** オブジェクト指向コンピューティングに関するワークショップWOOC(昭和60年3月18日~19日) **

対象指向型言語と論理型言語の融合方式に関する考察

中所武司、芳賀博英(日立製作所システム開発研究所)

[内容梗概]

対象指向型言語をベースにして、メソッドを論理型言語で記述する知識情報処理システム記述言語Super-LONLI[1]を開発中であるが、その設計思想は次の通りである。

(1) 人間世界のマクロな計算モデルの自然な記述には対象指向型言語(OOP)が適する。

(2) 一方、ミクロな計算規則と事実の正確な記述には論理型言語(LP)が適する。

本論文ではOOPとLPの融合上の問題点と解決方法について論じる。

まず、LP側から見たOOP機能の問題点として、

①オブジェクト変数による副作用

②オブジェクトの階層及び継承規則とLPの閉世界仮説との矛盾、

③プログラム(インスタンス)の動的生成機能、

④オブジェクトの並行処理性とPrologの遂次処理性の矛盾、

などがある。

また、OOP側から見たLP機能の問題点として、

⑤バックトラック機能とメッセージ伝達による問題解決法の矛盾、

⑥上記④と同じ、

などがある。

そこで、OOPとLPの各々の特徴を両立させながら、かつ、両者の境界を明確にするために、次のような融合方式を提案する。

(1) OOPとの融合方式としては、LP with OOP[2~4], LP and OOP[5,6], OOP with LPの3種類のうち、第3の方式を採用する。

(2) LP拡張機能として、OOP用述語を導入する。

(3) 継承規則の解釈としては、スネカジリルールとコピールールが考えられるが、後者を採用する。

(4) メッセージ送信用述語として、並行処理用、遂次処理用、遅延評価用の3種類を設ける。

1. はじめに

最近注目されている対象指向型言語[7](以下OOPと略す)は、次のような特徴を有し、人間世界の計算モデルのより自然な表現に適している。

(1) データとその操作手続きをカプセル化したオブジェクトを基本要素とする。

(2) オブジェクトは階層構造を構成でき、上位オブジェクトから下位オブジェクトへの性質の継承ができる。

(3) 静的に定義されたオブジェクトは、それを型(クラスオブジェクト)として動的に複数個(インスタンスオブジェクト)生成できる。

(4) オブジェクト間の実行制御はメッセージ伝送によって行なう。

このような特徴は、計算モデルの全体的記述には適するが、第(1)項の操作手続きをどのように記述するかという問題がある。OOPの代表的言語であるSmalltalk-80[8]では、その部分の記述に対してもOOPの思想を徹底しているため、プログラムの理解容易性の面で疑問が残る。一方、第(1)項と同じ思想であるデータ抽象化機能を有するSimula67,CLU,Mesa,SPL,Adaなどの言語では、従来通りの手続き型の記述形式をとっているため、人間世界との記述レベルのギャップが大きく、プログラム作成者への負担が大きい[9]。このような点から、人間世界に近い表現ができ、かつ、詳細な計算規則(アルゴリズム)の記述が可能な言語としては論理型言語(以下LPと略す)が適していると思われる。

そこで、我々は次のような理由からOOPとLPを融合した知識処理言語S-LONLIを開発中である。(図1参照)

(1) 人間世界のマクロな計算モデル(知識)の自然な表現には、OOPが適する。

(2) 一方、ミクロな計算規則と事実の正確な表現には、LPが適する。

しかしながら、異なるパラダイム(プログラミング方法論)に基づく複数の言語の融合には種々の問題がある。本論文では、それらの技術課題について検討する。なお、LPとしてProlog[10]相当のものを考える。

2．対象指向型言語と論理型言語の融合上の問題

2．1　論理型言語からみた問題点

前章で述べたＯＯＰの４つの特徴は、ＬＰの観点からは、各々、次のような問題がある。

まず第１に、プログラムの基本要素となるオブジェクトは変数を有する。これは、使い方にも依るが、一般にグローバル変数の役割を果すため、時間と共に変化する状態を有することになる。一方、ＬＰはこのようなグローバル変数の存在を認めていないため、この存在によって生じるプログラム実行時の副作用をどう解釈するかという問題がある。

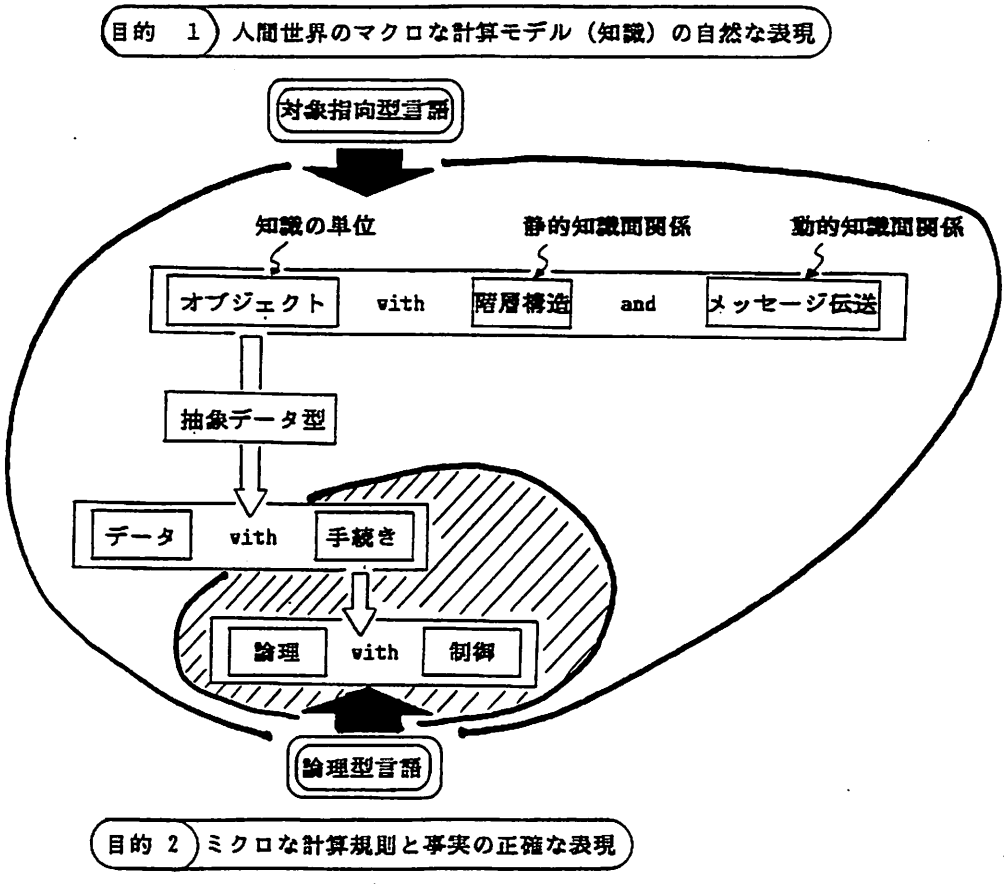
第２には、オブジェクトの階層化と上下間での性質の継承機能のために、各オブジェクトの機能は自分自身で閉じたものとはなっていない。一方、ＬＰでは、自分が知っておくべき知識はすべてそのプログラム（知識ベース）の中に含まれるという閉世界仮説（ the closed world assumption)が前提になっているため、この継承規則をどのように解釈するかという問題がある。

第３には、静的に定義されたオブジェクトは、それを型としてそのインスタンスを動的に生成可能である。このような、プログラムを動的に生成する機能はＬＰにはない。ただ、実用上、Prologでは、知識ベースに節と呼ばれるプログラム要素を加えることができるが、インスタンスの動的生成とは異なる。本機能については、上記の２つの問題との絡みでその解釈を明確にする必要がある。

第４には、オブジェクト間の実行制御はメッセージ伝送によって行なわれる。そこで、個々のオブジェクトは並列実行できることが前提になっている[11]。一方、本論文では、ＬＰとしては実用化の可能性の高い逐次型Prologを前提にしており、その視点からのメッセージ送信の解釈を明確にする必要がある。

2．2　対象指向型言語から見た問題点

次にＯＯＰの観点からのＬＰとの融合上の問題点について検討する。ＬＰとして逐次型Prologを前提にした場合、その主な特徴はユニフィケーションとバックトラッキングである。このうち、前者については、ＯＯＰにおけるメソッド選択の一手法としてとらえることができるが、後者の概念はＯＯＰにはない。即ち、並列実行されるオブジェクト間でメッセージ伝送しながら全体の目的を達成するというＯＯＰの枠組みの中で、目的達成に失敗すれば、以前の状態に戻って他の選択枝を再実行できるというバックトラッキング機能をどのように解釈するかが第１の大きな問題である。



図．１　知識処理言語の設計思想

第２には、並列実行を基本とするＯＯＰの世界の中で、Prologの逐次実行性をどのように解釈するかという問題がある。これは、先の第１の問題とも関係が深い。

３．対象指向型プログラミング用述語の設定

３．１　対象指向型言語のモデル設定

前章で述べた課題を検討する前に、まずＯＯＰのモデルを次の手順で設定する。

（１）個々のオブジェクトの機能仕様はメソッドの集合とする。各メソッドはそれを指定したメッセージの受信によって起動され、所定の機能を果す。その際、必要ならば入出力パラメータを有する。

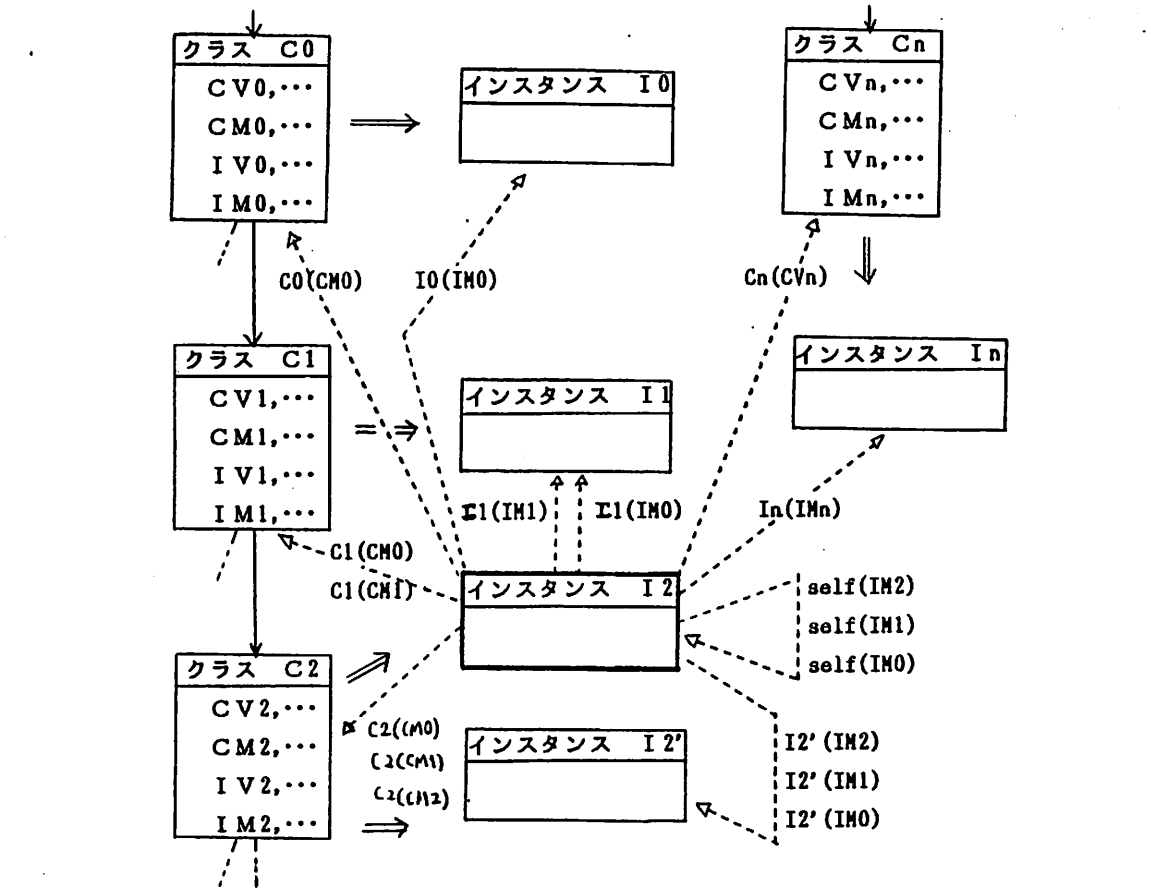
（２）個々のオブジェクトは必要ならば変数表現されたデータを有する。このデータには、そのオブジェクトの機能を象徴するようなもの、即ち人間世界のモデル化により必然的に導入されるものと、そのオブジェクトの機能の実現方式に依存して導入されるものがあるがここでは区別はしない。

（３）データ抽象化の観点から、これらのデータへのアクセスはそれらと同じオブジェクト内のメソッドからのみ可能とする。

（４）メソッドが同じで、データの内容が異なるオブジェクトを効率良く作成するために、メソッドが同じオブジェクトを型としてそのインスタンスを動的に複数個生成可能とする。そして、型となるものをクラス（オブジェクト）、生成される方をインスタンス（オブジェクト）と呼ぶ。

（５）この時、データには個々のインスタンスに固有のものとクラスに固有のものがあり、各々をインスタンス変数およびクラス変数に分離する。

（６）さらに上記(3)の方針に合せて、インスタンス変数の操作用メソッドとクラス変数操作用メソッドを分離し、各々をインスタンスメソッドおよびクラスメソッドと呼ぶ。



（記号の説明）

Ck ⟹ Ik	：	クラスCkからインスタンスIkを生成。	CVk ： クラス変数
Ck ⟶ Cm	：	クラスCkはクラスCmの上位クラス。	CMk ： クラスメソッド
A(m) ⇢	：	オブジェクトAにメソッドmの実行を要求するメッセージ送信。	IVk ： インスタンス変数
			IMk ： インスタンスメソッド

（図中は、インスタンスI2からのメッセージ送信例を示す）

図．２　対象指向型言語モデルの例

（7）次に、メソッドの集合が少しずつ異なるオブジェクトを効率良く作成するために、クラスに階層構造を導入する。そして、下位のクラスは上位のクラスの性質すなわちメソッドを継承できるものとする。

以上のモデル形成過程の中で、１章で述べたＯＯＰの４つの基本要素はすべて導入されている。図２にこのモデルの概念図を示す。図中にはインスタンスＩ２から送信可能なメッセージの例を示している。実際のメッセージ送信の記述はクラスＣ２内のインスタンスメソッド内にある。

３．２　対象指向型プログラミング用述語

本論文で提案する知識処理言語は、前章で述べたＯＯＰのモデルをベースにして、メソッドの部分をProlog風のＬＰで記述するものである。そのためには、ＬＰの側に次のようなＯＯＰ用述語を拡張機能として導入する必要がある。なお、述語名は、現在開発中の知識処理言語Ｓ－ＬＯＮＬＩに合せた。

（１）gen(a)：このメッセージを受信したクラスは、そのインスタンスを生成し、aと名付ける。

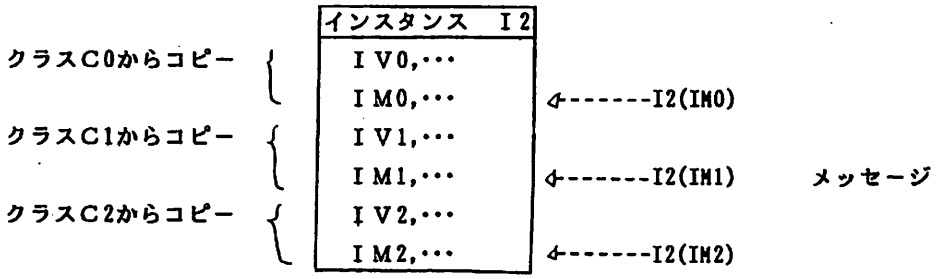
（２）change(x,v)：変数 x の値を v とする。

（３）send(a,m)：aという名前のオブジェクトにメッセージ m を送信する。

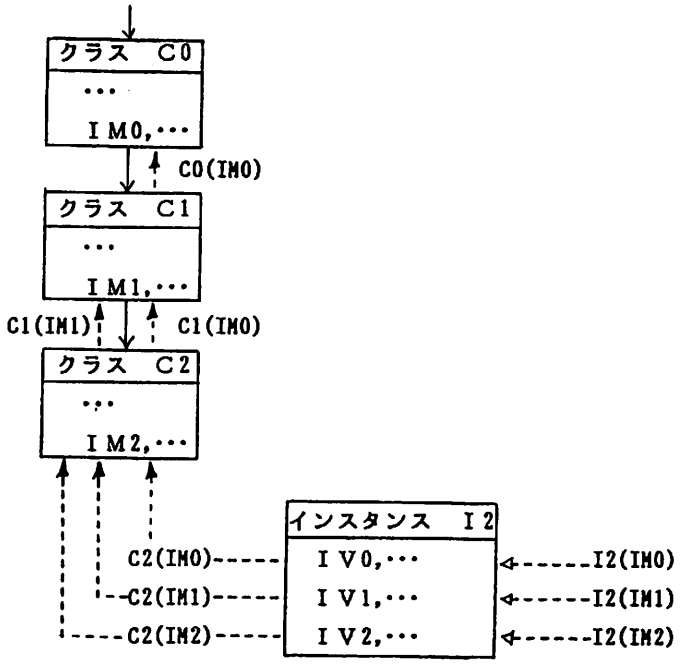
この他にも変数の値を参照する述語などが必要であるが、ここでは、今後の議論の対象となる副作用を伴うもの、あるいはその可能性のあるものに限定した。

４．オブジェクトの階層構造と継承規則の解釈

次に、ＬＰでは、自分が知っておくべき知識はすべて知識ベース中に存在するという閉世界仮説を前提にしているため、この観点から、クラスの階層構造における継承規則をどのように解釈するかという問題がある。この継承規則の解釈には、次の２通りがあり、各々コピールールとスネカジリルールと呼ぶ。



（１）　コピールールの例



（２）　スネカジリルールの例

図．３　オブジェクトの階層構造における継承規則の解釈方法

（1）コピールール：

インスタンスの場合、その元のクラスおよびその上位クラスのインスタンス変数とインスタンスメソッドがすべてそのインスタンス内にコピーされたと考える。図3(1)は図2のインスタンスＩ２の例である。なお、クラスの場合は、その上位クラスのクラス変数とクラスメソッドがすべてそのクラス内にコピーされたと考える。

（2）スネカジリルール：

メッセージを受信したクラスあるいはメッセージを受信したインスタンスの元のクラス内に、メッセージで指定したメソッドの定義がなければ、自分の親クラスにそのメッセージの処理を依頼する。図3(2)は図2のインスタンスＩ２が受信したメッセージの処理であり、メソッドＩＭ０の処理はインスタンスＩ２からクラスＣ２に依頼され、更にクラスＣ１を経由してクラスＣ０で実際の処理が行われている。なお、この場合もインスタンス変数についてはコピールールと同様に各インスタンスにコピーされる。

以上の２方式はＯＯＰのモデルの世界では同じ意味と考えられる。しかしながら、ＬＰの場合は、閉世界仮説の観点から、(1)の解釈が妥当である。このことから、あるメソッド内で他のメソッドを呼び出す方法を次のように規定できる。

（1）あるオブジェクトのメソッドから、コピールールに従ってコピーされるべきメソッドを引用する場合は通常の述語と同じように直接引用して良い。

（2）上記以外のメソッドを引用する場合は、send述語を用いて、そのメソッドのあるオブジェクトへメッセージを送る。

５．バックトラッキング機能の制約

５．１　副作用とバックトラッキング

ＯＯＰ用述語の実行に伴う副作用は、ＬＰの特徴的な機能であるバックトラッキング機能とは矛盾する。即ち、バックトラッキング時には、genの場合は生成したインスタンスを消滅したり、changeの場合は変数の値を前の値に戻したり、sendの場合はメッセージの受信側オブジェクトの実行内容をすべて無効にする必要がある。特にＯＯＰでは複数オブジェクトの並列実行を前提にしているため、生成されたインスタンスにメッセージを送ったオブジェクトや変数の新しい値を既に参照したオブジェクトなどもバックトラッキングの対象となり、いわゆる分散バックトラッキング(distributed backtracking) [12]が生じる。このような複雑な動きを許すことはプログラミング言語としては実用的でないため、何らかの制約が必要である。

５．２　ＯＯＰ用述語の成功／失敗の解釈

ＯＯＰ用述語の実行がＬＰの観点で失敗(fail)する場合として、

（1）単純な文法規則エラーの他に、

（2）changeで指定された変数が存在しないか値が不適当、

（3）genで指定された名前のインスタンスが既に存在する、

（4）sendで指定したクラスが存在しないとか、指定メソッドが無い、

（5）sendで指定したメソッドの実行が失敗した、

などが考えられる。

このようなＯＯＰ用述語の成功／失敗の解釈は次の２通りが考えられる。

［A］ＬＰの観点から、他のＬＰの述語と同様に成功／失敗の対象として扱う。

［B］ＯＯＰの観点から、常に成功とみなす。

このうち、方式Ａでは、実用上プログラミングの逐次実行を前提にする必要があると思われるが、即座に実行チェックできる上記(1)~(4)は別として、(5)の場合、メッセージの送信側がその実行終了まで常に待つのではＯＯＰの特徴である並列処理性が失われる。一方、方式Ｂでは、副作用が野放しになる危険性があり、メソッド定義をＬＰで行う利点が失われる。

そこで、我々は基本的方針を次のように設定した。

（a）genとchangeは即時実行チェックが可能であり、ＯＯＰの観点との矛盾もないので、方式Ａを採用する。従って、失敗の時はバックトラッキングを生じる。

（b）sendの失敗条件のうち、上記第(4)項は(a)と同様の理由で方式Ａとする。一方、第(5)項はＯＯＰの並行処理性を優先させ、方式Ｂとする。

しかしながら、実際のプログラム開発に際しては、sendで指定したメソッドの実行結果の成功／失敗によって後の処理をかえるような記述が必要な場合がある。これは、より一般的には、メソッドの実行結果を受け取って次の処理に進む機能が必要であるということである。そのためにはsendの他に次のようなsendw (send and wait)述語を設ければ良い。

（1）send：本述語の実行後、送信側はその結果を待たずに次の実行へ移る。

（2）sendw：本述語の実行後、送信側はその結果が戻されるまで実行を中断して待つ。

その使用例を以下に示す。

```
appoint( DATE ) :-
    sendw( secretary,schedule( DATE,ITEM ) ),
    (ITEM==vacant,answer( ok ) ; answer( no )).
```

この例では、オブジェクトsecretaryにDATEで指定した日時の予定を尋ねるメッセージを送った後、結果を待つ。そして、結果としてITEMの値がvacantならばok，それ以外はnoの返事を返す。

５．３　バックトラッキングの制約方法

次に、gen,change,sendが実行に成功し、副作用が生じた後、それに続く処理でバックトラッキングが起きた場合のこれらの副作用の扱いについて検討する。本来、ＬＰの観点では、副作用はすべて元の状態に戻してバックトラックするのが良い。しかし、この方式では、分散バックトラッキングが生じてＯＯＰの世界が複雑になることと実行処理系の負担が大きいことから、副作用を無視してバックトラックする方式にせざるを得ない。

この場合、副作用はすべてユーザ責任となるため、ユーザ側でその制御ができることが望ましい。genについてはインスタンスを消滅させる組み込み述語killを用いて、またchangeについてはもう一度changeを用いて元の値に戻すことによって一次的副作用を取消すことができる。

一方、sendについては、その制御をユーザが比較的容易に行えるようにするために、send の遅延評価（lazy-evaluation）機能として、次のような述語sendq（send via queue）を設ける。

sendq：メッセージの受信に基づいて起動されたメソッドの実行中にsendqを用いたメッセージ送信が生じた場合、その順にすべてキューに保存する。途中バックトラッキングが生じた時は関連する送信メッセージをキューから除く。そして、最後にメソッド全体が成功した時点でキュー内のメッセージを送信する。

この述語の導入により、ＬＰに特徴的なバックトラッキング機能とＯＯＰに特徴的なメッセージ送信機能を矛盾なく融合でき、かつ、ＬＰとＯＯＰの境界を明確にできる。

なお、本機能をより一般的に解釈すれば、Prologに次のような制御機能を導入することを意味する。

queue：素命題 x がqueue(x)の形で用いられた時、x はその現れた順にキューに保存され、バックトラキングが生じるとキューから除かれる。途中、後で述べるdequeueが実行されるとキュー内の x はその順に実行される。もし、dequeueが無ければ、全体のゴールの成功時に実行される。

dequeue：その時点でキューに保存されている素命題を順に実行する。

従って、sendに限らず、Prologの入出力述語なども例えばqueue (write (…)) のように使用することにより、バックトラッキングによる副作用の混乱を避けることができる。

６．　おわりに

以上の結果からもわかるように、ＯＯＰとＬＰの各々の特徴を両立させながら、かつ、両者の境界が明確になるような融合方式を達成できたと考えられる。特に今回はマクロな記述に用いるＯＯＰをベースにして検討した結果、ＬＰ側に部分的拡張機能が生じているが、ＯＯＰ用述語を使用しなければ従来通りのＬＰと見てよい。その場合でも、階層構造やメソッド定義の枠はＬＰの大きな欠点であるモジュール化機能の欠点を補うものとして有効である。また変数を値の変更されない定数として用いることもできる。

謝辞

本研究の機会を与えて頂いた日立製作所システム開発研究所所長川崎淳博士ならびに日頃御指導頂く青山義彦第２部部長、渡邊坦主幹研究員に感謝いたします。またＳ－ＬＯＮＬＩの開発メンバの１人として有意義な御討論を頂いた大藤淑子嬢に感謝致します。

参考文献

［１］芳賀、中所：知識情報処理システム記述言語Ｓ－ＬＯＮＬＩの提案：日本ソフトエア科学会第１回大会論文集、2D-2,167-170（昭59-12）

［２］Chikayama,T.：ESP Reference Manual:ICOT Technical Report TR-044(Feb.1984)

［３］溝口、他２名：オブジェクト指向概念を導入した知識表現言語：姿と鏡の設計とその応用：Proc.Logic Programming Conf.'84,2.1（昭58-11）

［４］古川、他２名：MANDALA：Concurrent Prolog上の知識表現システム：情報処理学会　知識工学と人工知能研究会資料32,32-1（昭58-11）

［５］Takeuchi,I.,et al.：TAO---A Harmonic Mean of Lisp,Prolog and Smalltalk：ACM SIGPLAN Notices,18,7,65-74(July 1983)

［６］石川、他３名：オブジェクト指向型知識表現言語　Orient84/K：日本ソフトウエア科学会第１回大会論文集、1D-3,57-60（昭59-12）

［７］米澤：オブジェクト指向型プログラミングについて：コンピュータソフトウエア、1,1,29-41（昭59-4）

［８］Goldberg,A. and Robson,D.：Smalltalk-80 The Language and its Implementation,Addisin-Wesley (1983)

［９］中所：プログラミング言語とその会話支援環境：情報処理，24,6,715-721（昭58-6）

［１０］Bowen,D.:DEC system-10 Prolog User's Manual:Dept. of Artificial Intelligence,Univ. of Edinburgh(Dec.1981)

［１１］Hewitt,C. and Baker,H.:Law for Communicating Parallel Processes:Proc.IFIP'77,987-992(1977)

［１２］竹内：論理型並列プログラミング言語 Concurrent Prolog：コンピュータソフトウエア，1,2,25-37（昭59-7）