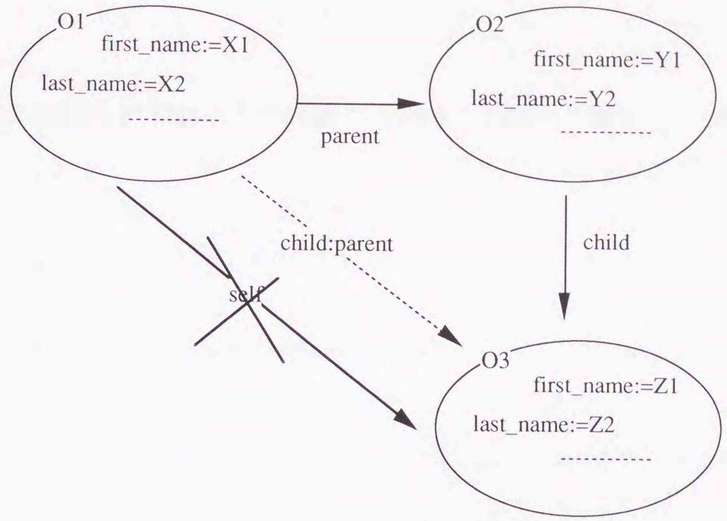


図4.1.5　siblingの関係

●派生形容詞brother

デスティネーション側のオブジェクトがソース側のオブジェクトの男の兄弟であるということを表す形容詞brotherを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの兄弟であるということを表す形容詞siblingはすでに定義ずみである．これにさらに，デスティネーション側のオブジェクトが男であるという制約を付加すればよい．デスティネーション側のオブジェクトが男であるという制約を表す形容詞maleはすでに定義ずみである．したがって，形容詞siblingと形容詞maleの論理積をとることで，デスティネーション側のオブジェクトがソース側のオブジェクトの男の兄弟であるという関係を記述できる．よって，形容詞brotherは次のように定義できる．

adjective(brother)::= sibling&male.

●派生形容詞sister

デスティネーション側のオブジェクトがソース側のオブジェクトの女の兄弟であるということを表す形容詞sisterを定義する．デスティネーション側のオブジェクトがソース側のオブジェクトの兄弟であるということを表す形容詞siblingはすでに定義ずみである．これにさらに，デスティネーション側のオブジェクトが女であるという制約を付加すればよい．デスティネーション側のオブジェクトが女であるという制約を表す形容詞femaleはすでに定義ずみである．したがって，形容詞siblingと形容詞femaleの論理積をとることで，デスティネーション側のオブジェクトがソース側のオブジェクトの女の兄弟であるという関係を記述できる．よって，形容詞sisterは次のように定義できる．

adjective(sister)::= sibling&female.

●派生形容詞cousin

デスティネーション側のオブジェクトがソース側のオブジェクトの従兄であるということを表す形容詞cousinを定義する．従兄というのは，自分自身を中心に考えてみると，親の兄弟の子供ということができる．デスティネーション側のオブジェクトがソース側のオブジェクトの親，兄弟，子供であるという関係を表す形容詞はそれぞれ，parent, sibling, childとしてすでに定義ずみである．これらの形容詞を使って，この関係を図示すると図4.1.6のようになる．したがって，形容詞cousinは次のように定義できる．

adjective(cousin)::= (child:sibling):parent

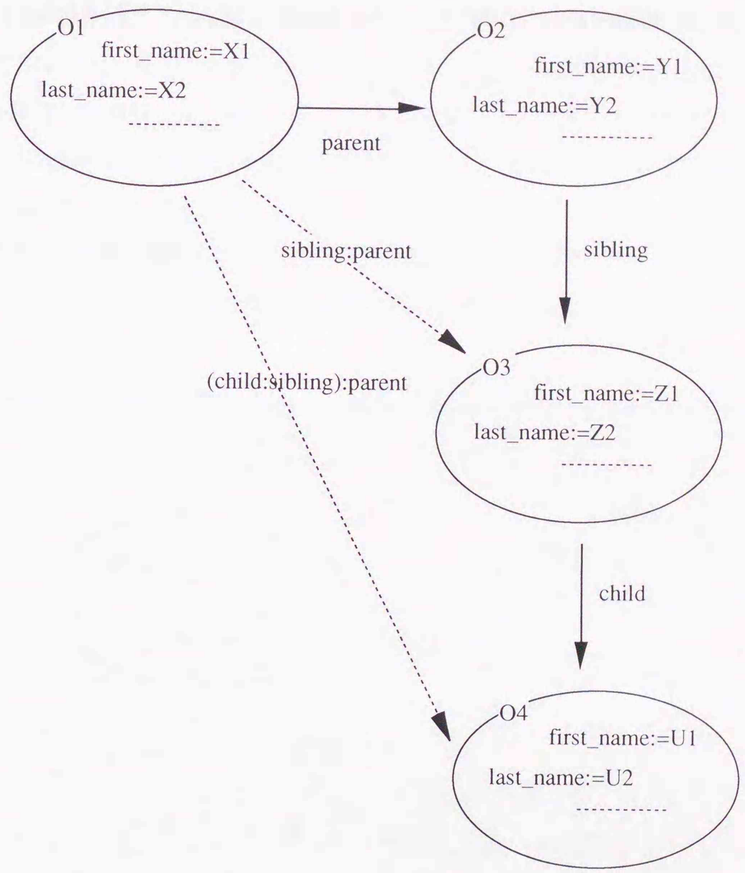


図4.1.6　cousinの関係

●派生形容詞niece

デスティネーション側のオブジェクトがソース側のオブジェクトの姪であるということを表す形容詞nieceを定義する．姪というのは，自分自身を中心に考えてみると，兄弟の子供で性別が女である人ということができる．デスティネーション側のオブジェクトがソース側のオブジェクトの兄弟，子供であるという関係を表す形容詞はそれぞれ，sibling, childとしてすでに定義ずみである．また，デスティネーション側のオブジェクトが女であるという制約を表す形容詞femaleも定義ずみである．この関係を図示すると図4.1.7のようになる．したがって，形容詞nieceは次のように定義できる．

adjective(niece) ::= (child:sibling)&female.

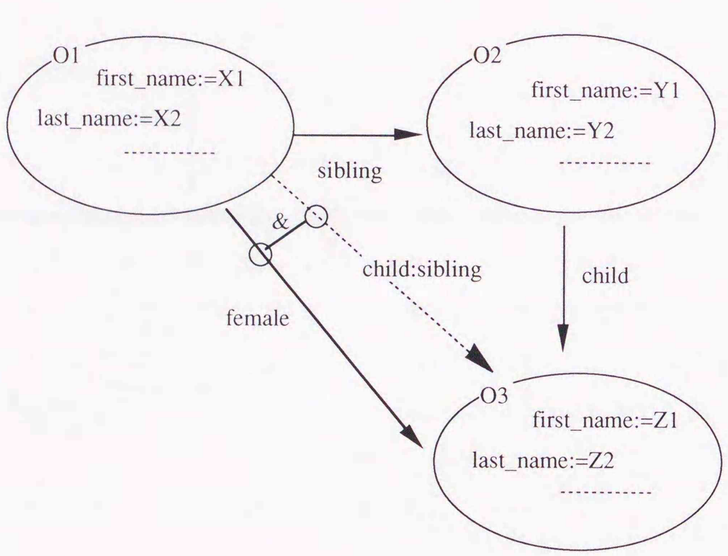


図4.1.7　nieceの関係

●派生形容詞nephew

デスティネーション側のオブジェクトがソース側のオブジェクトの甥であるということを表す形容詞nephewを定義する．甥というのは，自分自身を中心に考えてみると，兄弟の子供で性別が男である人ということができる．デスティネーション側のオブジェクトがソース側のオブジェクトの兄弟，子供であるという関係を表す形容詞はそれぞれ，sibling, childとしてすでに定義ずみである．また，デスティネーション側のオブジェクトが男であるという制約を表す形容詞maleも定義ずみである．この関係を図示すると図4.1.8のようになる．したがって，形容詞nephewは次のように定義できる．

adjective(nephew) ::= (child:sibling)&male.

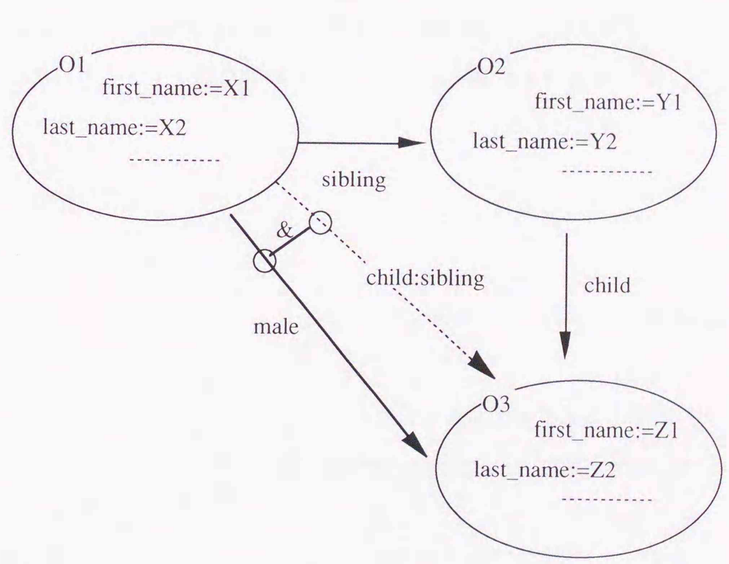


図4.1.8　nephewの関係

●派生名詞person

first_nameとlast_nameのリストを派生名詞personとして定義する．

noun(person) ::= [first_name,last_name].

●派生名詞number_of_children

子供の数を値として持つような派生名詞number_of_childrenを定義することを試みる．いま，personという名詞で各人を一意に特定できるとする．すると，

(child@person)/person

というグループ・バイ演算を用いた名詞句は，各人の子供の名前(first_nameとlast_nameの取る値からなるリスト)の集合をその値としてとる．その要素数を数え上げれば子供の数となる．したがって，要素数を数え上げる関数countを用いて，次のように定義できる．

noun(number_of_children) ::= count((child@person)/person).

以下にこれらの語彙を用いた質問例を示す．

(1)'Ken'という名前の人の兄弟の名前Xを求めよ．

?- [first_name:='Ken', sibling@first_name:=X].

(2)'Taro'という名前の人の兄弟の数Xを求めよ．

?- [first_name:='Taro',
count((sibling@person)/person):=X].

(3)'Ken'という名前の人の兄弟で，'Ken'と同じ血液型の人の名前Xを求めよ．

?- [first_name:='Ken', blood_type:=Y,
sibling@[first_name:=X, blood_type:=Y]].

4.2　文法規則の表現例

DCGで，次のように表されている文法規則を考える．

```
sentence —> noun_phrase,verb_phrase.
noun_phrase —> proper_noun.
noun_phrase —> posses, noun.
verb_phrase —> intrans_verb.
verb_phrase —> trans_verb, object.
object —> noun_phrase.
proper_noun —> [john].
posses —> [my].
noun —> [dog].
intrans_verb —> [runs].
trans_verb —> [likes].
```

この文法規則をVocalogで表すことを試みる．

最初に，差分リストを表す述語d_listを定義する．

```
d_list(X,Y) :- X==Y, !.
d_list(X,Y) :- nonvar(X), !, X=[X1|X2], d_list(X2,Y).
d_list(X,X).
```

差分リストとは，二つのリストの差として実際のリストを表現するものである．例えば，図4.2.1にあるように，一つの長いリストがあり，リストL1はその先頭から全てを，リストL2は先頭から三番目以降のリストであるとする．この二つのリストの差L=L1-L2が

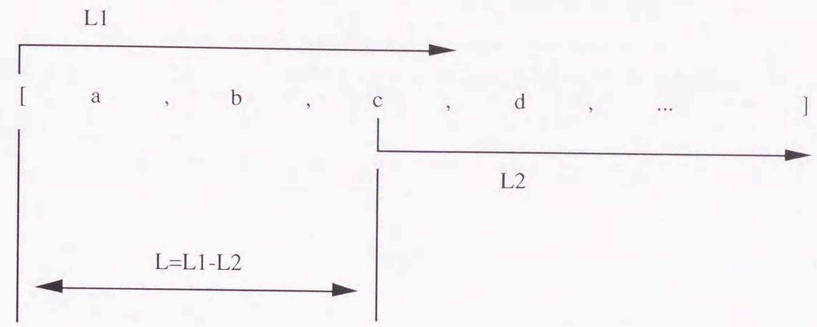


図4.2.1　差分リスト

リスト[a,b]を表しているとみるのが差分リストの考え方である．差分リストにおける空リストはX-Xである．述語d_list(X,Y)はX-Yが差分リストであることを表している．

差分リストにおいて，後ろ側すなわちL1-L2のL2を変数のままにしておくことで，リストの結合をO(1)で実現できる．差分リストの結合は次の事実として定義できる．

```
d_list_append(X1-Y1, Y1-Y2, X1-Y2)
```

これにより第1引数として与えられた差分リストに第2引数として与えられた差分リストを結合したものが第3引数に得られる．この仕組を図示すると図4.2.2のようになる．第1引数の差分リストの後ろ側は変数なので，その変数に第2引数の差分リストの先頭側のリストを代入する．この結果得られるリストが結合した結果の差分リストの先頭側のリストである．結合した結果の差分リストの後ろ側は第2引数として与えられた差分リストの後ろ側の変数となる．

さらに以下に示すような述語を定義する．

```
proper_noun([john|X],X).
posses([my|X],X).
noun([dog|X],X).
```

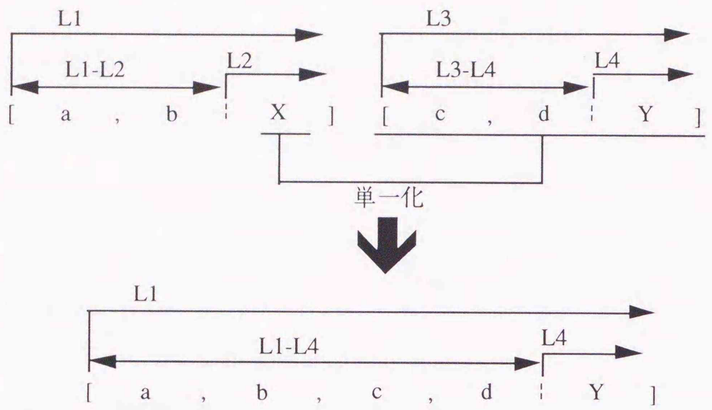


図4.2.2　差分リストの結合

```
intrans_verb([runs|X],X).
trans_verb([likes|X],X).
```

これらの事実は，差分リストを使って，各単語を定義している．例えばproper_nounとして，[john|X]-Xとしてリスト[john]を表現している．このように差分リストを使って表現しておくことで，単語を並べる操作が差分リストの結合として容易に実現できる．例えば，

```
?- proper_noun(X,Y), intrans_verb(Y,Z).
```

という質問によりjohnとrunsを並べたリストが，差分リスト表現として[john,runs|Z]-Zのように求められる．

次に，述語d_listと対応させて，基本名詞d_top, d_end を次のように定義する．

```
basic_predicate(d_list) ::= [d_top, d_end].
```


基本形容詞として，proper_noun, posses, noun, intrans_verb, trans_verbを以下のように定義する．

●基本形容詞proper_noun

ソース側の差分リストの先頭にproper_nounであるような単語を加えたものがデスティネーション側の差分リストとなるような基本形容詞proper_nounを定義する．これは，proper_nounである単語を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これを図示すると図4.2.3のようになる．したがって，基本形容詞proper_nounは次のように定義される．

```
basic_adjective(proper_noun)::=
        [source([d_top,d_end]):=[Y, Z],
        destination([d_top,d_end]):=[X,Z],
        proper_noun(X,Y)].
```

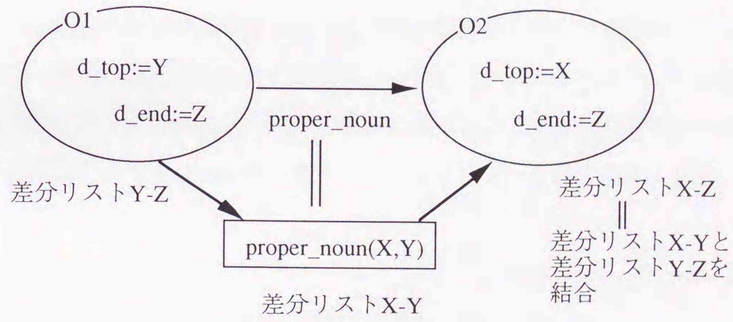


図4.2.3　基本形容詞proper_noun

●基本形容詞posses

ソース側の差分リストの先頭にpossesであるような単語を加えたものがデスティネーション側の差分リストとなるような基本形容詞possesを定義する．これは，possesである単語を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これは，基本形容詞proper_nounの説明で示した図4.2.3で示されたのと同様な操作である．したがって，基本形容詞possesは次のように定義される．

```
basic_adjective(posses)::=
        [source([d_top,d_end]):=[Y,Z],
        destination([d_top,d_end]):=[X,Z],
        posses(X,Y)].
```

●基本形容詞noun

ソース側の差分リストの先頭にnounであるような単語を加えたものがデスティネーション側の差分リストとなるような基本形容詞nounを定義する．これはnounである単語を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これは基本形容詞proper_nounの説明で示した図4.2.3で示されたのと同様な操作である．したがって，基本形容詞nounは次のように定義される．

```
basic_adjective(noun)::=
        [source([d_top,d_end]):=[Y,Z],
        destination([d_top,d_end]):=[X,Z],
        noun(X,Y)].
```


●基本形容詞intrans_verb

ソース側の差分リストの先頭にintrans_verbであるような単語を加えたものがデスティネーション側の差分リストとなるような基本形容詞intrans_verbを定義する．これは，intrans_verbである単語を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これは，基本形容詞proper_nounの説明で示した図4.2.3で示されたのと同様な操作である．したがって，基本形容詞intrans_verbは次のように定義される．

```
basic_adjective(intrans_verb)::=
        [source([d_top,d_end]):=[Y,Z],
        destination([d_top,d_end]):=[X,Z],
        intrans_verb(X,Y)].
```

●基本形容詞trans_verb

ソース側の差分リストの先頭にtrans_verbであるような単語を加えたものがデスティネーション側の差分リストとなるような基本形容詞trans_verbを定義する．これは，trans_verbである単語を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これは，基本形容詞proper_nounの説明で示した図4.2.3で示されたのと同様な操作である．したがって，基本形容詞trans_verbは次のように定義される．

```
basic_adjective(trans_verb)::=
        [source([d_top,d_end]):=[Y,Z],
        destination([d_top,d_end]):=[X,Z],
        trans_verb(X,Y)].
```

以上のように文法規則を表現する上での基本概念を基本語彙として定義することにより，以下に示すように文法に関する制約を派生語彙として定義できる．

●派生形容詞noun_phrase

ソース側の差分リストの先頭にnoun_phraseであるような単語列を加えたものがデスティネーション側の差分リストとなるような形容詞noun_phraseを定義する．noun_phraseは，proper_nounである単語，またはpossesである単語とnounである単語を並べたものである．これはnoun_phraseである単語列を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これを図示すると図4.2.4のようになる．したがって，基本形容詞noun_phraseは次のように定義される．

adjective(noun_phrase)::=proper_noun+posses:noun.

●派生形容詞object

ソース側の差分リストの先頭にobjectであるような単語列を加えたものがデスティネーション側の差分リストとなるような形容詞objectを定義する．objectは，noun_phraseである単語列である．したがって，派生形容詞objectは次のように定義できる．

adjective(object)::=noun_phrase.

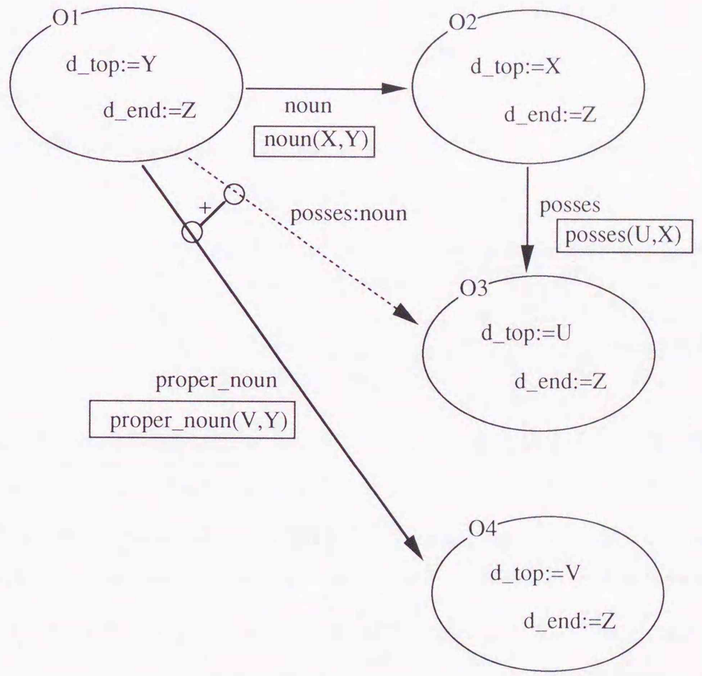


図4.2.4　派生形容詞noun_phrase

●派生形容詞verb_phrase

ソース側の差分リストの先頭にverb_phraseであるような単語列を加えたものがデスティネーション側の差分リストとなるような形容詞verb_phraseを定義する．verb_phraseは，intrans_verbである単語，またはtrans_verbである単語とobjectである単語を並べたものである．これはverb_phraseである単語列を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これを図示すると図4.2.5のようになる．したがって，派生形容詞verb_phraseは次のように定義できる．

adjective(verb_phrase)::=intrans_verb+trans_verb:object.

●派生形容詞sentence

ソース側の差分リストの先頭にsentenceであるような単語列を加えたものがデスティネーション側の差分リストとなるような形容詞sentenceを定義する．sentenceは，noun_phraseである単語列とverb_phraseである単語列を並べたものである．これはsentenceである単語列を表す差分リストに，ソース側の差分リストを結合したものをデスティネーション側の差分リストとする操作である．これを図示すると図4.2.6のようになる．したがって，派生形容詞sentenceは次のように定義できる．

adjective(sentence) ::= noun_phrase:verb_phrase.

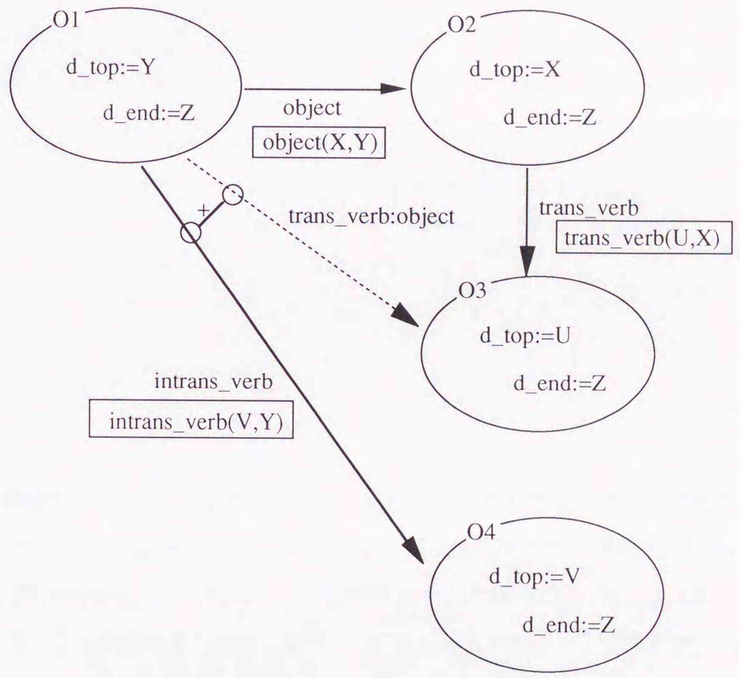


図4.2.5　派生形容詞verb_phrase

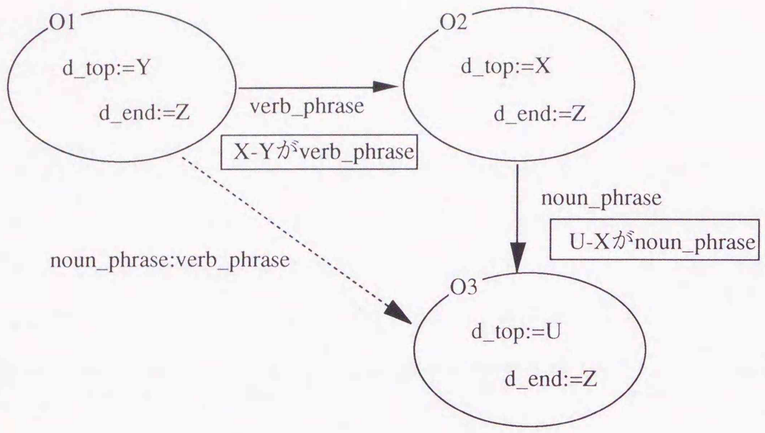


図4.2.6　派生形容詞sentence

●派生名詞sentence

与えられた文法規則から生成される文を値として取るような名詞sentenceを定義する．形容詞sentenceにより修飾された側の差分リストは与えられた文法規則から生成される文を表す．形容詞sentenceにより修飾された側の差分リストの後ろ側(d_end)の変数をnil([])に指定すれば，普通のリストとしてd_topの値として与えられた文法規則から生成される文が得られる．d_endの値をnil([])に指定するには，射影演算を用いればよい．したがって，派生名詞sentenceは次のように定義できる．

```
noun(sentence)::= sentence@(d_top<<d_end:=[]).
```


文法に関する制約を語彙という形で記述することにより，文に対する制約を語の組み合せにより表現することができる．以下に質問例を示す．

(1)　‘john likes my dog’という文は，この文法規則から作ることができますか．

```
?-[sentence:=[john,likes,my,dog]].
```

(2)　この文法規則から作られる文で，固有名詞から始まり，その後に他動詞が続く文Xを求めよ．

```
?-[(sentence&((proper_noun:trans_verb)@d_top)):=X ].
```

4.3　リスト処理

任意のオブジェクトを表す述語thing(X)，リストを表す述語is_list(X)が次のように定義されているとする．

```
thing(X) .
is_list([]).
is_list([X¦Y]) :- is_list(Y).
```

この述語と対応させて，基本名詞thing,listを定義する．

```
basic_predicate(thing) ::= [thing].
basic_predicate(is_list) ::= [list].
```

これらの基本名詞を用いた質問実行例を以下に示す．

```
?- [thing:=a].
yes

?- [list:=[a,b,c], thing:=c].
no
```

最初の例では，述語thingは必ず成功するので，システムは‘yes’を返す．2番目の例では，基本名詞listとthingは同一の述語と対応づけられて定義されていないので，システムは‘no’を返している．

さらに，次のような述語が定義されているとする．


```
list([X¦Y], X, Y).
```

3引数の述語listは，第1引数のリストを，その先頭要素と先頭を取り除いたリストに分解し，それぞれを第2引数と第3引数に返す．

述語listと対応させて基本形容詞head, tailを，そして，基本形容詞last_stopを定義する．

●基本形容詞head

ソース側のリストの先頭要素がデスティネーション側のオブジェクトのthingの値となるような基本形容詞headを定義する．これを図示すると図4.3.1のようになる．したがって，基本形容詞headは次のように定義できる．

```
basic_adjective(head) ::=
        [source(list):=X,
        destination(thing):=Y,
        list(X, Y, _)].
```

この基本形容詞headを用いた質問実行例をあげると次のようになる．

```
?- [list:=[a,b,c,d], head@thing:=X].
X = a ;
no
```

これではリスト[a,b,c,d]の先頭要素が変数Xに求められている．ユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．

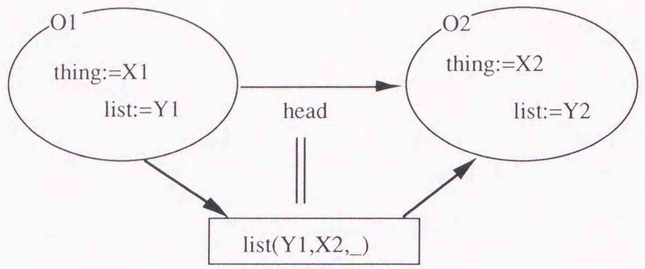


図4.3.1　基本形容詞head

●基本形容詞tail

ソース側のリストの先頭要素を取り除いたリストがデスティネーション側のオブジェクトのlistの値となるような基本形容詞tailを定義する．これを図示すると図4.3.2のようになる．したがって，基本形容詞tailは次のように定義できる．

```
basic_adjective(tail) ::=
        [source(list):=X,
        destination(list):=Y,
        list(X, _, Y)].
```

この基本形容詞tailを用いた質問実行例をあげると次のようになる．

```
?- [list:=[a,b,c,d], tail@list:=X].
X = [b,c,d] ;
no
```


これではリスト[a,b,c,d]の先頭要素を取り除いたリストが変数Xに求められている．ユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．

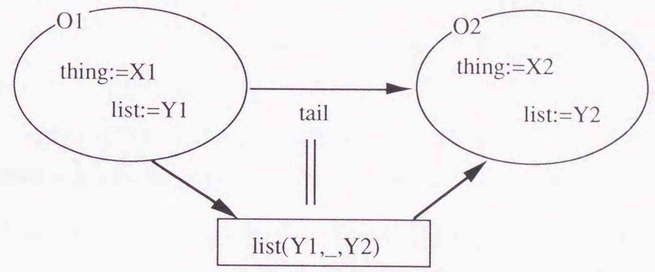


図4.3.2　基本形容詞tail

●基本形容詞last_stop

ソース側のリストがただ一つの要素を含むときにその要素がデスティネーション側のオブジェクトのthingの値となるような基本形容詞last_stopを定義する．これを図示すると図4.3.3のようになる．したがって，基本形容詞last_stopは次のように定義できる．

```
basic_adjective(last_stop) ::=
        [source([head@thing, tail@list]):=[X,Y],
        destination(thing):=Z,
        Y=[],X=Z].
```

この基本形容詞last_stopを用いた質問実行例をあげると次のようになる．

```
?- [list:=[a,b,c,d], last_stop@list:=X].
no

?- [list:=[d], last_stop@thing:=X].
X = d;
no
```

最初の例では，リスト[a,b,c,d]がただ一つの要素を含んでいるのではなく，複数要素を含んだリストなのでシステムは‘no’を返している．2番目の例ではリスト[d]はただ一つの要素dを含んでいるので，それを変数Xに返している．ユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．

以上のような基本語彙を定義しておくことで，次のような リスト処理に関する語彙を定義することができる．

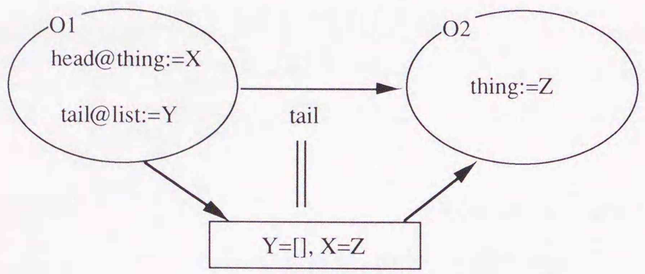


図4.3.3　基本形容詞last_stop

●派生形容詞member

ソース側のリストの要素の内の一つがデスティネーション側のオブジェクトthingの値となるような形容詞memberを定義する．「ソース側のリストの要素の内の一つ」ということを言い換えると，「ソース側のリストの先頭要素，またはソース側のリストから先頭要素を取り除いたリストの先頭要素，またはソース側のリストから先頭要素を取り除いたリストから先頭要素を取り除いたリストの先頭要素，または．．．」ということである．これを図示すると図4.3.4のようになる．したがって，推移閉包演算を使って次のように定義できる．

```
adjective(member) ::= head+(head:tail*).
```

この派生形容詞memberを用いた質問実行例をあげると次のようになる．

```
?- [list:=[a,b,c,d], member@thing:=X].
X = a;
X = b;
X = c;
X = d;
no
```

この例では，リスト[a,b,c,d]の要素が変数Xに求められている．最初に，リスト[a,b,c,d]の先頭要素であるaが求められている．ユーザが別解を求めるためにセミコロン‘;’を入力して，次の要素としてbが求められている．同様なことを繰り返して，要素c，要素dが変数Xに求められ，さらにユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．

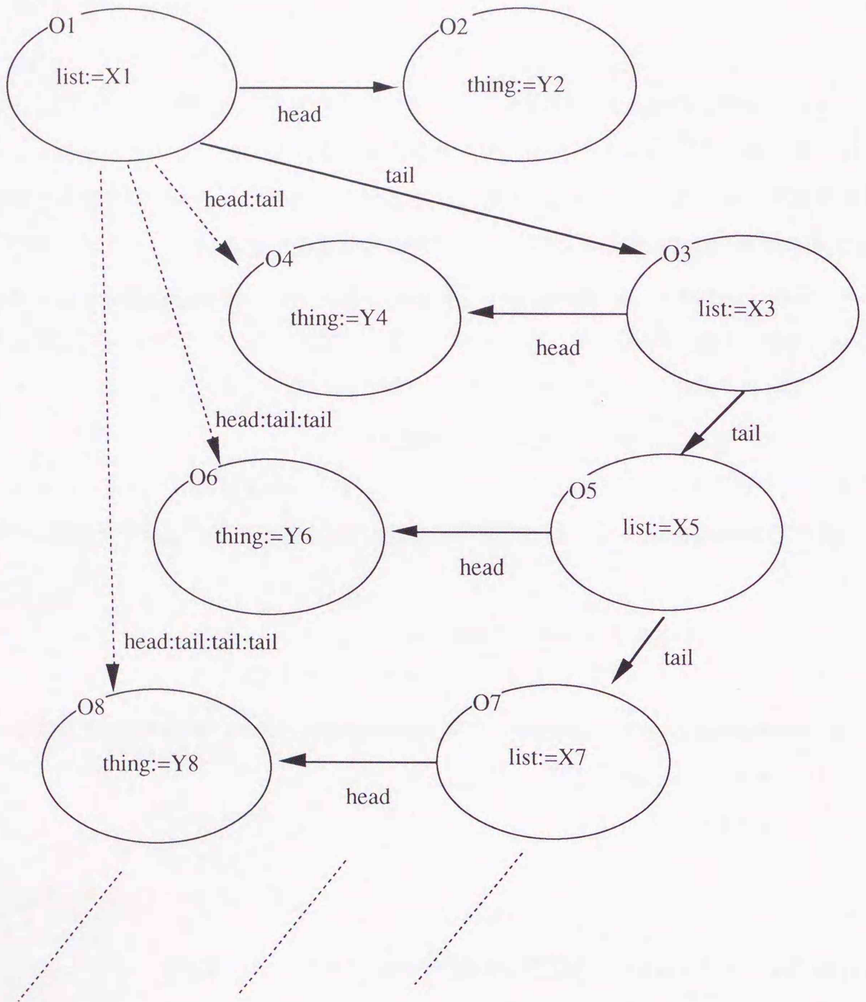


図4.3.4　形容詞member

●派生形容詞firstとlast

ソース側のリストの先頭要素がデスティネーション側のオブジェクトthingの値となるような形容詞firstと，ソース側のリストの最終要素がデスティネーション側のオブジェクトthingの値となるような形容詞lastを定義する．形容詞firstに関しては，基本形容詞headそのものである．したがって，次のように派生形容詞firstを定義できる．

```
adjective(first) ::= head.
```

形容詞lastに関しては，ただ一つの要素を含むリストが得られるまでソース側のリストから先頭要素を取り除き，ただ一つの要素を含むリストの先頭要素を取り出すようにすればよい．これを図示すると図4.3.5のようになる．したがって，推移閉包演算を使って次のように定義できる．

```
adjective(last) ::=
    last_stop+last_stop:tail*.
```

これらの派生形容詞を使った質問実行例を以下に示す．

```
?- [list:=[a,b,c,d], first@thing:=X].
X = a;
no

?- [list:=[a,b,c,d], last@thing:=Y].
Y = d;
no
```

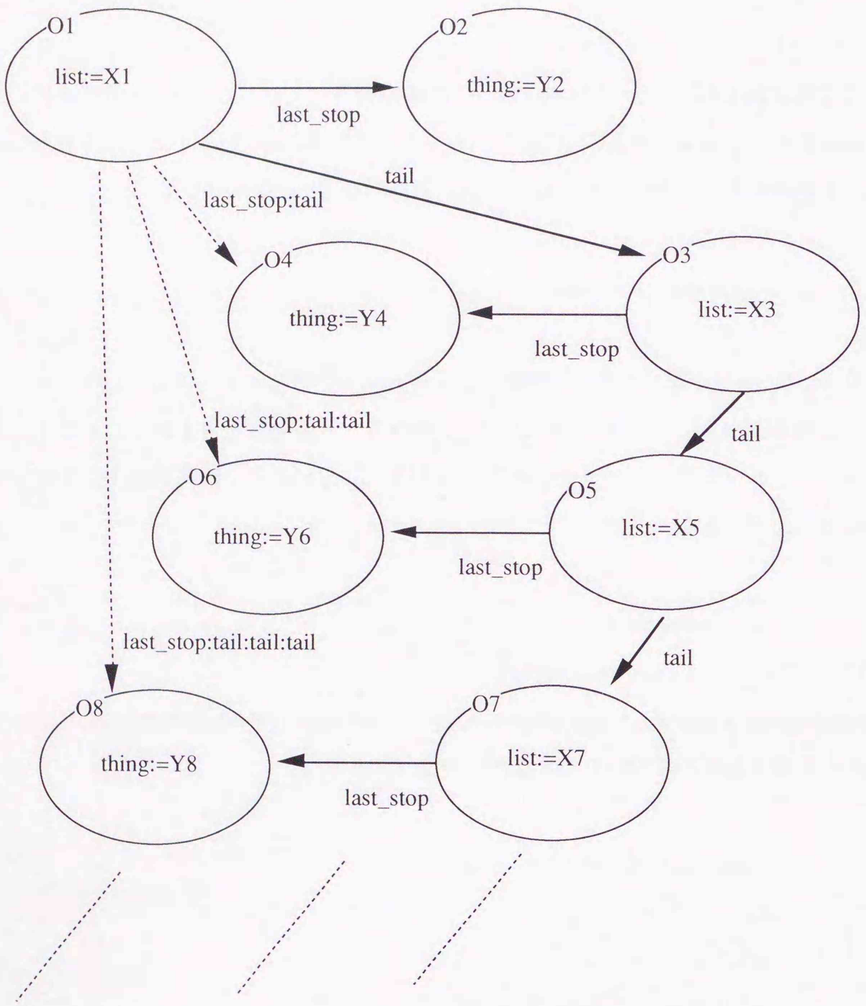


図4.3.5　派生形容詞last

最初の例では，リスト[a,b,c,d]の先頭要素を変数Xに求めている．変数Xの値としてaが得られている．さらにユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．2番目の例では，リスト[a,b,c,d]の最終要素を変数Yに求めている．変数Yの値としてdが得られている．さらにユーザが別解を求めるためにセミコロン‘;’を入力しているが，他に解がないのでシステムは‘no’を返している．

4.4　まとめ

Vocalogによるプログラミング例として，家系に関する語彙の構築例，文法規則の表現例，リスト処理に関する語彙の構築例を示した．Vocalogにおけるプログラミングは，基本語彙を述語と対応させて定義し，それらの基本語彙に含まれる語を組み合わせて新たな語を定義していく語彙構築の過程に他ならないことを示した．

第5章　Vocalogプログラムの可視化

5.1　概略

プログラミングにおける視覚化には，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化がある．プログラムの作成，デバッグ，そしてプログラムの利用はそれぞれ独立したものではなく相互に関連している．これらの環境を個々独立に視覚化したのでは，プログラム利用のための視覚オブジェクトの動きを見ながら視覚的なデバッグを行うといった，機能連携が行えない．したがって，これらの環境を一つのシステムに統合して視覚化すべきである．その際に，プログラムの作成モード，プログラムのデバッグモード，プログラムの利用モードとして個々独立に視覚化するのではなく，これらのモードが一つの環境に混在しうる統合システムにすべきである．これにより，視覚化された個々のモードを独立に利用するのみならず，モード間をわたった機能連携を行うことが可能となる．

エンドユーザーレベルで3つのモード間の自在な機能連携を行うためには，3つのモードをつなぎめなく統合することが必要となる．すなわちモード間の差異にとらわれずに視覚オブジェクトを統一的に操作することが可能となるように，どのモードにおいても視覚オブジェクトの表現形態が同一であるようにすべきである．それぞれのモード用の視覚オブジェクトの操作も含んだ見せ方を用意しておくことで，それぞれのモードに適した視覚オブジェクトの操作及び表示が可能となる．これにより，3つのモード間の自在な機能連携を行うことができ，かつ各モードに特有の視覚オブジェクトの操作及び表示が可能となる．

本章では拡張論理型プログラミング言語Vocalogに対する視覚的統合プログラミング環

境VPE(**V**ocalog **P**rogramming **E**nvironment)を提案する．

5.2 論理プログラミングにおける可視化

オブジェクトをテキスト表現だけではなく視覚的に表現することは，ユーザーインターフェースやCAIシステムなどに有用である．例えば，BarwiseとEthemendyによるTariski's World[19]というシステムは，一階述語論理における視覚的CAIシステムである．Tariski's Worldでは，一階述語論理におけるオブジェクトとそれらの関係を視覚化することで，論理式の理解などを視覚的に支援している．また，Prologシステムに関しても，プログラム中でウィンドウなどの視覚オブジェクトを扱えるようにした，SICStus Prolog上のGraphics Manager[18]やQuintus Prolog上のProXTやProXL[28]などが開発されている．

プログラミングの視覚的支援は，プログラムの効率的な作成・デバッグ，プログラムの効率的な利用に有効である．プログラミングにおける視覚化には，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化がある．テキストのみを用いたプログラミングに比べて，プログラム実行の流れを視覚化することでその流れが把握しやすくなったり，テキストだけを用いたユーザーインターフェースよりも，よりユーザーフレンドリーなインターフェースを与えることができる．論理プログラミングの視覚化に関する研究も，この3つのうちのどこをターゲットにしているかにより分類することができる．

Prologプログラムの作成に関しては，プログラムを可視化することで，プログラム全体の構造を把握しやすくしたり，データの表現をわかりやすいものにすることが可能となる．Prologプログラムの作成を視覚的に支援する研究として，LadretとRueherのVLP[24]，PauとOlasonのVPP[26]，YeungのMPL[32]，渋谷と田中によるVisiLog[33]などが行われてきた．

Prologにおけるプログラミングでは，プログラマーは主に問題が何かというWhatの部分を記述する．どのように問題を解くかというHowの部分はPrologシステムが自動的に

行ってくれる．これは，プログラムを記述するという面では利点ではあるが，プログラムをデバッグするという面では問題となる．すなわち，プログラムをデバッグする際には，プログラマーが記述した‘問題が何かというWhatの部分’だけではなく，Prologシステムが自動的に実行する‘どのようにして問題を解くかというHowの部分’もトレースしていく必要があるということである．Prologプログラムの実行の流れを可視化することで，デバッグを支援する研究として，LadretとRueherのVLP[24]，EisenstadtとBrayshawのTPM[23]，NumaoらによるPROEDIT2[25]，DewarとClearyのDEWLAP[22]，Polakらによるシステム[27]，YeungのMPL[32]などが行われてきた．

Prologプログラムの利用も，テキストを使って質問をするのではなく，視覚的インターフェースを与えることで，ユーザーがより利用しやすい環境を提供することが可能となる．Prologプログラムに対して，スプレッドシートをインターフェースとして用いる研究として，Emdenらによるもの[31]，BonsignoriらによるLOGIFORM[5]，渋谷と田中によるVisiLog[35]などがおこなわれてきた．

以上の論理プログラミングにおける視覚化システムがそれぞれ，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化のうちのどの部分を行っているかをまとめてみると，表5.1のようになる．これらのシステムは，プログラムの作成環境の視覚化，プログラムのデバッグ環境の視覚化，そしてプログラムの利用環境の視覚化のうちの一部分を行っているにすぎない．

Vocalogの視覚的プログラミング環境VPEは，3つの環境を統合して，つなぎ目のない視覚化を行っている．

5.3 VPEシステムの概略

VPEは，Tanakaらにより提案されているシンセティック・メディア・システムIntelligentPad[29][30]を用いて開発されている．IntelligentPadでは，すべての視覚オブジェクトは，機能を持った紙として表現される．この紙をパッドと呼んでいる．パッドの上には複数のパッドをはることができる．パッドの上にパッドをはることで，レイアウトを

表5.1 論理プログラミングにおける視覚化システムの比較

1. プログラムの作成環境の視覚化
2. プログラムのデバッグ環境の視覚化
3. プログラムの利用

システム	1	2	3
VLP	○	○	×
TPM	×	○	×
PROEDIT2	×	○	×
VPP	○	×	×
DEWLAP	×	○	×
Polakらによるシステム	×	○	×
MPL	○	○	×
Emdenらによるシステム	×	×	○
LOGIFORM	×	×	○
VisiLog	○	×	○
VPE	○	○	○

行うとともに機能の合成を行うことができる．

VPEでは，プログラムの作成環境，プログラムのデバッグ環境，プログラムの利用環境という3つのモードにおいてパッドという単一の表現形態を用いて視覚化を行っている．

VPEでは，Vocalogにおける名詞や形容詞など，プログラムの部品となるものすべてをパッドとして視覚化している．それらのパッドを視覚的に組み合わせてプログラムを構成するシンセティック・プログラミングを行うことができる．形容詞of_the_parent, of_the_child, of_the_selfと語構成子‘&’，‘−’それぞれを視覚化したパッドを組み合わせて得られる合成パッドの例を図5.1に示す．図5.1において四角い枠で示されるものがすべてパッドである．図5.1の合成パッドは4.1節において示した「兄弟であるという関係」を表す形容詞句に対応する．なお，図5.1には語構成子‘:’に対応するパッドはないが，これは形容詞に対応するパッドどうしを図のように組み合わせること自体が‘:’の機能を実現しているからである．

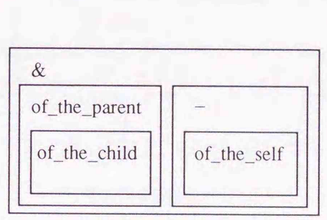


図5.1　VPEにおける視覚的な合成例

VPEでは，プログラムの部品となるパッドは各モードに応じた見せ方を持つことができる．変数Xを表すパッドの各モードにおける見せ方の例を図5.2に示す．プログラムの作成時には変数名を表示する(図5.2(a))．プログラムのデバッグ時には，その変数が束縛されていれば束縛された値を表示し，それ以外では変数名をそのまま表示する(図5.2(b))．プログラムの利用モードでは，変数Xに束縛された数値をパーメータで表示したり，パーメータを操作して変数Xを適当な数値に束縛することができる(図5.2(c))．図5.2

の3つのパッドは，パッドの大きさが異なったり表示の仕方が異なるが，すべて同一のパッドである．

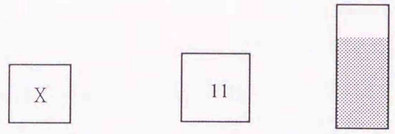


図5.2　異なる表示パッドを使った同一変数の種々の見せ方

次に，VPEにおいてパッドとして視覚化されるものを明らかにする．そして，パッドのはりあわせにより得られる合成オブジェクトについて説明する．そして，モードに応じたオブジェクトの見せ方について説明する．

VPEにおいて視覚化されるオブジェクトとそのパッドとの対応を表5.2に示す．表5.2に示すように，これらのパッドはVPE基本パッドとVPEシステムパッドの2種類にわけられる．VPE基本パッドは，プログラムを構成する部品がパッドとして視覚化されたものである．VPEシステムパッドは，システムの構成要素がパッドとして視覚化されたものである．ルールベースパッドは名詞や形容詞の定義と節集合を格納する機能を持つ．Vocalogパッドは Vocalogインタープリタの機能を持つ．

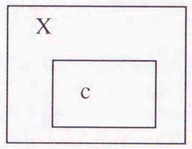
表5.2　VPEにおける視覚オブジェクト

種類	パッド名	オブジェクト
VPE基本パッド	定数パッド	定数
	変数パッド	変数
	複合項パッド	複合項
	基本名詞パッド	基本名詞
	基本形容詞パッド	基本形容詞
	派生名詞パッド	派生名詞
	派生形容詞パッド	派生形容詞
	語構成子パッド	語構成子
	リスト述語パッド	リスト述語
	ヘッドパッド	節の頭部
	ボディパッド	節の本体
	節パッド	節
	基本形容詞定義パッド	基本形容詞の定義式
	派生語定義パッド	派生語の定義式
VPEシステムパッド	ルールベースパッド	ルールベース
	Vocalogパッド	Vocalog インタープリタ

VPEにおけるパッドのはりあわせがどのような操作に相当するかを，主な組み合せに関して説明する．

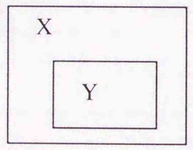
●変数パッドへの貼付

次の図のように変数パッドXの上に定数パッドcをはることで，変数Xを定数cに単一化できる．



X=c

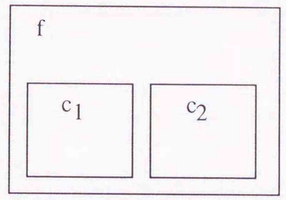
同様に，変数パッドの上に複合項パッドや他の変数パッドをはることで，それらと単一化できる．



X=Y

●複合項パッドへの貼付

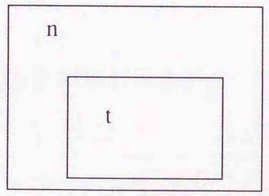
複合項パッドの上に，変数パッドや定数パッド，または別の複合項パッドをはることで引数を指定できる．次の図では，$f(c_1,c_2)$という複合項がはりあわせにより得られている．



$f(c_1, c_2)$

●名詞パッドへの貼付

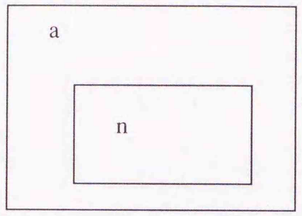
基本名詞パッドと派生名詞パッドをあわせて名詞パッドと呼ぶことにする．名詞パッドの上に，定数パッドや変数パッド，または複合項パッドをはることで名詞のとる値を指定できる．次の図では，名詞nのとる値としてtを指定している．



n:=t

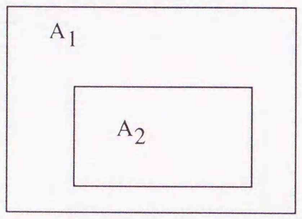
●形容詞パッドへの貼付

基本形容詞パッドと派生形容詞パッドをあわせて形容詞パッドと呼ぶことにする．形容詞パッドaの上に，名詞パッドnを貼付することでa@nという句を表すパッドを構成できる．



a@n

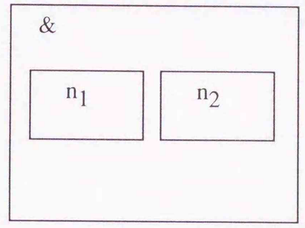
形容詞パッドA_1の上に，別の形容詞パッドA_2をはることで，合成演算子を使った形容詞句A_2:A_1を表すことができる．



A_2:A_1

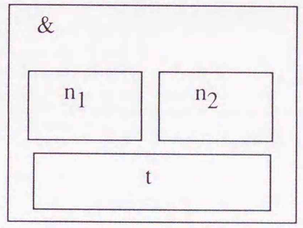
●語構成子パッドへの貼付

語構成子パッドの上に名詞パッドをはることで名詞句を構成できる．さらにその語構成子パッドの上に定数パッドなどを貼付することで，その名詞句の取る値を指定できる．例えば，次の図では，n_1&n_2という名詞句が構成さている．



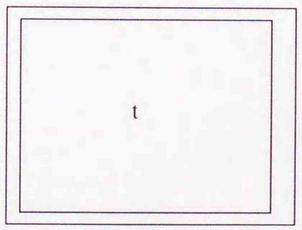
(n_1&n_2)

さらに次の図では，その値としてtが指定されている．



(n_1&n_2):=t

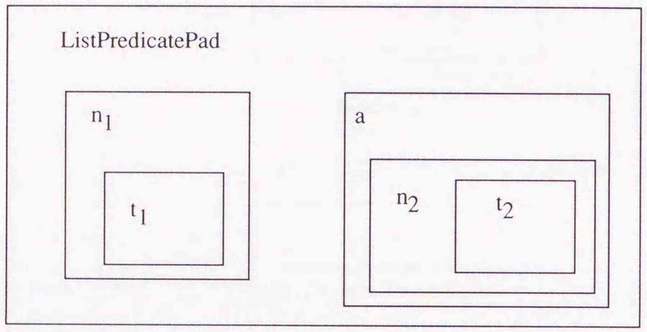
また，tを表すパッドをリサイズして次の図のようにみせることができる．形容詞句を表すパッドの構成およびその形容詞句が修飾する名詞の指定も同様に行う．



(n_1&n_2):=t

●リスト述語パッドへの貼付

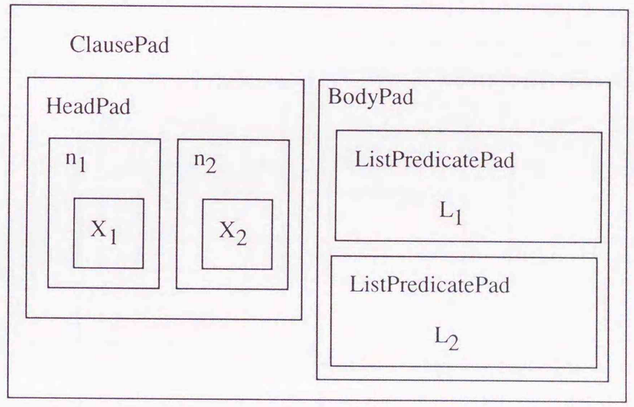
リスト述語パッドの上に，名詞パッドや，名詞パッドを修飾している形容詞パッドをはることで，それらに対応する語を並べたリスト述語を構成できる．次の図では，[n_1:=t_1, a@n_2:=t_2]というリスト述語が構成されている．



[n_1:=t_1, a@n_2:=t_2]

●節パッドへの貼付

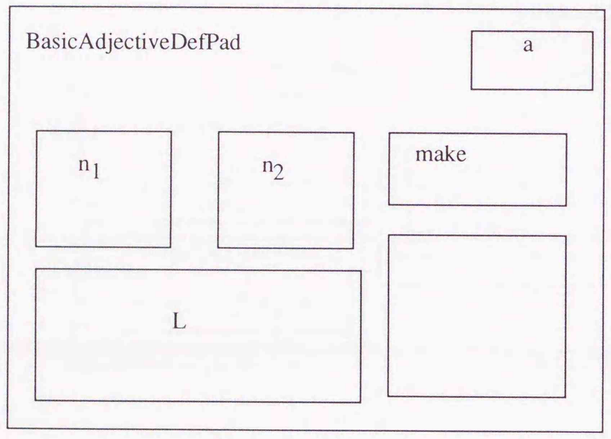
節パッドの上にヘッドパッドとボディパッドをはることで節が構成される．ヘッドパッドの上には，複数の基本名詞パッドがはられる．ボディパッドの上には，複数のリスト述語パッドがはられる．次の図では，節の頭部が[n_1:=X_1, n_2:=X_2]で，節の本体がL_1とL_2の連言である節が構成されている．



[n_1:=X_1, n_2:=X_2] :- L_1, L_2.

●基本形容詞定義パッドへの貼付

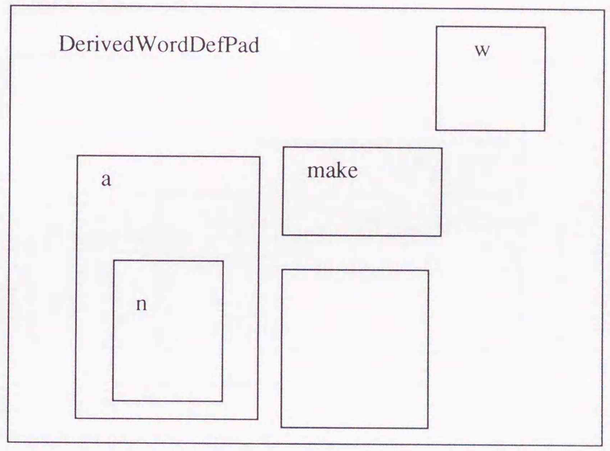
次の図のように基本形容詞定義パッドの上に，2枚の名詞パッドn_1, n_2とリスト述語パッドLをはることで，基本形容詞の定義を構成できる．makeボタンを押すと，makeボタンの下のパッド上に対応する基本形容詞パッドが生成される．それをdrag outして利用できる．



basic_adjective(a) ::=
[source(n_1), destination(n_2), L].

●派生語定義パッドへの貼付

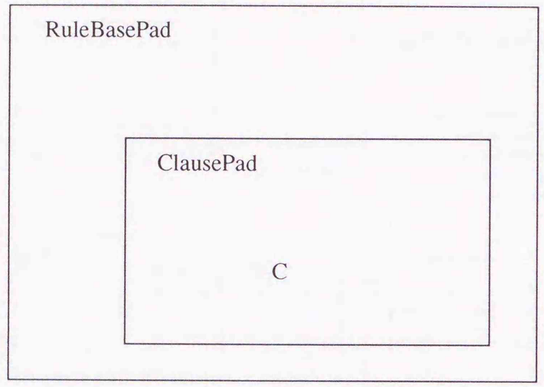
次の図のように派生語定義パッドの上に，名詞句や形容詞句に対応する合成パッドをはることで，派生語の定義を構成できる．makeボタンを押すと，makeボタンの下のパッド上に対応する派生語のパッドが生成される．それをdrag outして利用できる．



名詞句a@nを派生語wとして定義

●ルールベースパッドへの貼付

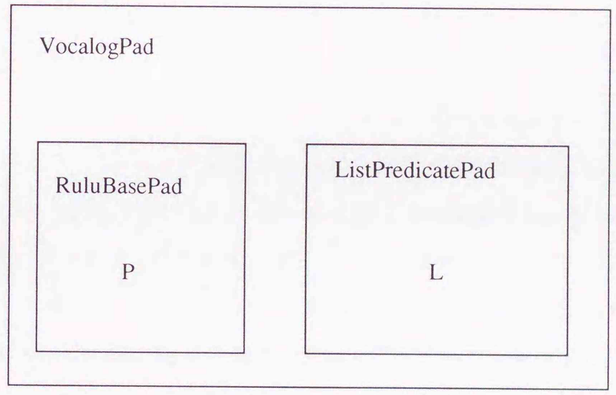
ルールベースパッドは，節集合を格納する機能を持つ．ルールベースパッドの上に節パッドや基本形容詞定義パッドや派生語定義パッドをはると，それらに対応する定義がルールベースに追加される．ルールベースパッドの上にすでにはられているパッドをはがすと，そのパッドに対応する定義がルールベースから削除される．



ルールベースへの節Cの追加

●Vocalogパッドへの貼付

Vocalogパッドは質問を実行する機能を持つ．Vocalogパッドの上には，ルールベースパッドとリスト述語パッドがはられる．貼付されたルールベースパッドに格納されている節の集合が質問の実行に使われる．貼付されたリスト述語パッドに対応するリスト述語が質問として実行される．



ルールベースPを用いた質問 ?- L.の実行

VPEには，プログラムの作成モードとデバッグモード，そしてプログラムの利用モードがある．VPE基本パッドとVPEシステムパッドは，どのモードに設定されているかというモード情報を持つ．デフォールトでは作成モードである．モード変更機能はVocalogパッドが持つ．Vocalogパッドの上に貼付されたパッドは，Vocalogパッドで設定されたモードとなる．

VPE基本パッドとVPEシステムパッドは複数の見せ方を持つことができる．この見せ方のことを表示スタイルと呼ぶ．各モードに対して適当な表示スタイルがデフォールトで対応づけられている．モードと連動して表示スタイルが切り替わる．
例えば，変数パッドはデフォールトで次の3つの表示スタイルを持つ．

(1) 変数名を表示するスタイル
(2) 変数の束縛状況を表示するスタイル
(3) (2)の機能に加えて，変数への束縛をキーボードから与えることができるスタイル

デフォールトで(1), (2), (3)の表示スタイルがそれぞれ作成モード，デバッグモード，利用モードに対応づけられている．

パッドの表示スタイル変更は，図5.3に示す表示スタイル変更パッドをその上にはり，矢印のボタンを操作して表示スタイルを変更した後，表示スタイル変更パッドを剥離することで行う．モードと表示スタイルとの対応関係の変更，モードと連動した表示スタイルの変更設定解除も表示スタイル変更パッドを用いて行う．

VPEでは，これらの見せ方も種々のパッドのはりあわせで実現している．デフォールトでは，各モードに対応した3つの表示スタイルが用意される．IntelligentPadシステムにプリミティブとして用意されているパッドや，新たにユーザーが作ったパッドを表示スタイルとして追加できる．デフォールトの3つの表示スタイルの他に表示スタイルを追加するための特殊スタイルがある．

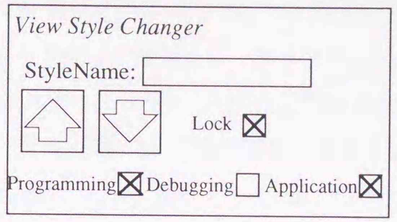


図5.3　表示スタイル変更パッド

5.4 まとめ

拡張論理型プログラミング言語Vocalogの，視覚的統合プログラミング環境VPEを示した．VPEはシンセティック・メディア・システムIntelligentPadを用いて開発された．

VPEでは，プログラムの作成モード，デバッグモード，プログラムの利用モードという3つのモードを視覚的に統合してつなぎめなく扱えるようにするために，次の二点を実現した．第一に，3つのモードすべてにおいて視覚オブジェクトをパッドという同一の表現形態で統一した．第二に，同一のオブジェクトに対して種々の見せ方(表示スタイル)を与えることができるようにした．これにより，同一オブジェクトを複数のモードで共有して，なおかつ，それぞれのモードに適した見せ方を与えることができた．

Vocalogのプログラムを構成する部品をパッドとして視覚化した．これらの部品を組み合わせることで，シンセティック・プログラミングが行えることを示した．システムを構成するものとして，節集合を格納するルールベースと質問の実行機能を持つVocalogインタープリタをそれぞれルールベースパッド，Vocalogパッドとして視覚化した．質問の実行に使われるルールベースの指定を，Vocalogパッドの上に指定したいルールベースパッドを貼付することで行えるようにした．質問をVocalogインタープリタに与えることも，質問に相当するリスト述語パッドをVocalogパッドの上に貼付することで行えるようにした．

VocalogはPrologに名詞や形容詞といった語彙を導入し利用できるようにしたシステムなので，VPEをPrologの視覚的プログラミング環境として用いることも可能である．VPEでは，Vocalogの名詞や形容詞を視覚オブジェクトとし，それらを組み合わせてプログラムを構成するシンセティックプログラミングとIntelligentPadのシンセティックプログラミングとの間に緊密な対応関係をとることで，従来Prologでは不可能であったシンセティックプログラミングを論理プログラミングに導入することが可能となった．

VPEでは，IntelligentPadシステムの資源を使うことができるので，メディアオブジェクトとしてのパッドをより多く取り入れていくことで，視覚的なプログラミング環境をさらに充実していくことが可能である．

第6章　VPEシステムの実現

6.1　概要

この章では，前章で述べたVPEシステムの実現手法を説明する．最初に，IntelligentPadとその動作機構を説明する．

6.2 IntelligentPad

IntelligentPadはTanakaらにより提案されているシンセティック・メディア・アーキテクチャを実現したシステムである．IntelligentPadでは，計算機上にのせることが可能なすべてのメディアを機能を持った「紙」状のオブジェクトとして可視化を行っている．この機能を持った「紙」状のオブジェクトはパッドと呼ばれる．

IntelligentPadは，次の5つの原則にしたがっている．

(1)　ユーザの操作対象はすべてパッドである．
(2)　すべてのパッドは永続オブジェクトである．
(3)　すべてのパッドは機能を持つ．
(4)　すべてのパッドは他のパッドと機能合成できる．
(5)　すべてのパッドは他のユーザとの間で交換できる．

定義されているすべてのパッドに対して，ユーザは以下の操作を行うことができる．

(1)　移動

(2)　コピー

(3)　貼る／はがす

(4)　属性変更

(5)　ファイルへの保存とその再現

(1)は画面上のパッドを自由に移動させて，レイアウトを決めることができるということである．(2)によりパッドを自由にコピーして，他のパッドの部品として利用することができる．(3)の貼り合わせにより，機能の合成が可能となる．(4)の属性変更により，パッドの背景色，描画色，枠線の太さなどを変更することができる．(5)により，ユーザが作成したパッドをファイルに保存し，それを他のユーザとの間で交換することが可能となる．

IntelligentPadにおけるパッドは，Smalltalk-80のMVC構造を単純化した構造を持つ．Mでパッドの内部状態が定義され，Vで表示の仕方が定義され，Cでマウス操作などに対する反応が定義される．Mにおけるいくつかの内部状態がスロットとして公開される．パッド内部でのメッセージ交換の様子を示すと図6.1のようになる．

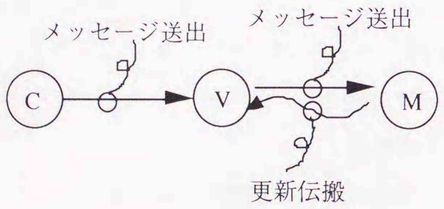


図6.1　パッド内部のメッセージの流れ

パッド間でのデータ通信は，はりあわせ構造を介した，set, update, gimmeという3つのメッセージとパッドの幾何属性を変更するジオメトリメッセージに限られている．この様子を図6.2に示す．

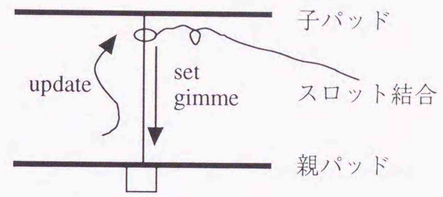


図6.2　パッド間標準インターフェース

パッドのはりあわせ構造において，下のパッドを親パッド，上のパッドを子パッドと呼ぶ．子パッドは自分より下のパッドのスロットのうちの一つだけに結合できる．setメッセージにより子パッドは自分が結合しているスロットに値を入れることができる．親パッドの状態が変わると子パッドにupdateメッセージが送られる．updateメッセージが送られると，子パッドは親パッドにgimmeメッセージを送り，自分が結合しているスロットの値を読み出す．set，gimmeのメッセージは，親パッドが対応するスロットを持たなければ親の親に送られる．

パッドはコピーをとることができる．コピーには，非共有コピーと共有コピーがある．この違いは，内部状態を共有しないかするかの違いである．単一パッドの非共有コピーは，オリジナルと同一の内部状態を持ったものが別に作られる(図6.3)．それに対して，単一パッドの共有コピーをとったものは，M部が共有されて，VとCは個別に持つパッドが作られる(図6.4)．複数のパッドがはりあわさった合成パッドの非共有コピーは，単一のパッドの場合と同様にオリジナルと同一の内部状態を持ったものが別に作られる．複数のパッドがはりあわさった合成パッドの共有コピーをとると，はりあわせ構造の一番下にあるパッドのM部が共有され，他の部分は個別に持つ合成パッドが作られる(図6.5)．

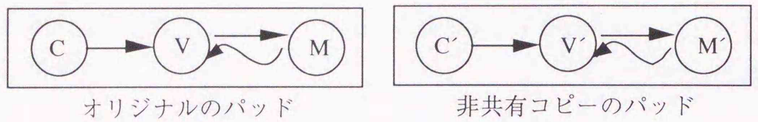


図6.3　単一パッドの非共有コピー

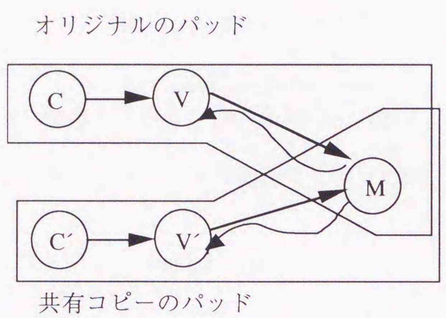


図6.4　単一パッドの共有コピー

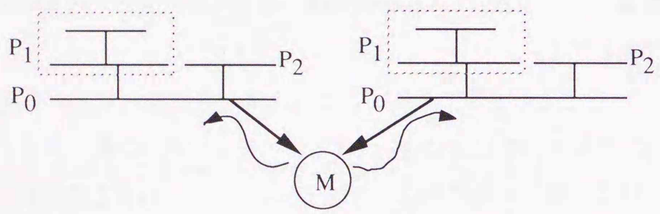


図6.5　合成パッドの共有コピー

6.3　実現手法

本節では，VPEにおける質問実行機能の実現手法と，表示スタイルの切り替え機構について説明する．

質問を実行する際のパッドのはりあわせ構造は図6.6のようになる．一番下に，Vocalogパッドがあり，その上に質問に相当するリスト述語パッドが貼付される．図6.6に，

?- [name:=taro, of_the_parent@name:=X].

という質問を実行する際のデータの流れを示す．図6.6における番号が以下の説明の番号に対応する．

(1)　queryスロットに初期化のための特殊シンボル#initializeがセットされる．

(2)　Vocalogパッドからupdateメッセージが子パッドに送られる．子パッドは親パッドにgimmeメッセージを送り，初期化フェーズであることを知る．

(3)　(2)のupdateメッセージが親から子へと伝搬していき，Vocalogパッドの上方にあるすべてのパッドに対して，初期化フェーズであることが伝わる．

(4)　はりあわせ構造の一番上にあるパッドから順々にsetメッセージが親パッドに送られていき，質問に相当するリスト述語を表す文字列が構成されていく．

(5)　Vocalogパッド上のリスト述語パッドのスロットlistPredicateに，質問に相当するリスト述語を表す文字列として，

[name:=taro, of_the_parent@name:=X]

が構成され，それがVocalogパッドのqueryスロットにセットされる．

(6)　Vocalogパッドにおいて質問が実行される．質問の実行が成功して，特殊シンボル#yesと正解代入{X=ken}がVocalogパッドのqueryスロットに入れられる．(質問の実行が失敗した場合は，特殊シンボル#noがVocalog パッドのqueryス

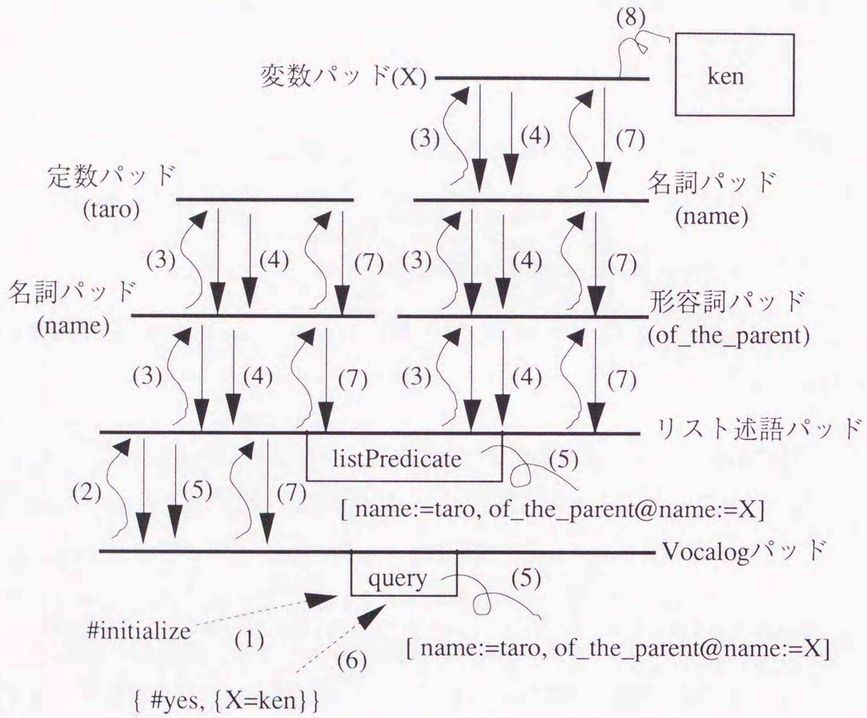


図6.6　質問実行時のはりあわせ構造とデータの流れ

ロットにセットされる．)

(7)　Vocalogパッドから子パッドにupdateメッセージが送られる．(6)で得られたqueryスロットの内容が子パッドに伝搬する．

(8)　各パッドで表示が更新されて，変数パッドの表示が束縛された値kenになる．

VPE基本パッドとVPEシステムパッドは，複数のパッドのはりあわせにより構成されている．変数パッドの例(図6.7)を用いて表示スタイルの切り替え機構を説明する．

変数パッドは変数に関するデータ等を保持するパッドP_0と表示スタイル切り替え機能を持つパッドP_{MF}と表示スタイル用のパッドP_1,P_2,P_3から構成され，図6.7に示すはりあわせ構造を持つ．

パッドP_0には，変数名Xが格納されているスロットvarName，変数Xの束縛されている値kenが格納されているスロットvarValue，モードがデバッグモードであることなどのデータが格納されているスロットvocalogDataがある．

P_1とP_2はスロットvalueの値を表示する機能を持つ．P_1のスロットvalueには，スロット結合によりP_0のスロットvarNameの値である変数名Xが格納される．P_2のスロットvalueには，スロット結合によりP_0のスロットvarValueの値kenが格納される．これにより，P_1と

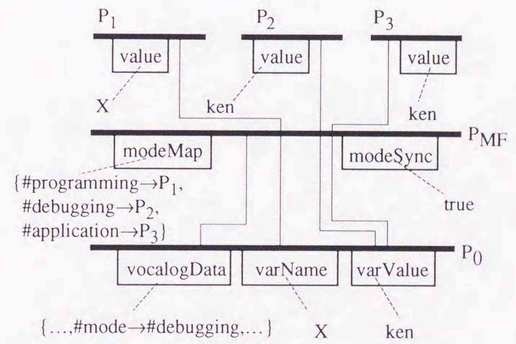


図6.7　変数パッドの見せ方の切り替え機構

P_2はそれぞれ変数名Xと変数Xが束縛されている値kenを表示する．P_3はスロットvalueの値を表示する機能とスロットvalueにキーボードから文字列をセットする機能を持つ。P_3のスロットvalueには，スロット結合によりP_0のスロットvarValueの値kenが格納される．パッドP_3を使って変数Xに対する束縛を変更できる．パッドP_3上でキーボードから定数taroを入力すると，スロット結合によりパッドP_0のスロットvarValueの値がtaroに変わる．

パッドP_{MF}はマルチファセットパッドと呼ばれる種類のパッドである．マルチファセットパッドはその上にはられている複数のパッドのなかから一つだけを表示し，かつ自分自身の大きさを表示したパッドの大きさにあわせる．パッドP_{MF}には各モードと表示スタイル用のパッドとの対応を格納するスロットmodeMapがある．図6.7ではP_1,P_2,P_3がそれぞれ作成モード，デバッグモード，利用モードと対応づけられていることを示すデータがスロットmodeMapに格納されている．パッドP_{MF}はスロットmodeSyncの値がtrueのときにだけ，結合しているスロットから得られるモードと対応づけられている表示スタイル用のパッドに表示変更する．この機能によって図6.7の状態ではP_2が表示されている．

図6.7の変数パッドの共有コピーを取って，一方のパッドのP_{MF}のスロットmodeSyncの値をfalseにしてシステム全体のモード(デバッグモード)と切り離してから，利用モードの視覚表示に変更することによりデバッグモードと利用モードの両方の視覚表示をみることができる．これは変数パッドの共有コピーを取った場合，実際に共有されるのは一番下のパッドP_0が持つ内部状態であり，その上にあるマルチファセットパッドP_{MF}が保持するデータに関しては，それぞれのパッドが個別に持つことができるからである．

6.4 まとめ

本節では，VPEにおける質問実行機能の実現手法と，表示スタイルの切り替え機構をどのようにして実現しているかを説明した．質問実行機能は，IntelligentPadのパッド間のスロット結合を介して，質問文に相当する文字列を順次構成していく．一番下にあるVocalogパッドに最終的な質問文に相当する文字列がわたされ，Vocalogインタープリタによりその質問が実行される．Vocalogインタープリタにより返された値は，パッド間のスロット結合を介して順次Vocalogパッドの上方のパッドに伝えられていく．そして，各パッドは自分の状態が変化していれば，変数を具体化された値に表示しなおすなどの処理を行う．表示スタイルの切り替え機構は，複数の見せ方を与えることができるマルチファセットパッドにより実現された．

第7章　VPEシステムの使用例

世界の都市に関するデータベースの例をあげる．図7.1，図7.3，図7.4がそれぞれ，プログラムの作成，デバッグ，利用のためのパッドである．これらのパッドを同時にみながら，プログラムの作成，デバッグを行っていくことが可能である．

データベース中には，都市の名前と緯度・経度が次のようなファクトで格納されているとする．

```
[ city:=tokyo,
    lattitude_deg:=35, longitude_deg:=139,
    lattitude_min:=42, longitude_min:=46,
    lattitude_ns:=north, longitude_ew:=east ] .
...
```

基本名詞cityは値として都市名をとる．基本名詞lattitude_deg, lattitude_minは緯度の度と分に関する数値を値としてとる．基本名詞lattitude_nsは北緯の場合はnorthを，南緯の場合はsouthを値としてとる．経度に関する基本名詞も同様である．

これらのファクトの作成モードにおける表示例を図7.1に示す．図7.1において，はりあわせ構造の一番下にあるパッドはVocalogパッドである．作成モードと対応づけられているルールベースパッドが表示されている．ルールベースパッドでは，格納されているファクトの一つを表示している．ルールベースパッド上の矢印のボタンパッドを押すことで，別のファクトを表示できる．Addと表示されているボタンパッドは，ルールベース

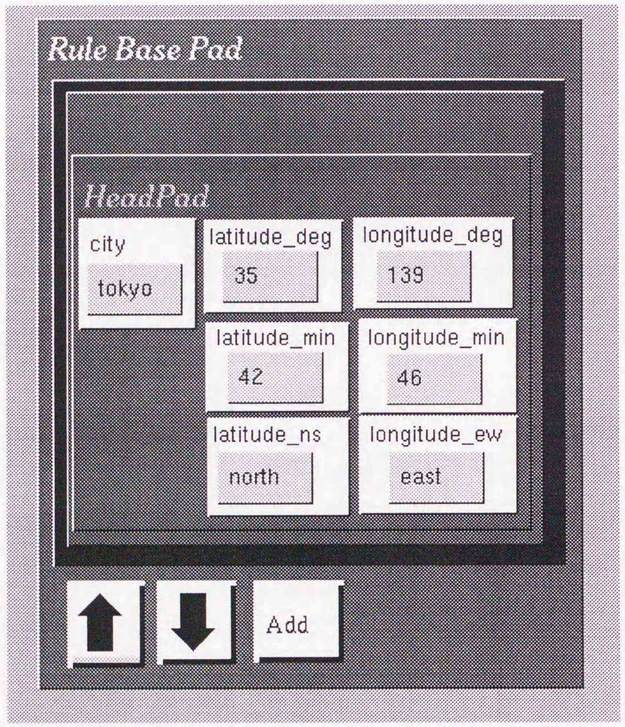


図7.1　作成モードにおける表示例

にファクトを追加する時に使う．

ルールベースには，さらに国名とその首都名とが基本名詞countryとcapitalを用いて，次のようなファクトで格納されているとする．

```
[ country:=japan, capital:=tokyo ].
    ...
```

VPEシステムでは，首都の緯度情報などを得るための基本形容詞capital'sの定義:

```
basic_adjective(capital's) ::=
    [ source(capital):=X,
      destination(city):=Y,
      X=Y ].
```

も基本形容詞定義パッドを用いて視覚的に行える．その基本形容詞定義パッド上で生成された基本形容詞パッドを用いた例が図7.2である．図7.2のパッドは，次のリスト述語の作成モードにおける表示例である．

```
[ country:=japan,
  capital's@[city:=U1, lattitude_deg:=V1]]
```

このリスト述語は，国japanの首都はU1でその緯度はV1度であるということをあらわす．

デバッグモードにおける表示例を図7.3に示す．図7.3に示されている3つのパッド(左上，左下，右)は同一のVocalogパッドの共有コピーである．マルチファセットパッドを用いて表示スタイルを切り替えている．Vocalogパッドがデバッグモードにおいてデフォールト表示するのは，質問に相当するリスト述語パッド(図7.3の左上のパッド)である．図7.1と同様な基本名詞パッドが使われている．基本名詞パッドの上に変数名表示のある変

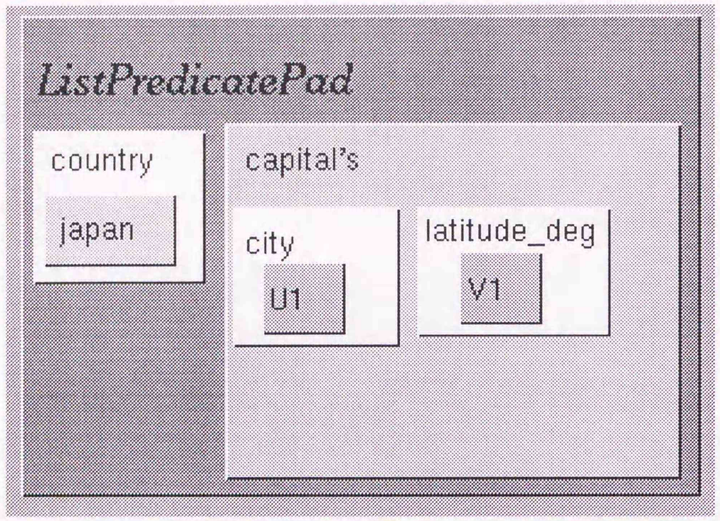


図7.2　形容詞パッドの使用例

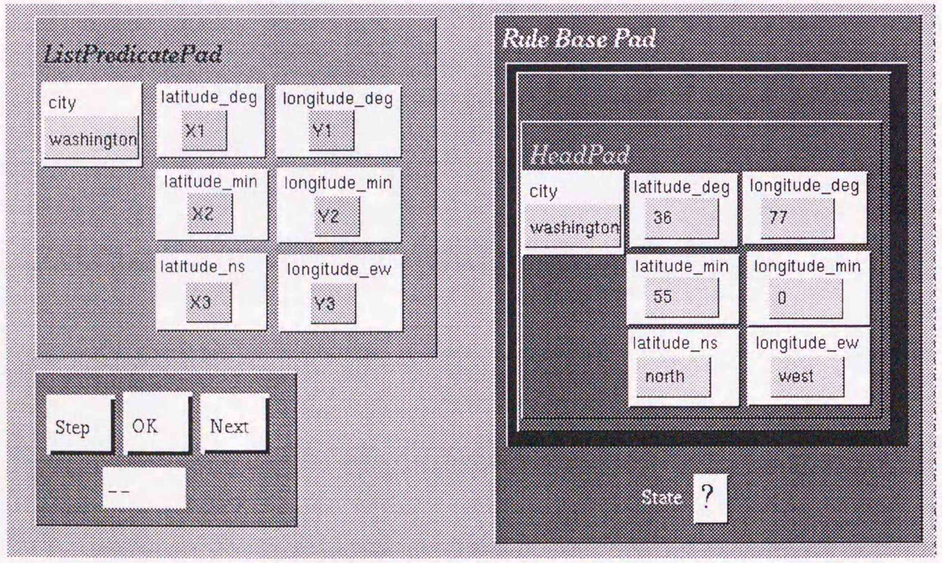


図7.3　デバッグモードにおける表示例

数パッドが貼付されている．図7.3の左下にあるパッドは，Vocalogシステムにコマンドを与えるための表示スタイルに設定されている．このパッド上のstepボタンは実行を1ステップ進めるために使う．実行の結果，それで良い場合にOKボタンを，別解を求めるときにNextボタンを使う．その下の表示器は，質問が最終的に成功した場合‘yes’，失敗したときは‘no’を，質問の実行中は‘--’を表示する．図7.3の右側のパッドは，ルールベースパッドを表示するスタイルに設定されている．ルールベースパッドでは，現在選択されているファクトを表示している．ルールベースパッドの下のほうにある表示器が‘?’を示しているのは，この節の選択が成功したかどうかが未定であるからである．

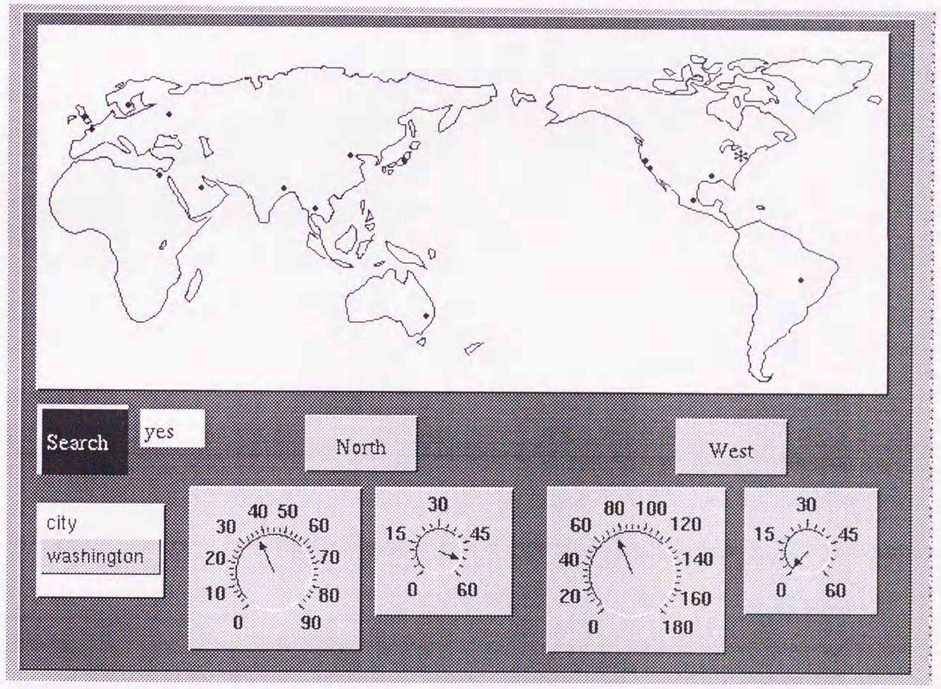


図7.4　利用モードにおける表示例

成功すると‘○’に，失敗すると‘×’に変わる．図7.3の左下のパッド上のstepボタンを押して実行が成功すると，左上のリスト述語パッド上で変数の束縛状況が示される．

図7.4のパッドでもはりあわせ構造の一番下にあるパッドはVocalogパッドである．利用モードに対応づけられているリスト述語パッドが表示されている．このリスト述語パッドは図7.3の左上のパッドと同一のものであるが，表示スタイルが異なっている．都市名‘washington’をテキストで入力して，Searchボタンを押した後の状態が図7.4である．Searchボタンを押すことで質問が実行されている．緯度と経度がメーターを使って表示されている．地図上で‘washington’の場所が‘＊’印で示されている．地図上の‘＊’と‘・’は同じ種類のパッドである．それらのパッドは都市名を保持しており，リスト述語パッド中の都市名に関する内部状態を参照して，それと同じ都市名のものが‘＊’を表示し，他のものは‘・’を表示する．Searchボタンの横の表示器はその都市が見つかれば‘yes’を，見つからなければ‘no’を表示する．

第8章　結論

自然言語における語彙は，人と人との間のコミュニケーションや，概念構築において重要な役割を担っている．本論文では，自然言語における語彙の持つ特徴をPrologに導入した拡張論理型プログラミング言語Vocalogを提案した．

Vocalogでは，述語の引数の役割を表すラベルとして基本名詞を導入した．基本名詞を用いることで，述語の引数を直接的に指定することができるようになった．さらに，名詞に対する語構成子を導入し，語構成子を用いて名詞を組み合わせた名詞句で，複雑な概念を表すことができた．名詞に対する語構成子として，代入(‘:=’)，論理演算(‘+’，‘&’，‘−’)，グループ・バイ演算(‘／’),射影演算(‘<<’)，関数，リスト化，形容詞句を導入した．

名詞句と述語との間の関係として基本形容詞を導入した．基本形容詞により，名詞間に新しい関係を定義でき，扱える概念の幅を増やすことができた．形容詞に対する語構成子を導入し，これを用いて形容詞を組み合わせた形容詞句で，複雑な関係概念をジェネリックに表現できた．形容詞に対する語構成子としては，逆演算(‘~’)，論理演算(‘+’，‘&’，‘−’)，合成演算(‘:’)，推移閉包演算(‘*’)を導入した．

定義された語やそれらを組み合わせた句に対して名前付けを行い，この名前を新たに語として用いることを許す機能として，語彙構築機能を導入した．語彙構築機能により，基本語彙から，使える語彙を増やしていくことが可能となった．さらに，基本語彙は，プログラムと対応させて導入したが，語彙構築により導入された派生語彙は，基本語彙をもとにレキシカルに定義することでプログラムとは独立に導入することができた．

拡張論理型プログラミング言語Vocalogの，視覚的統合プログラミング環境VPEを示した．VPEはシンセティック・メディア・システムIntelligentPadを用いて開発された．

VPEでは，プログラムの作成モード，デバッグモード，プログラムの利用モードという3つのモードを視覚的に統合してつなぎめなく扱えるようにするために，次の二点を実現した．第一に，3つのモードすべてにおいて視覚オブジェクトをパッドという同一の表現形態で統一した．第二に，同一のオブジェクトに対して種々の見せ方(表示スタイル)を与えることができるようにした．これにより，同一オブジェクトを複数のモードで共有して，なおかつ，それぞれのモードに適した見せ方を与えることができた．

Vocalogのプログラムを構成する部品をパッドとして視覚化した．これらの部品を組み合わせることで，シンセティック・プログラミングが行えることを示した．システムを構成するものとして，節集合を格納するルールベースと質問の実行機能を持つVocalogインタープリタをそれぞれルールベースパッド，Vocalogパッドとして視覚化した．質問の実行に使われるルールベースの指定を，Vocalogパッドの上に指定したいルールベースパッドを貼付することで行えるようにした．質問をVocalogインタープリタに与えることも，質問に相当するリスト述語パッドをVocalogパッドの上に貼付することで行えるようにした．

VocalogはPrologに名詞や形容詞といった語彙を導入し利用できるようにしたシステムなので，VPEをPrologの視覚的プログラミング環境として用いることも可能である．VPEでは，Vocalogの名詞や形容詞を視覚オブジェクトとし，それらを組み合わせてプログラムを構成するシンセティックプログラミングとIntelligentPadのシンセティックプログラミングとの間に緊密な対応関係をとることで，従来Prologでは不可能であったシンセティックプログラミングを論理プログラミングに導入することが可能となった．

VPEでは，IntelligentPadシステムの資源を使うことができるので，メディアオブジェクトとしてのパッドをより多く取り入れていくことで，視覚的なプログラミング環境をさらに充実していくことが可能である．

謝辞

本研究を行うにあたり，多数の方々のご援助をいただきました．

北海道大学工学部電気工学科　田中譲教授には，研究に望む姿勢，論文の作成に関する貴重なご助言，および叱咤激励をうけました．田中譲教授のご指導にまことに感謝致します．

北海道大学工学部電気工学科応用制御工学講座の皆様には，貴重な討論，および，研究上の支援・協力をいただいた．心より感謝致します．

参考文献

[1] Colmerauer, A., Kanaui, H., Roussel, P. and Pasero, R. : *Un systeme de Communication Homme-Machne en Francais*, Group de Recherche en Intelligence Artificielle, Université d'Aix-Marseille , 1973.

[2] Kowalski, R.A. : *Predicate Logic as a Programming Langueage*, IFIP 74, pp. 569-574, 1974.

[3] Kowalski, R.A. : *Algorithm = Logic + Control*, CACM, 22, 7, pp.424-436 , 1979.

[4] 横田一正：レコード・プログラミング，続新しいプログラミング・パラダイム，pp. 53-75，共立出版，1990.

[5] Ait-Kaci, H. : *.An Algebraic Semantics Approach to the Effective Resolution of Type Equations*, Theoretical Computer Science, Vol.45, pp.293-351 , 1986.

[6] Ait-Kaci, H. and Nasr, R. : *.LOGIN : A Logic Programming Language with Built-in Inheritance*, Journal of Logic Programming, Vol.3, pp.185-215 , 1986.

[7] Mukai, K. : *.Anadic Tuples in Prolog*, ICOT-TR 239 , 1987.

[8] Yokota, K. : *Deductive Approach for Nested Relations*, ICOT-TR, 1988.

[9] Tanaka, Y. : *Information Space Model*, Proc. 2nd Workshop on Formal Bases for Databases, Toulouse, 1979.

[10] Tanaka, Y. : *Vocabulary Building for Database Queries*, Lecture Notes in Computer Science, vol.147: RIMS Symposia on Software Science and Engineering, Springer-Verlag, pp.215-232 , 1983.

[11] Tanaka, Y. : *Roles of a Vocabulary in Knowledge-Based Systems*, IFIP WG 10.1 Int'l Workshop on Knowledge Architecture, Gotenba, 1987.

[12] Tanaka, Y. : Vocabulary-Based Logic Programming, *Proc. InfoJapan'90*, Tokyo , Vol.2, pp. 25-32, 1990.

[13] Chen, W. and Warren, D. (1989).*C-Logic of Complex Objects*, PODS'89, pp369-378 .

[14] Kifer, M. and Lausen, G. (1989).*F-Logic : A 'Higher-Order' Logic for Reasoning about Objects, Inheritance, and Scheme*, SIGMOD '89 .

[15] Maier, D. (1986).*A Logic for Objects*, Proc. Workshop of Deductive Database and Logic Programming.

[16] Goto, F. and Tanaka, Y. : *Introducing a Large Vocabulary into Prolog*, In Proc. of Pacific Rim International Conference on Artificial Intelligence '90 , pp.816-821, Nagoya Japan,Ohmsha, 1990.

[17] 後藤文太朗, 田中譲 (1992).Vocalog: 語彙を用いたジェネリックな概念記述が可能な拡張論理型プログラミング言語, 情報処理学会論文誌第33巻第4号, pp. 512-520 .

[18] Almgren, J. , Andersson, S. , Flood, L. , Frisk, C. , Nilsson, H. , and Sundberg, J. : *SICStus Prolog Library Manual*, Swedish Institute of Computer Science, 1993.

[19] Barwise, J. and Etchemendy, J. : *THE LANGUAGE OF FIRST-ORDER LOGIC*, CSLI Lecture Notes Number 23, CSLI, 1990.

[20] Bonsignori, A., Giannotti, F., Lucchesi, L., and Turini, F. : *The Logiform System*, Computers & Mathematics with Applications, Vol.20, No. 9/10, pp.83-99, 1990.

[21] Brayshaw, M. and Eisenstadt, M. : *A Practical Graphical Tracer for Prolog*, Int. J. Man-Machine Studies, vol.35, pp. 597-631, 1991.

[22] Dewar, A. D. and Cleary, J. G. : *Graphical display of complex information within a Prolog Debugger*, Int. J. Man-Machine Studies, Vol.25, pp. 503-521, 1986.

[23] Eisenstadt, M. and Brayshaw, M. : *The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming*, J. Logic Programming, Vol. 5, pp. 227-342 , 1988.

[24] Ladret, D. and Rueher, M. : *VLP: a Visual Logic Programming Language*, Journal of Visual Languages and Computing, Vol. 2, pp. 163-188 , 1991.

[25] Numao, M., Morishita, S., and Maruyama, H. : *How Should Prolog Computation Be Represented for Practical Use ?*, New Generation Computing, Vol. 8, pp. 95-112 , 1990.

[26] Pau, L. F. and Olason, H. : *Visual Logic Programming*, Journal of Visual Languages and Computing, Vol. 2, pp. 3-15, 1991.

[27] Polak, J. and Guest, S.P. : *A Graphical Representation of the Prolog Programmer's Knowledge*, Visualization in Human-Computer Interaction (7th Interdisciplinary Workshop on Informatics and Psychology, Selected Contributions), pp.82-104 , 1990.

[28] *Quintus X Window Interface*, Quintus Corporation ,1991.

[29] Tanaka, Y. , Nagasaki, A., Akaishi, M., and Noguchi, T. : *A Synthetic Media Architecture for an Object-Oriented Open Platform*, Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992, pp.104-110 , 1992.

[30] Tanaka, Y. and Imataki, T. : *IntelligentPad: A Hypermedia System allowing Functional Composition of Active Media Objects through Direct Manipulations*, Proc. of IFIP World Computer Congress '89, San Francisco, pp. 541-546, 1989.

[31] van Emden, M.H., Ohki, M., and Takeuchi, A. : *Spreadsheets with Incremental Queries as a User Interface for Logic Programming*, New Generation Computing, Vol.4, pp.287-304 ,1986.

[32] Yeung, R. : *MPL - A Graphical Programming Environment for Matrix Processing Based on Logic Programming, 1988 IEEE Workshop on Visual Languages*, pp.137-143 , 1988.

[33] 渋谷正弘, 田中譲 : スプレッド・シートを介した論理型プログラミング, 情報処理学会論文誌第30巻第6号, pp.709-718, 1989.

[34] 後藤文太朗, 田中譲 : VPE: 論理プログラミングにおける視覚的統合プログラミング環境, 情報処理学会論文誌採録決定ずみ.

[35] Apt, K.R. and van Emden, M. H. : *Contributions to the Theory of Logic Programming*, JACM 29, 3, pp. 841-862, 1982.