1985,12
共立出版
知識と論理指向…

3. オブジェクト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）

3.0 内容梗概

オブジェクト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 S-LONLI[1] を開発中であるが，その設計思想は次のとおりである．

（1） 人間世界のマクロな計算モデルの自然な記述にはオブジェクト指向型言語 (OP) 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 OP 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 OP 機能の問題点として，

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において，そのデータの値に伴う副作用

②オブジェクトの階層および継承規則と 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 の逐次処理性の矛盾

などがある．

また，OP 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，OP と LP のそれぞれ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） OP との融合方式としては，LP with OP[2)-4)]，LP and OP[5),6)]，OP with LP の三種類のうち，第3の方式を採用する．

（2） LP 拡張機能として，OP 用述語を導入する．

（3） 継承規則によって上位オブジェクトから受け継ぐメソッドはすべて自分のオブジェクト内にコピーして使用するという解釈により，LP の閉世界仮説と一致させる．

（4） メッセージ送信用述語として，並行処理用，逐次処理用，遅延評価用の三種類を設ける．

3.1 はじめに

最近注目されているオブジェクト指向型言語[7)]（以下 OP 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，（1）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．OP 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 OP 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，（1）と同じ思想であるデータ抽象化機能を有する Simula 67, CLU, Mesa, SPL[16)], Ada などの言語では，従来どお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が適していると思われる[14)]．

そこで，われわれは次のような理由から OP と LP を融合した知識処理言語 S-LONLI を開発中である（図 3.1 参照）．

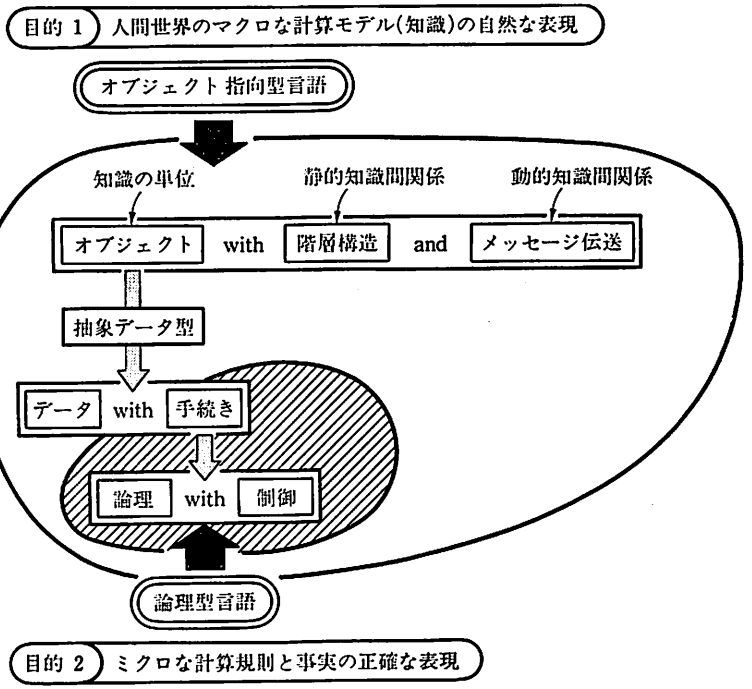


図 3.1 知識処理言語の設計思想

(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, OP 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. そこで, 本論文と同様に OP と LP を融合した言語について見てみる. まず, LP をベースに OP 機能を組み込んだ言語 (LP with OP 方式) として, Prolog ベースの ESP[2], 鏡[3], Intermission[18], および Concurrent Prolog ベースの Mandala[4] などがある. また, ベース言語としては Lisp を用い, LP と OP を対等に融合した言語 (LP and OP 方式) として, 通研の TAO[5] がある. このように, Prolog や Lisp などの既存言語をベースに OP 機能を組み込む方式では, ベース言語の影響が強く, OP の全体モデルが不明確になっていると思われる. すなわち, 異なるパラダイムに基づく

複数の言語を融合して完全な単一言語にする方式には無理があると思われる.

そこで, われわれは, 前に述べたような設計思想に基づき, OP をベースに LP 機能を組み込んだ言語 (OP with LP 方式) を開発するに当たり, 単一言語化ではなく, むしろ両者の境界を明確にするような融合方式を採った. 本論文では, そ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.

3.2 オブジェクト指向型言語と論理型言語の融合上の問題

3.2.1 論理型言語からみた問題点

前節で述べた OP の4つの特徴は, LP の観点からはそれぞれ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, 使い方にもよるが, 一般にオウン変数の役割を果たすため, 時間とともに変化する状態を有することになる. 一方, LP はこのようなオウン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, 知識はすべてそのプログラム (知識ベース) の中に含まれており, その知識ベースに含まれないものは否定的事実として扱うという閉世界仮説 (the closed world assumption)[15] 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, オブジェクト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.

3.2.2 オブジェクト指向型言語から見た問題点

次に OP 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，OP 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 OP にはない．すなわち，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 OP 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．

第2には，並列実行を基本とする OP 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．

3.3 オブジェクト指向型プログラミング用述語の設定

3.3.1 オブジェクト指向型言語のモデル設定

前節で述べた課題を検討する前に，まず OP 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，所定の機能を果たす．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，すなわ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．

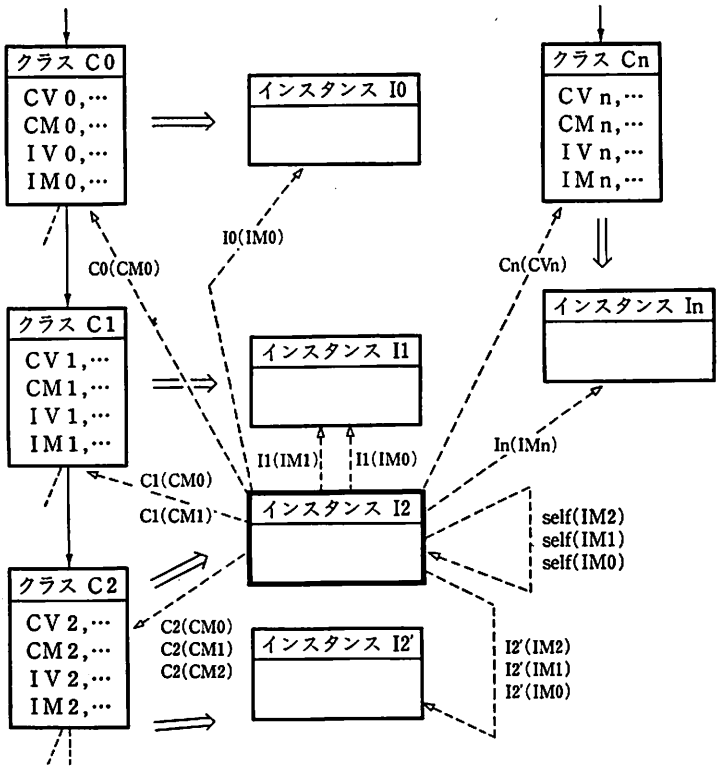
（3） データ抽象化の観点から，これらのデータへのアクセスはそれらと同じオブジェクト内のメソッドからのみ可能とする．

（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よく作成するために，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），生成されるほうをインスタンス（オブジェクト）と呼ぶ．

（5） このとき，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，それぞれをインスタンス変数およびクラス変数に分離する．

（6） さらに前記（3）の方針に合わせて，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，それぞれをインスタンス・メソッドおよびクラス・メソッドと呼ぶ．

（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よく作成するために，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



（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 3.2 オブジェクト指向型言語モデルの例

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，3.1節で述べたOPの4つの基本要素はすべて導入されている．図3.2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．

本論文の以降ではこのモデルに沿って議論するが，本モデルのレベルでのSmalltalk-80との相違点としては，本モデルでは，データ抽象化の考えを徹底し，クラス変数へのアクセスは同一クラス内のクラス・メソッドからのみ，またインスタンス変数へのアクセスは同一クラス内のインスタンス・メソッドからのみ可能としている点である．Smalltalk-80ではインスタンス・メソッドからクラス変数へのアクセスが可能になっており，これは，従来のブロック構造言語のスコープルールに似ているが，Smalltalk-80のクラス-メタクラス関係とインスタンス-クラス関係を同一視する思想には反していると思われる．

3.3.2 オブジェクト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，前項で述べたOPのモデルをベースにして，メソッドの部分をPrologふうの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OP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に合わせた．

（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と名づける．

（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．

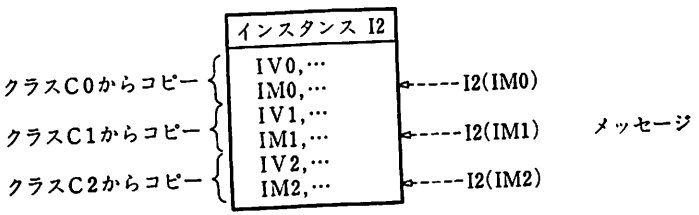
このほかにも変数の値を参照する述語などが必要であるが，ここでは，今後の議論の対象となる副作用を伴うもの，あるいはその可能性のあるものに限定した．

3.4 オブジェクトの階層構造と継承規則の解釈

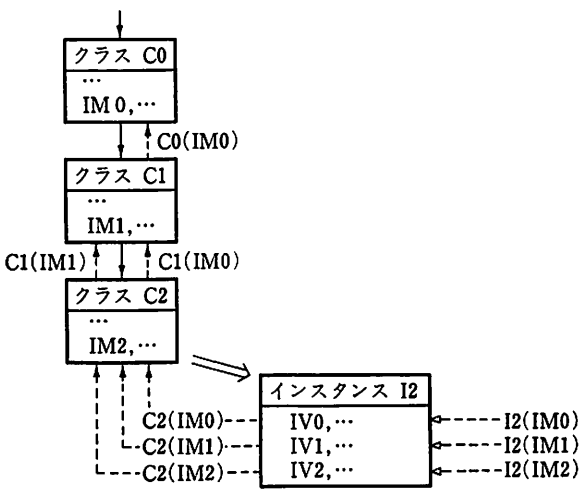
次に，LPでは，知識はすべて知識ベース中に存在するという閉世界仮説を前提にしているため，この観点から，クラスの階層構造における継承規則をどのように解釈するかという問題がある．この継承規則の解釈には，次の2通りがあり，それぞれコピールールとスネカジリルールと呼ぶ．

（1） コピールール

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3.3（1）は図3.2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．



（1） コピールールの例



（2） スネカジリルールの例

図3.3 オブジェクトの階層構造における継承規則の解釈方法

（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3.3（2）は図3.2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，さらにクラス C1 を経由してクラス C0 で実際の処理が行なわれている．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 OP 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よい．

（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．

たとえば，図3.2の例において，クラス C2 の IM2 の定義の中で IM0 を引用する場合，

im2 :- ……, send(self, im0), ……

の代わりに，

im2 :- ……, im0, ……

と記述してよい．しかし，IMn を引用する場合は，

im2 :- ……, send(in, imn), ……

の代わりに，

im2 :- ……, imn, ……

と書くことはできない．

3.5 バックトラッキング機能の制約

3.5.1 副作用とバックトラッキング

OP 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．すなわち，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，OP 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

い値をすで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ためには時間とともに膨大な量になる実行履歴の管理が必要になり，プログラミング言語としては実用的でないため，なんらかの制約が必要である．

3.5.2 OP 用述語の成功／失敗の解釈

OP 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すでに存在する，

（4） send で指定したオブジェクトが存在しないとか，指定メソッドがない，

（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 OP 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］ OP 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 OP 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 で行なう利点が失われる．

そこで，われわれ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，OP の観点との矛盾もないので，方式 A を採用する．したがって，失敗のとき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（4）は（a）と同様の理由で方式 A とする．一方，（5）は OP 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／失敗によってあとの処理を変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 のほかに次のような sendw (send and wait) 述語を設ければよい．

（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あと，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．

3.5.3 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，副作用が生じたあと，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．たとえば，

```
p(X) :- send(cl, gen(X)), change(last, X),
        send(monitor, register(X)), r.
```

という節の実行において，最後の r の実行が失敗した場合の問題である．

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よい．しかし，この方式では，分散バックトラッキングが生じて OP 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1次的副作用を取り消すことができる．

一方，send については，その制御をユーザが比較的容易に行な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ときは，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 OP 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 OP の境界を明確にできる．上の例は monitor へのメッセージ送信を

sendq(monitor, register(X))

と記述すればよい．

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) の形で用いられたとき，x はその現われた順にキューに保存され，バックトラッキングが生じるとキューから除かれる．途中，あとで述べる 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 がなければ，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．

上の例は

p(X):- send(cl, gen(X)), change(last, X),

queue(send(monitor, register(X))), r, dequeue.

と記述すればよい．したがって，send に限らず，Prolog の入出力述語なども，たとえば 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．

3.6 おわりに

以上の結果からもわかるように，OP と LP のそれぞれ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 OP 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，OP 用述語を使用しなければ従来どおりの 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如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いただ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ご指導いただく青山義彦第2部部長，渡邊坦主幹研究員に感謝いたします．また，S-LONLI の開発メンバーの1人として有意義なご討論をいただいた大藤淑子嬢に感謝いたします．最後に，本論文をご査読くださり，適切なるご助言をいただいた東京大学工学部計数工学科 鈴木則久助教授†に深謝いたします．

参考文献

1) 芳賀，中所：知識情報処理システム記述言語 S-LONL I の提案：日本ソフトウェア科学会第1回大会論文集，2D-2，pp. 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, pp. 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 Orient 84/K：日本ソフトウェア科学会第1回大会論文集，1D-3，pp. 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ェア，1, 1, pp. 29-41 (昭 59-4).
8) Goldberg, A. and Robson, D.: Smalltalk-80 — The Language and its Implementation, Addison-Wesley (1983).
9) 中所：プログラミング言語とその会話型支援環境：情報処理，24, 6, pp. 715-721 (昭 58-6).
10) Bowen, D.: DEC system-10 Prolog User's Manual: Dept. of Artificial Intelligence, Univ. of Edinburgh (Dec. 1981).

† 1985年8月より IBM Thomas J. Watson Research Center に勤務.

11) Hewitt, C. and Baker, H. : Law for Communicating Parallel Processes : Proc. IFIP '77, pp. 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ェア，1，2，pp. 25-37 (昭 59-7).

13) Kahn, K. : Intermission — Actors in Prolog : Logic Programming (Ed. Clark, K. and Tarnlund, S.), Academic Press, pp. 213-228 (1982).

14) Kawalski, R : Logic Programming : IFIP '83, pp. 133-145 (Sep. 1983).

15) Clark. K. : Negation as Failure — Logic and Databases (Ed. Gallaire, H. and Minker, J.), Plenum Press, pp. 293-322 (1978).

16) 中所他 3 名：段階的詳細化，データ抽象化を支援する言語 SPL のコンパイル技法：情報処理学会論文誌，21. 3，pp. 223-229 (昭55-5).