第五世代コンピュータプロジェクトにおける基礎研究の概要

古川 康一

（財）新世代コンピュータ技術開発機構

furukawa@icot.or.jp

概要

第五世代コンピュータプロジェクトは，知識情報処理に適した並列コンピュータの開発を目指して，1982年に開始された．一般に，記号計算に基づく知識情報処理を並列化する事は大変困難であると信じられてきたが，我々は，**「論理プログラミングの技術がそれを可能にする」**と言う仮説の基に，研究を開始した．

我々は，論理プログラミングの，見掛上異なる二つの目標に向かって研究に取り組んだ．その第一は，新しい情報処理技術の確立であり，第二は，人工知能およびソフトウエア工学の基礎の追及である．

前者の目標に対して，我々は，並列論理プログラミング言語GHCとその実際的な並列処理のための拡張がなされた言語KL1を開発した．GHC/KL1の開発により，その言語を使いこなすソフトウエア技術とその言語のための専用並列ハードウエアの二つの研究開発を同時並行的に推進する事が可能となった．我々は，さらに応用システムの開発に適した制約論理プログラミング言語もいくつか開発した．それらの大部分は，逐次Prologの技術に基づいているが，我々は，現在制約論理プログラミングと並行論理プログラミングを統合した言語として，GDCCを開発中である．

後者の目標に対しては，我々は，人工知能およびソフトウエア工学の多くの基礎的な問題を取り上げて，研究を進めた．それらは，仮説推論，類推，知識表現，定理証明，部分評価，プログラム変換などである．

研究開発の結果，我々は，論理プログラミングが，情報処理の多くの側面において強固な基盤を与えている事を示す事が出来た．それらは，人工知能およびソフトウエア工学における新しいソフトウエア技術に始まり，システム・プログラミング，並行プログラミング，さらには，並列アーキテクチャにまで及んでいる．

研究開発は，現在も進められており，これまでの研究結果に加えて，最新の研究結果は，我々の立てた仮説の正しさおよび我々のアプローチの適切さを強く支持している．

1　はじめに

第五世代コンピュータプロジェクトでは，二つの研究目標が掲げられた．それらは，知識情報処理の確立と並列処理の追及である．論理プログラミングは，それら二つの目標を同時の達成するための鍵となる技術として，採用された．プロジェクトの開始当初は，プロジェクトの研究全体を推進する牽引車として，Prologが選ばれた．我々のプロジェクト以前は，Prologに基づいた組織的な研究がなされていなかったので，なすべき事は沢山あった．それらは，研究推進のためのワークステーションの開発，Prolog上に知識ベースシステムを構築する技術の追及，Prologの並列化の研究などである．我々は，多くの研究テーマについて研究を立ち上げ，短期間の内に成果を挙げる事が出来た．

これらの研究の中で特記すべき成果は，以下の三つである．第一は，Prologにオブジェクト指向の機能を拡張した言語ESP(Extended Self-contained Prolog)と，その言語の専用ワークステーションPSIの開発である．我々は，そのワークステーションのためのオペレーティングシステムをすべてESPによって記述した[Chikayama 88]．

第二の成果は，部分評価技術のメタプログラミングへの応用である．この技術によって，新たなプログラミング言語のコンパイラを，その言語のインタプリタを与え，それを部分評価する事によって作る事が可能となった．我々は，この技術を用いて，分脈自由文法のボトムアップ・パーサを導き出す事に成功した．言い換えれば，部分評価によって初めて，メタプログラミング技術が実用に耐えるものとなった．

第三の成果は，制約論理プログラミング言語の開発である．我々は，CILとCALの二つの制約論理言語を開発した．CILは，自然言語処理を目的として開発され，状況理論での“Complex Indeterminates”と言う概念を表現するための不完全データ構造が導入されている．それによって，Minskyのフレームのようなデータ構造の表現が可能であり，スロット値間の任意の関係は，制約によって表現出来る．CILはDUALSと呼ばれる自然言語理解システムの開発に用いられた．もう一方の制約論理言語CALは，非線形方程式を対象とした言語である．CALでは，Gröbner Basisを求めるためのBuchbergerアルゴリズムを用いて，非線形連立方程式を解いている．

Prologを基にした研究のこのような成果にもかかわらず，我々は一つの困難な問題に遭遇した．それは，「Prologの基本的枠組の中に並行性の概念がない」と言う事実であ

る．我々は，並列処理技術の開発における「並行性」の果たす役割の重要性を認識していたので，その概念を持った他の論理プログラミング言語を探し始めた．

我々は，Keith Clark と Steve Gregory による Relational Language [ClarkGregory 81] および Ehud Shapiro による Concurrent Prolog [Shapiro 83] に注目した．この二つの言語は，「並行性」を導入するために，committed choice nondeterminism を採用している．我々は，これらの言語を詳細に検討し，その結果として，上田は，この二つの言語とほぼ同じシンタックスを持っているが，より単純で，しかも表現力もほとんど劣らない第三の並行論理プログラミング言語 **Guarded Horn Clauses (GHC)** [Ueda 86a], [UedaChikayama 90] を開発した．我々は，GHC の重要性を認識し，直ちにそれを我々のプロジェクトの核言語 KL1 の中核として採用した．KL1 の導入により，プロジェクトは，KL1 の専用並列ハードウエアとその言語の上のソフトウエア技術の開発の二つの部分に分けて，並行して推進する事が可能となった．この点において，GHC の開発が第五世代コンピュータプロジェクトの遂行に果たした役割は，大変大きい．

これらのプログラミング言語指向の研究の他，我々は，人工知能およびソフトウエア工学の分野で，論理および論理プログラミングに基づく多くの基礎的な研究を手掛けてきた．それらは，非単調推論，仮説推論，発想推論 (abduction)，帰納推論，知識表現，定理証明，部分評価，プログラム変換などである．我々は，これらの問題が論理および論理プログラミングに基づく並列処理との相性の良さから，将来我々の並列コンピュータの重要な応用分野になるであろうと予測した．これは，既に起こりつつある．

以下，この論文では，初めに並行論理プログラミング，および制約論理プログラミングについて述べる．次に，ソフトウエア工学と人工知能分野における我々の最新の研究成果を紹介する．最後に，将来の研究の方向性について触れる．

2 並行論理プログラミング

この章では，本プロジェクトにおける並行論理プログラミングの二つのトピックスを取り上げる．第一に，我々の並行論理プログラミング言語 **Flat GHC (FGHC)**[1] [Ueda 86a], [UedaChikayama 90] の設計原理について述べる．第二に FGHC における探索パラダイムについて述べる．後で述べるように，FGHC は，論理プログラミング言語としてみた時，Prolog に備わっている探索機能に欠けている，と言う欠点を有する．この機能は，論理プログラミングにおける完全性の概念に深く係わっているので，この機能を取り戻す事は，大変意味がある．

[1]FGHC は，GHC のサブセットで，KL1 の基になっている言語である．

2.1 FGHC の設計原理

FGHC の最も重要な特徴は，立て棒 "|" で表される，コミットメント・オペレータと呼ばれるものだけが Prolog に対する構文的拡張になっている点である．コミットメント・オペレータは，節全体を**ガード部**（立て棒の左）と**ボディ部**（立て棒の右）に二分する．節のガード部は，二つの重要な役割を担っている．第一に，ガード部は，その節が引き続く計算を行う節として選ばれる条件を与えており，第二に，それは同期の条件を与えている．FGHC における同期の一般規則は，**データフロー同期**である．これは，計算を行うのに充分なデータが来るまでその計算が中断される事，を意味する．FGHC の場合では，呼びだし条件を判定するのに足るだけ充分に呼びだし元が具体化されるまで，ガードの計算が中断される．例として，切符の自動販売機がどのように動くかを考えてみよう．コインが挿入された後，切符自動販売機は利用者が行き先のボタンを押すのを待たなければならない．この待ち合わせは，ガードを持った節によって，「もし利用者が 160 円のボタンを押せば，それは 160 円の切符を発行する」といった規則として表現される．

重要な点は，データフロー同期が，ゴールが実行されてそのゴールに相当する FGHC の節が呼び出される時のヘッド・ユニフィケーションのルールによって実現される，という事実である．そのルールは，**ヘッド・ユニフィケーションでの情報の流れは，呼びだし元から呼び出される節への一方向でなければならない**，というものである．例として，窓口でのサービスを表す述語を考えてみよう．その述語は，次のような二つの節によって定義される．第一の節は，日中，客が来続ける時のためのもので，第二は，業務終了後で，もう客が来ない時のためのものである．それらの節は，次のように定義される：

```
serve([First | Rest]) :- <extra-condition> |
      do_service(First), serve(Rest).
serve([]) :- true | true.
```

ここには定義を与えないが，*serve* プロセスの他に，サービスを受ける人の待ち行列を生成する *queue* プロセスも必要となることに注意しよう．ここで，トップレベルのゴールは，次のようになる：

```
?- queue(Xs), serve(Xs).
```

ここで "?-" は，端末ユーザのためのプロンプトである．このゴールを実行すると，変数 Xs を共有する queue と serve の二つのプロセスが生成される．この**共有変数**は，一方のプロセスから他方のプロセスへデータを転送するためのチャネルとして振る舞う．この例では，queue プロセスが変数 Xs を具体化し，serve プロセスがその値を読む．言い換えると，queue が Xs の値の生産者として働き，serve が消費者として働く．queue プロセスは，Xs を

[<first-servee>, <second-servee>,...] の形をしたサービス受益者の並びに具体化するか，空リスト [] に具体化するかの，いずれかである．また，具体化する前は，Xs の値は，未定義である．いま，Xs は未定義であったとしよう．するとゴール serve(Xs) によって引き起こされるヘッド・ユニフィケーションは，中断される．というのは，方程式 Xs = [First | Rest] および Xs = [] が共に Xs を具体化せずに解く事が出来ないが，そのような具体化は，一方向ユニフィケーションのルールに違反するからである．ここで，serve の頭部に現れる [First | Rest] は，その節が，その引き数の値として空でないリストが与えられるのを期待している事を意味する．一方，項 [] は，そこに空リストが来ることを期待している．この例から，情報の流れの一方向性がデータフロー同期を実現している事が理解出来るであろう．

この原理は，二つの点で重要である．第一に，このプログラミング言語が，並行性を表現する自然な道具となっている点であり，第二に，その同期機構が，効率の良い並列ハードウエアの実現を可能にするに足るだけ，充分に単純である点である．

2.2 FGHC における探索パラダイム

FGHC は，committed choice の性質そのものに由来する重大な欠点を有している．それは，Prolog の最も重要な特徴である自動探索機能を備えていない点である．Prolog では探索機能は自動後戻り機構によってもたらされているが，committed choice 機構は引き続く計算のための節を一意的に決めてしまうので，自動後戻り，あるいはそれに類する方法によって他の可能性を調べる事が出来ない．探索機能は論理プログラミングの計算手続きに付随する完全性の性質に関連しており，その能力の欠如は，この点において重大な欠陥である．

探索機能は，OR 並列探索によって簡単に実現出来るのではないか，と思われるかも知れない．すなわち，これまでの変数の束縛情報を持った計算環境を複製して，各選択枝毎に並列に計算を続ける方法である．しかし FGHC では変数が複製出来ないので，この方法は使えない．その理由は，FGHC では変数への束縛がいつ起こるかが分からないからである．すなわち，ある変数に対する束縛があるプロセッサで実行された後でも，別のプロセッサではその同じ変数が暫くは未定義であり続ける事が起こり得るのである．

一方，我々は何故 Prolog のような探索機能を備えた言語を核言語として選択しなかったのか，と疑問に思われるかも知れない．これには，二つの理由がある．第一の理由は上に述べたように Prolog は並行性を表現出来ない，と言う点である．並行性は，並行アルゴリズムにとっての鍵となる特性であるばかりでなく，AND 並列計算の制御にとっても極めて重要な役割を果たす．第二の理由は，Prolog のような探索機能を備えた言語の実行機構は，効率の良い並列処理機構を実現するには複雑過ぎる，と言う点である．

我々は，プログラミング言語を出来るだけ単純に保ったままでプログラミング技術の工夫によって探索機能を取り戻す事を試み，そのようなプログラミング技法を幾つか開発した．以下の三つは，その代表例である．

(1) Continuation 法 [Ueda 86b]
(2) Layered stream 法 [OkumuraMatsumoto 87]
(3) 質問コンパイル法 [Furukawa 92]

この論文では，この中で (1) と (3) を取り上げる．と言うのは，その二つが，相互に補完し合っているからである．

continuation 法は，再帰アルゴルズムで表現されるような全解探索問題に適している．例えば，append プログラムによって，与えられたリストのすべての 2 分割を求めるような場合である．この方法は，Prolog の OR 並列実行を模擬している．Prolog での AND 逐次計算は，スタック上に計算の継続点 (continuation point) を記憶する方法に翻訳される．計算の中間結果は，この継続点スタックを経由して，次のゴールへと渡される．この方法は，一つのゴールから他のゴールへの一方向のデータフローを仮定しているので，この方法を用いるためには，入出力のモード解析が必要である．このため，データベースの質問応答のように一つの質問リテラルに対して多数の入出力関係があり得る場合には，この方法は非現実的である．

質問コンパイル法は，この問題を解決している．この方法は，渕が KL1 によってボトムアップ型定理証明器を実現した時に初めて導入された [Fuchi 90]．このプログラミング技法では，環境の複製問題は，データベースの質問応答プログラムの Prolog による最も直接的実現方法での呼びだし元と呼びだし先の役割を入れ替える事によって，回避している．すなわち，ゴールとして与えられた質問のパターンに一致するレコードを節集合の中から探し出す代わりに，この方法は，ゴールとして与えられたレコードに一致する質問成分を表す節を探索し，この処理を全レコードに対して再帰的に繰り返す．レコードはすべて変数を含まない基底リテラルであるので，呼びだし元のパターンは変数を含まない．変数の具体化は，質問成分が探索され，現在のレコードにマッチする節が見つかった時に起こる．質問とデータベースの表現を Prolog における直接的な表現の逆にした事により，情報の流れは呼びだし元（データベース）から呼びだし先（質問成分）へとなり，これは，KL1 のユニフィケーションの方向性と合致している．これによって，KL1 でデッドロックの発生を防いでいる．もう一つの重要な仕掛けは，質問成分節が呼ばれる度に，その中に含まれる各変数に対して新しい未定義変数が作られ，それによって OR 並列の各計算枝のための計算環境が作られる事である．これらの仕掛けによって，KL1 変数をデータベースの質問中の変数の表現に用いる事が可能となり，データベース全体を質問の各入出力モード毎にコンパイルする必要がなくなる．

ここで述べた新しいコーディング技術は一般性があり，多くの応用プログラムの開発に利用可能である．その制限は，データベースが質問に比べてより具体化されていなければならない，と言うものであり，ボトムアップ型定理証明では，この制限は，各公理の値域制限性 (range-restrictedness) と呼ばれている．値域制限性は，公理の前提を満足するモデル要素（の組）を見出すのに成功した時，その公理の帰結部に現れる新たにモデル要素となるべきリテラルは基底項になっていなければならない事を意味する．

この制約は，一見大変強そうである．確かに，定理証明の分野ではこの条件を満たさない問題が数多く見られる．そのような問題に対しては，トップダウン型定理証明器が必要である．しかしながら，多くの実際的な応用では，ほとんどの場合有限の具体的なモデルを有しているので，値域制限性の条件を満足している．そのような問題には，VLSI-CAD，電子機器などの故障診断，計画問題，スケジューリング問題などがある．

我々は，MGTP (Model Generation Theorem Prover) と呼ばれる並列ボトムアップ型定理証明器を開発した [FujitaHasegawa 91]．それは，Manthey と Bry によって開発された SATCHMO と呼ばれる定理証明器に基づいている [MantheyBry 88]．我々は，その定理証明器を利用した新しい応用を開拓している．その一例として，MGTP を用いた abduction の計算アルゴリズムを第 5 節に示す．

3 制約論理プログラミング

我々は，プロジェクトの開始とほぼ同時に，自然言語の研究に関連して制約論理プログラミングの研究開発を開始した．向井と安川 [MukaiYasukawa 85] は，状況意味論の計算モデルを開発する事を目的として，CIL (Complex Indeterminates Language) と呼ばれるプログラミング言語を開発した．*complex indeterminate* は，不定個の引き数を持つ部分確定項を表現出来るデータ構造である．この言語の開発中に，彼らは Colmerauer による Prolog II [Colmerauer 86] に現れる *freeze* に注目し，それを CIL の制御構造として採用した．

制約解消の観点から見ると，CIL は受動的な制約解消しか行っていない．すなわち，制約伝播や連立方程式を解くなどと言った能動的な制約解消を行っていない．その後，我々は，能動的制約解消を含む制約論理プログラミングの研究開発を開始した．我々は，CAL と呼ばれるプログラミング言語を開発した．その言語は，非線形方程式によって，制約条件を表現する事が出来る．CAL の開発は，三つの出来事がその契機となった．第一に我々は，METIS と呼ばれる線形代数の定理証明器の開発を進めていた [OhsugaSakai 91]．第二に我々は，非線形方程式の解を与える Gröbner 基底を求めるための Buchberger アルゴリズムを知る機会を得た．そのアルゴリズムは，項書き換えシステムのための Knuth-Bendix 完備化アルゴリズムの一種であるので，METIS の経験を生かして，我々はそのアルゴリズムを直ちに実装する事に成功した．第三に，Jaffar と Lassez による CLP(X) の枠組の開発が挙げられる [JaffarLassez 86]．それは，制約論理言語の条件を与えるもので，我々は CAL がその条件を満たす事を確かめた．

一般的な記号処理の領域における制約論理型言語の研究として特筆すべきものに，津田の cu-Prolog [Tsuda 92] がある．cu-Prolog では，制約は展開 / 畳込み変換 (これについては，ソフトウェア工学における最適化技法として，後に本論文中で詳述する) というプログラム変換により解消される．ここでは，展開 / 畳込み変換は，項の間の組合せ的制約を解くための基本操作として用いられている．変換が行なわれる毎に，制約 (プログラム) は構文的に見てより制約を含まない形に変形される．この基本操作は，CAL の基本操作である項書換えに似ていることに注意されたい．これら二つの操作はいずれも，プログラムを書き換えて，何らかの標準形を得ることを目的としている．cu-Prolog のアイデアは橋田 [Hasida 92] の依存伝播 (dependency propagation) の研究において導入されたものである．彼は，文脈自由規則を記述したプログラムを依存伝播により実行することにより，チャートパーザと同等の効率で構文解析が行えることを示している．

橋田 [Hasida 92] は人工知能および認知科学の基礎的問題に関する，計算モデルの観点からの研究を続けている．彼の力学プログラミング (dynamnical programming) という計算モデルでは，計算は要素制約，節，ユニフィケーションに付随するさまざまなポテンシャルエネルギーにより制御される．ポテンシャルエネルギーは，制約に対する違反の度合を表し，したがって，ポテンシャルエネルギーを減少させることは制約を解くことに相当する．

制約論理プログラミングは，論理プログラミングの持つ記述力を大幅に向上させたもので，Prolog の対象とする領域を拡張することにより，人工知能の問題のほとんどをカバーするような，強力なプログラミング環境を提供できるのではないかと期待されている．

我々のプロジェクトの扱う問題のひとつに，記述力と効率の両方を求めて，制約論理プログラミングと並列論理プログラミングとをいかに統合するかという問題がある．

この統合を行なうことは，(1) 制約論理プログラミングは，それぞれの制約評価のスキーマで効率の良い実行を行なうためのコントロールの問題に焦点を合わせていることと，(2) 制約論理プログラミングは本質的には探索のパラダイムを含むものであり，自動バックトラックのような適当なサポートメカニズムを必要とすること，の二つの理由により，そう容易なものではない．

最初の問題点は，制約論理プログラミングスキーマにデータフロー制御機構を導入することにより，解決可能である．我々は，GDCC (Guarded Definite Clauses with Constraints) という並列制約論理プログラミング言語の試作を KL1 を用いて行なった [HawleyAiba 91]．GDCC は Maher により提唱され [Maher 87]，saraswat により拡張された [Saraswat 89] ask-tell メカニズムに基づいてい

る．GDCCにおいては，ガードにおける計算メカニズムを単純な一方向のユニフィケーションから，ask操作を用いてその時点の制約集合に対するガード部の条件のチェックを行なうという，より一般的なものに拡張している．一方本体部にある制約はtell操作を用いて制約の集合へと付加されるようになっている．制約集合における情報の不足からガード条件の判定が出来ない場合には，計算は一時中断される．また，制約集合のもとでガード条件が成り立たない場合には，そのガードの計算は失敗する．このようなガードの判定を用いることで，制約評価の実行順序を制御することが可能となる．これは，ある節の本体部に現れる制約は，その節のガード条件が成立するようになるまでは，制約集合に付け加えられないからである．

二番目の並列制約論理プログラミング言語に枠組において探索を実現する問題については，これまでのところ，まだ十分には解決されてはいない．ひとつの簡単な方法としては，論理変数を基礎項で表すことでフルユニフィケーションエンジンを実現し [Koshimura et al. 91]，これを用いてOR並列の探索機構を実現することである．しかし，この方法で実現したユニファイアは，組み込みのそれと比較して10ないし100倍効率が悪く，実際的な方法とは言い難い．次に，前節で述べたような，新しいコーディング技法を適用する方法が考えられる．このようなコーディング法を導入することにより，探索を効率良く実行できるようになるのではないかと考えている．このパラダイムは，並列推論マシンを記述力と効率の両方において高いレベルを要求されるような多くの応用において有用なものとするための鍵ともなるものである．

4 ソフトウエア工学の新技術

ソフトウエア工学は，ソフトウエア開発における諸側面の支援を追及している．それらは，ソフトウエアの生産性の向上，高品質ソフトウエアの開発，維持管理の容易なソフトウエアの追及などである．論理プログラミングは，ソフトウエア工学における多くの側面に対して，大なる可能性を秘めている．論理プログラミングの明らかな特長は，与えられた仕様に対する正当性の検証への適合性である．自動虫取りは，それに関連している．さらに，与えられた仕様に対する証明過程からのプログラムの合成，あるいは，帰納推論による例からのプログラムの合成も，論理プログラミングは適している．プログラムの最適化も論理プログラミングの枠組が威力を発揮する方向である．

この節では，プログラムの最適化に関する二つの話題を取り上げる．第一は，メタプログラミングの部分計算による最適化で，第二は，展開・畳み込みプログラム変換である．

4.1 メタプログラミングと部分計算

我々は，論理プログラミングの枠組で知識処理システムを開発するための道具立てとして，BowenとKowalskiの仕事にヒントを得てメタプログラミング技術の研究を行った [BowenKowalski 83]．知識同化システムも，メタプログラミングを用いれば，知識ベースが満たさなければならない統合性制約をメタルールとして捉えることにより極めて直接的に実現することができる．ただ，このアプローチの最大の難点は，対象プログラムを解釈するメタインタプリタのオーバヘッドのために実行効率が悪いということである．この問題に挑戦した結果，竹内－古川によって，部分計算に基づく最適化技法をメタプログラムに適用するという方法でこの問題の突破口が切り開かれた [TakeuchiFurukawa 86]．確信度つきのメタインタプリタが与えられたとき，コンパイルされて効率の向上した確信度計算つきエキスパートシステムプログラムを導出することができたのである．このプログラムでは最適化前に比べ，3倍以上の速度向上が得られた．次に，我々は，構文解析のインタプリタと文法規則を与えて効率のよいコンパイルされた構文解析プログラムを得る，というより高度な問題に挑戦した．その結果，松本の開発した**BUP** [Matsumoto et al. 83]と称するボトムアップ型の構文解析プログラムと同型のプログラムを導出することに成功した．BUPのメタインタプリタは制御構造がPrologのトップダウン制御と正反対であって，“Prolog in Prolog”のようなヴァニラインタプリタの自明な拡張ではないという点を考えると，この結果は大きな意義をもつ．

メタプログラミングに部分計算技法を適用することで大きな成功を収めた後，我々は次のステップとして，自己適用可能な部分計算プログラムの開発を行った．その結果，藤田－古川によって簡単な自己適用可能部分計算プログラムを得るのに成功した [FujitaFurukawa 88]．この部分計算プログラムは，次のような“Prolog in Prolog”に極めて相似なメタインタプリタで実現できることが示されている．

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

ここで，対象プログラム節 $H:-B$ に対応して単位節 clause(H,B) がプログラムデータベースに登録されているものとする．このPrologの自己インタプリタsolveの簡潔な定義から，次のような部分計算プログラムpsolveが示唆される．

```
psolve(true,true).
psolve((A,B),(RA,RB)):-psolve(A,RA),psolve(B,RB)
psolve(A,R):-clause(A,B),psolve(B,R).
psolve(A,A):-'$suspend'(A).
```

この部分計算プログラムpsolve(G,R)は，与えられたゴールGを部分的に解いて中間結果Rを返す．この中間結果Rは，所期ゴールGに対する剰余ゴールと呼ばれる．剰余ゴールはもし所期ゴールが完全に解ける場合にはtrue，そうでない場合はサブゴールの連言で与えられる．各サブゴール R_i は何らかの理由により '$suspend'($R_i$) によって評

価が止められたものである．予約述語 '$suspend'(P) は，評価を中断すべきゴールパタンP毎に，ユーザによって定義される．

psolve は solve と次のように関連づけられることに注意しよう．

```
solve(G):-psolve(G,R),solve(R).
```

すなわち，ゴールGが最終的に解けるということは，まず部分的にこれを解いたときに剰余ゴールRが得られたとすると，このRが最終的に解けるときに限る．つまり，ゴールGを解く“全計算”が，剰余ゴールRを導く部分計算とRを解く全計算という二つの一連の計算に分けられるということである．

我々は，上の psolve に手を加えることによって自己適用可能な部分計算プログラムを実現した．そのポイントは，Prolog の組み込み述語の扱い方と，部分計算プログラムを定義する述語それ自身に対し，それらを自己適用可能化する工夫である．我々は，この部分計算プログラムを自己適用すること，すなわち psolve プログラムで同じ psolve を部分計算することによって，インタプリタからコンパイラを得ることに成功した．さらにコンパイラコンパイラを導くのにも成功した．すなわち，異なるインタプリタ毎にコンパイラを得るために psolve を psolve に特化すべく，psolve を用いて部分計算することに成功したのである．

部分計算プログラムの自己適用によってコンパイラやコンパイラコンパイラが得られるということは，理論的には古くから知られていた [Futamura 71]．関数型言語の分野では，自己適用可能な部分計算プログラムを得ようとする試みが長い間にわたり数多く行われてきたが，その成果が現れ出したのはつい最近になってからである [Jones et al. 85] [Jones et al. 88] [GomardJones et al. 89]．これに対し，我々は論理型言語の分野において極めて短期間に極めて簡潔な形でこれを実現することに成功した．このことはまた，ユニフィケーションに基づく論理型言語が関数型言語に優っている一つの例証を与えるものといえよう．

4.2 展開・畳み込みプログラム変換

プログラム変換は，プログラムの形式的仕様あるいは宣言的ではあるが必ずしも効率の良くないプログラムを，より効率の良いプログラムに変換する技術である．宣言的に書かれたプログラムは，Prolog の標準である左から右への制御では効率が良くない事が多い．その典型例は，生成・試験型のプログラムに見られる．世木と古川は，その種のプログラムに対する展開・畳み込みプログラム変換法を開発した [SekiFurukawa 87]．ここで，その基本的アイディアを多少詳しく説明しよう．いま，gen_test(L) を以下のように定義される述語としよう．

```
gen_test(L) :- gen(L), test(L).
```

ここで，Lは，リストを表す変数で，gen(L) はリストLの生成器であり，test(L) はその試験器である．いま，gen と test が共に漸進的で，以下のように定義できるとしよう．

```
gen([]).
gen([X|L]) :- gen_element(X), gen(L).

test([]).
test([X|L]) :- test_element(X), test(L).
```

このような場合には，展開・畳み込み変換によって，以下に示すように，gen と test の二つのプロセスを融合させる事ができる．

```
gen_test([X|L]) :- gen([X|L]), test([X|L]).
```

unfold at gen and test

```
gen_test([X|L]) :- gen_element(X), gen(L),
    test_element(X), test(L).
```

fold by gen_test

```
gen_test([X|L]) :- gen_element(X),
    test_element(X), gen_test(L).
```

もし試験器が漸進的でなければ，このような展開・畳み込み変換はできない．そのような例の一つは，リスト中の要素が互いに異なっていなければならない場合である．その試験器は，以下のように定義される．

```
test([]).
test([X|L]) :- non_member(X,L), test(L).

non_member(_,[]).
non_member(X,[Y|L]):-
    dif(X,Y), non_member(X,L).
```

ここで，dif(X,Y) はXがYと等しくないかどうかを判定する述語である（その定義は，省略した）．さて，この述語 test は，漸進的ではない．と言うのは，リストの先頭要素Xに対する試験に，リスト全体の情報が必要となるからである．我々の与えたこの問題に対する解は，述語 test を，それと等価な漸進的なプログラムに置き換える事である．そのような等価なプログラム test' は，test に計算の中間結果を蓄える作業領域を引き数に追加する事によって，以下のように定義される．

```
test'([],_).
test'([X|L],Acc) :-
    non_member(X,Acc), test'(L,[X|Acc]).
```

test と test' の関係は，次の定理によって与えられる．

Theorem

```
test(L) ≡ test'(L,[])
```

この時，始めに与えた gen_test プログラムは，次のようになる．

```
gen_test(L) :- gen(L), test'(L,[]).
```

次に，展開・畳み込み変換を行いたいが，そのためには，次のような新述語の導入が必要である．

```
gen_test'(L,Acc) :- gen(L), test'(L,Acc).
```

この新述語に対して，前述と同様の変換過程を施すと，次のような融合された再帰的プログラム gen_test' を得る．

```
gen_test'([],_).
gen_test'([X|L],Acc) :- gen_element(X),
    non_member(X,Acc), gen_test'(L,[X|Acc]).
```

二つのゴール

```
?- test([X1,...,Xn]).

?- test'([X1,...,Xn]).
```

を記号実行し，その結果を比較する事により，作業領域の導入による各対の比較の順序の変更が，
$\Sigma_{i=1}^{N}\Sigma_{j=i}^{N}x_{ij} = \Sigma_{j=1}^{N}\Sigma_{i=1}^{j}x_{ij}$
のような二重加算の交換則の効果に類似している事が分かる．そのため，我々は，この性質を構造的交換則と呼んでいる．

展開 / 畳込み変換における重要な問題の一つとして，上の例の gen_test' のような新述語の導入が挙げられる．川村 [Kawamura 91] は適切な新述語を発見するための統語的規則を提案した．従来，差分リスト表現を用いて append プログラムを最適化する技法のように，領域に依存した発見的な知識を用いて適切な新述語を発見する研究は行なわれていた．川村の研究では，新述語の候補を選択するための一般的な基準を提供している．この方法では，まず変換するプログラムを解析して新述語定義に現れ得るパターンを求める．この解析は次のように行なわれる．プログラムを展開し，その結果現れるすべてのパターンを有限個の異なるパターンで表現できるよう，これらのパターンの一般化を行なう．しかし，このとき過度の一般化は避けなければならない．過度な一般化を避けるには Plotkin の提案した最小汎化 (least general generalization) [Plotkin 70] を用いるのが有効である．これに加えて川村は不必要な一般化を抑制するため次のような戦略を導入した．各パターンに対し，頭部がそのパターンとユニファイ可能な節の集合を関係づける．最小汎化はこの関係づけられた節集合が同一であるものの間でのみ行なわれる．ゴール・パターンはそのパターンに関係づけられた節集合中の節によってのみ展開できるので，上で述べたようにして過度な一般化を抑制すると，展開による不必要な節の増加を抑制できる．

5 論理に基づく人工知能研究

演繹的推論は，長い間論理および論理プログラミング研究において，中心的役割を演じてきた．しかし，最近になって，発想推論および帰納推論の二つが，多大な関心を集め，それらについての研究が急速に進展しつつある．これらの研究の方向は，問題の性質が外界に向けて開かれているような本質的に困難な人工知能の問題に関連している．そのような問題には，フレーム問題，機械学習，分散問題解決，自然言語理解，常識推論，仮説推論，および類推などが含まれる．これらの問題を解決するためには，非演繹的な推論能力を必要とする．

歴史的には，これらの問題に対するこれまでの研究は，殆んどすべてが問題領域に依存した個別的な発見的方法を採用しており，一般的な傾向として，論理に基づく形式的な方法を避ける傾向が見られた．それは，その形式化が表現および問題解決能力に限界があると広く信じられていたからである．しかしながら，その限界は，論理の演繹的な側面のみに由来するものであり，近年，論理に基づく発想推論や帰納推論がその種の問題に対して強力な枠組を提供する事が，広く認められるようになってきた．この見方を支持する証拠は，沢山ある．以下は，その内の代表例である．

- 自然言語理解の研究において，ユニフィケーション文法が構文解析，意味解析，および談話理解を統合するための道具として，重要な役割を演じている．
- 非単調推論において，極小限定や暗黙推論などの論理に基づく形式化や，それらの論理プログラムへのコンパイルなどの研究が急速に進展している．
- 機械学習の研究において，Model Inference System，逆融合，最小汎化などの論理的枠組を基にした成果が数多く見られる．
- 類推において，その推論は，演繹的推論と類似の推論規則によって，自然に記述できる．そして，その推論は発想推論と深く係わっている．

以下に，これらの問題に関連した，三つの話題を取り上げる．それらは，仮説推論，類推，および知識表現である．

5.1 仮説推論

仮説推論の論理的枠組は Poole ら [Poole *et al.* 87] などにより提案されている．仮説推論を暗黙推論や極小限定 (circumscription) と比べると，仮説推論は単純かつ効率的に実

現でき,他の推論形態も仮説推論を用いて実現することが可能であることにより,仮説推論が最も優位であり有望であるとも言われている.近年これらの推論形態の関係がより詳細に調べられており,非単調推論の問題を失敗による否定(negation as failure,以下NAFとする)付きの論理プログラムに翻訳する試みも数多く行われている.

他の研究の方向としては,アブダクションの定式化とNAFとの関係がある.一般論理プログラム(general logic program)は,各ルールの本体(body)にNAF式の出現を許した論理プログラムであるが,そのモデル論もアブダクションとの関係において提案されている.つまり,各NAF式を否定情報を含んだ仮定可能(abducible)述語を用いて置換することにより,アブダクションを用いてNAFが定式化できる[EshghiKowalski 89].

ここで一般論理プログラムの意味論としては,安定モデル意味論(stable model semantics)[GelfondLifschitz 88]が用いられている.この意味論は最小不動点意味論の自然な拡張にもなっている.ところが,安定モデルを直接的に計算するようなT_Pオペレータは存在しない点が異なっている.この理由は,ボトムアップに不動点計算を行うときにNAF式の真偽値を判定するためには,求めようとしている完全なモデルが必要となるからである.それゆえ,安定モデルの定義にそのモデルを参照する必要があり,安定モデルの計算を困難なものにしている.最も直接的な方法は,すべての可能な解釈を仮定し,各々についてプログラムを満足する最小モデルになっているかどうかを調べる方法である.このアルゴリズムではプログラムから生成され得る基礎アトムの集合のすべての部分集合を探索する必要があり,現実的な方法とは言い難い.

井上ら[Inoue *et al.* 92]は,統合性制約を含む一般論理プログラムの安定モデルを計算するために,はるかに効率的な方法を提案している.彼らのアルゴリズムはボトムアップなモデル生成法に基づいている.そこではNAF式は仮定的なモデルを導入するために使われる:つまり,NAF式$not\ A$に対して,そのうちの一つはAを真であると仮定し,他では偽であると仮定する.また仮定されたリテラルを表現するために,ある種の様相記号が導入されている.より正確には,プログラム中で

$$A_l \leftarrow A_{l+1} \wedge \ldots \wedge A_m,\ not\ A_{m+1} \wedge \ldots \wedge not\ A_n$$

という形をした各ルールは,

$$A_{l+1} \wedge \ldots \wedge A_m \rightarrow$$
$$(\mathsf{NK}A_{m+1} \wedge \ldots \wedge \mathsf{NK}A_n \wedge A_l) \vee \mathsf{K}A_{m+1} \vee \ldots \vee \mathsf{K}A_n .$$

というNAF式を含まないような選言的な節に変換される.ここで節の前件部を左辺に書く理由は,この節をボトムアップな方法で(すなわち,左辺から右辺を導くように)使うことを意図しているからである.上の表現で,$\mathsf{NK}A$はAが偽であることを仮定することを意味し,$\mathsf{K}A$はAが真であることを仮定することを意味する.KおよびNKは様相記号であるが,ここでは$\mathsf{K}A$および$\mathsf{NK}A$をAとは独立した新しいアトムとして扱うために,次のような制約が公理スキーマとして加えられる:

$$\mathsf{NK}A,\ A \rightarrow \quad \text{for every atom } A. \tag{1}$$

$$\mathsf{NK}A,\ \mathsf{K}A \rightarrow \quad \text{for every atom } A. \tag{2}$$

以上の変換によって,一階述語論理上の節集合が得られるため,2章で示したボトムアップ型定理証明器MGTPを使って,すべての極小モデルを計算することが可能となる.これらすべてのモデルを計算した後,次の条件を満たすモデルMの集合を選択すれば,それらが求めるべき安定モデル集合となる:

$$\text{For every ground atom } A, \text{ if } \mathsf{K}A \in M, \text{ then } A \in M. \tag{3}$$

上の変換方式が,NAF式を含む元の一般論理プログラムを一階述語論理を用いてコーディングする方法を定義していることに注意されたい.さらに,同様の手法はアブダクションの計算にも適用することができる.つまり,観測を説明するような仮説集合で,与えられた制約と矛盾しないようなものを見つけることが同様にして行える.

佐藤と岩山[SatohIwayama 92]は上とは独立に,統合的制約を含む一般論理プログラムの問い合わせ評価法を提案している.これはEshghiとKowalski[EshghiKowalski 89]によるトップダウン手続きを拡張したものであり,制約(3)で示されるように,ある命題があるモデルで成立しなければならない,ということを正確に評価しようとしたものである.

岩山と佐藤[IwayamaSatoh 91]は,統合的制約を含む一般論理プログラムの安定モデルを計算するために,ボトムアップ法とトップダウン法を混合した戦略を提案している.この手続きは基本的にはボトムアップ法であり,トップダウン計算は制約(3)の計算に関連している.つまり,仮説$\mathsf{K}A$があるモデルに付け加えられるとすぐに,トップダウン的にAを証明することが試みられる.

アブダクションの定式化は,計算機支援による設計や診断を含む非常に広範な応用を持っている.われわれのアプローチはそのような問題を表現し解くための統一的な方法を与えている.これはまた,それらの複雑なAI問題を解くためにわれわれの並列推論マシンPIMを活用する方法をも提供しているのである.

5.2 類推への形式的アプローチ

類推は人間の問題解決で最も重要な推論方法の一つである.ある問題単独では解くのが大変難しいものを解くのに類推は大変役立つ.与えられた問題の解き方を類推を働かせ類似の問題の解き方を指針にして得ることができる.類推のもう一つの面は,答を引き出すのに情報が十分ない場合でさえ,良い推測を引き出すのに使えることである.

類推を実現するためには3つの大きな問題を解決せねばならない[Arima 92].

- 与えられたターゲットに対して適当なベースを見つけること.
- ベースとターゲットが共有する重要な性質を選択すること.
- ベースからターゲットへある類比によって投射される性質を選択すること.

類推を実現する多くの研究がなされてきたが，これらの研究の多くは上記に述べた課題について部分的にしか答えていなかった．有馬 [Arima 92] はすべての問題を一度に答える試みを行なった．彼のアイデアを説明する前に，いくつかの技術用語について準備する必要がある．

類推を以下の推論規則で表そう:

$$\frac{S(B) \wedge P(B) \quad S(T)}{P(T)}$$

ここで T はターゲットと呼ばれる個体，B はベースと呼ばれる個体，S は T と B の間の類似性，P は投射性を表している．

この推論規則は，T と B が共通の性質 S を満たすという点で似ている場合に，B が別の性質 P を満たしているならば，T もまた P という性質を持つと類推できることを表している．この規則と三段論法との構文的な類似に注意するべきである．もし，B を全量化された x に一般化し，*and* 接合子を *implication* 接合子に置き換えると，最初の式は $\forall x.(S(x) \supset P(x))$ となり，この規則は三段論法を表すことになる．

有馬 [Arima 92] は $S(B) \wedge P(B)$ を

$$\forall x.(J(x) \wedge S(x) \supset P(x)), \qquad (4)$$

に変換することによって類推と演繹を関連づけようとした．ここで $J(x)$ は $P(x)$ を論理的帰結にするのに必要になる仮説である．もしそのような仮説を導けられれば，類推は演繹的な推論になる．例えば，オーケストラ部の部員で研究をさぼってばかりいる学生 ($Student_B$) がいるとしよう．もしたまたま別の学生 ($Student_T$) がオーケストラクラブに属していることを知ると，我々は彼もまた研究をよくさぼると考えがちである．我々がそのような結論を導くのは，オーケストラクラブが非常に活動的でそういった忙しいクラブにはいっている学生は研究をさぼりがちであると推測したからである．この推測が上記の仮説 $J(x)$ の例になっている．

有馬は上記 $J(x)$ の構文的な構造を類推が使われる状況を慎重に観察することによって分析した．まず，述語 J の適当な引数を見つけなければならない．この仮説は個体のみならず類似性，投射性にも依存するから，類似性，投射性をそれぞれ表す個体変数 s,p を導入して，J は $J(x,s,p)$ で表せるとする．

類推の性質から，個体 x と投射性 p の間に直接の関係があると仮定することはできない．それゆえ，$J(x,s,p)$ 全体は以下の 2 つの部分に分けることができる．

$$J(x,s,p) \;=\; J_{att}(s,p) \wedge J_{obj}(x,s), \qquad (5)$$

最初の構成要素 $J_{att}(s,p)$ はベースから引き出される情報に対応している．それが x に依存しないのは，類推においてベースから得られる情報は個体に関する情報それ自身とは無関係で，類似性と投射性の組合せの情報が重要であるという観測から得られる．

2 つめの構成要素 $J_{obj}(x,s)$ は類似性から引き出される情報に対応し，それゆえその引数に p を含まない．

例題: 怠慢学生

まず，われわれは先の推測でなぜオーケストラクラブの部員が研究をさぼるのかを説明することができた．これは形式的に表現すると以下のような知識を我々があらかじめ知っていたことに対応する:

$$\begin{aligned}\forall x,s,p.(\; &Enthusiastic(x,s) \wedge BusyClub(s)\\ &\wedge Obstructive\_to(p,s) \wedge Member\_of(x,s)\\ &\supset Negligent\_of(x,p)\;) \end{aligned} \qquad (6)$$

ここで x,s,p はそれぞれ人間，クラブ，人間の行動を表す変数である．それぞれの述語の意味は容易に理解されるので説明は省く．$Student_B$ と $Student_T$ は両者ともオーケストラ部の部員であることを知っているので，$Members\_of(x,s)$ は (4) 式の類似性 $S(x)$ に対応している．一方，ある学生が何をさぼるかに興味があるので，投射性 $P(x)$ は $Negligent\_of(x,p)$ に対応する．よって，(6) 式の残りの部分 $Enthusiastic(x,s) \wedge BusyClub(s) \wedge Obstructive\_to(p,s)$ は，$J(x,s,p)$ に対応している．この論理式の構文的な特徴からさらに，

$$J_{obj}(x,s) = Enthusiastic(x,s),$$
$$J_{att}(s,p) = BusyClub(s) \wedge Obstructive\_to(p,s)$$

と対応づけることができる．

$J_{obj}(x,s)$ が必要である理由は，我々がいつも $Enthusiastic$ のように重要な類似性を知ることができるわけでないためである．それゆえ，$Member\_of$ のような表面的に現れた類似性から表に現れない重要な類似性を推論せねばならない．この推論は上の類推を行なうために上記以外のなんらかの知識から推論されねばならない．

$J(x,s,p)$ のこの構文上の制限は与えられたターゲットから正しいベースへの探索空間を刈り取るのに使うことができる．この機能は事例に基づく推論に類推を応用する際に特に重要になる．

5.3 知識表現

知識表現は，人工知能研究における中心的な課題のひとつである．知識表現に関する問題は，多種多様な知識を表現でき，簡潔であり，同時に，効率的であるような，単一の表現スキーマがなかったことである．論理は，その候補として最も有望なものの１つであるが，構造的な知識の表現能力や，世界・状態の変化の扱いに関して弱点を持っていた．我々は，論理および論理プログラミングの枠組みに基づいて知識表現の枠組みを開発しようとしているが，その目標は，これら２つの問題点を解決することである．

構造的な知識の表現という観点から，非正規形のデータを対象とする関係データベースシステムと，その上のプログラミング言語 CRL [Yokota et al. 88b] を開発した．そこで用いられている表現形式により，論理に基づく枠組みで，構造的なデータベースを記述することが可能となっている．

また，最近にいたり，論理に基づく新しい知識表現言語 Quixote [YasukawaYokota 90] を提案した．Quixote は，CRL と CIL で示されたアイディアを統合・拡張したものである．すなわち，CRL からオブジェクト指向性を，CIL から部分項のアイディアを引き継ぎ，それらを拡張している．オブジェクト指向性に関する基本的な特性としてオブジェクト識別性という概念があるが，Quixote では，オブジェクト指向性を，単に原子的な (atomic) データだけでなく，複合的な構造体を用いて表現することができる [Morita 90]．さらに，再帰的構造を持つオブジェクト表現を取り扱うことも可能である．そのため，オブジェクトやオブジェクトのユニフィケーションは，P. Aczel の非整礎集合論 (non-wellfounded set theory) により数学的な形式化がおこなわれており，実装手法としては，無限木のユニフィケーション・アルゴリズム [Colmerauer 82] が用いられている．

6 おわりに

この論文では，FGCS プロジェクトの基礎研究の紹介を行った．ここでは，我々の研究活動を論理プログラミングにおける二つの異なる研究方向に分けて論じた．その第一は，論理プログラミング言語を主題とする研究の取り組みで，そこでは制約論理プログラミングと並行論理プログラミングの二つの話題に焦点を当てた．第二は，人工知能とソフトウエア工学の両分野における論理および論理プログラミングに基づく基礎研究である．

このプロジェクトは，*ジグソーパズル*を解くのに似ている．それは，「論理と論理プログラミングを手掛りとしてパズルを完成させる」，のように譬える事ができる．解決すべき研究課題は，このイメージから自然に導き出された．制約論理プログラミングも，並行論理プログラミングもこのような構図から，自然に本プロジェクトの中心課題として取り上げられた分けである．それらの中には，困難な研究課題も幾つかあった．課題によっては，その結果を評価する客観的な尺度を持ち会わせていない事さえあった．GHC の選択は，そのような例の一つである．そのような状況において，このプロジェクト全体の絵が，研究を適切な方向に誘導する役割を果たした．

この絵は，未だ完成されていない．残った空隙を埋めるために，さらに研究開発を継続させる事が必要である．これまで作られた絵に付け加えられるべき最も重要なものは，制約論理プログラミングと並行論理プログラミングの統合である．我々は，そのような試みの一つとして，GDCC の開発を進めてきたが，それは，今だ完成には致っていない．それは，実際の応用問題の記述にとって有用で，しかも KL1 を介して並列ハードウエア PIM 上で効率良く実行されるものでなければならない．追及されるべき第二の研究課題は，KL1 上でのデータベースの実現である．我々は，実際並列データベースの開発を精力的に進めているが，現在は未だ開発の初期段階である．データベースは，並列処理に適しており，データベースを含んだ多くの応用問題が，膨大な並列性を抱えていると考えられる．追及されるべき第三の研究課題は，発想推論，および帰納推論の並列化である．近年，論理および論理プログラミングの枠組を用いた発想推論および帰納推論の研究が盛んになってきている．それらの研究成果は，知識獲得および機械学習に関する研究の基礎を与えるものと期待されている．さらに，発想推論と帰納推論の両者共に，膨大な記号計算を必要としているので，それらは，共に PIM アーキテクチャに大変適した問題であると考えられる．

我々の研究成果が幅広い分野の大規模な応用問題に対して有用である事を示すためにはさらに研究開発を進める事が必要であるが，我々は，このプロジェクトのアプローチが正解である事を確信している．

謝辞

この論文は，第五世代コンピュータプロジェクトのすべての基礎研究の成果を反映したものである．著者は ICOT および本プロジェクトの研究開発に係わってこられた関連会社の研究員すべてに感謝したい．特に，相場亮，有馬淳，藤田博，橋田浩一，井上克己，岩山登，川村正，佐藤健，津田宏，上田和紀，安川秀樹，横田一正の各氏には，本論文の作成に当たり，内容を含めてその改善に多大な協力を頂いた．最後に，本論文を書く機会を与えていただいた ICOT 研究所所長渕一博博士に深く感謝する．

参考文献

[Arima 92] J. Arima, Logical Structure of Analogy. *In Proc. of the International Conf. on Fifth Generation Computer Systems 1992*, Tokyo, 1992.

[Aczel 88] P. Aczel, *Non-Well Founded Set Theory*. CLSI Lecture Notes No. 14, 1988.

[Aiba *et al.* 88] A. Aiba, K. Sakai, Y. Sato, D.J. Hawley, and R. Hasegawa, Constraint Logic Programming Language CAL. In *Proc. of the International Conf. on Fifth Generation Computing Systems 1988*, Tokyo, 1988.

[BowenKowalski 83] K. Bowen and R. Kowalski, Amalgamating Language and Metalanguage in Logic Programming. In *Logic Programming*, K. Clark and S. Tarnlund (eds.), Academic Press, 1983.

[ClarkGregory 81] K. L. Clark and S. Gregory, *A Relational Language for Parallel Programming*. In Proc. ACM Conf. on Functional Programming Languages and Computer Architecture, ACM, 1981.

[ClarkGregory 86] K. L. Clark and S. Gregory, *PARLOG: Parallel Programming in Logic*. Research Report DOC 84/4, Dept. of Computing, Imperial College of Science and Technology, London. Also in *ACM. Trans. Prog. Lang. Syst.*, Vol. 8, No. 1, 1986.

[Chikayama 88] T. Chikayama, Programming in ESP – Experiences with SIMPOS –, In *Programming of Future Generation Computers*, Fuchi and Nivat (eds.), North-Holland, 1988.

[Colmerauer 82] A. Colmerauer, Prolog and Infinite Trees. In *Logic Programming*, K. L. Clark and S. Å. Tärnlund (eds.), Academic Press, 1982.

[Colmerauer 86] A. Colmerauer, Theoretical Model of Prolog II. In *Logic Programming and Its Applications*, M. Van Caneghem and D. H. D. Warren (eds.), Albex Publishing Corp, 1986.

[FuchiFurukawa 87] K. Fuchi and K. Furukawa, *The Role of Logic Programming in the Fifth Generation Computer Project*. New Generation Computing, Vol. 5, No. 1, Ohmsha-Springer, 1987.

[EshghiKowalski 89] K. Eshghi and R.A. Kowalski, Abduction compared with negation by failure, in: *Proceedings of the Sixth International Conference on Logic Programming*, Lisbon, Portugal, 1989.

[Fuchi 90] K. Fuchi, *An Impression of KL1 Programming - from my experience with writing parallel provers -*. In Proc. of KL1 Programming Workshop '90, ICOT, 1990 (in Japanese).

[FujitaFurukawa 88] H. Fujita and K. Furukawa, *A Self-Applicable Partial Evaluator and Its Use in Incremental Compilation*. New Generation Computing, Vol. 6, Nos.2,3, Ohmsha-Springer, 1988.

[FujitaHasegawa 91] H. Fujita and R. Hasegawa, *A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm*. In Proc. of the Eighth International Conference on Logic Programming, Paris, 1991.

[Furukawa 92] K. Furukawa, Logic Programming as the Integrator of the Fifth Generation Computer Systems Project, *Communication of the ACM*, Vol. 35, No. 3, 1992.

[Futamura 71] Y. Futamura, Partial Evaluation of Computation Process: An Approach to a Compiler-Compiler. *Systems, Computers, Controls 2*, 1971.

[GelfondLifschitz 88] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, Seattle, WA, 1988.

[GomardJones 89] C. K. Gomard and N. D. Jones, Compiler Generation by Partial Evaluation: A Case Study. In *Proc. of Information Processing 89*, G. X. Ritter (ed.), North-Holland, 1989.

[Hasida 92] K. Hasida, Dynamics of Symbol Systems - An Integrated Architecture of Cognition. *In Proc. of the International Conf. on Fifth Generation Computer Systems 1992*, Tokyo, 1992.

[HawleyAiba 91] D. Hawley and A. Aiba, *Guarded Definite Clauses with Constraints – Preliminary Report*. Technical Report TR-713, ICOT, 1991.

[Inoue *et al.* 92] K. Inoue, M. Koshimura and R. Hasegawa, Embedding Negation as Failure into a Model Generation Theorem Prover. *To*

appear in CADE-11: The Eleventh International Conference on Automated Deduction, Saratoga Springs, NY, June 1992.

[IwayamaSatoh 91] N. Iwayama and K. Satoh, *A Bottom-up Procedure with Top-down Expectation for General Logic Programs with Integrity Constraints*. ICOT Technical Report TR-625, 1991.

[JaffarLassez 86] J. Jaffar and J-L. Lassez, *Constraint Logic Programming*. Technical Report, Department of Computer Science, Monash University, 1986.

[Jones *et al.* 85] N. D. Jones, P. Sestoft, and H. Søndergaard, An Experiment in Partial Evaluation: The Generation of a Compiler Generator. In J-.P. Jouannaud (ed.), *Rewriting Techniques and Applications*, LNCS-202, Springer-Verlag, pp.124–140, 1985.

[Jones *et al.* 88] N. D. Jones, P. Setstoft and H. Søndergaard, MIX: a self-applicable partial evaluator for experiments in compiler generator,Journal of LISP and Symbolic Computation, 1988.

[Kawamura 91] T. Kawamura, *Derivation of Efficient Logic Programs by Synthesizing New Predicates*. Proc. of 1991 International Logic Programming Symposium, pp.611 – 625, San Diego, 1991.

[Koshimura *et al.* 91] M. Koshimura, H. Fujita and R. Hasegawa, *Utilities for Meta-Programming in KL1*. In Proc. of KL1 Programming Workshop'91, ICOT, 1991 (in Japanese).

[Maher 87] M. J. Maher, *Logic semantics for a class of committed-choice programs*. In Proc. of the 4th Int. Conf. on Logic Programming, MIT Press, 1987.

[MantheyBry 88] R. Manthey and F. Bry, *SATCHMO: A Theorem Prover Implemented in Prolog*. In Proc. of CADE-88, Argonne, Illinois, 1988.

[Matsumoto *et al.* 83] Yuji Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi and H. Yasukawa, BUP: A Bottom-up Parser Embedded in Prolog, *New Generation Computing*, Vol. 1, 1983.

[Morita *et al.* 90] Y. Morita, H. Haniuda and K. Yokota, *Object Identity in Quixote*. Technical Report TR-601, ICOT, 1990.

[MukaiYasukawa 85] K. Mukai, and H. Yasukawa, Complex Indeterminates in Prolog and its Application to Discourse Models. *New Generation Computing*, Vol. 3, No. 4, 1985.

[OhsugaSakai 91] A. Ohsuga and K. Sakai, Metis: A Term Rewriting System Generator. In *Software Science and Engineering*, I. Nakata and M. Hagiya (eds.), World Scientific, 1991.

[OkumuraMatsumoto 87] Akira Okumura and Yuji Matsumoto, Parallel Programming with Layered Streams, In *Proc. 1987 International Symposium on Logic Programming*, San Francisco, September 1987.

[Plotkin 70] G. D. Plotkin, *A note on inductive generalization*. In B. Meltzer and D. Michie (eds.), Machine Intelligence 5, 1970.

[Poole *et al.* 87] D. Poole, R. Goebel and R. Aleliunas, Theorist: A logical Reasoning System for Defaults and Diagnosis, N. Cercone and G. McCalla (eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer-Verlag, pp.331–352 (1987).

[SakaiAiba 89] K. Sakai and A. Aiba, *CAL: A Theoretical Background of Constraint Logic Programming and its Applications*. J. Symbolic Computation, Vol.8, No.6, pp.589-603, 1989.

[Saraswat 89] V. Saraswat, *Concurrent Constraint Programming Languages*. PhD thesis, Carnegie-Mellon University, Computer Science Department, 1989.

[SatohIwayama 92] K. Satoh and N. Iwayama, A Correct Top-down Proof Procedure for a General Logic Program with Integrity Constraints. In *Proc. of the 3rd International Workshop on Extensions of Logic Programming*, E. Lamma and P. Mello (eds.),

Facalta di Ingegneria, Universita di Bologna, Italy, 1992.

[SekiFurukawa 87] H. Seki and K. Furukawa, *Notes on Transformation techniques for Generate and Test Logic Programs.* In Proc. 1987 Symposium on Logic Programming, IEEE Computer Society Press, 1987.

[Shapiro 83] E. Y. Shapiro, *A Subset of Concurrent Prolog and Its Interpreter.* Tech. Report TR-003, Institute for New Generation Computer Technology, Tokyo, 1983.

[Sugimura *et al.* 88] R. Sugimura, K. Hasida, K. Akasaka, K. Hatano, Y. Kubo, T. Okunishi, and T. Takizuka, A Software Environment for Research into Discourse Understanding Systems. In *Proc. of the International Conf. on Fifth Generation Computing Systems 1988,* Tokyo, 1988.

[TakeuchiFurukawa 86] A. Takeuchi and K. Furukawa, *Partial Evaluation of Prolog Programs and Its Application to Meta Programming.* In Proc. IFIP'86, North-Holland, 1986.

[Taki 88] K. Taki, *The Parallel Software Research and Development Tool: Multi-PSI system.* In Programming of Future Generation Computers, K. Fuchi and M. Nivat (eds.), North-Holland, 1988.

[Taki 89] K. Taki, *The PGCS Computing Architecture.* In Proc. IFIP'89, North-Holland, 1989.

[TanakaYoshioka 88] Y. Tanaka, and T. Yoshioka, Overview of the Dictionary and Lexical Knowledge Base Research. In *Proc. FGCS'88,* Tokyo, 1988.

[Tsuda 92] H. Tsuda, cu-Prolog for Constraint-based Grammar. *In Proc. of the International Conf. on Fifth Generation Computer Systems 1992,* Tokyo, 1992.

[Ueda 86a] K. Ueda, *Guarded Horn Clauses.* In Logic Programming '85, E. Wada (ed.), Lecture Notes in Computer Science, 221, Springer-Verlag, 1986.

[Ueda 86b] K. Ueda, *Making Exhaustive Search Programs Deterministic.* In Proc. of the Third Int. Conf. on Logic Programming, Springer-Verlag, 1986.

[UedaChikayama 90] K. Ueda and T. Chikayama, *Design of the Kernel Language for the Parallel Inference Machine.* The Computer Journal, Vol. 33, No. 6, pp. 494–500, 1990.

[Warren 83] D. H. D. Warren, *An Abstract Prolog Instruction Set.* Technical Note 304, Artificial Intelligence Center, SRI, 1983.

[YasukawaYokota 90] H. Yasukawa and K. Yokota, *Labeled Graphs as Semantics of Objects.* Technical Report TR-600, ICOT, 1990.

[Yokota 88a] K. Yokota, *Deductive Approach for Nested Relations.* In Programming of Future Generation Computers II, K. Fuchi and L. Kott (eds.), North-Holland, 1988.

[Yokota *et al.* 88b] K. Yokota, M. Kawamura and A. Kanaegami, Overview of the Knowledge Base Management System(KAPPA). In *Proc. of the International Conf. on Fifth Generation Computer Systems 1988,* Tokyo, 1988.