日本語に基づく論理プログラム表現

小谷 善行†

一般人の扱いやすい，日本語表現の論理型プログラミング言語の仕様を設計した．Prolog などの論理型言語は，表現が非日本語的であるだけでなく，述語の引数に存在する変数の間の対応をよく理解するには数学的素養が必要であり，なじみやすくはない．そこで単なるワープロ使用者にも利用可能とすることを想定し，自然な日本語の文構造や語彙体系を素直に反映したプログラミング言語の仕様を設計した．本言語仕様は，述語の引数は陽には記述しないという特徴をもつ．データは，入力と出力の２ポートに区分して述語に与えられる．プログラミングは，メタ述語と呼ばれる演算子で述語間の結びつきを指定することで行う．述語は主としてユーザが定義し，通常，日本語の名詞を用いると読みやすい設計となっている．メタ述語は主としてシステム組込みであり，助詞や接頭語で表される．メタ述語は日本文としての理解と一致するように，語義に基づき選定されている．プログラムは分かち書せず，「祖母とは親の母」のように，ほとんど日本語そのままのプログラム表現となる．本言語によるプログラムは，Prolog プログラムを日本語で自然に読み下した文に非常に近い．さらに意味ネットワークなどの知識を記述する観点からも，本仕様はその意味を自然に表現した形になっている．また，本仕様に対しその意味定義を与え，プログラム例による可読性および，システムを試作し効率を調査した．

Logic Program Description Based on Japanese Language

Yoshiyuki Kotani†

A logic programming language specification is discussed, which is expected to be friendly for general users. It may be difficult for them to deal with many variables occurred in arguments of Prolog predicates. Additionally it is not based upon Japanese language. We propose a logic programming language in which each predicate has no arguments and design a Japanese-language oriented style which reflects natural structure of sentences. The relation between predicates is described by a meta-predicate. Arguments of predicates are not described explicitly. Programming is specifying meta-predicates to make clauses. Meta-predicates are built-in names, for which postpositions and suffices of Japanese are used so that description agrees with semantics. Programs are not segmented. We investigated its semantics and readability. Also performance is measured by making the system written in Prolog.

1. 序　論

Prolog 言語を勉強し始めると，たいてい，member 述語の定義：

```
member(X, [X|L]).
member(X, [Y|L]) :-member(X, L).
```

に出会う．われわれは，member の定義を日本語で説明するときどうするだろうか．普通つぎのように説明するにちがいない．

「リストの先頭はそのメンバーである．それから，そのリストの先頭を除いた部分のメンバーであるものも，そのリストのメンバーである．」

一方，変数名を言いながら，

「XはXとLのリストのメンバーである．それからXがLのメンバーなら，XはYとLでできたリストのメンバーである．」

などということは少ない．引数や変数の意味を説明したいときにだけそうする．すなわち，「X」や「Y」の仮の名前，すなわち変数より，「リストの先頭」などといって，既存の概念の結合で説明することが多い．

この現象は，日本語や英語をはじめとする自然言語では，「変数」に相当するものがほとんど使われないことを反映している．自然言語で変数に相当するものとは，仮につける名前である．契約書などでは，甲，乙，丙などを使って特定のものを指示する．数学などではａ，ｂ，ｃなどを使う．しかし，そうした変数は，われわれの日常会話では出現しない．むしろ概念クラス名や代名詞を使って指示物を表現する．そのほうが自然であり，理解も速い．このことを，プログラ

† 東京農工大学工学部電子情報工学科
Department of Computer Science, Tokyo University of Agriculture and Technology

ム言語の問題の上に移せば，

「変数がない（少ない）ような言語設計で，使いやすいプログラム言語が実現する可能性がある」

ということが示唆される．

一昔前，goto 文は構造化プログラミングの考えのもとに批判された．ここで goto 文という言葉が指しているのは，goto 文自体とその行き先のラベルの組である．この組で，プログラムの構造に無関係なリンクがあまりにも自由に書けることが問題であった．同じように，プログラムのなかの変数も，場所と場所を結び付けるリンクである．変数が多いこともまたプログラムの性質として悪いことであると考えられる．

多くの一般大衆がプログラミングに適応できないのは，変数に適応できないことにもあると予想される．計算機システムにかかわらない社会生活においても，複雑な手順や複雑な論理的関係はつねに存在しており，一般の人間はそれを自由に処理している．しかし，一般の人間は，手続き型プログラム言語にせよ非手続き型プログラム言語にせよ，適応できない人の方が多い．したがって，自然言語に出現するのが稀な「変数」もその不適応の一因と考えることができる．

また，一般に，同一のアルゴリズムを書くために，コーディングの表現の形が多すぎないのが望ましい．なぜなら，多様な表現が許されるなら，利用者がプログラムを読んでそこで認識される意味も一つに定まらなくなることがある．たとえば，Prolog において，child(X, Y) という表現で親子関係データを表現するなら，XとYをどちらを親にし子にするかは，言語からは定まらない．それは，普通は「プログラミングのしつけ」，つまり，そうすべきだと指導することで規定するしかない．

grandchild(X, Z) :-child(X, Y), child(Y, Z).

という表現をみたすだけでは，Xが孫を表すのか祖父母を表すのか直感的にはわからない（同時に，理解するためには引数の対応関係もみなければならない）．実際，自然言語では「子供の子供」というように，こうした無駄な自由度は少ない．また，手順を表現する手法の分野でも，自由度が少ないことが表現を分かりやすくするということが主張されている[12]．

次に，今日までの引数のない（または少ない）言語表現について論じる．そうしたプログラム言語の一つとして Backus の関数型言語 FP がある[3]．その意義は，ユーザ・インタフェースよりもむしろ，並行処理によるフォンノイマン・ボトルネックの解決の可能性，およびプログラム代数を理論化することによるプログラム検証にあると思われる．また，FORTH を始めとするスタック言語など変数（そして関数の引数）が明示されない言語が存在するが，それらは手続き型言語である．

論理型プログラミング言語については，引数の位置にリテラル（述語呼出）が埋め込まれた，言語が考えられている[1),2),7),8)]．そこでは，Prolog のリテラルを，別の項の引数の部分に埋め込んだ形式が提案されている．これは，表記上は項書換えシステムに似た，論理プログラム言語系になっている．これは，出力変数が埋め込みによって消えることになるため，平均すれば述語１個について，引数が１個減ると理解される．これで引数そして変数の出現が減らされ，また，一般の関数表記と一致することになるため，可読性など利用者インタフェースが向上したと考えられる．異なる方法としては，述語の引数に命名することにより，概念記述を容易にすることも研究されている[9)]．これも意味から独立した変数表記をなくすという観点からとらえることができる．

自然言語処理においては，文法表現のモデルとして，DCG がある[10),11)]．この特徴は，表現が文脈自由文法に類似したものであり，かつ構文解析を行う Prolog プログラムが若干の変換により得られるということである．表現は主として，Prolog プログラムの引数にあるディファレンス・リスト（解析対象文の部分列を表す）を隠ぺいしたものであると考えることができる．これは言語処理分野での，引数をなくすることの利用者インタフェースの例である．

以上の議論をもとにすると，利用者親和性の立場から，「論理型プログラム言語における引数の隠ぺい」が重要な鍵であると考えられる．引数を隠ぺいすることによりはじめて，その言語表現は自然言語に近いものとなりうる．われわれは，ここで言語表現を設計するにあたって日本語を採用する．まず，日本語プログラム言語について概観する．

日本語プログラム言語は，プログラム言語が日本で用いられ始めた当初から考えられていた．しかしその普及は成功しなかった．その理由は第一には，プログラム言語を一度修得した人にとっては，その表現が日本語であることは，なんの役にも立たないということがある．新しく日本語プログラミング言語がもたらされたとしても，かえって新しく用語を覚える手間を生じる．第二には，歴史的に，プログラミング環境で効

率的に仮名漢字文を扱えなかったからである．

このうち第二の理由は現在では解消しつつある．第一の理由にかかわりなく，変数名や手続き名などの識別名を日本語化する実際的な日本語プログラム言語の研究がある[13),14)]．一方日本語ワープロやパソコンの出現により，日本で数百万人規模の人間が新たにプログラミングをしうる環境をもつこととなった．こうした人たちに対しては第一の理由は成立しない．アプリケーションソフトウェアのマクロ定義など，いくつかの日本語言語といえるものが，単に予約語を日本語に置き換えたものを含め，実際に用いられている（たとえば，日本語マインド，LOGO ライタ[15)]，ひらがな LOGO，桐などの言語やマクロ）．しかし，それらについて，自然言語との関係や計算モデルについて，まだ学術的な見地からの研究は行われていない．

本論文では上記のように，第一に引数をなくすことにより，利用者親和性をもたらすこと，第二に自然言語表現，特に日本語表現として適切な表現をとること，という観点から一つの論理的プログラミング言語の言語仕様（LOgical Natural lanGuage description；LONG，論具）を設計し，その有効性を検討する．

2. 機能設計の枠組み

仕様 LONG の表現の基本的単位を Prolog と同様，述語と呼ぶ．Prolog では，文法上，明示して述語に対して引数を与え，それによりデータを交換する．一方，本仕様の場合は上述の設計方針により，述語に対する引数を明示的には示さない．引数の代わりに，述語間の関係を表現する方法をとる．

2.1 述語のデータポート

Prolog のような論理プログラム言語では，仮引数と実引数間で変数が単一化でバインドされるため，両方向参照が可能である．しかし現実のプログラムでは，両方向参照の行われている引数が非常に少ないことが報告されている．両方向参照が一つもないプログラムも多い．一般の Prolog では，引数の両方向参照性はその構文に反映していない．

これを考慮し，LONG ではデータを受け渡す場所を二つに分割し，おのおの出力データと入力データを扱うものとする．その場所をおのおの，出力ポート，入力ポートと呼ぶ．Prolog では通常，モード宣言で入力データと出力データを指定することによりコンパイラを効率化している．したがってこの効率化のためにはユーザがモード宣言をする必要がある．

一方，LONG では両ポートは表現により自動的に定まるという方式をとる．ポートを他の述語のポートに接続する記述を行って述語を組み合わせることがプログラミングとなる．

データの方向については，アリティが2の Prolog の述語が2方向性ポートであるといえるのに対して，LONG は「1方向性ポート」であるといえる．

すなわち，1方向性ポートは FP と同様のデータ交換になる．ただし，FP と異なり，LONG は非決定的な機能をもつ．Prolog としてみると，両方の引数に対して入出力のモードを固定したものとみなせる．

逆に，1方向性ポートは逆方向計算ができず，また2引数とも入力であるような計算ができないが，1方向性ポートにすることにより，処理速度の高速化が図れる（なお，逆方向計算は，後述のメタ述語の一つである程度カバーする）．また，自動的にモード宣言をしたことになり，Prolog におけるような手間がない．

Prolog に比べて，引数が二つにまとまることは，内部的な変数の分類ができない点で，高速化しにくい点もある．しかし述語の形が単一形式となり，その単純性が効果を生む可能性もある．

2.2 述語の機能

述語は上のように1方向性としたことにより，次のような順序で動作する機能をもつ．

①入力ポートからデータを受け取る
②データにより計算を行い失敗または成功をする
③成功した場合，出力ポートにデータを出すとともに，出力側に述語の式があればそれを起動する
④失敗した場合，入力側の述語の処理にもどる
⑤出力側の述語式からもどってきたとき，さらに計算して再び②以下を行う

このように Prolog の計算と類似するが，データは授受の相手が固定的である．

ポートを通じてやりとりされるデータは，リスト（複合項）ないし単純な名前や数値の定数とする．すなわち，Prolog のデータと考えてよい．

3. 言語設計

本プログラム言語を設計するにあたって，自然言語，特に日本語の用法と一致させることを重視し，表現形式の設計の考えを述べる．

3.1 日本語表現とプログラム表現の対応付け

一般に自然言語の普通名詞は，外延的意味としてインスタンスの集合を定めるような，ひとつの名称とし

てとらえられる．しかし多くの場合，その機能だけではなく，普通名詞は，ある（名詞によっては特定の）カテゴリーの名詞（句）によって限定され，その集合の部分集合または要素を指し示す．たとえば「母」という言葉は，人間名詞というカテゴリーによって限定される．そして，限定された詞句（たとえば「太郎の母」）は限定している名詞の意味する人の母（たとえば「花子」）を示す．自然言語の運用においては名詞句による限定は必ずしも特定の限定ではない（たとえば，「瞼の母」）が，通常は特定の限定が頻度高く用いられる．そして「ジャイアンツのメンバーの母の年齢の平均」のように限定要素の意味が一意に定まることが多い．

ここで，二つのポートをもち，カテゴリーによる限定を入力として受け，部分集合（か要素）を出力するような本方式の述語を考える．そうすると，このことはその述語の表現形式として普通名詞（あるいはそれに近い言語表現）を使うのが適切であるといえる．数学的表現においても f(g(h(x))) という形は，日本語の表現にしたがえば「ｘのｈのｇのｆ」と読まれる．これは日本語の語順が逆ポーランド的であるためである．ｇ，ｆ，ｈという関数を要素とみた場合は，中置辞記法とみられる．ここで「の」はそれらを直列に接続する働きをもつ．また，「の」以外にも「父や母」における「や」などの等位表現助詞も同様の表現形式を接続し，別の意味を規定する．以上の議論を設計方針としてまとめると，次の３点となる．

（1） 述語は用法上，名詞的表現を使うことを想定する．

（2） 述語のあいだに「の」のような中置辞を置き，その組合せの意味を規定する．

（3） 述語を組み合わせる作業自身がプログラミングである．

3.2 構文の設計

この方針にしたがい，本言語仕様は基本的には述語を中置辞で結び付けつつ組み合わせたものとする．ここでのこの中置辞に相当するものを２項メタ述語と呼ぶ．また，後述の単項メタ述語も用意する．これらにより，概略的には基本単位である式は次の BNF で表現する．

〈式〉::=〈述語〉|〈式〉〈２項メタ述語〉〈式〉|
〈単項メタ述語〉〈式〉|〈式〉〈単項メタ述語〉

ここで，単項および２項のメタ述語は，式に対する演算子であり，結合した結果も式である．用意されているメタ述語は次の節で述べる．メタ述語の間には演算子優先順位があり，その順で部分式が結合する．また，括弧（ ）によって先に結合する部分式を示す．

LONG のプログラムは手続きからなる．手続きは同一の述語を頭部にもつ節の集まりである．手続きは一つの述語の機能を定義する．節は式の形をしているが，詳しくは，

〈定義節〉::=〈定義される述語〉とは〈一般の式〉.|
〈一般の式〉の〈定義される述語〉は
〈一般の式〉.

という形をもつ．これは Prolog の定義節に対応するものである．最初の形式は Prolog の語順に近いものである．二番目は意味ネットワークを表現することを考慮した形式である．

ゴール節に対応するのは，

〈ゴール節〉::=〈定数述語〉の〈一般の式〉.

という形をしている（定数述語は後述）．

全体の構文を付録１に示す．

3.3 日本語語彙の選択

特定の機能のメタ述語を設計し，そのあと，その名称を決定した．日本語を含め，どの自然言語も個々の語にははっきり規定された意味がある．その意味との整合性をとるように配慮した．２項メタ述語は可能な限り助詞を用いるようにした．したがってこれは平仮名表記となる．単項メタ述語はなるべく漢字の接頭語的な形になるようにした．

4. メ タ 述 語

メタ述語とは，引数として式をもつ演算子である．述語を組み合わせてより複雑な機能を表現するために用意されるものである．LONG では，述語をメタ述語で組み合わせることでプログラムを記述する．以下に個々のメタ述語とその機能を示す．その際，例となる節表現に対して，日本語の意味，および同じ機能をもつ Prolog 節を示すことにより解説する．

4.1 ２項メタ述語

２項メタ述語は二つの述語の間に置き，両者を組み合わせる．それらには仮名表記の，等位助詞などの助詞を用いる．一般文では名詞は漢字列または片仮名列（外来語の場合）が多い．名詞として表される述語名と明確に区別するために２項メタ述語は平仮名列で構成する．

とは

Prolog の「:-」にあたるもので，述語の定義をす

る．一般に，日本語では「と」は引用を指示する助詞である．「は」は，既知物（多くの場合文のテーマ）を文中に指示する．結果として「とは」は多くの場合，語を定義することを示す．したがってここに「とは」を用いることは適当である．例を示す．

日本語の意味：旦那とは夫のことである．

LONG の表現：旦那とは夫．

Prolog の表現：danna(X, DANNA) :-
otto(X, DANNA).

の

この演算子で二つの式を結合すると，第一の式の出力ポートが第二の式の入力ポートにつながれる．式が入力から出力への写像を定めるものと考えると，「の」は二つの写像を直列につなぐことを意味する．第二の式の出力が全体の出力になる．両者の出力がないと全体として出力がない．

「の」には名詞化の「の」など数種の用法があるが，名詞句どうしを結び付ける「の」は名詞を限定するという意味では単一の用法しかない．また，前述のようにその限定のしかたは「の」の後の名詞で決まる場合が多い．そこで上の形式において「の」を用いる．

また，格助詞「が」を「の」と同義のメタ述語として用いる．これは続く表現が真偽値を求めるときに使い，その表現を適切にする．

日本語の意味：孫とは子の子のことである．

Prolog の表現：mago(X, MAGO) :-
ko(X, KO), ko(KO, MAGO).

LONG の表現：孫とは子の子．

で

この演算子でつないだ二つの式に同じ入力を入れ，両者が同じ出力を出すならば，それを全体の出力とする．同じ出力がないならば，失敗する．すなわち，両式の共通部分を表す．

これは二つの表現の AND 結合を示す．しかし，日本語の表現を選択するにおいて，その翻訳である「と」や「そして」や「かつ」は不適切である．論理的に二つの事柄が並立するという意味で，断定の助動詞「だ」の連用形の「で」を採用した．なお，「である」の連用形である「であって」ももう一つの候補として考慮している．これは文字列として長いという欠点があるが，「で」のような多義性がない．

日本語の意味：兄とは兄弟であって
年上の人のことである．

Prolog の表現：ani(X, ANI) :-brother(X, ANI),
toshiue(X, ANI).

LONG の表現：兄とは兄弟で年上．

や

この演算子で二つの式を結合したものは，両者に同じ入力を入れ，順に出た出力を全体の出力とする．すなわち，両式の出力を非決定的に与えるものを表す．

ここでこの意味を表現する日本語の候補としては，他に「か」，「または」，「と」，「および」，「かつ」，「そして」などがある．「と」，「および」，「かつ」，「そして」は，結合した二つのものが両方成り立つという意味では正しい．しかしそれらは両者以外に成り立つものがない（exhaustive）というニュアンスがある．また，「かつ」や「そして」は名詞を結び付けるものではない．「と」は単に両方という意味ではなく，一組のものを合わせた全体というニュアンスがある．

一方「や」には両方が成り立つという意味をもつが，他にも成り立つものがあるということの可能性を示唆するニュアンスがある．この性質は他の句によってその述語が成り立つという非決定性を示すために都合がよい．その結果「や」を用いることに決定した．

日本語の意味：brother とは兄や弟のことである．

Prolog の表現：
brother(X, BR) :-ani(X, BR) ; otouto(X, BR).

LONG の表現：brother とは兄や弟．

なお，Prolog で，これを二つの節に分けられるのと同様，LONG でも，

brother とは兄．brother とは弟．

というように二つの節で表現することもできる．

なら（なら および ほかは）

二つの式に同じ入力を入れ，最初の式が成功して何らかの出力があるなら，（後の式が「ほかは」で結び付けた形でないとき）後の式の出力を全体の出力とする．最初の式が失敗すれば，全体としても失敗する．

後の式が「ほかは」で結合した形のとき，最初の式が成功して何かを出力すれば「ほかは」の前の述語に入力を入れその出力を全体の出力にする．そうでない場合は，「ほかは」の後の述語で同様のことをする．

日本語では条件や仮定を表すのに，他には「ば」，「と」がある．これらは用言に接続するのでここでは用いにくく，「なら」を採用した．

日本語の意味：絶対値とは，ある数が負なら，その数の符号を変えたもので，そのほかのときは，その数自身である．

Prolog の表現：abs(X, ABS) :-

(X>0->ABS=X ; ABS is -X).

LONG の表現：絶対値とは負なら符号反転ほかは自身.

[ ]

角括弧は，そのなかに並んだいくつかの式に同じ入力を入れ，おのおのの出力を並べたものを全体の出力とする．その式のなかで一つでも失敗するものがあれば全体として失敗する．

Lisp や Prolog と同様の，リスト表記とドット表記をもつが，これは syntax sugar である．つまり，角括弧の表現は「と」というメタ述語の式であり，

[式1|式2]

は，内部的には，“式1と式2”という形をしたものである．

ここで「と」を用いた理由は，上述のように「と」には，両者を合わせたもの，あるいは「いっしょに」という意味を含んでいるからである．そのため，データとして組み合わされたものを示すこととした．

日本語の意味：ある人の個人情報とは，その氏名自身，住所，電話番号からなる．

Prolog の表現：kojinjoho(X, [X, JUSHO, DENWA]) :-
juusho(X, JUSHO),
denwa(X, DENWA).

LONG の表現：個人情報とは [自身，住所，電話].

は

「は」は「とは」と同様の述語を定義するメタ述語である．これは特に単純な事実節をわかりやすくするために用いる．事実節は意味ネットワークのアークデータとして理解することもできる．

「とは」が用語の定義を表すニュアンスが強いのに対して，「は」は個別の状況に対する記述を表すニュアンスが強い．そこで別形式の定義節

〈式1〉の〈定義される述語〉は〈式2〉.

により，その定義される述語が起動された場合，それへの入力とを式1の出力が一致すれば，それを式2の入力とし，式2の出力を述語の出力とする．

日本語の意味：太郎の母は花子である．

Prolog の表現：haha (taro, hanako).

LONG の表現：「太郎」の母は「花子」.

(ここで「太郎」などは定数述語で，後述する)

これと同じ機能のものを「とは」で表現するのは，

LONG の表現：母とは「太郎」であるなら「花子」.

Prolog の表現：haha(X, Z) :-
(taro=Y, Y=X)->hanako=Z.

となる（詳細は次節）.

4.2 後置単項メタ述語「である」と定数述語

LONG のプログラムでは，要素はすべて述語かメタ述語である．定数自体も述語として扱い，定数述語という表現を用いる．これは表現のなかに一つの定数をパラメータとしてもつ述語である．

定数述語はプログラム中では，定数値を鉤括弧「 」でくくったもので表される．日本語では鉤括弧は，引用の形式として，述べられた表現を指示する場合がもっと多い．したがってこのような表記にすることは適切であると考えられる．その機能はどんな入力に対しても，一度だけ成功し，パラメータを出力する．

定数述語は定数を出力するものであり，その点を強調して出力定数述語とも呼ぶ．一方，入力と，ある定数との比較を行いたい場合がある．これに対して，後置辞型単項メタ述語「である」を用意する．これは，

〈式〉である

の形式をとり，式への入力と出力が一致する場合，一度だけ成功して「真」を出力する．一致しない場合は失敗する．したがって，式に定数述語を与えたもの，たとえば，”「太郎」である”という形式は，それへの入力が太郎であるときに一度だけ成功して「真」を出力する．そこで，出力定数述語に「である」を後置した表現を入力定数述語とよぶ．これにより，たとえば「母が（「太郎」の娘である）なら」といった表現でも文字どおりの条件判定を行える．

4.3 前置単項メタ述語

前置単項メタ述語はおもにデータの非決定性や集合としての処理に関連するものである．これらは漢字一字の接頭辞で表しやすい．こうすることにより，2項メタ述語の表現と干渉しないものとした．

各

全体の入力であるリストの各要素にその述語を作用させ，その出力をつないだリストを全体の出力とする．Lisp の map 関数，FP の α に相当するものである．

「各〜」という表現は集合の個別の要素を示す用法があるので，「各」を採用することとした．

日本語の意味：個人情報表とは各人の個人情報のリストである．

Prolog の表現：kojin_hyou([], []).
kojin_hyou([X|XL], [Y|YL]) :-
kojin(X, Y), kojin_hyou(XL, YL).

LONG の表現：個人情報表とは各個人情報.

総

全体の入力であるリストの先頭から二つの要素を取り出し，その組に対して述語を作用させ，それと第3の要素との組にさらにその述語を作用させる，というように繰り返して，最後まで計算した結果を全体の出力として出す．FP の「／」に相当するものである．ここでは全体で，という意味で「総」を用いた．

日本語の意味：合計とは一つ一つの数の総和のことである.

Prolog の表現：
```
goukei([X], X).
goukei([X, Y|L], SUM) :-
    XY is X+Y, goukei
        ([XY|L]), SUM).
```

LONG の表現：合計とは総和.

全

入力に対して述語を作用させ，成功して出力されるものを順に並べて，リストとしたものを全体の出力とする．Prolog の bagof に対応する述語である．

ここでは成り立つものすべてという意味で，「全」が意味的に一致するので用いた.

日本語の意味：すべての孫の表とは孫であるものをすべて並べたリストである.

Prolog の表現：
```
magohyou(X, HYOU) :-
    bagof(MAGO,
        mago(X, MAGO), HYOU).
```

LONG の表現：孫表とは全孫.

逆

式に対して，入力と出力を逆転した機能のものを作る．「は」による事実節に対して用いることができる.

日本語の意味：親とは子の反対の概念である.

Prolog の表現：oya(X. OYA) :-ko(OYA, X).

LONG の表現：親とは逆子.

以上で各種の組込みメタ述語および定数表現を示した．例で理解されるように LONG の表現は，(1)日本語表現に近く，(2)短い，という特徴をもつ.

5. その他の設計仕様

前節で述語を結合するためのメタ述語および定数の表現を述べた．ここでは日本語的表現のために必要な，語の区切り方式および組込みの述語を説明する.

5.1 語の句切り

今日までの日本語プログラム言語では，形態素ごとにスペースで区切るなど，すべて分かち書きで記述されていた．しかし，分かち書きは今一般に使われている日本語表現とは異なる．LONG では，一般の言語表現と同じ，べた詰めの表記を用いる.

そこで語すなわち，述語やメタ述語をシステム側が区切る必要が生じる．これに対し，LONG では，

「文字種の境目で区切る」

という方法を採る．これに従い，述語名となるのは

(1) 平仮名以外の単一の文字種からなる列

とした．なお，それ以外の文字列として，

(2) 二重引用符（"）でくくられた記号列

を述語名として許す．たとえば，

絶対値とは負ならマイナスほかは自身.

という列は，

絶対値/とは/負/なら/マイナス/ほかは/自身/.

と区切られる.

このようにした理由は，形態素の境界と文字種の境界とがよく一致する事実があることである．特に LONG の主要な要素である名詞のつながりについてはその性質が強い．そして境界が一致しないのは，多くが動詞の連用形を名詞化したものであり，その場合送りがなを省略しても理解可能である（たとえば「申し込み」を「申込」とするなど）.

上の方法以外に，(1)メタ述語の前後で区切る方法，および(2)仮名漢字変換辞書を利用して区切る方法を検討した．これらは文字種で区切る場合より，自然言語として正しい区切りになる場合もある．しかし，プログラマに対してプログラミング時に，よけいな判断をさせるべきではない，という点を考慮し，現在では上記の方式としている.

なお，区切りと実行のかかわりであるが，実行時にはすでに構文解析された結果が保持されているため，区切りが実行速度に影響することはない.

5.2 組込み述語

組込み述語としては，既存のリスト処理言語のもつリスト処理の多くをカバーするような述語を用意している．たとえば，「最初」(car にあたる)，「残」(cdr にあたる) などである．cons 系のものは Prolog のようにリストそのものの形を表面上はとる（付録のプログラム参照）．中身は上述のようにメタ述語「と」で表現されている.

データを後の処理に渡すために，「自身」という述語が用意される．これは入力されたものをそのまま出力するものである.

データをプログラムとして評価する組込み述語「意

味」が用意される．"「太郎」の母」の意味"により，"「太郎」の母"と同じ処理が行われる．

なお，今後の言語の拡張として，組込み述語「意味」と syntax sugar の定義を組み合わせてメタ述語のユーザ定義を行わせることを検討している．

6. 議　　論

本稿で提起した日本語論理型言語に関して，まず，その表現形態の意義を総合的に論じる．付加的事項として，さらに LONG の機能および効率について議論する．

6.1 表現形態の特徴

LONG は，日本語の名詞を中心とした修飾構造をモデルとした表現にもとづくものとして規定できる．それを可能とするための設計上の特徴は(1)引数がない，(2)非決定的である，(3)名詞的表現を基準としているという点にある．また，付加的な設計としては，(4)分かち書きをしないことがある．ここでプログラム例を示す（付録2）．このうち，血液型のプログラムは次のような日本語で規定されることがらを扱っている．

A型の遺伝子はAとAやAとOの対である．B型ではBとBやBとOの対である．AB 型ではAとBの対で，O型ではOとOの対である．花太の父は太郎で，母は花子である．太郎の血液型はOで，花子の血液型は AB である．子供の血液型は，父と母の各々の血液型の遺伝子対の一方を取り出して作った遺伝子に対応する血液型である．

このとき，花太の血液型は何か．

このようにみると，日本語による説明と，プログラムは，実際的にもかなり近いことがわかる．記述性および可読性に関して，量的測定を行うことは，慣れの問題に関わるので困難である．しかし，LONG による表現は，宣言的な表明における，日本語のニュアンスをほぼ反映していると考えられる．

次に，すべてのプログラム言語に普遍的に存在するような制限が，LONG にも存在することをふれておく．たとえば，プログラム言語は文法が曖昧でない（構文木が複数存在しない）のに対し，自然言語では曖昧である．LONG も，自然言語と明らかに異なる．実例をあげると，「の」の用法に関して，LONG で採用しているのは，名詞句が名詞句を制限的に修飾する場合（たとえば，「太郎の母」）だけである．たとえば「私は母の花子です」における叙述的修飾で同格の「の」は LONG で扱わない (LONG では「で」を用いる)．もちろん「私のです」や「赤いのがいい」などの名詞化辞としての「の」も扱わない．すなわち，個々の組込み語はすべて単一の用法しか許されていない．これは既存のプログラム言語が特定の予約語を決めて単一の意味を持たせているのと同様である．この意味で LONG はあくまでもプログラム言語であり，制限された自然言語ではない．

6.2 表現における情報の隠ぺい

次に，情報の隠ぺいについて論じる．プログラム言語の質に関する重要な指標の一つに，情報の隠ぺい性がある．それはプログラムの作成の容易さ，保守性，安全性をもたらす．LONG では引数の表示をなくすことにより情報の隠ぺいを容易にしている．ここでは応用例として，自然言語処理における2項メタ述語による処理の隠ぺいについて，その意味を述べる．

自然言語処理においては，論理的な文法表現として DCG が提案されている．これによると，処理対象を示す差リストを隠ぺいして，プログラム表現をほとんど形式文法表現と一致させることができる．例えば，文が主語（名詞句）と述語（動詞句）からなることを示す，

```
sentence(A, C) :-np(A, B), vp(B, C).
```

は，

```
sentence --〉 np, vp.
```

という形に表現される．さらに，生成される構文木や，部分木間で情報を伝達するための引数も隠ぺいすることができる．この場合，隠ぺいされたデータの処理は DCG 表現（--〉による節）をインタープリタが解釈する際に行われるか，またはコンパイルされる際にそれらの引数が付いたホーン節にもどして行われるかである．

これに対して，LONG を用いて一つの新しい方式を提起することができる．上の例に対応するものは，

```
sentence とは np＋vp.
```

と表現される（ただし，syntax sugar をもつ拡張版による．それがない場合は「sentence とは np と vp の結合」といった表現になる）．ここでいろいろなデータはすべて隠ぺいされている．その処理は利用者定義のメタ述語「＋」に書かれることになる．すなわち，DCG とは異なり，文法規則の右辺にある連接の機能として多様な処理を記述する．

6.3 言語表現に関する同値性

次に LONG の効率や機能の側面を論じる．まず，

言語表現に関する性質について述べる．LONG は設計上，純 Lisp や純 Prolog と同様，計算モデルとして純粋性をもつ．つまり，式の等価性を定義することによりプログラム代数[3),4)]を組み立てることができる．プログラム代数はプログラムの性質を証明したり変形したりする道具となる．本言語系では非決定性があるので，2種類の同値関係を考えうる．第一の同値関係「＝」は，同一の入力に対して，有限個の出力がある場合（0個も含む），同一の出力を同じ順序で生成すること，で定める．同様に，第二の同値関係「～」は，同一の出力を異なる順や重複出力を許して，生成すること，で定める．

それらに関する等価な式の組を示す．これにより次の式が得られる．ここで A, B, C は式であり，メンバはリストの要素を出力する述語，アペンドは二つのリストをつないだリストを出力する述語である．自明なものを除いてある．

A＝B ならば A～B
各Aのメンバ～メンバのA
各（AのB）～ 各Aの各B
各（AやB）～（各Aと各B）のアペンド
AとBのアペンド ～
　全（(Aのメンバ) や（Bのメンバ））
全Aのメンバ ＝ A
全（AのB）＝ 全Aの（各全Bの総アペンド）
全（AやB）＝（全Aと全B）のアペンド
（AやB）のC ＝（AのC）や（BのC）
（AでB）のC ～（AのC）で（BのC）
（AやB）でC ～（AでC）や（BでC）

6.4 言語の機能の性質

LONG の機能が，以下に示すような変数に制限を加えた純 Prolog の機能を含むことを述べる．すなわち，

純 Prolog において任意の項を計算するときに任意の引数について，

（1） 含まれている変数がすべて完全にインスタンシエートされている
（2） 一つの自由変数であり計算が成功した後に完全にインスタンシエートされる

のどちらかに静的に定まっているとする．この条件を満たす純 Prolog プログラムは LONG プログラムに変換できる．

この性質は次のように示される．純 Prolog の各節に対応させて LONG の節を作る．簡単のために，ある項のタイプ(1)の変数は，節の中の直前の項のタイプ(2)のなかにあるとする（こうでない場合は変数を付加すればよい）．ここで「:-」の位置には「とは」を，「，」の位置には「の」を置く．さらに条件部の各項（述語）の入力はタイプ(1)の引数のリストとし，出力は，新たにインスタンシエートした変数のリストとする．この際，入力の中の変数に対応して，直前の項の出力の最初，二番目，三番目…の変数を，「最初」，「残の最初」，「残の残の最初」…に置き換えてリストを作る．そしてこのリストに「の」を付けて当該項に前置する．また頭部からの出力にある変数が，条件の最後の項のタイプ(2)になければ同様に変数を加える．そして頭部からの出力がリストなら同様のものを節の最後に置く．

例を示す．次のものはリスト（第一引数）から基準値（第二引数）より大きい要素のリスト（第三引数）を作る述語を定義する節の一つである（同時に引数のタイプを示す）．

```
p([X|L], P, [X|DAI]) :- 〉(X, P), p(L, P, DAI).
   (1)    (1) (2)          (1) (1)  (1) (1) (2)
```

これに次の項の計算のために必要な変数を付加し，入出力ごとにまとめると

```
p([[X|L], P|A], [[X|DAI]|A]) :-
   〉[X, P, L, A], [X, L, P, A]),
   p([L, P, X, A], [DAI, X, A]).
```

となる．条件部の第一引数および頭部の第二引数の形をリストとして外に出すと，目的の

```
pとは
   [最初の最初，残の最初，最初の残，残の残]の〉の
   [残の最初，残の残の最初，最初，残の残の残の
    最初]
   のpの[[残の最初|最初]|残の残の最初].
```

がえられる．

なお，一般の Prolog のプログラムを LONG に直接に翻訳することはできない．Prolog の計算が両方向性をもつからである．もちろん通常の言語と同様，LONG でも Prolog プログラムとおなじことを行うプログラムは書くことは可能である．

6.5 試作系の効率

次に試作系による効率評価について述べる．LONG の処理系のインタープリタが Prolog の上に作られている．これを用いて実行効率を調べた．なるべく近い条件とするため，LONG と，Prolog の上に書かれた純 Prolog とを比較した．純 Prolog は単一化等で同一レベルまで Prolog で記述した．ベンチマークプ

表 1 ベンチマークプログラムの効率比較
Table 1 Performance comparison of benchmark programs.

処理系	cpu 時間 単位 msec
LONG のプログラム	29070
Prolog 記述の Prolog プログラム	44350
直接実行の Prolog プログラム	1580
ダミーループ	720

ログラムとして，おのおのの定義により，要素数 10 の member 述語の要素を順に取り出すことを，1万回行った．その結果この作業1回分の cpu 時間は，LONG によると 2.84 msec，純 Prolog によると 4.36 msec であった（**表1**，環境は sun 4/2 上および quintus-prolog コンパイラによった）．

実際の Prolog 処理系と同様に，比較用の Prolog 上の純 Prolog インタープリタでも，単一化の部分がある．そこでは引数が変数か定数か複合項かを判定し，また両方向の計算が可能なメカニズムを持っている．LONG 処理系の方は，一方向の計算だけであり，単一化の手間の分だけ処理が軽くなっている．このことが LONG 処理系の速度が速かった主な原因と考えられる．

この比較は Prolog の上に作成したインタープリタのものであり必ずしも厳密なものではないが，これにより LONG は効率の上でも評価できることが予想される．

7. まとめ

一般の利用者にとって使い勝手のよい論理型プログラム言語の言語仕様を検討し，通常の自然言語にみられる，引数（および変数）の非明示性が重要な意味をもつことを指摘した．そして，このことを背景としたひとつの計算モデルを提起した．また日本語プログラム言語の今日的意義を指摘し，そのモデルに基づいた日本語プログラム言語を設計した．この設計は非決定的ないし論理的表現を基本とし，その骨子は，(1)述語は入力と出力の二ポートを持ち，(2)メタ述語でポート間を接続すること自身がプログラミングであり，表現上は(3)述語として名詞的な言葉を用い，(4)メタ述語としては日本語の助詞等を用いることである．

ここにおいて，論理的な意味に対して，日本語表現を対応させることを行った．自然言語表現は一定の意味を内包するとともに，暗黙の，しかし厳密な用法が存在する．本研究では，そうした用法を基に，個々の論理的な意味に対応する適切な日本語表現を決定した．たとえば，メタ述語名の選定にあたって日本語の微妙な意味を反映させることに努めた．このことはプログラム直感的理解のために非常に重要である．

この言語について，表現能力を検討した．その結果，日本語として読みうるプログラムを作成できることを示せた．数学的性質としては，式の間の代数的関係を示した．また純 Prolog との関係を議論した．Prolog を用いて試作システムを作成し，効率を測定した．その限りでは十分な速度を得た．

今後の議論としては次のものが考えられる．第一に，日本語の文構造を反映した別の表現形式を検討する問題がある．本稿では，主として名詞と名詞の間の関係から名詞句を作り構造を組み立てている．拡張として，格助詞による用言の格支配構造は syntax sugar として組み込むことを検討している．第二には実際的システムの作成がある．LONG の述語は入出力が固定しているため，モード宣言が不要であり，また入力データによる節のインデクシングも容易である．こうした利点を用いたシステムを実現する研究は今後重要であると思われる．さらにヒューマンインタフェースの面からの分析も意義があると思われる．たとえば LONG では，引数がなくなったために視覚化が容易になると考えられる．また LONG を意味ネットワーク表現系としてみれば，知識自体がそのまま日本語で読み下せるという利点が活用できると考えられる．

謝辞 本研究は，文部省科研費（研究課題番号 62580018）の援助を受けた．また本研究のアイデアは情報処理学会の研究賞を受けた[5]．

参考文献

1) 小谷善行：変数記述を減らした Prolog の仕様，第 29 回情報処理学会全国大会論文集，pp. 519-520 (1984).
2) 小谷善行：LOGO の「ヒューマン・フレンドリ」性とその日本語 Prolog への応用，ヒューマンフレンドリーなシステムシンポジウム報告集，pp. 207-212，情報処理学会 (1986).
3) Backus, J.: Can Programming Be Liberated from the von Neuman Style? —A Functional Style and Algebra of Programs, *CACM*, Vol. 21, No. 8, pp. 613-641 (1978).
4) Backus, J.: The Algebra of Functional Programs: Functional Level Reasoning, Linear Equations, and Extended Definitions, *Lecture Notes in Computer Science*, No. 107, pp. 1-43, Springer-Verlag (1981).

5) 小谷善行：引数のない論理プログラム表現，情報処理学会第53回記号処理研究会研究報告，pp. 1-8 (1989).
6) 新田健二，矢野稔裕，瀧口伸雄，小谷善行，西村恕彦：自然言語指向の論理型言語処理系の実現方式，第44回情報処理学会全国大会論文集，Vol. 5, pp. 17-18 (1992).
7) 戸村 哲：TD Prolog：項記述可能な Prolog 処理系，*Proc. of the Logic Programming Conference '85*, pp. 237-245 (1985).
8) 戸村 哲：項記述単一化に基づく Prolog，コンピュータソフトウェア，Vol. 8, No. 6, pp. 53-65 (1991).
9) Tanaka, Y.: Vocabulary-Based Logic Programming, *Proc. of Info Japan '90* (1990).
10) Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Language Analysis —A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).
11) Dahl, V. and Saint-Dizier, P. *Natural Language Understanding and Logic Programming*, Elsevier (1985).
12) 二村良彦：プログラム技法—PAD による構造化プログラミング—，オーム社 (1984).
13) 高橋延匡：日本語プログラミング環境，情報処理，Vol. 30, No. 4, pp. 363-372 (1989).
14) 並木美太郎：OS/omicron 用システム記述言語C処理系 Cat のソフトウェア工学的見地からの方式設計，電子情報通信学会誌，D, Vol. J71-D, No. 4, pp. 652-659 (1988).
15) ロゴライターリファレンスマニュアル，ロゴジャパン (1988).

付録1 構文形式

〈定義節〉::=〈述語〉とは〈式〉.|〈式〉の〈述語〉は〈式〉.
〈ゴール節〉::=〈式〉.
〈式〉::=〈述語〉|〈リスト〉|
　〈式〉〈2項メタ述語〉〈式〉|
　〈前置単項メタ述語〉〈式〉|
　〈式〉〈後置単項メタ述語〉
〈式〉::=〈式3〉|
　　〈式〉〈2項メタ述語〉〈式3〉
〈式3〉::=〈式2〉{〈後置単項メタ述語〉}
〈式2〉::=〈式1〉|
　　〈前置単項メタ述語〉〈式2〉
〈式1〉::=〈述語〉|〈リスト〉|(〈式〉)
〈述語〉::=〈名前〉|〈定数述語〉
〈定数述語〉::=「〈式〉」
〈リスト〉::=[〈式列〉|〈式〉]|[]
〈式列〉::=〈式〉{,〈式〉}...
〈2項メタ述語〉::=の|で|や|なら|ほかは|と|とは|は
〈前置単項メタ述語〉::=各|総|全|逆
〈後置単項メタ述語〉::=である
〈名前〉::=〈漢字列〉|〈片仮名列〉|
　〈英字列〉|〈数字列〉|"〈文字列〉"

付録2 プログラム例

1. member 述語
　member とは最初.
　member とは残の member.
2. append
　アペンドとは最初が[]であるなら残.
　アペンドとは
[最初の最初|[最初の残|残]のアペンド].
(注：append([[]|X], X),
　append([[A|X]|Y], [A|Z]) :-
　　append([X|Y], Z).
　という定義に対応する)
3. 血液型計算プログラム
(1)本 体
「A」の遺伝子対は「[a, a]」や「[a, o]」.
「B」の遺伝子対は「[b, b]」や「[b, o]」.
「AB」の遺伝子対は「[a, b]」.
「O」の遺伝子対は「[o, o]」.
「花太」の父は「太郎」.
「花太」の母は「花子」.
「太郎」の血液型は「O」.
「花子」の血液型は「AB」.
血液型とは [父，母] の各（血液型の遺伝子対のメンバ）の表出型.
表出型とはソートの逆遺伝子対.
(2)プログラムの起動
「花太」の血液型.
(3)プログラムのトレース
「花太」の父は「太郎」.
「花太」の母は「花子」.
「花太」の [父，母] は「[太郎，花子]」.
「太郎」の血液型は「O」.
「O」の遺伝子対は「[o, o]」.
　(途中略)
「A」の遺伝子対は「[a, o]」.
「[a, o]」の逆遺伝子対は「A」.
「[o, a]」のソートの逆遺伝子対は「A」.

「[o, a]」の表出型は「A」.
「[太郎, 花子]」の各（血液型の遺伝子対のメンバ）の表出型は「A」.
「花太」の[父, 母]の各（血液型の遺伝子対のメンバ）表出型は「A」.
「花太」の血液型は「A」.　（以下略）

（平成4年8月25日受付）
（平成5年1月18日採録）

小谷　善行（正会員）

昭和24年生．昭和46年東京大学工学部計数工学科卒業．昭和52年同大学院博士課程工学系研究科修了．同年東京農工大学工学部数理情報工学科講師．現在同大学電子情報工学科（コンピュータサイエンス）助教授．記号処理言語を含むソフトウェア工学および知識処理に興味を持つ．人工知能学会，日本ソフトウェア科学会，電子情報通信学会，認知科学会各会員．