# 並列定理証明器とその応用

長谷川 隆三　　藤田 正幸

第五研究室
新世代コンピュータ技術開発機構
東京都港区三田 1 丁目 4-28
{hasegawa, mfujita}@icot.or.jp

## 概要

本論文では、ICOT 第 5 研究室で進めてきた自動推論システムの研究開発成果について述べる。主要な成果の一つは、KL1 言語を用いて並列推論マシン PIM 上に開発した、1 階述語論理のための並列定理証明システム MGTP (Model Generation Theorem Prover) である。現在、基底アトムのみからなるモデルを扱う基底版 (MGTP/G)、および変数含みのアトムからなるモデルを扱う非基底版 (MGTP/N)、の二種類の MGTP が稼動中である。128 PE 構成の PIM/m 上での、非基底版 MGTP/N の走行実験結果、分離の規則に関する数学的定理証明問題では 100 倍以上の速度向上が確認された。

MGTP の有望かつ具体的な応用領域として、非単調推論およびプログラム合成をとりあげた。基底版 MGTP/G は、ICOT 第 7 研究室において、法的推論システムの開発に実際に使用されている。また、MGTP の推論能力および応用領域の拡大を狙って、高次推論・学習システムの基礎的な検討も合わせて行った。MGTP の開発過程で、並列論理プログラミング技術およびメタプログラミングのためのユーティリティプログラムが KL1 を用いて開発された。これらの技術は、PIM 上での応用開発に広く用いられている。

## 1　はじめに

第五世代コンピュータシステム (FGCS) プロジェクトの最終目標は知的なユーザインタフェースと知識ベースシステムを備えた知識情報処理システムを並列推論マシン上に実現することである。この目標を達成するためには、高機能、高並列の推論機構の開発が重要な研究課題の一つとなる。

第五研究室では、第五世代コンピュータシステムの構成要素である問題解決プログラミング・モジュールの研究開発の一環として、上記課題に取り組んだ。我々の当面の目標は、KL1 言語並びに PIMOS オペレーティングシステムの利点を活用して、並列推論マシン PIM 上に、高効率、高並列の自動推論システムを構築することである。我々は、

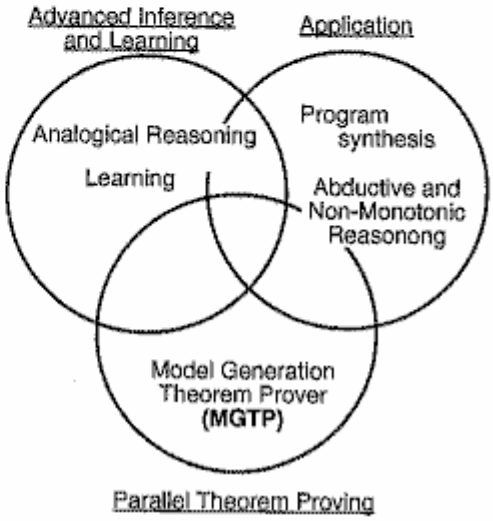


図 1: ICOT における自動推論システムの研究開発テーマ

このシステムを自然言語処理、知的知識ベース、数学的定理証明、自動プログラミングといった様々な分野に応用することを目指している。また、自動推論システムの開発過程において、KL1 による実装及び並列推論ハードウェア上での実験を通じて得られた知見を、論理型言語ならびにオペレーティングシステムにフィードバックすることも本研究の狙いの一つである。

自動推論システムの研究開発を進めるにあたって次にあげる三つの研究開発テーマを設定した (図 1)。

(1) PIM 上での並列定理証明技術の開発

KL1 言語の能力を最大限引き出すことによって、効率の良い並列定理証明器を PIM 上に開発することが、本開発課題の主要な命題である。我々は、超導出 (hyper-resolution) に基づく推論機構を有するモデル生成法に焦点をあて、基底問題および非基底問題の両方に対処するため、二種類のモデル生成型定理証明器を開発

することにした。PIM 上で最大限の性能を追究することを第一義とし、以下のような技術課題に的を絞った。

(a) 冗長計算の除去

モデル生成の過程における冗長計算を最小のオーバヘッドで除去することは重要な課題の一つである。モデル生成法には、超導出のステップで行われる連言照合、あるいは非ホーン基底問題を扱う時の場合分けに冗長性が潜んでいる。

(b) 過剰なモデル生成を防ぐことによる時間・空間量の削減

超導出に基づきボトムアップに実行を進めるモデル生成法にとっては、モデルの過剰生成は時間・空間の浪費を引き起こす本質的問題である。我々は、モデル生成法を生成・検査手続きとみなすことによりこの問題の解決を図った、遅延モデル生成と呼ぶ新たな制御機構を導入した。

(c) PIM 向きのスケーラブルな並列アーキテクチャの探究

PIM は低通信コストの MIMD マシンであり、この特長を最大限引き出しうるような、モデル生成定理証明器のための並列アーキテクチャを見い出すことが本課題の目標である。非ホーン基底問題用の MGTP/G では、場合分けの並列性に注目し、これを抽出するための OR 並列アーキテクチャを中心に検討した。一方、ホーン非基底問題用の MGTP/N では、連言照合および包摂テストに内在する並列性を抽出する AND 並列アーキテクチャに焦点を絞った。

KL1 による定理証明器開発の重要な狙いの一つは、並列論理プログラミングパラダイムの利点を KL1 から引き出すことである。この狙いを具体化するための足固めとして、KL1 によるメタプログラミング技術の開発にも力点を置いた。我々は、問題解決のための応用を幅広く開発すると共に、それらを支援しうるよう KL1 を拡張していくことが重要であると考えている。定理証明器を作成する過程で開発された KL1 プログラミング技術は、論理プログラミングおよびその拡張をベースとした PIM 上の様々な応用、例えば自然言語処理システムや知識ベースシステム等、の開発の一助になっており、共通技術として浸透しつつある。

(2) 応用

モデル生成型定理証明器は普遍的な推論能力を有しており、種々の AI 分野において広く用いられているタブロ法を効果的に模擬することができる。モデル生成定理証明器の用途拡大を図って、この上に、様相命題論理のための効率的なタブロ証明器を構築することを試みた。このアプローチは、診断等のより実際的な応用と密接に関連している、アブダクション (abduction) や失敗による否定 (negation as failure) を自然な形でモデル生成型定理証明器上に実現することを可能にした。

我々は、非ホーン論理の定理証明器の主要な応用分野の一つとして、困難な問題ではあるが、自動プログラミングを取り上げた。形式論理で書かれた仕様からプログラムを合成する試みは古くからある。本課題では、自動プログラミングの強力な支援ツールとなりうる、1 階述語論理の定理証明器の作成を目指し、

- 構成的数学における実現性解釈 (realizability interpretation) に基づく関数型プログラムの生成
- 時制論理に基づくプロトコル生成
- Knuth-Bendix 完備化技術に基づくコンカレントプロセスのインタフェース設計

の三点を試みた。

(3) 高次推論・学習

定理証明技術自身は、基本的な証明機構という点では、飽和した感がある。本課題では、演繹的アプローチに留まらず、類推、帰納、およびプログラム変換的アプローチをとることによって、この基本機構を拡張することを試みている。論理プログラムに関する機械学習や論理のメタ的使用がその際の主要な技術となる。

類推アプローチでは、人間が自然に行っている知的な推論を形式化し、模擬することを目指した。類推推論を実現することにより、情報が不完全であったり、演繹ステップが非常に長いために、通常の演繹システムでは解を得るすべがない場合でも、何らかの結果を得ることができるようになる。

帰納アプローチでは、帰納推論の計算量を考慮に入れ、新述語の発明および最小汎化 (least-general generalization) を用いた論理プログラムの学習理論を検討した。

プログラム変換アプローチでは、展開 / 畳込み変換操作を用いて、論理プログラムの新述語の生成を行った。

以下の節では、第五研究室における自動推論研究の三つの研究開発テーマについて、成果概要を述べる。

## 2 PIM 上の並列定理証明技術

本節では、Multi-PSI および PIM 上で走行する MGTP 定理証明器について述べる。また、MGTP の効率改善の

ために開発した、KL1 プログラミング技術ならびにアルゴリズムのエッセンスを紹介する。

## 2.1 並列モデル生成定理証明器 MGTP

並列定理証明の研究では、知的な知識情報処理システムを構築するための根幹技術の一つである、高機能、高並列の推論機構の実現を目指している。我々は、この並列定理証明研究プロジェクトを約二年半前に開始した。本研究の当面の目標は、KL1 言語並びに PIMOS オペレーティングシステムの特長を活用して、並列推論マシン PIM 上に高速な並列自動演繹システムを構築することである。我々は、このシステムを知的データベース、自然言語処理、自動プログラミング等の様々な分野に適用することを目指している。

本研究にあたり、以下の項目を主要課題として設定した。

- 高速な 1 階述語論理定理証明器の開発
  KL1 プログラミング技術を用いた定理証明器の開発にあたり、その第一歩として、モデル生成法を採用し、これに基づく定理証明器 MGTP を作成した。その理由は、後述するように、モデル生成法が KL1 プログラミングに極めて適合していたからである。MGTP の開発経験に基づき、証明トレーサや定理証明器の動的振舞いを可視化するビジュアライザ等の有用な機能を提供する、定理証明器開発支援システムも合わせて作成した。

- 応用の開発
  1 階述語論理の定理証明器は AI 領域のほとんどの問題を扱える潜在能力があるが、Prolog ほどには普及していない。その理由の一つは、証明手続きの効率が悪いためであり、もう一つは、有用な応用が欠如していたためである。しかしながら、形式的仕様からのプログラム生成 [Hasegawa *et al.*, 1990]、回路検証、法的推論 [Nitta *et al.*, 1992] 等の研究を通して、1 階述語論理の定理証明器は種々の分野で有効に使用できるという見通しがたってきた。現在、この確認のため、自動プログラム生成システム、電話交換機の仕様記述システム、アブダクションシステム、非単調推論システム、等を MGTP 上に開発中である。

- KL1 プログラミング技術の開発
  定理証明器の開発を通して KL1 プログラミング技術を蓄積することも本研究の目的の一つであった。MGTP 作成過程で得られた KL1 プログラミング技術の代表的なものとしては、与えられた 1 階述語論理の問題節を KL1 節に変換する KL1 コンパイル技術、KL1 の論理変数およびストリームデータ型を活用した MGTP 並列化手法、等がある。KL1 コンパイル技術によって、基底問題に対しては KL1 の高速な照合 (matching) 機能が利用可能となり、単一 PE でも高性能を達成した。

- KL1 メタプログラミング技術の開発
  KL1 言語で、メタプログラミングをいかに実現するかは、定理証明器開発の際の重要な課題となる。我々は、メタライブラリと呼ぶプログラマのための基本的なメタプログラミングツールを開発した。本ライブラリは、同一化 (unification)、照合、変数管理等のルーチンを提供する KL1 プログラム群である。

### 2.1.1 KL1 言語による定理証明器

最近の論理プログラミングの進展に伴い、1 階述語論理の定理証明器が効率良く実現できるようになってきた。典型的な例としては、Stickel [Stickel 1988] による PTTP、Manthey と Bry [Manthey and Bry 1988] による SATCHMO がある。

PTTP はモデル除去法 (model elimination method) に基づく後ろ向き推論定理証明器であり、完全性、健全性を損なうことなく、任意の 1 階の論理式をホーン節形式で扱うことができる。

一方、SATCHMO はモデル生成法に基づく前向き推論定理証明器である。SATCHMO は本質的には超導出定理証明器であり、値域限定 (range-restricted) [1] という条件を節に課すことにより、基底アトムのみを導出することができる。本定理証明器は基本的には前向き推論を行うが、ホーン節に対しては、Prolog を用いて後ろ向き推論が行えるようにしている。

これらのシステムの主な利点は、入力節が Prolog 節で表現され、演繹の大部分が通常の Prolog 実行の形で行えることにある。

上述のことを踏まえて、オブジェクトレベルの変数を KL1 でどう扱うかについて、次の二つの方法を検討した。

(1) KL1 の基底項として表現する。

(2) KL1 変数で表現する。

最初の方法は、メタプログラミングの正道ともいうべきもので、オブジェクトレベルとメタレベルが厳密に区別され、明確な意味づけが可能となる。しかし、この方法は、ユーザに同一化、置換、変数の名前換え、その他変数及び環境に対する諸々の操作を記述することを強いる。これらの操作コードは主プログラムに比較してもかなりの量であり、複雑なものとなって、オーバヘッドを大きくしてしまう。

二番目の方法においては、変数及び環境に対する操作は、ユーザの記述したコードではなく、システムの側で行われる。これによって、メタプログラマは煩雑なルーチンを書かずに済み、実行効率も高いものとなる。さらに、Prolog では、必要とあらば var 述語を用いて、出現検査 (occurrence check) を記述し、同一化を健全なものにすることも

[1] 節中に現れるいずれの変数も、その前件に少なくとも一箇所出現しているとき、その節は値域制限を満たすという。

可能である。厳密に言うと、この方法は常に採用して構わないというわけではない。オブジェクトレベルとメタレベルの区別が曖昧になるからである。しかし、プログラムの複雑さや効率の観点からは、本方法で得られる実益はかなりのものがある。

しかし、KL1 においては、Prolog のように二番目の方法がいつも可能であるとは限らない。KL1 の意味論によれば、var のようなものは許されないからである。さらに、KL1 の組み込み同一化は Prolog のそれとは全く異なるものである。即ち、KL1 節のガード部における同一化は変数の束縛に関して一方向に制限されており、ボディ部においては、同一化の失敗は後戻り (backtrack) 可能な単なる失敗ではなく、意味論上のエラーあるいは例外として扱われる。こういった制約はあるものの、基底モデルを扱う定理証明器を実装する際には、我々は二番目の方法をとることができ、KL1 の機能を最大限に利用することができる。

上述の議論を勘案し、基底版と非基底版の二種類の定理証明器を開発することを決め、扱う問題領域に応じてこれらを効果的に使いわけることができるようにした。

基底版 MGTP/G では、データベース処理や自然言語処理など多様な分野の大抵の問題が含まれる、有限領域の問題を支援することを目指している。

基底モデルの場合は、問題節が値域限定条件を満たしていれば、モデル生成定理証明器は照合のみを行えば良く、同一化は必要としない。

これは KL1 の頭部同一化 (head unification) を使うだけで十分であることを示唆しており、オブジェクトレベルの変数を表現するにあたって、我々は KL1 変数で表現する方法を採用することができる。

MGTP/G で開発した KL1 プログラミング技術の要点は次の通りである (詳細は次節で述べる)。

- 第一に、与えられた問題節を対応する KL1 節集合に変換する。この変換は極めて簡単である。
- 第二に、節の前件リテラルとモデル要素との連言照合を、KL1 の頭部同一化を用いて行う。
- 第三に、頭部同一化の際、KL1 節を呼び出すことによって新変数を自動的に確保し、使用されるリテラルに対して異なるインスタンスが得られるようにする。

一方、非基底版 MGTP/N は無限領域の支援を目指している。この領域の典型的な例としては、群論や含意計算 (implicational calculus) 等の数学的定理があげられる。

非基底モデルの場合は、出現検査機能付きの同一化が必要であり、KL1 基底項 でオブジェクトレベルの変数を表現する方法をとらざるをえない。しかし、必ずしも KL1 のプロセスを用いて、変数と束縛値の対を管理する必要はない。通常の言語処理系でよく用いられているように、KL1 でもベクタ機構を用いることができるからである。このベクタを用いた実装法は、プロセスによる実装法よりも数百倍速いという実験結果が得られている。

しかし、この場合は、MGTP/G で開発されたプログラミング技術を使用することができず、従来通りの方法を用いなければならない。すなわち、与えられた問題節を KL1 節に変換して実行するかわりに、それを解釈することになる。

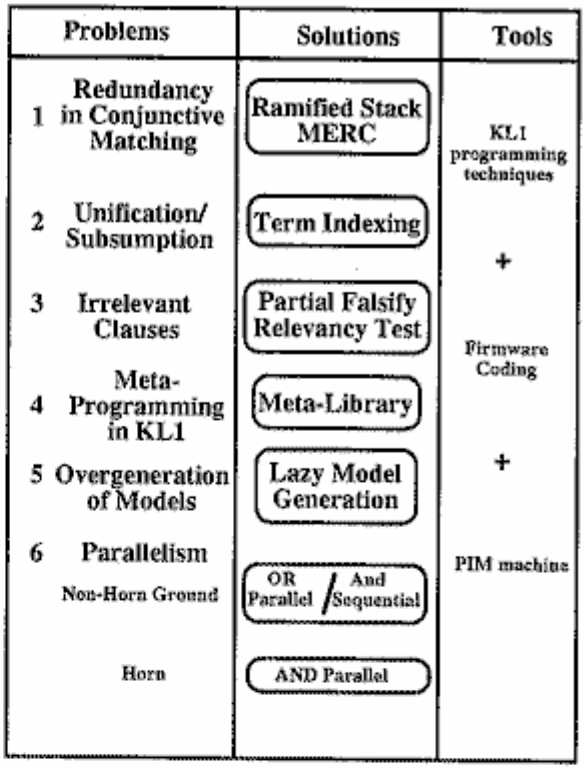


図 2: 主な問題点と技術的解決策

### 2.1.2 効率改善のための主要技術

モデル生成定理証明器の効率改善法を探究する過程で、いくつかのプログラミング技術が開発された。図 2 に性能を劣化させる要因となる問題点と我々がとった解決策の一覧を示す。以下の節では、これらについて要点を簡単に述べる。

#### 連言照合における冗長性

モデル生成定理証明器の性能を向上させるには、連言照合時の冗長計算をできるだけ回避することが必須となる。前件部に二つのリテラルを持つ節 $C$ を考える。証明過程のあるステージにおけるモデル候補を $M$ としよう。節の前件リテラルとモデル要素との連言照合を行うためには、$M$ から可能な要素対をすべて選ぶ必要がある。

今、節の後件にあって $M$ には含まれないようなモデル拡張アトム $\Delta$ で $M$ を拡張するとしよう。そうすると、

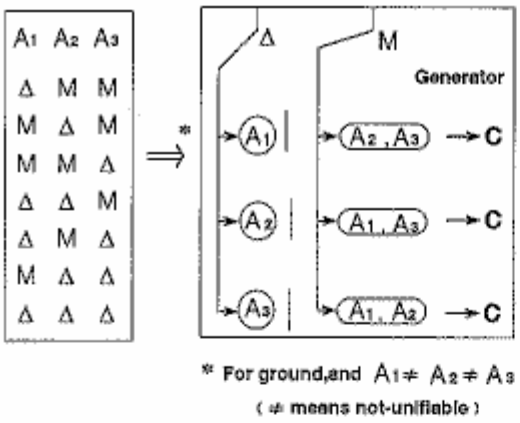


図 3: MERC 法

次のステージでは $M \cup \Delta$ の中から要素対を選ぶことになる。その要素対の総数は、

$$(M \cup \Delta)^2 = M \times M \cup M \times \Delta \cup \Delta \times M \cup \Delta \times \Delta.$$

となる。ところが、これを単純に行うと冗長を生じることになる。というのは、$M \times M$ 個の対は、この前のステージで既に選ばれているからである。したがって、少なくとも $\Delta$ を一つ含む対のみを選ばなければならない。

(1) RAMS 法

RAMS (ramified stack) 法 [Fujita and Hasegawa 1990] の要点は、連言照合で得られる中間結果、即ちリテラルとモデル要素との照合結果、をリテラルインスタンス・スタックに保持することである。RAMS アルゴリズムは、冗長性を含むことなく、$\Delta$ と $M$ から取り出されたモデル要素の重複組合せを計算する。

非ホーン節の場合、リテラルインスタンス・スタックは、場合分けが起きる度に分岐し、木状に成長する。分岐スタック (RAMS) という名称はこれにちなんだものである。

RAMS 法は、連言照合時の冗長性を回避するのみならず、共通モデルの共有を可能にする。しかし、欠点が一つあり、中間のリテラルインスタンスを保持しなければならないため、必要メモリ量が増大する傾向がある。

(2) MERC 法

MERC 法 (multi-entry repeated combination) [Hasegawa 1991] は上述の RAMS 法の問題の解決を図ったものである。

MERC 法の概要を図 3 に示す。3 個の前件リテラルを持つ節 $A_1, A_2, A_3 \rightarrow C$ の場合、7 個の節を用意する。各節は $\Delta$ と $M$ の重複組合せの一つのパタンに対応しており、そのパタンに従って、連言照合を実行する。例えば、$[M, \Delta, M]$ という組合せパタンに対応する節は、まず、リテラル $A_2$ と $\Delta$ の照合を行う。それが成功すると、残りのリテラル $A_1$ および $A_3$ に対して、$M$ から取り出されたモデル要素との照合を行う。ここで、各組合せパタンは少なくとも一つ $\Delta$ を含み、$[M, M, M]$ パタンは除外されていることに注意されたい。実際の実装では、この組合せパタンをテンプレートとして持ち、定理証明器側でこれを解釈することにより、連言照合を実行する $\Delta$-M 法を採っている。$\Delta$-M 法では、組合せパタンに応じて節を複数本用意する必要はなく、1 本で済む。

RAMS 法と MERC 法および $\Delta$-M 法の間では、いくつかのトレードオフ点が存在する。RAMS 法では、リテラル $A_i$ とモデル要素の照合結果の内、成功したものはすべて記憶し、同じリテラルとモデル要素どうしで再照合しないようにする必要がある。一方、MERC 法も $\Delta$-M 法も連言照合の各断片に関する情報を記憶するためのメモリは必要としない。しかしながら、いずれの方法もある種の冗長計算を含んでいる。例えば、$[M, \Delta, \Delta]$ 、$[M, \Delta, M]$ というパタンに対する連言照合計算において、共通の部分パタン $[M, \Delta]$ は再計算されてしまう。しかし RAMS 法では、この種の冗長性をも除去することができる。

### 同一化 / 包摂テストの高速化

MGTP においては、全計算時間の内、ほとんどが同一化と包摂テスト (subsumption test) で消費される。項インデキシングは、これらの処理を必要な対象に対してのみ、1 回の操作で複数の同一化 / 包摂テストが行えるように改善するための、古典的手法ではあるが、唯一の方法である。

図 4 に、ある典型的な問題に関する MGTP/G 上での項メモリ (term memory) 効果を示す。

### 選言節の最適使用

Loveland ら [Wilson and Loveland 1989] は、基底版モデル生成定理証明器において、選言節 (disjunctive clause) を適切に使用しないと、不要な場合分けを生じ、無駄な探索を行ってしまうことを指摘した。MGTP でも、勝手に選言節を選ぶと、冗長なモデルの組合せ的爆発を引き起こす危険がある。人工的だが示唆に富んだ例を図 5 に示す。

この例題を扱うため、MGTP/G では二つの方法を採った。一つは、"upside-down meta-interpretation" (UDMI) [Stickel 1991] を MGTP/G に導入する方法である。これを使用することにより、図 5の問題は図 6 に示すようなボトムアップ・ルールにコンパイルされる。

この変換後の節は値域限定の条件に反しているが、Prolog の同一化を用いて安全に実行できることに注意。

● Execution Time and
No. of Reductions(Instruction Unit of KL1)

| | |
|---|---|
| (1) With TM. | 784197 red / 14944 msec |
| (2) Without TM. | 1629421 red / 28174 msec |

● The Most Dominant Operations

(1) With TM (The eight best predicates are of TM operations)

| Predicate | Red |
|---|---|
| assocs / 4 | 466990 |
| term Type / 2 | 43791 |
| termNodes / 4 | 39771 |
| termNodes1 / 4 | 20338 |
| bindConstant / 5 | 20304 |
| Others | 193003 |

0 500,000

(2) Without TM(member predicate takes the first rank)

| Predicate | Red |
|---|---|
| member / 3 | 1214048 |
| c / 6 | 178464 |
| satisfyLiteral / 9 | 133255 |
| satisfyLiteral / 7 | 46655 |
| do / 7 | 27063 |
| Others | 29936 |

0 500,000 1,000,000

図 4: 項メモリによる速度向上

もう一つは、推論過程で得られた正の選言節を保持する方法である。共通のモデル候補である単位モデル集合に、選言モデル要素中のリテラルを一つ付加したものを、リテラル個数分用意し、それぞれに対して独立に棄却テストを行う。もしテストに成功すればそのリテラルを除去する。選言モデルは長さ順にソートしてもよい。この方法は、Nクィーン問題や有限代数の数え上げ問題に対してかなりの速度向上を達成している。

### KL1 におけるメタプログラミング

効率の良い定理証明器を作成するには、同一化や照合プログラムといったメタプログラムの高速化が必要となる。実際に証明過程の大部分は、このようなメタプログラムの実行で占められる。高速な定理証明器には、高速なメタプログラムが必須となる。

```
false :- p(c,X,Y)                    (1)
false :- q(X,c,Y)                    (2)
false :- r(X,Y,c)                    (3)
s(a)                                 (4)
s(b)                                 (5)
s(c)                                 (6)
s(X),s(Y),s(Z) ->
p(X,Y,Z);q(X,Y,Z);r(X,Y,Z)          (7)
```

図 5: 適合性テスト (relevancy test) の例題

```
true -> gp(c,X,Y).                   (1-1)
gp(c,X,Y),p(c,X,Y) -> false.         (1-2)
true -> gq(X,c,Y).                   (2-1)
gq(X,c,Y),q(X,c,Y) -> false.         (2-2)
true -> gr(X,Y,c).                   (3-1)
gr(X,Y,c),r(X,Y,c) -> false.         (3-2)
true -> s(a)                         (4)
true -> s(b)                         (5)
true -> s(c)                         (6)
s(X),s(Y),s(Z),
gp(X,Y,Z),gq(X,Y,Z),gr(X,Y,Z) ->
p(X,Y,Z);q(X,Y,Z);r(X,Y,Z)           (7)
```

図 6: UDMI のコンパイルコード

PTTP や SATCHMO のような Prolog のプログラミング技法を生かした定理証明器においては、オブジェクトレベルの変数 (与えられた節集合に現れる変数) は、Prolog 変数で直接表現される。この表現法を採用すると、変数および変数環境に関する多くの操作を Prolog の組み込みの機能にまかせることができる。つまり Prolog の提供する高速な機能を使うことにより、高効率を得ることができる。さらに、必要となれば Prolog 述語 var 利用することによって Prolog の同一化に出現検査機能を付加してこれを健全にするようなことも容易にできる。

しかし、残念ながら KL1 ではこのような技法を使うことはできない。この理由には以下のことがあげられる。

1) KL1 においては、var のような述語に Prolog と同じ意味を持たせることはできない。

2) KL1 の組み込み同一化の機能は、Prolog とは異なる。ガード部の同一化は一方向の同一化であるし、

3) ボディ部の同一化の失敗は、後戻りではなくプログラムエラーもしくは例外事象を引き起こす。

したがって、オブジェクトレベルの変数は KL1 変数で表現することは一般的にはできない。そこでオブジェクトレベルの変数をある特殊な定数 (基底項) で表現するすることにした。これにともなって、同一化、置換や、変数およびその環境に関する多くの操作プログラムを記述する必要が生じる。これらのプログラムは、主プログラムに比して巨大で複雑になる可能性があり、またプログラミングが面倒臭く、そのコストは無視できない。

プログラマーのこのような負荷を軽減するために、メタライブラリを開発した [Koshimura *et al.*, 1990]。これは KL1 のメタプログラミングを容易にするためのプログラム集であり、出現検査機能付き同一化や変数管理プログラ

ムを含んでいる。メタライブラリの速度は比較的速い。例えば同一化プログラムは、KL1 の組み込みとして提供されているボディ部の同一化に比べて 0.25 ～ 1.25 倍の速度である。

メタライブラリの主な機能は以下の通りである。

```
unify(X,Y, Env,^NewEnv)
unify_oc(X,Y, Env,^NewEnv)
match(Pattern,Terget, Env,^NewEnv)
oneway_unify(Pattern,Terget, Env,^NewEnv)
copy_term(X,^NewX, Env,^NewEnv)
shallow(X,Env, ^NewNnv)
freeze(X,^FrozenX, Env)
melt(X,^MeltedX, Env)
create_env(^Env, Size)
fresh_var(Env, ^VarAndNewEnv)
equal(X,Y, Env, ^YN)
is_type(X,Env, ^Type)
unbound(X,Env, ^YN)
database(RequestStream)
get_object(KL1Term, ^Object)
get_kl1_term(Object, ^KL1Term)
```

### モデルの過剰生成

モデル生成に基づく定理証明器の効率に関するより重要な問題は、証明過程で要する計算量およびメモリ量の総量を削減することである。

モデル生成定理証明器は次の三つの操作を実行しなければならない。

- モデル拡張アトム $\Delta$ の集合とモデル候補集合 $M$ を用いて、モデル拡張規則を与えられた生成節に適用し、新しいモデル要素を生成する (モデル拡張)。
- 生成されたアトムに対し包摂テストを施し、それが既に生成されたアトムの集合、通常は現モデル候補、に包摂されるか否かを調べる。
- モデル棄却規則をテスタ節に適用し、未包摂のモデル要素が偽 (false) を生じるかどうかを調べる (棄却テスト)。

モデル生成法の問題は、生成されるアトムの総数、および時間 / 空間量の観点から計算コストが膨大になることである。この問題は、Lukasiewicz 問題のように深い推論 (長い証明) を要する困難な問題を扱う場合により深刻になる。

この問題を解決するためには、証明過程を生成・テストの過程とみなし、テストが必要としたときのみ生成を行うことが重要である。

そこで我々は、証明を得るのに要する計算量およびメモリ量を削減することができる遅延モデル生成アルゴリズム [Hasegawa *et al.*, 1992] を提案した。

表 1: 複雑度の比較 (単位テスタ節の場合)

| アルゴリズム | T | S | G | M |
|---|---|---|---|---|
| 基本 | $\rho m^2$ | $\mu\rho^2 m^{4\alpha}$ | $\rho^2 m^4$ | $\rho^3 m^4$ |
| 全テスト / 遅延 | $\rho m^2$ | $\mu m^{2\alpha}$ | $m^2$ | $\rho m^2$ |
| 遅延 & 先読み | $m^2$ | $(\mu/\rho)m^\alpha$ | $m/\rho$ | $m$ |

† $m$ は基本アルゴリズムにおいて偽が検出された時点のモデル候補中の要素数である。

‡ $\rho$ は生成されたアトムの生存率、$\mu$ は連言照合の成功率 ($\rho \leq \mu$)、$\alpha(1 \leq \alpha \leq 2)$ は包摂テストの効率を表す指標である。

表 1 にモデル生成法のいくつかのアルゴリズム[2]の複雑度を比較した結果を示す。

簡単な解析から、モデル拡張および包摂テストの複雑度は、遅延制御なしのアルゴリズムでは $O(m^4)$ であるが、遅延制御ありでは $O(m)$ まで減少するという結果が得られている。詳細は、[Hasegawa *et al.*, 1992] を参照されたい。

### MGTP の並列化

MGTP 定理証明器の証明過程には、主に次の三つの並列性がある。

- 証明における複数のモデル候補。
- 複数の節の中のどれにモデル生成規則を適用するか。
- 連言照合における複数のリテラル。

モデル生成法を用いる主な目的が解となるモデルを見つけることにあるとしよう。このとき与えられた問題に対して、選択の対象となる解もしくはモデルが複数あるかも知れない。これは OR 並列性とみなせ、複数の解を同時に探索することができる。いいかえると、複数のモデル候補や複数の節があるとき OR 並列性を引き出すことができる。

一方、AND 並列性は前件部の複数のリテラルから引き出せる。これは、一つの節中のすべてのリテラルが一つの解に寄与することによる。このとき、節中の共有変数は矛盾しない値を取らなければならない。

非ホーン基底の場合は、複数の解を同時に探索する OR 並列性を利用することによって十分な並列効果が得られる。しかしながら、ホーン節のみを対象とする場合にはこのような OR 並列性はないので、AND 並列性を利用しなけ

[2]OTTER [McCune 1990] でとられている基本アルゴリズムは、生成されたアトムに対する棄却テストが完了する以前に、新たなアトムを集合を生成する。全テストアルゴリズムは次のサイクルに入る前に棄却テストを完了するが、基本アルゴリズム同様、一度に新たなアトム集合を生成する。先読みは、全テストおよび遅延アルゴリズムよりも広い空間をテストするための最適化手法である。

ればならない。これには連言照合と包摂テストの並列化がある。非ホーン非基底の場合の並列化は、今後の課題である。

(1) MGTP/G の並列化

現在のところ、基底版 MGTP/G では OR 並列性のみを引き出すことを考えている。

(a) プロセッサ割付

プロセッサ割付法としては、「有界 OR」並列を実現した。これは、証明プロセスの OR 並列的な起動を抑制して、資源が限られた状況に適合させる方法である。

このような方法を実現する簡単な方法の一つがシンプル割付である。この方法では 1 台のマスタープロセッサが空モデルから始めてモデル候補を拡張し、モデル候補の数が利用できるプロセッサ数を越えると、続きのタスクをスレーブプロセッサに分配する。各スレーブプロセッサは割り当てられた枝を自分で探索し、他のプロセッサにさらにタスクを分配するようなことはない。このシンプル割付法は、通信コストを最小化できるという点で簡便な方法である。

(b) Multi-PSI での性能測定

並列実行による速度向上効果を見るため、N クイーン問題を用いた。問題は次のような節集合で表現される。

$$
\begin{aligned}
C_1 :&\quad true \rightarrow p(1,1); p(1,2); \ldots; p(1,n).\\
&\quad \ldots\\
C_n :&\quad true \rightarrow p(n,1); p(n,2); \ldots; p(n,n).\\
C_{n+1} :&\quad p(X_1,Y_1), p(X_2,Y_2),\\
&\qquad unsafe(X_1,Y_1,X_2,Y_2)\\
&\qquad\quad \rightarrow false.
\end{aligned}
$$

最初の N 個の節は N×N のチェス盤にクイーンを置くあらゆる可能性を表している。最後の節はクイーンの対が満たすべき制約を表す。この問題が解けるのは、この節集合に対してあるモデル (一つの解) か全てのモデル (すべての解) が得られたときである。

シンプル割付法の効果を Multi-PSI 上の MGTP/G を用いて測定した。時間は全てのモデルを得るのに要した時間である。16 台のプロセッサを用いたときに得られた台数効果を図 7に示す。10 クイーン問題では、プロセッサ数の増加に対してほぼ線形に速度が向上する。4 クイーン問題だけは速度向上効果がみられない。これはおそらく問題の規模が小さいので、一定量の解釈オーバヘッドが証明自身のコストより大きくなるからであろう。

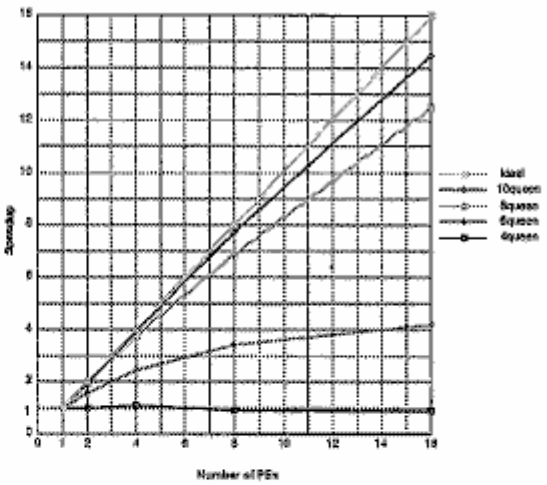


図 7: Multi-PSI 上の MGTP/G の台数効果 (N クィーン)

(2) MGTP/N の並列化

非基底版 MGTP/N については、ホーン問題に対して AND 並列性を引き出した。

モデル生成に基づく定理証明器で並列化を図るにあたって、以下のことに留意した。

1) PE 台数に応じて証明が変わる証明変動方式と変わらない証明不変方式。
2) モデル共有 (分散メモリアーキテクチャでは各 PE が同じモデルをコピーして持つ) 方式とモデル分散方式。
3) マスターありとマスターなし。

証明変動方式では、超線形台数効果が得られる可能性があるが、使用した PE 台数に見合う台数効果が常に得られるとは限らない。一方証明不変方式では、使用した PE 台数に見合う台数効果が期待できるが、それは高々線形である。

モデル共有方式の利点は、連言照合や包摂テストといった最もコストのかかる計算を最小の PE 間通信で行なえることである。一方モデル分散では、メモリ・スケーラビリティを得ることができるという効果がある。しかしながら、生成されたアトムは包摂テストのために全 PE を一順しなければならないため、PE 台数が増えるにしたがって通信が増大してしまう。

マスター・スレーブ方式では、逐次版 MGTP/N をスレーブ PE 上に置き、単にそれらとマスタ PE とつなぐことによって簡単に並列システムを構築することができる。しかし、マスタの負荷をできるだけ小さくする工夫が必要となる。リング結合のようなマスターなしの場合、負荷分散が図れるとともにパイプライン効果を得ることができるが、各 PE を協調的に制御することは難しくなる。

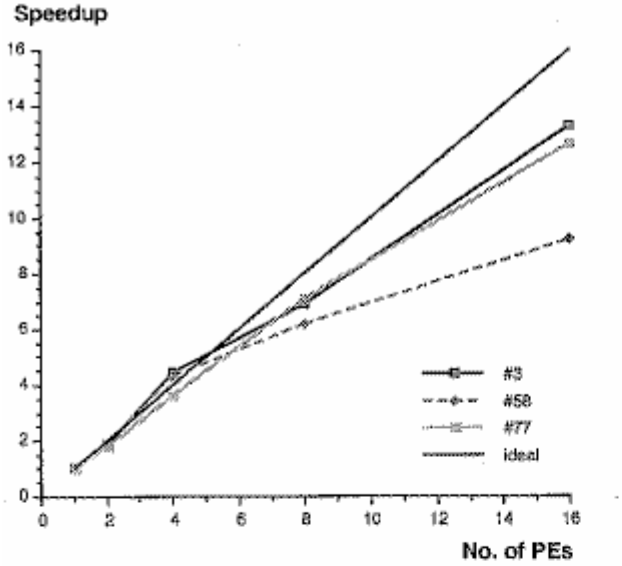


図 8: 速度向上比

並列定理証明器を開発する際の我々の方針は、得られた速度向上が並列化によるものなのか、戦略による探索空間の刈り込みによるものなのか、を明確に区別することである。証明変動方式においては、PE 台数を変えることは一種の賭けに過ぎない。この場合、速く解が得られるかどうかは実行させてみないと分からない。

以上のことを踏まえて、マスター・スレーブ型の遅延モデル生成に基づく証明不変方式の MGTP/N を実現した。ここでは、生成プロセスと包摂テストプロセスは要求駆動的に動作し、棄却テストプロセスはデータ駆動的に動作する。このシステムの特徴は以下の通り。

1) 証明不変方式なので、PE 台数に見合った速度向上が得られる。
2) KL1 の同期機構により包摂テストの逐次性を最小にすることができる。
3) スレーブプロセスは、暇になるとマスタープロセスからタスクを受けとれ、また各タスクの粒度はほぼ均等なので、良い負荷分散が得られる。
4) KL1 のストリーム通信により、要求駆動制御を容易にかつ効率良く実現することができる。

要求駆動制御により、不必要なモデル拡張や包摂テストを抑制でき、さらに高い稼働率を得ることができる。高稼働率の達成は線形台数効果得るためには必須である。

図 8 は [McCune and Wos 1991] にある分離の規則に関する問題 (condensed detachment problems) #3、

表 2: MGTP/N の性能 (Th 5 and Th 7)

| Problem | | 16 PEs | 64 PEs |
|---|---|---|---|
| Th5 | Time (sec) | 41725.98 | 11056.12 |
| | Reductions | 38070940 | 40759689 |
| | KRPS/PE | 57.03 | 57.60 |
| | Speedup | 1.00 | 3.77 |
| Th7 | Time (sec) | 48629.93 | 13514.47 |
| | Reductions | 31281211 | 37407531 |
| | KRPS/PE | 40.20 | 43.25 |
| | Speedup | 1.00 | 3.60 |

#58、#77 を解いたときの台数効果を示したものである。計測は 16 PE 版 Multi-PSI を用いて行なった。要した時間は 16 台でそれぞれ 218、12、37 秒である。図に示す通り、16 PE まででは速度向上の飽和現象は見られなかった。また、時間を要する問題ほど台数効果が得られているのが分かる。

表 2 は [Overbeek 1990] にある定理 5 と 定理 7 を 64 PE 版 Multi-PSI を用いて解いたときのものである。これらの定理も分離の規則に関する問題である。

図 8 および表 2 の問題を解くに当たって、ソーティングといったヒューリスティクスは用いずに項サイズによる足切りとトートロジー除去のみを用いた。同一化プログラムは KL1 ですべて書かれており、この速度は SUN/3 や SPARC 上の C によるものに比べて 30 倍から 100 倍低速である。

表 2 において、1 台当たりの稼働率は 64 PE の方が 16 PE に比べて若干速いのに注意してもらいたい。これらの結果や他の例題による結果からも、ほとんど線形な台数効果を得られることが判明した。

最近利用可能となった 128 台版の PIM/m で定理 5 を解いたところ、127 PE で 2870.62 秒、約 440 億リダクション、稼働率 120 KRPS/PE という結果が得られた。PIM/m の CPU の速度は Multi-PSI のそれに比べると 2 倍強であることを考慮に入れると、少なくとも 128 PE までは線形台数効果が得られることが分かる。

## 2.2 リフレクションおよび並列メタプログラミングシステム

リフレクションとは、ある計算システムの現在の内部状態を感知したり、動的にそれを変更したりする能力のことである。ここで、我々が興味を持っているのは、最初に [Smith 1984] によって提案された、計算リフレクションである。我々は、論理型言語におけるメタレベル計算や計算リフレクションについて、様々な方向から研究を進めてきた。

まず、基礎的な研究として，リフレクティブな逐次的論理型言語 *R-Prolog** の提案を行った [Sugano 1990]。 R-Prolog* は，いくつかのコーディングオペレータを用意することによって，言語の構文要素と意味要素を言語自身で記述し、操作することを可能にしている。また，計算リフレクションの概念を言語に組み入れることによって、自身の計算状態を計算システムが動的に認識し、変更することを可能にしている。結果として、 Prolog のいくつかの論理外述語を、統一的な枠組のもとで再定義することができる。次にリフレクティブな並列論理型言語として、RGHC (Reflective Guarded Horn Clauses) を提案した [Tanaka and Matono 1992]。この RGHC では、リフレクティブ述語を用いて、メタレベルの無限階層を持つリフレクティブ・タワーを動的に構築・消去することが可能である。RGHC の試作システムは既に実装がなされており、RGHC の特色は、リフレクションの実装の容易さにあると思われる。また、本システムでは、メタレベルの実行がオブジェクトレベルの実行と同じ速度で行なわれる。

また、我々は、対象レベルの計算とメタレベルの計算を並行に実行することの可能な「分散リフレクション」の形式化を試みた。そこでは、ローカルな環境を共有するゴールをグループ化することによって、リフレクションのスコープを定めている。このモデルはまた、制約伝播の遅延を記述することのできるモデルでもある。

我々は、メタプログラミングやリフレクションの応用として、様々な応用システムを構築した。すなわち、実験的プログラミングシステム ExReps [Tanaka 1991]、プロセス指向 GHC デバッガ [Maeda *et al.*, 1990, Maeda 1992]、性能管理シェル [Kohda and Maeda 1991a, Kohda and Maeda 1991b] などである。

ExReps は、並列論理型言語のための実験的なプログラミング環境であり、ユーザは、それを用いてプログラムを入力したり、ゴールを実行したりすることができる。システムは仮想マシン層と実行システム層から構成され、双方ともにメタプログラミング技法を用いて構成されている。これらの層の中で、様々なリフレクティブな操作が実装されている。

プロセス指向デバッガは、GHC プログラムで頻繁に用いられる仮想的なプロセスとストリームをサポートし、抽象度の高いイメージのもとで実行をトレースしたり、テストするための機能を提供する。例えば、ゴール列をテキストで書き下した従来の実行トレースに代わり、プロセス間のデータ依存関係とストリームの連結関係がグラフィカルに表示される。また、ゴール単位での実行制御に代わり、プロセスの振る舞いを決定するデータに対する操作、すなわちストリーム上のデータの流れの遮断 / 復旧、データの修正を可能としている。

性能管理シェルは、負荷分散戦略データベースを管理する。ユーザジョブが投入されると、現時点における最適戦略と数個の実験的な戦略が負荷分散戦略データベースから選ばれる。次いで、最適戦略タスクと実験的戦略タスクの実行が開始される。タスクの実行が終了した時点で、性能管理シェルは戦略どうしの相対性能を比較し、次回の最適戦略を決定する。

## 3 自動推論の応用

自動推論システムは、論理プログラミングやプログラミングの形式的アプローチと結び付けることにより、応用範囲がより広がる。まず我々は、MGTP を拡張することによって様相論理を扱えるようにした。この拡張を皮切りに MGTP を利用した様々な応用分野の研究を行なった。AI システムのアブダクションや失敗による否定 (negation as failure) を含む論理プログラムの研究がそれである。これらは、故障診断や法的推論のように実際的応用プログラムに広く結びついている。我々はまた、古典的研究テーマであるプログラム生成に関する研究も行なった。これは、形式的論理系の記述からプログラム特に並列プログラムを生成するものである。

自動推論システムの研究開発は推論自体の研究開発に留まらず、ここで述べる各種応用の基盤技術となるべく進めている。

### 3.1 MGTP 上の命題様相タブロ

MGTP とタブロ法の証明手続きは、操作的に良く似ている。タブロ法 [Smullyan 1968] の規則は、MGTP の入力節としてほとんどそのままに表現することができる。いいかえると、ホーン節が Prolog のプログラミング言語であるように、MGTP の入力節をタブロ法の実現言語としてみなすことができる。

MGTP は与えられた節集合のモデルをボトムアップに生成しようと試みる。モデル生成に成功した場合は、その節集合は充足可能であることが分かり、そうでない場合は、充足不能であることが判明する。

$$apply(MGTP, ASetOfClauses) = \begin{cases} satisfiable \\ unsatisfiable \end{cases}$$

一方タブロ法では、入力論理式のモデルを論理式の分解により生成しようと試みる。ここで MGTP を単なる推論システムとみなすことにより、MGTP 上に命題様相タブロ [Fitting 1983, Fitting 1988] を構築することができる。

$$apply(MGTP, TableauxProver(Formula)) = \begin{cases} satisfiable \\ unsatisfiable \end{cases}$$

タブロ法の閉条件 / 入力論理式 / 分解規則は、MGTP 入力節の負節 / 正節 / 混合節として自然に表現できる [Koshimura and Hasegawa 1991] 。

証明器の上に別種の証明器を作成する本方法では、証明の対象領域の性質 (推論規則) は入力節集合として表現され、推論機構は MGTP のプログラムとして表現される。これにより、対象領域の性質と推論機構の記述が明確に分離され、見通しのよい証明器の作成が可能となった。

## 3.2 アブダクションと非単調推論

不完全な知識に基づく推論のモデル化はAIの重要な研究テーマの一つである。この種の推論は、ある特有の状況に対処する知識システムを実現する高度な機構であるのみでなく、常識推論を扱うために必要な本質的な機構であると考えられる。常識推論は、人間も計算機も現実世界における日常的な状況に関連するすべての情報を持つことはできないという観点から極めて重要な機構である。完全な情報なしで推論を機能させるためには、閉世界仮説あるいはデフォルト推論等を適用することにより、いくつかの健全でない結論を導いたり、所与の理論を増大させたりする必要がある。このような推論は、情報が増大すれば結論が増大するとは限らない推論形態であり、非単調推論と呼ばれ、信念の翻意の可能性を前もって考慮しておかなければならない。

不完全な情報を用いた推論を以下では仮説推論 [Inoue 1988] として扱うことにする。仮説推論においては、理論と矛盾しない限り仮説を導入し、結論の集合が拡張される。仮説推論のうち、特に、観測事実を説明することのできる仮説集合を計算することをアブダクションと呼ぶ。この説明という概念は、診断、合成、設計および自然言語理解のような種々のAI問題の基本的な概念として有用である。我々は、これまでに、このような仮説推論の方法論を種々の観点から研究してきており、いくつかのアブダクションシステムや非単調推論システムを開発してきた。

本節では、MGTP [Fujita and Hasegawa 1991] 上に構築したいくつかの仮説推論システムについて述べる。これらのシステムは、MGTPが古典論理のみでなく、仮説推論をも扱えるようにしたものである。これらに共通する基本的な考えは、失敗による否定 (Negation as Failure) やアブダクションにおける説明の無矛盾性のような特有の性質を持った論理式の集合を ある様相演算子を持った論理式集合に変換することである。MGTPは選言によるモデル候補の分岐や矛盾するモデル候補の棄却を効率的に扱うが、これらを用いて、その特有の性質を生成検査問題に帰着させているのである。以下に、MGTPを用いて、失敗による否定を含む論理プログラムやアブダクションの計算を行うシステムについて述べる。

### 3.2.1 失敗による否定を含む論理プログラムと選言的データベース

論理プログラミングや演繹データベースの最近の研究により、拡張された論理プログラムの宣言的意味論が与えられている。これらのプログラムにおいては、失敗による否定を非単調な様相演算子として表現する方法が考察されており、特に、強力な知識表現ツールとして、失敗による否定 ($not$) と古典論理における否定 ($\neg$) の両方を含んでいるような論理プログラムあるいは演繹データベースとして用いることができる。また、拡張論理プログラムは、不完全な知識を用いた推論 [Gelfond and Lifschitz 1991]、デフォルト推論およびアブダクション [Inoue 1991a] と非常に深い関連がある。しかしながら、この拡張された論理プログラムのクラスは、プログラムの評価が局所的に行えないので、トップダウンアプローチだけによっては計算できない。例えば、一般論理プログラムの安定モデル (*Stable Model*) を計算する健全なトップダウン手続きは存在しない。そこで、失敗による否定を含んだ論理式の正当な評価のために、関数を含まないすべてのクラスの論理プログラムの解集合 (*Answer Set*) をボトムアップに計算する手続きが [Inoue *et al.*, 1992a] に示されている。これは、証明手続きが明らかにされていなかった拡張選言的データベース [Gelfond and Lifschitz 1991] に対しても適用できるものである。ボトムアップに $not\,P$ を評価するためには、$P$ が現時点でまだ証明されていなくても、それに続く推論で $P$ が証明されるかもしれないので、それをボトムアップ計算の不動点において評価するというものである。つまり、$not\,P$ を次のような従来とは全く異なった方法で扱うことを提案している。現在のモデル候補で、$not\,P$ を評価しなければならないとすると、そのモデル候補を

1. $P$ が成り立たないと仮定するモデル候補
2. $P$ が成り立つことを要求するモデル候補

の二つに分岐させる。各々の失敗による否定 $not\,P$ は、信念を表現する様相演算子を用いて、その正負リテラルに変換される。すなわち、"$P$ を信じていない" ($\neg \mathrm{K}P$ と記述する) または "$P$ を信じている" ($\mathrm{K}P$ と記述する) とに変換される。上記の議論に基づき、あらゆるクラスの論理プログラム (失敗による否定を含む) をMGTPによって計算できる正選言的プログラム (失敗による否定を含まない) に変換できる。以下に、一般論理プログラムの変換例を示す。$\Pi$ を次のような形式のルールから構成される一般論理プログラムとする。

$$A_l \leftarrow A_{l+1}, \ldots, A_m, not\,A_{m+1}, \ldots, not\,A_n\,. \quad (1)$$

ここに、$n \geq m \geq l \geq 0$、$1 \geq l \geq 0$、かつ $A_i$ はアトムである。ヘッドが空のルールは、統合的制約 (*Integrity Constraint*) と呼ばれ、形式 (1) において、$l = 0$ で表現される。形式 (1) の $\Pi$ 中の各ルールは、

$$A_{l+1}, \ldots, A_m, \rightarrow \neg \mathrm{K}A_{m+1}, \ldots, \neg \mathrm{K}A_n, A_l \,|\, \mathrm{K}A_{m+1} \,|\, \ldots \,|\, \mathrm{K}A_n \quad (2)$$

なるMGTPルールに変換される。形式 (2) の各MGTPルールについて、もし、モデル候補 $S'$ が $A_{l+1}, \ldots, A_m$ を充足するならば、$S'$ は、$n-m+l$ ($n \geq m \geq 0,\ 0 \leq l \leq 1$) のモデル候補に分岐する。また、信じられているリテラルと信じられていないリテラルに関して枝刈りを行うために、次のように表現される 統合的制約をMGTPのオブジェクトレベルのスキーマとして用いる。

$$\neg \mathrm{K}A, A \rightarrow . \quad (3)$$

$$\neg \mathrm{K}A, \mathrm{K}A \rightarrow . \quad (4)$$

ここに、$A$ は任意のアトムである。二つのスキーマ (3) および (4) と与えられた一般論理プログラム $\Pi$ の各ルール (1) をルール (2) で置き換えることにより得られる MGTP ルールから構成されるルールの集合を $tr(\Pi)$ と記述する。MGTP は、$\mathcal{M}(tr(\Pi))$ によって記述されるモデル候補の不動点を計算することになる。その計算は、MGTP の基本的な操作をそのまま用いることで実現される。不動点が計算された後で、$\mathcal{M}(tr(\Pi))$ 中の各モデル候補が信じられているアトムを含んでいる場合、そのようなアトムがすべて実際に導かれているかどうかを確認する必要がある。この検査は、$S'$ を $\mathcal{M}(tr(\Pi))$ 中の各要素、$A$ を基礎アトムとして、次の制約を用いることにより、極めて容易に行うことができる。

$$\text{もし } \mathsf{K}A \in S' \text{ ならば } A \in S'. \qquad (5)$$

MGTP を用いたこの計算は、次のような安定モデル意味論に関して、健全かつ完全である。すなわち、$\mathcal{M}(tr(\Pi))$ 中の条件 (5) を充足する任意のモデル候補を $S'$ とし、もし、$S$ が、そのモデル候補 $S'$ から K 演算子付きの全てのリテラルを削除することによって得られるアトムの集合の一つであるならば、かつそのときに限り、$S$ は $\Pi$ の解集合 (または 安定モデル) である。

例題: $\Pi$ を以下の 4 つのルールから構成される一般論理プログラムとする。

$$\begin{aligned} R &\leftarrow not\, R, \\ R &\leftarrow Q, \\ P &\leftarrow not\, Q, \\ Q &\leftarrow not\, P. \end{aligned}$$

これらのルールは次のような MGTP ルールに変換される。

$$\begin{aligned} &\rightarrow \neg\mathsf{K}R, R \mid \mathsf{K}R, \\ &Q \rightarrow R, \\ &\rightarrow \neg\mathsf{K}Q, P \mid \mathsf{K}Q, \\ &\rightarrow \neg\mathsf{K}P, Q \mid \mathsf{K}P. \end{aligned}$$

この例題において、最初の MGTP ルールはスキーマ (3) により選言の最初の式は刈られ、次のようにさらに簡略化される。

$$\rightarrow \mathsf{K}R.$$

ゆえに、そのルールは統合的制約

$$\leftarrow not\, R$$

と同じ効果を有している。この 統合的制約 はすべての 解集合 で、$R$ を含んでいなければならないことを意味している。すなわち、$R$ が導かれなければならない。

さて、$\mathcal{M}(tr(\Pi)) = \{S_1, S_2, S_3\}$ なることは容易にわかる。ここに、

$$\begin{aligned} S_1 &= \{\mathsf{K}R, \neg\mathsf{K}Q, P, \mathsf{K}P\}, \\ S_2 &= \{\mathsf{K}R, \mathsf{K}Q, \neg\mathsf{K}P, Q, R\}, \\ S_3 &= \{\mathsf{K}R, \mathsf{K}Q, \mathsf{K}P\} \end{aligned}$$

である。(5) を充足する唯一のモデル候補は $S_2$ である。これは、$\{Q, R\}$ が $\Pi$ の唯一の安定モデルであることを示している。$S_1$ は $\mathsf{K}R$ を含んでいるが $R$ を含んでいないから、$\{P\}$ は安定モデルでないことに注意されたい。

古典論理による否定、失敗による否定、選言を含むような拡張選言的データベースに対する同様な変換方法もまた [Inoue *et al.*, 1992a] に示されている。この変換方法によれば、MGTP は後戻りなしで、並列かつインクリメンタルにすべての解集合 を計算することができる。提案した方法は極めて簡単であり、計算量を正選言的プログラムの極小モデルを計算する計算量と比較して増加させるものではない。この手続きは、既に MGTP 上に実装されており、法的推論システムに適用されている。

### 3.2.2 アブダクション

従来、観測の説明を生成するためのアブダクションに関する多くの研究がある。以下で考察されるアブダクションの定義は、[Poole *et al.*, 1987] と同様である。いま、$\Sigma$ を論理式の集合、$\Gamma$ をリテラルの集合、$G$ を閉論理式とする。$E$ を $\Gamma$ の基礎例の集合 として、もし、

1. $\Sigma \cup E \models G$,
2. $\Sigma \cup E$ は無矛盾.

であるならば、$E$ は、$(\Sigma, \Gamma)$ からの $G$ の説明であるという。$(\Sigma, \Gamma)$ からの $G$ の説明を計算することは、理論 $\Sigma$ と矛盾しない $\Gamma$ からの仮説集合を導入することによって $G$ を証明することとみなせる。また、アブダクションは、結論発見問題 [Inoue 1991b] としても形式化できる。これは、あるリテラルに関して、それを証明するかわりに、仮定 (または Skip) することを許容するものである。すると、演繹の最後に空節が導かれるかわりに、それらの Skip されたリテラルだけが含まれている新しい定理が導かれる。このようにアブダクションは、特にトップダウン、後向き定理証明手続きによる演繹の拡張によって実現できる。例えば、Theorist [Poole *et al.*, 1987] や SOL- 導出 [Inoue 1991b] は、モデル消去定理証明手続き [Loveland 1978] の拡張である。

一方、アブダクションはボトムアップ、前向き推論手続きによっても実現できる。実際、前向き推論エンジンと ATMS [de Kleer 1986] とから構成された APRICOT/0 [Ohta and Inoue 1990] と呼ばれるアブダクションシステムが開発されている。ここで、ATMS は、繰り返されるサブゴールの証明と同一の結果を導く異なった仮説集合間の重複した証明とを避けるために、それらの推論結果を導いた根拠を保存するために用いられている。

アブダクションのための、これら二つの推論方法はお互いに相補的な利点と欠点を持っている。トップダウン推論は目標主導型であり、与えられたゴールの証明に無関係なサブゴールの証明は行わないが、先のような重複した計算

を行う可能性がある。また、ボトムアップ推論は、重複計算は行わないが、ゴールの証明と無関係な証明を行ってしまう可能性がある。これらは、ボトムアップ推論によってトップダウン推論を模擬する方法や、トップダウン推論にキャッシュを統合するような方法で各々の欠点を克服できることを示唆している。

例えば、upside-down meta-interpretation [Bry 1990] 法は、[Stickel 1991] によりアブダクションに適用され、[Ohta and Inoue 1992] により無矛盾性検査と統合されるように拡張された。

以下に概要を示すように、ボトムアップ型の定理証明器 MGTP を用いたいくつかの並列アブダクションシステムが既に開発されている。

1. **MGTP+ATMS** (図 9).

   これは、APRICOT/0 [Ohta and Inoue 1990] を並列化したものであり、無矛盾性検査のために ATMS を用いている。MGTP は、前向き推論エンジンとして用いられている。ATMS は、現時点で信じられているデータの集合 M を保存している。ここに、M は、それぞれが成り立つ極小の仮説集合によってラベル付けされた基礎アトムの集合である。また、この構成において、ボトムアップ推論にゴール情報を付加させる upside-down meta-interpretation を無矛盾性検査と統合されるように拡張している [Ohta and Inoue 1992]。

   この構成においては、ラベル計算を並列化した ATMS を用いている。しかしながら、MGTP と ATMS との間の通信路は 1 本であるため、MGTP が、しばしば、ATMS の計算結果を待つことになり、並列化の効果には限界がある。

2. **MGTP+MGTP** (図 10).

   これは、[Stickel 1991] によって提案された方法の並列化版である。さらに、無矛盾性検査に別の MGTP (*MGTP_2*) を用いている。このシステムにおいては、$\Gamma$ の要素 $H$ は $fact(H,\{H\})$ のように変換され、さらに、$\Sigma$ 中の各ホーン節

   $$A_1 \wedge \ldots \wedge A_n \supset C$$

   は

   $$fact(A_1,E_1),\ldots,fact(A_n,E_n) \rightarrow fact(C,\mathrm{cc}(\bigcup_{i=1}^{n} E_i))$$

   のような MGTP ルールに変換される。

   ここに、$E_i$ は、アトム $A_i$ が依存する仮説集合 $\Gamma$ の部分集合である。また、cc は次のように定義される関数である。

   $$\mathrm{cc}(E) = \begin{cases} E & \Sigma \cup E \text{ が無矛盾;} \\ \mathsf{nil} & \text{その他.} \end{cases}$$

   このシステムでは、現時点で信じられている基礎アトムの集合 M が、$fact(A,E)$ なる形で *MGTP_1* に保存される。ここに、$fact(A,E)$ は $\Sigma \cup E \models A$ なるメタ的な意味を表す。*MGTP_1* が新しい基礎アトムを導くたびに、それと統合される仮説集合の無矛盾性が *MGTP_2* によって検査される。

   この構成においては、一度に生成される複数の *MGTP_2* に並列性があり、並列化の効果は先のシステムよりも良いという結果が得られている。しかしながら、*MGTP_1* が並列化できないので、並列化の効果は、どれだけ多くの無矛盾性検査が並列になされるかに大きく依存している。

3. **全モデル生成方式.**

   前述の MGTP+MGTP 方式は、並列化によって良い性能を得たが、二つの異なったコンポーネントから構成されているので、並列化の可能性には限界がある。一方、MGTP はモデル生成と無矛盾性検査との両方の機能を備えており、これを単に推論エンジンとして使用するだけではなく、生成検査機構として使用することができる。

   MGTP をこのように用いるために、MGTP が提供するモデル候補の生成と棄却とを 1 台の MGTP で行うようにした。このような方式を用いることにより、大域的な信念の集合 M を保存するかわりに、複数のモデル候補を分散メモリ中に各々保存すればよいようにでき、上記二つの方式よりも多くの並列性を引き出すことができる。

   この全モデル生成方式は、仮説を用いた推論を実現する最も直接的な方法である。$\Gamma$ の各要素 $H$ について、

   $$\rightarrow H \mid \neg \mathsf{K} H \qquad (6)$$

   なる形の MGTP ルールを用意する。ここに、$\neg \mathsf{K}\, H$ は、“そのモデルのなかで $H$ を真であると仮定しない”ことを意味する。すなわち、各仮説により、それが成り立つモデル候補と成り立たないモデル候補とに分岐される。このシステムでは、$2^{|\Gamma|}$ 個のモデル候補が生成されるので、しばしば、いくつかの実用的な応用については、モデル候補の数が爆発してしまう可能性がある。

4. **Skip 方式.**

   全モデル生成方式において可能な限り生成されるモデル候補の数を減少させるために、仮説によるモデル分岐を遅らせる方法を示す。この Skip 方式は、前述の MGTP による失敗による否定の処理 [Inoue *et al.*, 1992a] と似ている。それは、$\Gamma$ の各要素に対して、形式 (6) のようないかなる MGTP ルールをも用意しない。そのかわりに、仮説を必要としたところでそれを導入する。$\Sigma$ の任意の節に仮定可能な述語 $H_1,\ldots,H_m$ ($H_i \in$

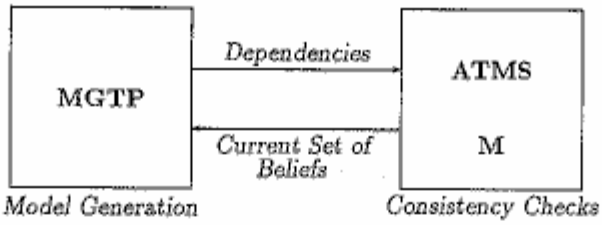


図 9: MGTP+ATMS

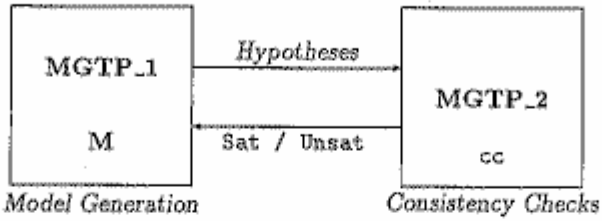


図 10: MGTP+MGTP

$\Gamma, m \geq 0$) が負に出現するものを含んでいたとき、すなわち、

$$A_1 \wedge \ldots \wedge A_l \wedge \underbrace{H_1 \wedge \ldots \wedge H_m}_{abducibles} \supset C$$

を以下のような形式の MGTP ルールに変換する。

$$A_1, \ldots, A_l \rightarrow H_1, \ldots, H_m, C \mid \neg \mathrm{K}H_1 \mid \ldots \mid \neg \mathrm{K}H_m . \quad (7)$$

この変換において、前提部における各仮説は、照合の対象から除外され、右辺に *Skip* される。この操作は、[Inoue 1991b] で定義されたトップダウンアプローチにおける Skip ルールと類似した方法である。ここで、前述のスキーマ (3) と全く同じように、各仮説 $H$ について、$H$ と $\neg \mathrm{K}H$ とをモデル候補に含んでいたならば、そのモデル候補が棄却されるように、次のスキーマを用意する。

$$\neg \mathrm{K}H, H \rightarrow .$$

以上のようなアブダクションシステムを計画や設計の問題に適用した場合の評価結果は、[Inoue *et al.*, 1992b] に示されている。現在、さらに並列化による実行効率向上のために改良を行っている。なお、 Skip 方式において、構成されるモデル候補の組合せ爆発をさらにどのようにして、減少させるかという課題が残されているが、並列性の観点から Skip 方式またはその改良が最も有望であると考えられる。また、知識ベースに仮定可能な述語と 失敗による否定の両方が含まれているような場合 [Inoue 1991a]、Skip 方式は、比較的容易に 失敗による否定 と統合できる利点を有する。

### 3.3 実現可能性解釈によるプログラム生成

定理証明器より得られる証明から実行可能なコードを取り出すため、構成的数学における実現可能性解釈 [Howard 1980] [Martin 1982] を用いる。自動証明技術と実現可能性解釈を融合したアプローチには、以下のような利点がある。

- このアプローチでは定理証明器は独立に機能するため、定理証明器を自由に取り替えることができる。
- 実現可能性解釈の正当性は数学的に証明されている。
- 実現可能性解釈は、並列プログラムを網羅している。

このアプローチの実用的効力を示すものとしてのソートアルゴリズム生成実験では、 ICOT において開発された MGTP システムと PAPYRUS システムを利用している (図 11)。

さて、MGTP は KL1 上に実現され、 Multi-PSI 上で動作し、論理式として表現された仕様の証明を検索する。MGTP は超導出をベースとしたボトムアップ（前提よりゴールを推論する）定理証明器であり、KL1 のプログラミング技術により、単純だが値域限定条件を満たす問題では非常に高速に動作する。 この推論方式は、 原理的には STCHMO [Manthey and Bry 1988] と類似のものである。また、超導出は、推論システムが構成的である点でプログラム生成に向いているが、これは、無駄な検索を排除するための制約が必要ないからである。

PAPYRUS(PArallel Program sYnthesis by Reasoning Upon formal System) は、構成的論理からプログラムを生成することを目的として作成された、論理フリーであるシステムである。このシステムでは、LF (Edinburgh Logical Framework) [Harper *et al.*, 1987] を証明チェックに利用しているため、論理をユーザが自由に定義できる。

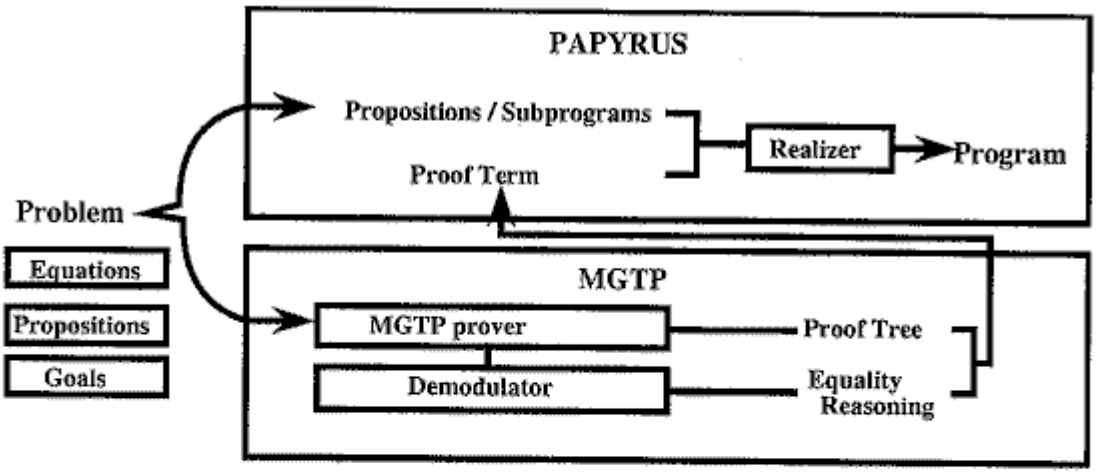


図 11: MGTP によるプログラム生成

LF における型付ラムダ項は証明を表現し、この型付ラムダ項からプログラムを生成することができる。また、このシステムではユーザにより定義される実現可能性解釈による論理式のモデルとしてプログラムを扱う。PAPYRUS は、論理証明を扱う統一的環境であり、PX [Hayashi and Nakano 1988], Nuprl [Constable *et al.*, 1986], Elf [Pfenning 1988] に似た機能を提供する。

MGTP、PAPYRUS によるプログラム生成の研究において、以下の二つの問題に直面した。

- 節形式で表現された論理式の証明からいかにしてプログラムを生成するか。
- 帰納法、等号関係をいかに扱うか。

最初の問題は、排中律に関わる問題である。論理式を節形式に変換するアルゴリズムには排中律が仮定されているので、構成的論理では論理式を節形式に変換することは一般的にはできないのである。しかしこの問題は、仕様としての論理式を始めから節形式で与えることにすることで解決される。二番目の問題は帰納法のスキーマが 2 階の命題となるところにある。これをそのまま扱うためには 2 階の同一化が必要になるのである。しかしながら、もしプログラムの領域が確定しているならば、この 2 階の命題は 1 階の命題に落すことができる。

また、等号関係についてはその証明が変換後のプログラムのアルゴリズムに関わらないことから、組み込みの手続きとしての等号関係として効率的なアルゴリズムを使用することができる。

### 3.3.1 相互プロセス生成のための論理

構成的証明からプログラムを生成するための様々な研究が行なわれている [Howard 1980, Martin 1982, Sato 1986], [Hayashi and Nakano 1988]。これらの方法では、各論理式に対する解釈が定義され、ある論理式の正しい証明が与えられれば、その証明をその論理式の解釈を満たすプログラムに変換することができる。よって各論理式は、その解釈により定められるプログラムの仕様と、証明はプログラムと、さらに証明チェックはプログラムチェックと同一視化することができる。これらの方法には、例えば証明チェックは自動化が可能であり、生成されたプログラムの仕様に対する正当性を保証できるなど、様々な利点があるが、ここでいうプログラムとは単なるラムダ項にすぎない。つまり、外界と通信を行なうような並列プログラムを生成することはできないのである。

我々は、このような並列プログラムを生成するための構成的論理として体系 $\mu$ を提案した。この論理は、新たに $\mu$、$\natural$ という論理演算子を導入することにより拡張したものである。$\mu$ は、論理式上の不動点演算子である。$\mu$ と $\natural$ により、回数に上限を持たないような入出力を表現することができる。

この論理の証明から、CCS [Milner 1989] のような並列プログラムを生成する具体的アルゴリズムを示し、さらにそのアルゴリズムの正当性、体系 $\mu$ の無矛盾性を示した。

## 3.4 交換システムの仕様記述への応用

定理証明技法を交換システムの仕様記述系の作成に利用した。その結果、仕様記述言語の処理系を MGTP で記述することにより、生成されたモデルからシステムの状態遷移図を獲得できることを確認した。

ここで紹介する仕様記述系の特徴は、以下の 三点である。

1) プロトコル仕様記述言語 Ack の提案
2) Ack の図形式表現
3) MGTP による Ack 処理系の実現

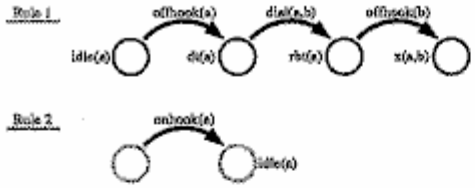


図 12: Ack の図形式表現の例

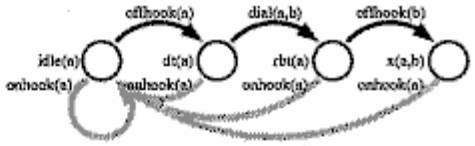


図 13: Ack 仕様の解釈結果

プロトコル仕様記述言語 Ack の特徴は、交換システムの仕様を状態遷移の生成規則として記述することである。これにより、すべての状態遷移を具体的に記述する必要がなくなり、一つの規則からいろいろな場合の状態遷移を導くことを可能としている。

Ack は、遷移システム $(S, s_0, A, T)$ に基づいて形式化されている。ここで、$S$ は状態の集合を、$s_0$ $(\in S)$ はシステムの初期状態を、$A$ は動作の集合を、$T(T \subseteq S \times A \times S)$ は遷移関係の集合を表している。

Ack の図形式表現は、ラベル付きノードとアークから構成されている。ノードとアークは、状態と動作にそれぞれ対応している。また、それらは灰色と黒色からなる二種類の色を持っており、"灰色で記述した状態遷移が存在するならば、黒色で記述した状態遷移も存在する"ことを意味している。

Ack のテキスト形式表現は、以下に記す 1 階述語論理式で表現される。

$$\forall \mathbf{X} \exists \mathbf{Y}(\mathbf{A}[\mathbf{X}] \rightarrow \mathbf{B}[\mathbf{X}, \mathbf{Y}]).$$

ここで、$\mathbf{A}[\mathbf{X}]$ と $\mathbf{B}[\mathbf{X}, \mathbf{Y}]$ は、以下に記す原始論理式の連言である。

$\mathbf{state(S)}$ – S は、状態である。

$\mathbf{trans(A, S_0, S_1)}$ – 動作 A は、状態 $\mathbf{S_0}$ から状態 $\mathbf{S_1}$ への遷移を引き起こす。

ただし、$\mathbf{A}[\mathbf{X}]$ は灰色で、$\mathbf{B}[\mathbf{X}, \mathbf{Y}]$ は黒色で記述された状態遷移にそれぞれ対応している。

Ack の処理系は、MGTP で記述されている。以上に示したプロトコルの状態遷移の規則を表現する述語論理式は、MGTP の形式に容易に変換することができる。記述された仕様に矛盾がない場合、MGTP はモデルを生成する。このモデルはプロトコルの完全な状態遷移を表している。逆に、仕様に誤りがある場合、モデルは生成されない。

図 12 に Ack の図形式表現の例を示す。

図 12 の一つめの規則は、初期状態 $idle(a)$ から $offhook(a)$, $dial(a,b)$, $offhook(b)$ と続く動作系列の存在を意味している。これは、次に記す論理式で表現される。

$$\rightarrow \mathbf{trans(offhook(a), idle(a), dt(a)),}$$
$$\mathbf{trans(dial(a, b), dt(a), rbt(a)),}$$
$$\mathbf{trans(offhook(b), rbt(a), x(a, b)).}$$

図 12 の二つめの規則は、動作 $offhook(a)$ は任意の状態を $idle(a)$ に遷移させることを意味している。これは、次に記す論理式で表現される。

$$\forall \mathbf{S}(\mathbf{state(S)} \wedge \mathbf{state(idle(a))} \rightarrow$$
$$\mathbf{trans(onhook(a), S, idle(a))})$$

図 13 は 図 12 に記す仕様の解釈結果を表している。

この例では、以下に記す 4 個の状態遷移が新たに追加されている。

- 状態 $idle(a)$ から $idle(a)$ への動作 $onhook(a)$
- 状態 $dt(a)$ から $idle(a)$ への動作 $onhook(a)$
- 状態 $rbt(a)$ から $idle(a)$ への動作 $onhook(a)$
- 状態 $x(a,b)$ から $idle(a)$ への動作 $onhook(a)$

## 3.5 MENDLES ZONE: 並列プログラムの知的プログラミング環境

MENDELS ZONE は並列プログラムを対象としたソフトウェア開発環境である。対象とする並列プログラムは MENDEL を用いて記述される。MENDEL はペトリネットをベースとしており、KL1 に変換されマルチ PSI 上で実行される。MENDEL は KL1 のユーザ言語の一つとしても位置づけられ、また、協調動作をする離散事象システムの効率的な記述にも有効な言語である。MENDELS ZONE の主な機能は次のとおりである。

1) データフローダイアグラムの可視化ツール [Honiden *et al.*, 1991]
2) 項書換え系: Metis[Ohsuga *et al.*, 1990][Ohsuga *et al.*, 1991]
3) ペトリネットと時相論理によるプログラミング環境 [Uchihira *et al.*, 1990a][Uchihira *et al.*, 1990b]

1) については、我々は、データフローダイアグラムの展開規則を規定し、展開したデータフローダイアグラムからMENDEL部品を抽出している。また、データフローダイアグラムと等式論理を組み合わせることにより、詳細な仕様化プロセスも合わせて規定した。

2) については、Metis は等式論理を対象として、最近の研究成果をいち早く実装し評価するための実験環境（この核は Knuth-Bendix (KB) 完備化手続きである）であるが、MENDEL 部品の検証系として用いている。MENDEL 部品はペトリネットに変換される。

3) については、次の機能を有している。

1. 図形エディタ

   ユーザは図形エディタを用いて、所望のソフトウェア・アーキテクチャのペトリネットモデルを構成する（生成、削除、置き換え）ことができる。また、ペトリネットの縮退、展開も可能である。

2. メソッドエディタ

   メソッドエディタはペトリネットモデルに対して種々の詳細記述を付加するのに役立つ。すなわち、KL1 における条件やアクションを記述することができる。

3. 部品ライブラリ

   MENDEL 部品は部品ライブラリに格納されている。これらの部品の表示や検索機能も用意されている。

4. 検証・生成系

   ペトリネットモデルが MENDEL 部品から自動的に抽出される。検証系はペトリネットモデルが与えられた時相論理による制約を満たすかどうかの観点で検証を実施する。

5. マルチ PSI 上のプログラム実行系

   MENDEL プログラムは KL1 プログラムに変換され、マルチ PSI 上で実行される。実行中においては、図形エディタ上に実行したメソッドが表示され、値はメッセージウィンドウに表示される。したがって、ユーザはプログラムの動的な振る舞いを視認できる。

## 4 高次推論・学習

近い将来、ソフトウェア開発にあらゆる人間がたずさわったとしても需要が満たされない状況 - “ソフトウェア危機”と呼ばれる状況 - におかれると予測されている。この危機の本質的な打開策は、未知の情報に対応し新しい情報を生み出す推論能力、個々の経験から全体を把握する学習能力、およびそれに裏付けされた強力な問題解決能力を計算機に備えさせることであり、高次推論・学習の研究目的はまさにそれらの実現方法を探ることにある。

このような先駆的な研究には確立したアプローチが存在せず、網羅的で包括的な検討が必要である。そこで、(1) 論理をベースとしてそれを拡張する論理学的アプローチ、(2) 例からの帰納的プログラム合成を形式言語理論に基づいて行なう計算論的アプローチ、(3) 仕様から演繹的プログラム合成を目指す経験論的アプローチの三つのアプローチをとった。論理学的アプローチでは、人間の問題解決の中心的役割を持つと考えられる類推を形式的に分析し、類推のための計算メカニズムを探究してきた。計算論的アプローチでは、論理プログラムの学習において最も困難な課題の一つである新述語の発見について研究し、また極小多重汎化と呼ばれる新技術を論理プログラムの構成的学習に応用する研究を進めてきた。経験論的アプローチでは、健全な効率良いプログラムを引き出すのによく研究されている展開 / 畳み込み変換に基づく論理プログラム変換・合成を特に研究してきた。

以下にそれらの研究と成果を簡単に示す。

### 4.1 類推

類推はしばしば人間の問題解決の中心的役割を持つといわれ、人工知能の分野でも比較的早い時期から研究が行なわれてきたが、未だ多くの未解決問題を抱えている。二つの事柄がある共通の性質 ($S$: 類似性) を有している時、一方 ($B$: ベース) の持つ性質 ($P$: 投射性) を他方 ($T$: ターゲット) も持つと推定する一般的な類推について考察しよう。その場合、以下が本質的な問いであり続けている。

1) “あるターゲットに対し何をベースとするか”

2) “どの性質をもって類似性というか”

3) “ある類似性に対してどのような性質が投射されうるか”

我々の目的は与えられた公理 $\mathcal{A}$ のもとで $T,B,S,P$ が必然的に満たしている論理的関係をできる限り形式的に求めることであり、それらの論理的関係を求めることは上記問題のすべてに答えることに他ならない。[Arima 1992, Arima 1991] の一連の研究でそのような一つの関係を求めており、上記問題のすべてに対する一般的解であることを示している。

述語論理による類推の分析を行なうことにより、以下が合理的な結論であることを示すことができた。

- 類推は、類推原子規則 (APR) と呼ぶある特定の型のルールが与えられた理論の演繹的な定理である時に限り、可能である。類似性 $S(x) = \Sigma(x,S)$、投射性 $P(x) = \Pi(x,P)$ とすると、APR とは以下の形の論理式である。

$$\forall x,s,p.\ J_{att}(s,p) \wedge J_{obj}(x,s) \wedge \Sigma(x,s) \supset \Pi(x,p).$$

ここで、$J_{att}(s,p)$, $J_{obj}(x,s)$, $\Sigma(x,s)$ と $\Pi(x,p)$ はそれぞれ引数以外の変数は自由に現れない述語である。

- 類推結論は、APR と 2 つの推測結果から演繹される。一つの推測は、ベースに関する情報、$\Sigma(B,S)\,(=S(B))$、および、$\Pi(B,P)\,(=P(B))$ から、$J_{att}(S,P)$ を得るものであり、他方は、ターゲットに関する情報、$\Sigma(T,S)(=S(T))$、から $J_{obj}(T,S)$ を得る推測である。

また、"アブダクション + 演繹" がこれら二つの推測を得るのに適当な推論体系であることを示した。

これらの結果に基づいて、類推のためのプロトタイプシステムを試作した。公理 $A$ のもとで、ターゲット $T$、および、投射される属性 $\Pi(x,p)$ が与えられると (質問 "?-$\Pi(T,p)$" から得られる)、このシステムは類似性 $\Sigma(x,S)$ を動的に決定し、ベース $B$ を選び、それによって投射性 $\Pi(x,P)$ を求めることができる (すなわち、先の質問に対しては "$\Pi(T,P)$" なる解が得られる).

## 4.2 論理プログラムの機械学習

機械学習は人工知能の分野におけるもっとも重要な課題の一つである。学習能力は膨大な量の知識情報処理や保守等だけでなくユーザにとって好ましいインターフェイスを実現する上でも必要である。我々は、論理プログラムの学習においてもっとも重要な問題の一つである新述語発見の問題について考察を行なった。また、論理プログラムの構成的学習に対する極小多重汎化の応用についても検討を行なった。

### 4.2.1 述語発見

Shapiro によるモデル推論は論理プログラムの学習に対する極めて重要な戦略、すなわち、矛盾点追跡による漸増的な仮説の探索法を与えている。しかし、彼の理論では、目標のモデルを記述するのに十分な述語を持つ初期仮説言語が学習者に与えられることを仮定している。さらに、教師はそれらすべての述語の意図されたモデルを知っていることが仮定される。このような仮定は論理プログラム学習の現実的なアプリケーションにとってかなり厳しく制限的なものであるので、取り除かれるべきである。そのような仮定を必要としない学習システムを構築するためには述語発見の問題を扱わなければならない。

この興味深くかつ難しい問題に対するいくつかのアプローチが最近提案されている [Muggleton and Buntine 1988, Ling 1989]。しかしながら、それらのほとんどは、仮説言語が増大するような学習過程における計算量に関する十分な解析を与えていない。我々は、この述語発見の問題を文法推論における非終端記号発見の問題として議論した。よく知られているように、任意の文脈自由文法は特殊な形の論理プログラム、すなわち DCG (definite clause grammar) によって表現できる。すなわち、文法推論における非終端記号発見の問題は述語発見の問題に対応するのである。

我々は、非終端記号発見と矛盾点追跡手法に基づいた、単純決定性言語のクラスに対する多項式時間学習アルゴリズムを提案した [Ishizaka 1990]。単純決定性言語のクラスは正則言語のクラスを真に包含するので、この結果は我々の以前の結果 [Ishizaka 1989] の拡張になっている。

### 4.2.2 極小多重汎化

論理プログラムの学習におけるもう一つの重要な問題は構成的な学習アルゴリズムの開発にある。Shapiro によるモデル推論システムのように、多くの帰納的学習アルゴリズムは探索や枚挙といった手法に基づいている。たしかに探索や枚挙は極めて強力な手法ではあるが、それらは非常に効率の悪いものである。構成的手法の方が探索手法よりも効率的である。

構成的な論理プログラム学習においては、最小汎化 [Plotkin 1970] の概念が中心的な役割を演ずる。極く最近、最小汎化の自然な一般化である極小多重汎化の概念が有村によって提案された [Arimura 1991] 。例えば、通常の append プログラム中の二つの節の頭部の対は append プログラムの Herbrand モデルに対する極小多重汎化の一つになっている。したがって、極小多重汎化は目標のプログラムの頭部の推論に適用可能であると考えられる。有村はさらに、極小多重汎化を求める多項式時間アルゴリズムを与えている。我々は、現在、極小多重汎化を用いた効率的な学習手法について考察中である。

## 4.3 論理プログラム変換合成

高次推論の研究の重要な課題の一つとして自動プログラミングが挙げられる。自動プログラム変換合成の分野では、正しく効率的なプログラムを導出する技法として展開 / 畳込み変換 [Burstall and Darlington 1977, Tamaki and Sato 1984] の研究が行われてきた。

展開 / 畳込み変換はプログラム開発のための強力な技法であるが、効率的なプログラムを得るにはこれらの変換規則を適用する戦略を考慮する必要がある。論理プログラムの展開／畳込み変換では、畳込み変換を行って述語の再帰的な定義を求めることによってプログラムの効率が向上することが多い。また、適当な新述語の導入によって畳込みの適用が可能になる場合が多いことが知られている。このように、新述語の発見はプログラム変換の研究の最も重要な問題の一つである。

展開／畳込み変換は論理プログラム合成の研究でも用いられる。これらの研究では、畳込み変換を適用して限量子を消去することによって一階論理式で記述されたプログラムの仕様から確定節論理プログラムを導く。しかしながら、従来の研究の多くでは展開 / 畳込み変換を非決定的に適用しており、確定節を導く一般的な方法は知られていなかった。

以上のような観点から、我々は展開／畳込み変換による論理プログラム変換合成の研究を行い、次の成果を得た。

(1) 展開／畳込み変換に基づく論理プログラム変換技法の研究を行った [Kawamura 1991]。この方法では畳込

み変換を適用できるように新述語が自動的に合成される。また、この方法を拡張すると、展開／畳込み変換とゴール置換変換 [Tamaki and Sato 1984] を組み合わせて用いる場合にも適用できることを示した。

(2) 自動的に確定節論理プログラムを導くことができる一階論理式のクラスを示した [Kawamura 1992]。これらの論理式はホーン節の本体に全称限量された含意式が現れることを許した論理式である。これらの論理式は、ある統語上の制約を満たしているとき、一般化展開 / 畳込み変換 [Kanamori and Horiuchi 1987] に基づくプログラム合成手続きによって等価な確定節論理プログラムに変換できる。

## 5 結論

ICOT における並列自動推論システムの研究開発の概略を述べた。並列定理証明、応用、高次推論・学習の三つのテーマについて研究開発を進めてきたが、各テーマで以下のような有望な技術的成果が得られている。

(1) 並列定理証明器と PIM 上での実現技術

基底モデル用 MGTP/G と非基底モデル用 MGTP/N の二種類のモデル生成定理証明器を提示し、分散メモリアーキテクチャのマルチプロセッサ Multi-PSI および PIM 上で性能評価を行った。

値域限定の条件を満たす問題は同一化を必要とせず、照合で済む。KL1 言語の特徴を活用することによって、MGTP/G では高効率が達成されている。

非値域限定の問題を解くため、MGTP/N では、ホーン節問題に限定し、同一化や変数管理機能を支援するメタライブラリと呼ぶ KL1 メタプログラミングツールを使用した。

MGTP の効率改善を図って、連言照合における冗長計算を回避する RAMS 法および MERC 法を開発した。

分離の規則に関する問題のような困難な数学的定理を証明する際に必要とされる、膨大な時間 / メモリ量を軽減するため、基本アルゴリズムの時間およびメモリに関する複雑度を数桁下げることができる、遅延モデル生成法を提案した。実験結果からも、遅延アルゴリズムを使用することにより、時間 / メモリ量が大幅に節約できることが確認された。

非ホーン基底問題については、場合分けが MGTP/G の OR 並列の源となる。この種の問題は、通信コスト低減のためにできるだけ粒度を大きくする必要がある、Multi-PSI のような MIMD マシンにうってつけの問題である。負荷分散のための簡単な割り付け方式を用いて、N クィーン、ビジョンホール等の問題を Multi-PSI 上で走行させたところ、予想どおり、ほぼ線形の台数効果が得られた。

一方、ホーン非基底問題の場合は、連言照合や包摂テストに内在する AND 並列性を抽出しなければならない。ここで開発した MGTP/N の AND 並列化方式は良好な性能向上とスケーラビリティを達成しうることがわかった。

特に、PIM/m 上の MGTP/N を用いて得られた最近の実験結果から、分離の規則に関する定理については、少なくとも 128 PE までは (128 PE 以上は未測定) 線形の台数効果が得られることが確認された。

MGTP/N で使用している KL1 で書かれた同一化プログラムは、SPARC 上の C で書かれたものよりも 100 倍程度遅い。現在、この差を埋めるため、ファームウェアルーチンとして組み込み化することを検討している。しかし、これに代わる方法としては、非基底モデル用の KL1 コンパイル技法の開発が PIM 上の並列論理プログラミングにさらに貢献するものと思われる。

MGTP の開発を通して、KL1 は コンカレントシステムのラピッドプログラミングの言語として、強力なツールとなり得ること、そして、並列推論システムを容易且つ効果的に構築しうることが実証された。

(2) 応用

MGTP/G 上の様相論理定理証明器には二つの利点がある。一つは、冗長性の除去と MGTP/G の並列化によって高性能を達成することが可能になっている点である。二つめは、含意形式の MGTP 入力節によって様相論理のタブロルールを直接表現しているため、性能に対するヒューリスティックスを容易に付加することができることである。この様相論理定理証明器は優れたベンチマーク結果を示している。

MGTP 上での非単調およびアブダクションに関する基本的なアイデアは、MGTP をメタインタープリタとして用い、これらのシステムに特有な性質、例えば非単調性を有する証明 (失敗による否定) や仮説推論の無矛盾性を古典論理で扱えるような形式に変換することである。こうすることによって、これらの特有な性質はモデル候補の生成検査問題に置き換えることができ、MGTP の場合分けや矛盾したモデル候補の棄却機能によって効果的に取り扱うことができる。

また我々は、MGTP の応用として、プログラム生成について以下のような二つの方法で実現した。

第一のアプローチは、実現可能性解釈を用いる方法である。これは Curry Howard Isomorphism の拡張と考えられるが、効率的に定理証明器から得られた証明に実行可能な意味を与えるものである。

我々は、現在までに ICOT で開発された MGTP と PAPYRUS の二つのシステムを使って、ソーティングアルゴリズムや中国剰余定理の証明からのプログラ

ム抽出実験を行ない、MLのプログラムを抽出することに成功している。

また、並列プログラムを抽出するために、我々は新しい論理体系 $\mu$ を提案した。これは新しい論理演算子 $\mu$ と $\nu$ を導入した構成的論理である。さらに我々は、体系 $\mu$ の証明から、CCSのような並列プロセスとしてプログラムを生成する方法を示した。我々はまた、体系 $\mu$ の無矛盾性の証明とプログラム合成の方法の妥当性も示した。

MGTPによる並列プログラム合成の別のアプローチは時制論理を使うことであり、その中では、仕様は有限状態ダイアグラムとして以下のようにモデル化されている。

1) 仕様記述言語Ack状態遷移システムに基づいている。
2) Ackにおける視覚的表現
3) MGTPによるAckインタプリタ

Ackでは、いくつかの状態遷移システムは時制論理の定理証明系によって演繹的に導くことができるので、仕様のすべてを具体的に記述する必要はない。したがって、仕様がある程度曖昧なものであってもそこから完全な仕様を得ることができる。

もう一つのアプローチは項書換え系(Metis)を利用する方法である。MENDELS ZONEはこの方法を利用した並列プログラムのソフトウェア開発システムである。対象としているソフトウェア開発言語はペトリネットを表現した並列プログラミング言語MENDELである。MENDELはさらに並列論理型プログラミング言語KL1へ変換され、Multi-PSI上で実行される。

本システムでは、データフロー図に対して分解規則が定義され、それをもとにプログラムが抽出される。Metisは並列プログラムの実現や試験の実用的な技術研究を等式推論を利用して行なう実験環境を提供している。Metisの中核的機能はKnuth-Bendix(KB)の完備化手続きである。MENDELS ZONEではMetisをペトリネットの検証に利用している。

プログラムの検証や合成は有限のペトリネットに適用可能なので、ペトリネット構造の格子のみ(スロットやKL1コードは無視される)が自動的に消去される。また、検証ツールはペトリネットが与えられた時制論理の制約を満足するかどうかも判定する。

### (3) 高次推論と学習

類推システムの推論能力を拡張するため、論理学的、計算論的および経験的の三つのアプローチをとった。

論理学的アプローチでは、人間が問題解決を行なう際に中心的役割を果たすと考えられている類推を厳密に解析し、類推を実現する計算機構の研究を行った。このアプローチでは、与えられた理論に対して類推要因であるターゲット、ベース、類似性および投射性間の一般的な関係をできるだけ厳密に明らかにすることが目的であった。類推要因間の関係を決定することによって、類推に関する諸問題に対して最終的な答えを出すことになる。この研究では、類推要因間における関係を明らかにし、存在する諸問題の一般的な解答を得ることができた。

計算論的アプローチでは、新しい述語の発見について研究を行なった。新しい述語の発見は論理型プログラミングの学習において最も重要な問題である。この研究において、簡単な決定性言語のあるクラスに対して、多項式時間オーダーの学習アルゴリズムを提案した。このアルゴリズムは非決定的発見法と矛盾後戻りに基づいたものである。この言語クラスは正規言語を含んでいるので、この結果はこれまでの成果の拡張になっている。また極小多重汎化の論理型プログラミングの構成的学習への応用についても研究を行なった。極小多重汎化 とは有村によって最近に提案された概念であり、極小多重汎化を用いた効率的な学習法を研究している段階である。

経験的アプローチでは、展開/畳込み変換に基づいたプログラム変換および合成法を中心とした自動プログラミングに関する研究を行なった。展開/畳込み変換は正当性を持ち、効率的なプログラムを得るのに有力な手段として知られている手法である。この研究では、論理型プログラミングを対象としたプログラム変換の戦略を考案した。この方法では新しい述語が畳込みを通じて自動的に合成される。 さらにゴール置き換え変換に組み込めるようにこの方法の拡張を行なった。

また、確定節のプログラムが自動的に得られるような1階述語論理式の節の性質を明確にした。これらの論理式は全称限量子がついた含意形式の節によって拡張されたホーン節である。そして、一般化された展開/畳込み規則に基づいたプログラム合成手続きを与え、いくつかの文法的な制限があるもののこれらの論理式が同値な確定節のプログラムにうまく変換できることが明らかになった。

以上の結果は、AIの応用分野だけでなく、FGCSの中核技術である並列論理型プログラミングの基礎技術の面においても、FGCSの発展に貢献している。

## 謝辞

自動推論システムの研究は、メーカ5社の協力のもと、ICOT 第五研究室において進められた。ICOT 渕所長および古川次長をはじめ、ご支援と貴重なコメントを下さっ

た方々に感謝いたします。また、PTP、PAR、ANR、ALTのワーキンググループでは、多くの有益な議論が行なわれた。各ワーキンググループの主査およびメンバーの方々に感謝いたします。そして、本研究プロジェクトにおいて御助力頂いた各メーカの方々になによりも感謝いたします。

## 参考文献

[Fuchi 1990] K. Fuchi, Impression on KL1 programming – from my experience with writing parallel provers –, in *Proc. of KL1 Programming Workshop '90,* pp.131-139, 1990 (in Japanese).

[Hasegawa *et al.*, 1990] R. Hasegawa, H. Fujita and M. Fujita, A Parallel Theorem Prover in KL1 and Its Application to Program Synthesis, in *Italy-Japan-Sweden Workshop '90,* ICOT TR-588, 1990.

[Fujita and Hasegawa 1990] H. Fujita and R. Hasegawa, A Model Generation Theorem Prover in KL1 Using Ramified-Stack Algorithm, ICOT TR-606, 1990.

[Hasegawa 1991] R. Hasegawa, A Parallel Model Generation Theorem Prover: MGTP and Further Research Plan, in Proc. of the Joint American-Japanese Workshop on Theorem Proving, Argonne, Illinois, 1991.

[Hasegawa *et al.*, 1992] R. Hasegawa, M. Koshimura and H. Fujita, Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers, ICOT TR-751, 1992.

[Koshimura *et al.*, 1990] M. Koshimura, H. Fujita and R. Hasegawa, Meta-Programming in KL1, ICOT-TR-623, 1990 (in Japanese).

[Manthey and Bry 1988] R. Manthey and F. Bry, SATCHMO: a theorem prover implemented in Prolog, in *Proc. of CADE 88, Argonne, illinois,* 1988.

[Nitta *et al.*, 1992] K. Nitta, Y. Ohtake, S. Maeda, M. Ono, H. Ohsaki and K. Sakane, HELIC-II: a legal resoning system on the parallel inference machine, in *Proc. of FGCS'92,* Tokyo, 1992.

[Stickel 1988] M.E. Stickel, A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, in *Journal of Automated Reasoning* 4 pp.353-380, 1988.

[McCune 1990] W.W. McCune, OTTER 2.0 Users Guide, Argonne National Laboratory, 1990.

[McCune and Wos 1991] W.W. McCune and L. Wos, Experiments in Automated Deduction with Condensed Detachment, Argonne National Laboratory, 1991.

[Overbeek 1990] R. Overbeek, Challenge Problems, (private communication) 1990.

[Wilson and Loveland 1989] D. Wilson and D. Loveland, Incorporating Relevancy Testing in SATCHMO, Technical Report of CS(CS-1989-24), Duke University, 1989.

[Fitting 1983] M. Fitting, *Proof Methods for Modal and Intuitionistic Logic,* D.Reidel Publishing Co., Dordrecht 1983.

[Fitting 1988] M. Fitting, "First-Order Modal Tableaux", *Journal of Automated Reasoning,* Vol.4, No.2, 1988.

[Koshimura and Hasegawa 1991] M. Koshimura and R. Hasegawa, "Modal Propositional Tableaux in a Model Generation Theorem Prover", In *Proceedings of the Logic Programming Conference'91,* Tokyo, 1991 (in Japanese).

[Smullyan 1968] R.M. Smullyan, *First-Order Logic,* Vol 43 of Ergebnisse der Mathematik, Springer-Verlag, Berlin, 1968.

[Arima 1991] J. Arima, A Logical Analysis of Relevance in Analogy, in *Proc. of Workshop on Algorithmic Learning Theory ALT'91,* Japanese Society for Artificial Intelligence, 1991.

[Arima 1992] J. Arima, Logical Structure of Analogy, in *FGCS'92,* Tokyo, 1992.

[Kohda and Maeda 1991a] Y. Kohda and M. Maeda, Strategy Management Shell on a Parallel Machine, IIAS RESEARCH Memorandum IIAS-RM-91-8E, Fujitsu, October 1991.

[Kohda and Maeda 1991b] Y. Kohda and M. Maeda, Strategy Management Shell on a Parallel Machine, in poster session of ILPS'91, San Diego, October 1991.

[Maeda *et al.*, 1990] M. Maeda, H. Uoi, N. Tokura, Process and Stream Oriented Debugger for GHC programs, Proceedings of Logic Programming Conference 1990, pp.169-178, ICOT, July 1990.

[Maeda 1992] M. Maeda, Implementing a Process Oriented Debugger with Reflection and Program Transformation, in *Proc. of FGCS'92,* Tokyo, 1992.

[Smith 1984] B.C. Smith, Reflection and Semantics in Lisp, Conference Record of the 11th Annual ACM Symposium on Principles of Programming Languages, pp.23-35, ACM, January 1984.

[Sugano 1990] H. Sugano, Meta and Reflective Computation in Logic Programs and its Semantics, Proceedings of the Second Workshop on Meta-Programming in Logic, Leuven, Belgium, pp.19-34, April, 1990.

[Sugano 1991] H. Sugano, Modeling Group Reflection in a Simple Concurrent Constraint Language, *OOPSLA'91 Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming*, 1991.

[Tanaka 1991] J. Tanaka, An Experimental Reflective Programming System Written in GHC, *Journal of Information Processing*, Vol.14, No.1, pp.74-84, 1991.

[Tanaka and Matono 1992] J. Tanaka and F. Matono, Constructing and Collapsing a Reflective Tower in Reflective Guarded Horn Clauses, in *Proc. of FGCS'92*, Tokyo, 1992.

[Arimura 1991] H. Arimura, T. Shinohara and S. Otsuki, Polynomial time inference of unions of tree pattern languages. In S. Arikawa, A. Maruoka, and T. Sato, editors, *Proc. ALT '91*, pp. 105–114. Ohmsha, 1991.

[Ishizaka 1989] H. Ishizaka, Inductive inference of regular languages based on model inference. *International journal of Computer Mathematics*, 27:67–83, 1989.

[Ishizaka 1990] H. Ishizaka, Polynomial time learnability of simple deterministic languages. *Machine Learning*, 5(2):151–164, 1990.

[Ling 1989] X. Ling, Inventing theoretical terms in inductive learning of functions - search and constructive methods. In Zbigniew W. Ras, editor, *Methodologies for Intelligent Systems, 4*, pp. 332–341. North-Holland, October 1989.

[Muggleton and Buntine 1988] S. Muggleton and W. Buntine, Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, pp. 339–352, 1988.

[Plotkin 1970] G.D. Plotkin, A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pp. 153–163. Edinburgh University Press, 1970.

[Burstall and Darlington 1977] R.M. Burstall and J. Darlington, "A Transformation System for Developing Recursive Programs", J.ACM, Vol.24, No.1, pp.44–67, 1977.

[Kanamori and Horiuchi 1987] T. Kanamori and K. Horiuchi, "Construction of Logic Programs Based on Generalized Unfold/Fold Rules", Proc. of 4th International Conference on Logic Programming, pp.744–768, Melbourne, 1987.

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

[Kawamura 1992] T. Kawamura, "Logic Program Synthesis from First Order Logic Specifications", to appear in International Conference on Fifth Generation Computer Systems 1992, Tokyo, 1992.

[Tamaki and Sato 1984] H. Tamaki and T. Sato, "Unfold/Fold Transformation of Logic Programs", Proc. of 2nd International Logic Programming Conference, pp.127–138, Uppsala, 1984.

[Bry 1990] F. Bry, Query evaluation in recursive databases: bottom-up and top-down reconciled. *Data & Knowledge Engineering*, 5:289–312, 1990.

[de Kleer 1986] J. de Kleer, An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[Fujita and Hasegawa 1991] H. Fujita and R. Hasegawa, A model generation theorem prover in KL1 using a ramified-stack algorithm. In: *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pp. 535–548, MIT Press, Cambridge, MA, 1991.

[Gelfond and Lifschitz 1991] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Inoue 1988] K. Inoue, Problem solving with hypothetical reasoning, in *Proc. of FGCS'88*, pp. 1275–1281, Tokyo, 1988.

[Inoue 1991a] K. Inoue, Extended logic programs with default assumptions, in *Proc. of the Eighth International Conference on Logic Programming* (Paris, France), pp. 490–504, MIT Press, Cambridge, MA, 1991.

[Inoue 1991b] K. Inoue, Linear resolution for consequence-finding, To appear in: *Aritificial Intelligence*, An earlier version appeared as: Consequence-finding based on ordered linear resolution, in *Proc. of IJCAI-91*, pp. 158–164, Sydney, Australia, 1991.

[Inoue *et al.*, 1992a] K. Inoue, M. Koshimura and R. Hasegawa, Embedding negation as failure into a model generation theorem prover, To appear in *CADE 92*, Saratoga Springs, NY, June 1992.

[Inoue *et al.*, 1992b] K. Inoue, Y. Ohta, R. Hasegawa and M. Nakashima, Hypothetical reasoning systems on the MGTP, ICOT-TR 1992 (in Japanese).

[Loveland 1978] D.W. Loveland, *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[Ohta and Inoue 1990] Y. Ohta and K. Inoue, A forward-chaining multiple context reasoner and its application to logic design, in: *Proceedings of the Second IEEE International Conference on Tools for Artificial Intelligence*, pp. 386–392, Herndon, VA, 1990.

[Ohta and Inoue 1992] Y. Ohta and K. Inoue, A forward-chaining hypothetical reasoner based on upside-down meta-interpretation, in *Proc. of FGCS'92*, Tokyo, 1992.

[Poole *et al.*, 1987] D. Poole, R. Goebel and R. Aleliunas, Theorist: a logical reasoning system for defaults and diagnosis, In: Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pp. 331–352, Springer-Verlag, New York, 1987.

[Stickel 1991] M.E. Stickel, Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction, ICOT TR-664, 1991.

[Constable *et al.*, 1986] R.L. Constable et al, *Implementing Mathematics with the Nuprl Proof Development System*, Prenticd-Hall, NJ, 1986.

[Hayashi and Nakano 1988] S. Hayashi and H. Nakano, *PX: A Computational Logic*, MIT Press, Cambridge, 1988.

[Harper *et al.*, 1987] R. Harper, F. Honsell and G. Plotkin, A Framework for Defining Logics, in *Symposium on Logic in Computer Science*, IEEE, pp. 194-204, 1987.

[Pfenning 1988] F. Pfenning, Elf: A Language for Logic Definition and Verified Meta-Programming, in *Fourth Annual Symposium on Logic in Computer Science*, IEEE, pp. 313-322, 1989.

[Takayama 1987] Y. Takayama, Writing Programs as QJ Proof and Compiling into Prolog Programs, in *Proc. of IEEE The Symposium on Logic Programming '87*, pp. 278-287, 1987.

[Howard 1980] W.A. Howard, "The formulae-as-types notion of construction", in Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, pp.479-490, 1980.

[Martin 1982] P. Martin-Löf, "Constructive mathematics and computer programming", in Logic, Methodology, and Philosophy of Science VI, Cohen, L.J. et al, eds., North-Holland, pp.153-179, 1982.

[Sato 1986] M. Sato, "QJ: A Constructive Logical System with Types", France-Japan Artificial Intelligence and Computer Science Symposium 86, Tokyo, 1986.

[Milner 1989] R. Milner, "Communication and Concurrency", Prentice-Hall International, 1989.

[Honiden *et al.*, 1990] S. Honiden et al., An Application of Structural Modeling and Automated Reasoning to Real-Time Systems Design, in *The Journal of Real-Time Systems*, 1990.

[Honiden *et al.*, 1991] S. Honiden et al., An Integration Environment to Put Formal Specification into Practical Use in Real-Time Systems, in *Proc. 6th IWSSD*, 1991.

[Ohsuga *et al.*, 1991] A. Ohsuga et al., A Term Rewriting System Generator, in *Software Science and Engineering*, World Scientific ,1991.

[Ohsuga *et al.*, 1990] A. Ohsuga et al., Complete E-unification based on an extension of the Knuth-Bendix Completion Procedure, in *Proc. of Workshop on Word Equations and Related Topics*, LNCS 572, 1990.

[Uchihira *et al.*, 1990a] N. Uchihira et al., Synthesis of Concurrent Programs: Automated Reasoning Complements Software Reuse, in *Proc. of 23th HICSS*, 1990.

[Uchihira *et al.*, 1990b] N. Uchihira et al., Verification and Synthesis of Concurrent Programs Using Petri Nets and Temporal Logic, in *Trans. IEICE*, Vol. E73, No. 12, 1990.