モデル生成型定理証明系：MGTP
マニュアル

Model Generation Theorem Prover : MGTP Manual

(財) 新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

平成 7 年 3 月
March 1995

目次

1 MGTP の機能説明 **4**
- 1.1 まえがき . . . 4
- 1.2 モデル生成法 . . . 5
- 1.3 モデル生成型定理証明系 MGTP . . . 6
  - 1.3.1 MGTP の基本構成 . . . 6
  - 1.3.2 KL1 による MG 節集合の表現 . . . 9
  - 1.3.3 証明系本体の実現 . . . 9
- 1.4 連言照合における冗長性除去 . . . 10
  - 1.4.1 連言照合の冗長性 . . . 10
  - 1.4.2 RAMS 法 . . . 12
  - 1.4.3 MERC 法 . . . 13
- 1.5 RAMS 法と MERC 法の比較 . . . 13
- 1.6 まとめ . . . 18

2 MGTP 入力言語仕様 **21**
- 2.1 MGTP 入力節シンタクス . . . 21
- 2.2 例題 . . . 22

3 MGTP/G マニュアル **26**
- 3.1 MGTP/G の機能説明 . . . 26
  - 3.1.1 Prolog 版 . . . 26
  - 3.1.2 KLIC 版 . . . 27
  - 3.1.3 KL1 版 . . . 27
- 3.2 MGTP/G の操作方法 . . . 27
  - 3.2.1 Prolog 版 . . . 27
  - 3.2.2 KLIC 版 . . . 31
  - 3.2.3 KL1 版 . . . 34

4 MGTP/N マニュアル **40**
- 4.1 MGTP/N の機能説明 . . . 40
  - 4.1.1 Prolog 版 . . . 40
  - 4.1.2 KL1 版 . . . 40

4.2 MGTP/N のアルゴリズム 41
4.2.1 逐次アルゴリズム 41
4.2.2 モデル共有型並列アルゴリズム 42
4.2.3 モデル分散型並列アルゴリズム 47
4.3 操作方法 48
4.3.1 Prolog 版操作法 48
4.3.2 KL1 版 (モデル共有型) 操作法 52
4.3.3 KL1 版 (モデル分散型) 操作法 56

5 CMGTP マニュアル **61**
5.1 CMGTP の機能説明 62
5.1.1 概要 62
5.1.2 準群問題 62
5.1.3 制約 MGTP 64
5.1.4 並列化方式 67
5.1.5 並列実行に関する考察 70
5.1.6 まとめ 71
5.2 Prolog 版操作マニュアル 77
5.2.1 インストール 77
5.2.2 トランスレータのコンパイル 77
5.2.3 問題の KL1 プログラムへの変換 77
5.2.4 CMGTP 実行オブジェクトの作成 78
5.2.5 実行 78
5.2.6 実行結果 78
5.3 KLIC 版操作マニュアル 81
5.3.1 インストール 81
5.3.2 トランスレータのコンパイル 81
5.3.3 問題の KL1 プログラムへの変換 81
5.3.4 コンパイル 82
5.3.5 実行 82
5.3.6 実行結果 82
5.4 KL1 版操作マニュアル 84
5.4.1 KL1 版の特徴 84
5.4.2 PIM/m 上での実行方法 84

はじめに

本マニュアルは (財) 新世代コンピュータ技術開発機構 (ICOT) において開発されたモデル生成型定理証明システム MGTP に関する機能説明および操作説明書です。

MGTP は一階述語論理に基づくボトムアップ型の定理証明システムで、連言照合における冗長性を極力排除し、また実装上の工夫などにより並列マシン上で極めて効率的に動作する問題解決器です。

MGTP はその問題記述の簡便性から、一階述語論理による知識表現言語と見ることもでき、 Prolog ではフォローできない Non-Horn 論理式を取り扱うことができるほか、並行論理型言語 KL1 では失われている探索の完全性が保証されているなどの利点を持っています。

MGTP は、適用すべき問題の性質によって、グランド版 MGTP (MGTP/G)、ノングランド版 MGTP (MGTP/N)、および制約充足問題を解くために拡張された制約 MGTP (CMGTP) の 3 種類が用意されており、それぞれ KL1、 Prolog、 Klic で実装されています。グランド版 MGTP においては生成されるモデル候補はすべて基底項、すなわち変数を含まない項であり、ルールの後件部に選言を記述することができます。ノングランド版 MGTP では、変数を含む項が許されますが、現在の実装ではルールの後件部に選言を記述することはできません。また、 CMGTP は、制約充足問題を解くために拡張された MGTP で、枝刈りを効率的に行なうために負アトムを用いて制約伝搬ルールを表現することが可能です。並列化に関しては、グランド版 MGTP および CMGTP では OR 並列が、またノングランド版 MGTP では AND 並列がそれぞれ実現され、いくつかのベンチマーク問題に対して線形に近い台数効果が達成されています。

IFS ではこれらのすべてのソフトウェアを公開しており、本マニュアルでは、これらのシステムの機能概要と各バージョンにおける操作説明方法について説明します。

本マニュアルに関するコメント、あるいはソフトウェアに関するバグ報告、コメントなどは下記までお願い致します。

(財) 新世代コンピュータ技術開発機構

長谷川隆三　(hasegawa@icot.or.jp)
越村 三幸　(koshi@icot.or.jp)
白井康之　(shirai@icot.or.jp)

Phone: (03) 3456-4365
Fax: (03) 3456-1618

本マニュアルの構成

本マニュアルの構成は以下のようになっている。

第 1 章 MGTP の機能説明
MGTP の基本的な機能について、特にモデル生成法の枠組、冗長性排除の方法などについて説明する。グランド版、ノングランド版、制約 MGTP に固有な機能についてはそれぞれのシステムの機能説明を参照されたい。

第 2 章 MGTP 入力言語マニュアル
MGTP の入力節のシンタクスについて、例題を交えながら説明する。

第 3 章 トランスレータの説明
MGTP においては、ユーザが記述した入力節を実行させるべき言語環境 (KL1、 Klic、 Prolog) に変換するトランスレータが必要である。トランスレータの機能概要についてここで説明する。

第 4 章 MGTP/G マニュアル
グランド版 MGTP の機能説明と、 KL1 版、 Klic 版、 Prolog 版のそれぞれについて操作説明を行なう。

第 5 章 MGTP/N マニュアル
ノングランド版 MGTP の機能説明と、 KL1 版、 Klic 版、 Prolog 版のそれぞれについて操作説明を行なう。

第 6 章 CMGTP マニュアル
制約 MGTP (CMGTP) の機能説明と、 KL1 版、 Klic 版、 Prolog 版のそれぞれについて操作説明を行なう。

第 1 章

MGTP の機能説明

この章では、 MGTP の基本的な機能について、特にモデル生成法の枠組、冗長性排除の方法などについて説明する。

1.1 まえがき

第 5 世代コンピュータプロジェクトにおいて技術の根幹に据えられた、ホーン論理を基盤とする論理プログラミングは、革新的な計算パラダイムとして多くの研究者の関心を集め、著しい発展を遂げた。定理証明の分野では、論理プログラミング分野での主要成果である Prolog に注目が寄せられ、そこに駆使された効率の良い実装技術を利用して高効率な定理証明系を作る試みが出てきた [Sti88] [MB88]。これらのシステムはいずれも、ホーン節に対する推論が Prolog プログラムの実行として極めて効率良く実現できるという事実を利用している。

さて、 Prolog などの論理型言語ではプログラミング言語としての効率性が優先されたため、1) 単一化から出現検査が省かれている、 2) 選言的頭部を持つ節 (非ホーン節) を扱えない、 3) 否定はいわゆる “失敗としての否定” であり、古典論理的否定が扱えないなど、一階述語論理を扱う上で必要な論理的性質の一部が犠牲にされている。

一方、並行論理型言語 KL1 は論理変数など Prolog の持つ利点を受け継ぎつつ、さらに並行プロセスとプロセス間通信の概念に基づく並行プログラムを容易に記述することを可能としている。 KL1 は同期に関するバグが発生しにくいなどの利点があり、並列処理システムを効率良く実現するのに優れた言語である。

しかしながら、 KL1 は論理的観点からは Prolog よりさらに後退したものになっている。実際、 KL1 節はもはや純粋なホーン節ではなく、コミット演算子のような非論理的要素を含んでいる。そして、単一化の失敗や節選択の再試行が許されないなど、探索空間に関する完全性が失われている。

上述のような問題に対しては、一般的にはメタプログラミング技法によって解決する手段を講じることが可能であるが、通常その代償として、実行効率の顕著な劣化が伴うという難点がある。

我々は、以上の問題点を解決し、 KL1 で一階述語論理の定理証明系を効率良く実現するために、証明方式として SATCHMO が基礎を置くモデル生成法を採用した。モデル生成法においては、値域限定という条件を満たす節集合を対象とした場合、出現検査付き単一化が不要に

なり、 KL1 言語処理系を活用した効率の良い実装が可能となった。

以下では、まずモデル生成法の原理を説明し、 KL1 言語によるモデル生成型定理証明系の簡潔な実装法を述べる。次いで、モデル生成法で生じる可能性のある冗長計算を避けるための二つの方式、 RAMS 法と MERC 法を説明する。

1.2 モデル生成法

任意の一階述語論理式は、常に節形式 (連言標準形) に変換可能である。 1 本の節は正負リテラルを任意の順で選言結合したものであるが、ここでは節 $\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$ を次のような含意式の形で表現する。

$$A_1, \ldots, A_n \rightarrow B_1; \ldots; B_m$$

ここで、 $A_i(1 \leq i \leq n), B_j(1 \leq j \leq m)$ はアトム、 ",” は論理積結合子、 ";” は論理和結合子、 "$\rightarrow$” は含意結合子である。 "$\rightarrow$” の左辺 (のアトム) を前件 (リテラル)、右辺 (のアトム) を後件 (リテラル) という。前件が空の場合 $A_0 = true$ と置き正節、後件が空の場合 $B_0 = false$ と置き負節と呼ぶ。

節 $A_1, \ldots, A_n \rightarrow B_1; \ldots; B_m$ において、

$$\bigcup_{i=1}^{n} var\,(A_i) \supseteq \bigcup_{j=1}^{m} var\,(B_j)$$

が成り立つとき、この節は値域限定 ($range\text{-}restrictedness$) の条件を満たすといわれる。ここで、$var\,(F)$ は式 F に現われるすべての変数の集合である。節集合 $\mathcal{S}$ に値域限定の条件を満たさない節が含まれる場合には、充足性について $\mathcal{S}$ と等価で、かつ、すべての節が値域限定の条件を満たすような節集合に変換することができる。この変換は、 dom という予約された述語を導入し、以下のように行なう。

- 値域限定の条件を満たさない節
  $A_1, \ldots, A_n \rightarrow B_1; \ldots; B_m$ を、

  $$A_1, \ldots, A_n, dom\,(X_1), \ldots, dom\,(X_k) \rightarrow B_1; \ldots; B_m$$

  で置き換える。ただし、

  $$\{X_1, \ldots, X_k\} = \bigcup_{j=1}^{m} var\,(B_j) \setminus \bigcup_{i=1}^{n} var\,(A_i)$$

  である。

- 節集合 $\mathcal{S}$ 中に現れる定数記号 (定数記号が一つも現れない場合は、定数記号 'a') の集合を $consts(\mathcal{S})$、関数記号の集合を $functs(\mathcal{S})$ とするとき、節集合 $\mathcal{S}$ に dom 定義節集合

  $$\{true \rightarrow dom\,(K) \mid K \in consts(\mathcal{S})\} \quad \cup$$
  $$\{dom\,(X_1), \ldots, dom\,(X_n) \rightarrow \{dom\,(F_n(X_1, \ldots, X_n)) \mid F_n \in functs(\mathcal{S})\}$$

を加える。

モデル生成法は、与えられた入力節 (MG 節と呼ぶ) 集合 $\mathcal{S}$ に対するモデルを、次のような手続きに従って構成的に求めようとするものである。以下では、構成途中のモデルを M で表し、これをモデル候補と呼ぶ。また、モデル候補の集合を $\mathcal{M}$ で表す。

1. $\mathcal{M}$ の要素として、初期モデル候補 $M_0 = \emptyset$ を設定する。
2. $\mathcal{M}$ のある要素 M に対し、後述のモデル拡張規則あるいはモデル棄却規則のいずれかが適用可能なとき、その規則を適用して $\mathcal{M}$ を更新する。
3. 上記 2 のステップを可能な限り繰り返す。
4. $\mathcal{M}$ のすべての要素 M に対して、いずれの規則も適用できなくなったとき、手続きは終了する。

この手続きが終了したとき、 $\mathcal{M}$ が空になっていれば、 $\mathcal{S}$ にはモデルが存在しない (すなわち、充足不能である) ことが結論できる。さもなければ、すべての要素 $M \in \mathcal{M}$ がいずれも $\mathcal{S}$ のモデルとなっている。

ここで、モデル拡張規則とモデル棄却規則は以下のとおりである。

- モデル拡張規則：モデル候補 M に対して、非負節 $A_1, \ldots, A_n \rightarrow B_1; \ldots; B_m$ ならびに、ある基礎代入 σ が存在して、 σA_i $(1 \leq i \leq n)$ がすべて M で充足され、かつ σB_j $(1 \leq j \leq m)$ のいずれも M で充足されないならば、 M を m 個のモデル候補 $M \cup \{\sigma B_j\}$ $(1 \leq i \leq m)$ で置き換えて拡張 (場合分け) する。
- モデル棄却規則：モデル候補 M に対して、負節 $A_1, \ldots, A_n \rightarrow \mathit{false}$ ならびに、ある基礎代入 σ が存在して、 σA_i $(1 \leq i \leq n)$ がすべて M で充足されるならば、 M を棄却する。

ここで、前件 $A_1, \ldots, A_n$ が M で充足されるような σ を求める操作を連言照合と呼ぶ。

例として次の節集合 (問題 S と呼ぶ) に関するモデル生成法の証明木を図 1.1に示す。

$$
\begin{aligned}
C_1: &\quad \mathit{true} \rightarrow p(a,a); q(b).\\
C_2: &\quad r(X, f(X)) \rightarrow \mathit{false}\ .\\
C_3: &\quad p(X,X), p(X,Y) \rightarrow r(X, f(Y)).\\
C_4: &\quad q(X) \rightarrow p(f(X), f(X)).
\end{aligned}
$$

1.3 モデル生成型定理証明系 MGTP

1.3.1 MGTP の基本構成

モデル生成法の推論過程で動的に生成されるのは、モデル候補要素としてのアトムのみである。このアトムは、値域限定条件を満たす MG 節集合を対象とする場合、常に変数を含まな

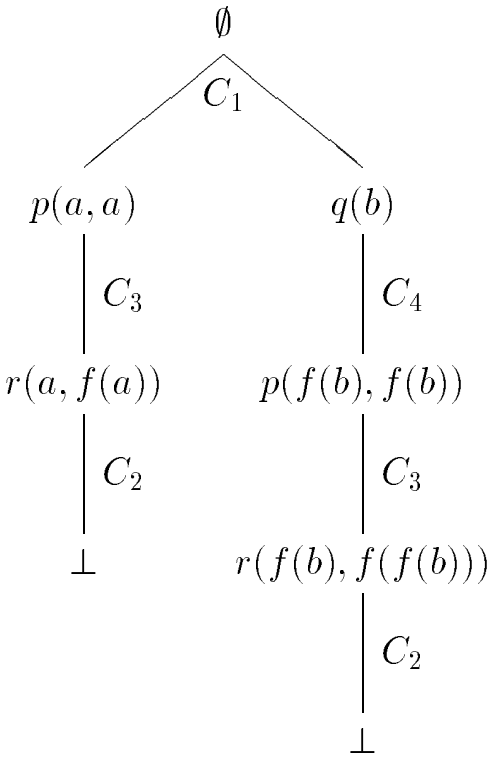

図 1.1: 問題 S の証明図

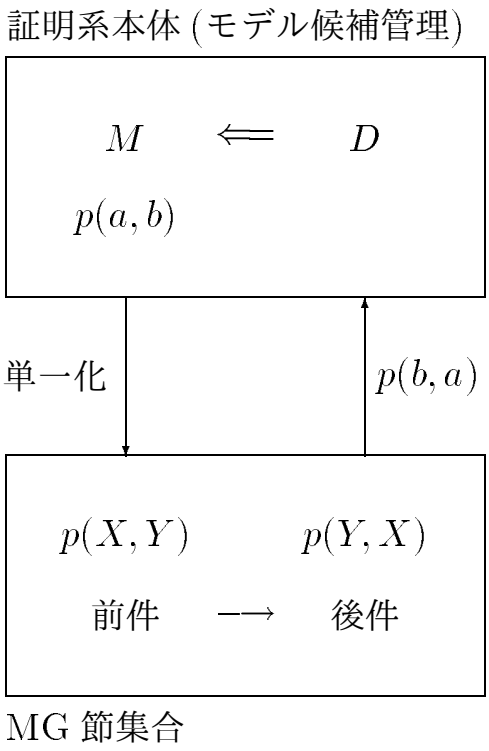

図 1.2: 証明系の KL1 による実現

い基底アトムとなる。節の前件の充足可能性テスト (連言照合) では、 MG 節の前件リテラルとモデル候補の要素であるアトムの単一化 (照合) が行なわれる。

さて、 KL1 ゴールと KL1 節の頭部間の頭部単一化は、並行動作における同期機構を簡潔に実現する目的で、 KL1 節側の変数しか束縛を許さない一方向の単一化になっている。したがって、 KL1 で実装する場合は、 MG 節を KL1 節で表現し、連言照合は KL1 節の頭部単一化を利用するのが簡便かつ効果的と考えられる。これにより、 MG 節の複製に伴う新変数の導入も KL1 実行系に委ねることができる。

そこで、モデル生成型定理証明系 MGTP を KL1 で実現する際、図 1.2のような基本構成をとるのが適切である。すなわち、モデル候補 M を管理する証明系本体と MG 節集合を表現する KL1 節の部分とを分離し、図中の MG 節 C_n : $p(X,Y) \rightarrow p(Y,X)$ は、以下のような KL1 節で表現する[1]。

```
c(n,p(X,Y),R):-true|R=p(Y,X).
```

これにより、節の前件の充足可能性テストにおける単一化は、たとえば証明系本体が M からアトム $p(a,b)$ を選び出し、 MG 節集合部の KL1 節を呼んで、その頭部で表現された前件リテラル $p(X,Y)$ に照合させる、というふうに実現することができる。

[1]実際には、照合が失敗する場合のために最後に otherwise 節が置かれる。

ここで、 $p(X,Y)$ と $p(a,b)$ の照合が成功すると、後件インスタンス $p(b,a)$ が証明系本体に返される。 $p(b,a)$ が M によって充足されなければ M を拡張すべきアトムの候補となるので、証明系本体で一旦モデル拡張候補集合 D に格納する[2]。

1.3.2 KL1 による MG 節集合の表現

前件が 2 個以上のリテラルからなる一般の場合、例えば、

$$C_n: \quad p(X,X), q(X,Z) \rightarrow p(X,Z).$$

のような MG 節についても、

```
c(n,(p(X,X),q(X,Z)),R):-true|R=p(X,Z).
```

のような 1 本の KL1 節で表現することができよう。この場合、連言照合の度にモデル候補中からアトムの対 $\langle A_i, A_j \rangle$ を作ることにすれば前件全体を一度に照合することができる。しかしこの方法には冗長性が含まれている。たとえば $\langle p(a,b), A_j \rangle$ の対はすでに第 1 リテラルにおいて $p(X,X)$ との照合で失敗することが明らかであるにもかかわらず、すべての A_j について無駄に連言照合を実行してしまうからである。

このような冗長性を省くためには、連言照合を前件リテラル個別に実行できるように、1 本の MG 節を複数の KL1 節に分離して表現すればよい。ただし、この場合には前件リテラル間の共有変数の同一性をいかに保つかという問題が生じる。これを解決するには、たとえば、

```
c(n,p(X,X),[],R):-true|R=(1:[X]).
c(n,q(X,Z),1:[X],R):-true|R=p(X,Z).
```

のような 2 本の KL1 節で表現すればよい。ここでは、 X が第 1 リテラルと第 2 リテラルおよび後件との共有変数であり、第 2 リテラルに対応する`c/4` の第 3 引数で $1:[X]$ を受けることにより、共有変数 X の同一性を表現している。

この表現方法に基づいて、前出の問題 S の MG 節集合を KL1 節集合で表現した実例を図 1.3に示す。

$c(N,A,V,R)$ において、 N は節番号、 A は前件リテラル、 V はこの前件リテラルの番号と束縛済み共有変数のリスト、 R は照合が成功した際証明系本体に返される値である。

1.3.3 証明系本体の実現

証明系本体は、図 1.4および 1.5に示す KL1 プログラムとして実現することができる。

メインループの`mgtp/5` は、モデル拡張候補集合 D、モデル候補 M、負節番号のリスト Cn、非負節番号のリスト Cg を受け、結果を Res に返す。 D が空の場合、 MG 節集合は M で充足可能 (M がモデル) であるという結果を得てメインループを終了する。 D が空でない場合、`pickup/3` によりモデル拡張候補 Δ を一つ選び、これに対し`subsTest/3` により M のもとでの

[2]実際には、後件の充足性テストはそれがモデル拡張候補として選ばれる直前に行なえばよい。

```
c(1,true,[],R):-true|R=(p(a,a);q(b)).
c(2,r(X,f(X)),1:[X],R):-true|R=false.
c(3,p(X,X),[],R):-true|R=(1:[X]).
c(3,p(X,Y),1:[X],R):-true|R=r(X,f(Y)).
c(4,q(X),[],R):-true|R=p(f(X),f(X)).
otherwise.
c(_,_,_,R):-true|R=fail.
```

図 1.3: 問題 S の KL1 節表現

充足性を調べる。もし Δ が M で充足されているならば、モデル拡張候補集合の残り $D1$ について再びメインループに入る。そうでなければ、 Δ が選言である場合、caseSplit/6 により場合分けを行なう。 Δ がアトムである場合、 M はこれを含むように拡張される。拡張されたモデル候補 $[\Delta|M]$ に対して、モデル棄却規則が適用できれば、 MG 節集合は M のもとで充足不可能である (M はモデルになり得ない) という結果を得てメインループを終了する。モデル棄却規則が適用できない場合、拡張されたモデル候補 $[\Delta|M]$ に対して、前件の連言照合を適用し、新たなモデル拡張候補集合 New とする。これをaddNew/3 によってモデル拡張候補集合の残り $D1$ に加え $NewD$ として再びメインループに入る。

連言照合手続きcjm/5 は、節番号 N のリストとモデル候補 M を受け、リスト中のすべての節について連言照合cjm1/6 を行ない、結果を差分リスト $Rh-Rt$ に詰めて返す。 1 本の MG 節に対する連言照合手続きcjm1/6 では、各前件リテラルに対し、 M 中のすべてのアトムについて連言照合を行なう。このとき、 MG 節集合の KL1 表現c/4 が呼ばれる。当該リテラルについて M 中のアトム A によるc/4 での照合結果が $fail$ の場合、このリテラル - アトム対の照合結果は空である。共有変数情報 (_ : _) が返された場合、引き続くリテラルについての連言照合を進める。それ以外の場合、前件全体の連言照合が成功して後件が返されているので、これを $Rh-Rm$ の差分リストに詰めて返す。当該リテラルについて M 中の他のアトムによる連言照合結果は、差分リスト $Rm-Rt$ に詰めて返される。

1.4 連言照合における冗長性除去

1.4.1 連言照合の冗長性

基本 MGTP の連言照合では、前件リテラルとモデル候補 M 中のアトムの同じ組合せに対して重複した計算が行なわれる可能性がある。以下では、このような連言照合における冗長計算を防ぐために考案した RAMS 法 [FH91] と MERC 法について述べる。

```
mgtp(D,M,Cn,Cg,Res):-true|
 empty(D,E),
 (E=yes->
   Res=sat;
  otherwise;true->
   pickup(D,Delta,D1),
   subsTest(Delta,M,S),
   (S=yes->mgtp(D1,M,Cn,Cg,Res);
    S=no,Delta=(_;_)->
     caseSplit(Delta,D1,M,Cn,Cg,Res);
    otherwise;true->
     cjm(Cn,Cn1,[Delta|M],F,[]),
     (F=[false|_]->
       Res=unsat;
      otherwise;true->
       cjm(Cg,Cg1,[Delta|M],New,[]),
       addNew(New,D1,NewD),
       mgtp(NewD,[Delta|M],Cn1,Cg1,Res)))).

caseSplit((A;B),D,M,Cn,Cg,Res):-true|
 caseSplit(A,D,M,Cn,Cg,R1),
 (R1=sat->Res=sat;
  otherwise;true->
   caseSplit(B,D,M,Cn,Cg,Res)).
otherwise.
caseSplit(A,D,M,Cn,Cg,Res):-true|
 addNew([A],D,NewD),
 mgtp(NewD,M,Cn,Cg,Res).

subsTest((A;B),M,S):-true|
 subsTest(A,M,S1),
 (S1=yes->S=yes;
  otherwise;true->subsTest(B,M,S)).
otherwise.
subsTest(A,[A|_],S):-true|S=yes.
otherwise.
subsTest(A,[_|M],S):-true|subsTest(A,M,S).
subsTest(_,[],S):-true|S=no.
```

図 1.4: 基本 MGTP 証明系

```
cjm([N|Cs],Cs1,M,Rh,Rt):-true|
 Cs1=[N|Cs2],
 cjm1(N,M,[],M,Rh,Rm),
 cjm(Cs,Cs2,M,Rm,Rt).
cjm([],Cs1,_,Rh,Rt):-true|Cs1=[],Rh=Rt.

cjm1(N,[A|M],V,Mh,Rh,Rt):-true|
 problem:c(N,A,V,R),
 (R=fail->Rh=Rm;
  R=(_:_)->cjm1(N,Mh,R,Mh,Rh,Rm);
  otherwise;true->Rh=[R|Rm]),
 cjm1(N,M,V,Mh,Rm,Rt).
cjm1(_,[],_,_,Rh,Rt):-true|Rh=Rt.
```

図 1.5: 基本 MGTP 証明系の連言照合部

1.4.2 RAMS 法

RAMS 法は、前件の照合の履歴を記憶する機構を設けることにより、連言照合における重複計算を避けるものである。図 1.6にこの方法を図示している。

一般に、 MG 節の前件 $A_1, \ldots, A_n$ の各リテラル A_i につき一つのインスタンススタック S_i を割り当てる。各 S_i には $A_1, \ldots, A_i$ のモデル候補 M との照合結果 (成功した場合の変数束縛情報) が蓄えられる。 S_1 には A_1 の照合結果が格納され、 $A_i(i > 1)$ の照合では、 S_{i-1} に蓄えられた $A_1, \ldots, A_{i-1}$ の各照合結果のもとで、 A_i 自体と M との照合を行なう。以下、 $A_i(i > 1)$ に対するこの照合演算を $S_{i-1} \circ M$ と表す。

各インスタンススタック S_i は、前ステージまでに積まれた部分 S_i^{k-1} と、現在のステージにおいて積まれる部分 δS_i^k との二つの部分に分けられる。ここで、指標 k はメインループのステージ番号を表す。また、前ステージまでのモデル候補 M^{k-1} を拡張するアトムをモデル拡張アトムと呼び、 Δ^k で表わす。

A_1 における作業 T_1^k は、 A_1 と Δ^k との照合であり、結果が S_1 に積まれる。各リテラル $A_i(2 \leq i \leq n-1)$ における作業 T_i^k は、連言照合の後、次のようにスタックの更新を行なう。

$$\begin{aligned} \delta S_i^k &:= \delta S_{i-1}^k \circ (\Delta^k \cup M^{k-1}) \cup S_{i-1}^{k-1} \circ \Delta^k \\ S_i^k &:= S_i^{k-1} \cup \delta S_i^k \end{aligned}$$

ただし、 T_n は連言照合の最後であり、照合結果に対応する後件が得られた後は $A_1, \ldots A_n$ の照合計算結果自体をスタックに積む必要がない。したがって、最後のリテラル A_n に対するスタック S_n も実際に割り付ける必要はない。

こうして、作業列 $T_1 \ldots, T_n$ をこの順に実行することにより、前件 $A_1 \ldots, A_n$ の連言照合を重複なしに実現することができる。図 1.6は、 2 ステージめで、 M にアトム 1 が積まれ、 Δ^2 がアトム 2 であるという状況を表している。まず、 A_1 と Δ^2 の照合が成功し、その結果 (図中、

簡単のため 2 と表記) が δS_1^2 として S_1 に積まれる。この δS_1^2 のもとで A_2 と Δ^2 の照合が成功し、その結果 (図中 22 で表す) が δS_2^2 の一つとして積まれている。

上で述べた方法と同様の効果を狙ったものに RETE があるが、これはホーン節に限定されている。一方、 RAMS 法はスタックを用いているので、非ホーン節に対しても容易に対応できる。

RAMS 法に基づいて、基本 MGTP 証明系の連言照合部を修正したものを図 1.8に示す。

1.4.3 MERC 法

MERC 法では、一般に MG 節の前件 $A_1, \ldots, A_n$ に対し、 M と Δ に対する連言照合を図 1.9に示すような組合せで行う。本図の第 1 行に示すパタンは、 A_1 に Δ を、 $A_2, \ldots, A_n$ に M 中のアトムを照合させることを表す。ここで、 $A_1, \ldots, A_n$ の少なくとも一つは Δ と照合するような組み合わせのみを選ぶようにし、 $A_1, \ldots, A_n$ のすべてが M に含まれるアトムと照合されるような組み合わせを排除すれば、ステージ間冗長性が避けられるわけである。

以下では、 Δ と照合させるリテラルを入口リテラルと呼び、 M 中のアトムを照合させるリテラルを後続リテラルと呼ぶことにする。連言照合は、まず入口リテラルに Δ を照合させ、これが成功した場合に後続リテラルを M 中のアトムで照合させるように進める。しかしながら、前節までに示した KL1 節表現ではリテラル順序が一つに固定されており、任意のリテラルを入口リテラルとすることができない。したがって、 MERC 法においては 1 本の MG 節に対し、入口リテラルの異なる取り方ごとにそれぞれ別個の KL1 節として表現する必要がある。

図 1.9中 (イ) に属する、入口リテラルが一つのパタンについては、各パタンにつき、 $A_1, \ldots, A_n$ の順序を入れ換えて入口リテラルを先頭とする 1 本の MG 節を考え、これに対応する KL1 節を用意する。一方、図中 (ロ) に属する、複数の前件リテラルが入口リテラルとなるパタンについては、 KL1 節を省ける場合がある。たとえば、二つのリテラル A_j, A_k が同時に Δ と照合するのは、これらが単一化可能な場合 ($A_j = A_k$ と表す)、すなわちファクタリング可能な場合に限られる。したがって、図中 (ロ) に属するパタンのうち、単一化不可能な入口リテラル対を含むものについては KL1 節を用意しなくてよい。一方、単一化可能な複数の入口リテラル $A_{i1} \ldots A_{il}$ については、ファクタリングを行なった結果のリテラル A_r $(A_{i1} = A_{i2} = \ldots = A_{il})$ を計算し、これを一つの新たな入口リテラルとした KL1 節を用意することにする。

例として、問題 S の KL1 節表現を図 1.10に示しておく。 MERC 法を採用する場合、基本 MGTP 証明系の連言照合部の一部は、図 1.11のように修正される。

1.5 RAMS 法と MERC 法の比較

RAMS 法では前件リテラル $A_1, \ldots, A_{i-1}$ に対するモデル候補 M との照合結果を記憶し、その記憶をもとに A_i と最新のアトムである Δ の照合を行なうのに対し、 MERC 法では $A_1, \ldots, A_{i-1}$ に対する照合を再計算するという冗長性がある。これを M-M 冗長性と呼ぶ。さらに、 MERC 法では 1 本の MG 節を入口リテラルの数だけ複製しているので、上記の冗長性は複製分だけ重複して現れる。

しかしながら、 MERC 法では入口リテラル A_i と Δ との照合を優先し、後続リテラル $A_1, \ldots, A_{i-1}$ と M 中のアトムとの照合を後に行なうのに対し、 RAMS 法では、与えられた MG 節の前件

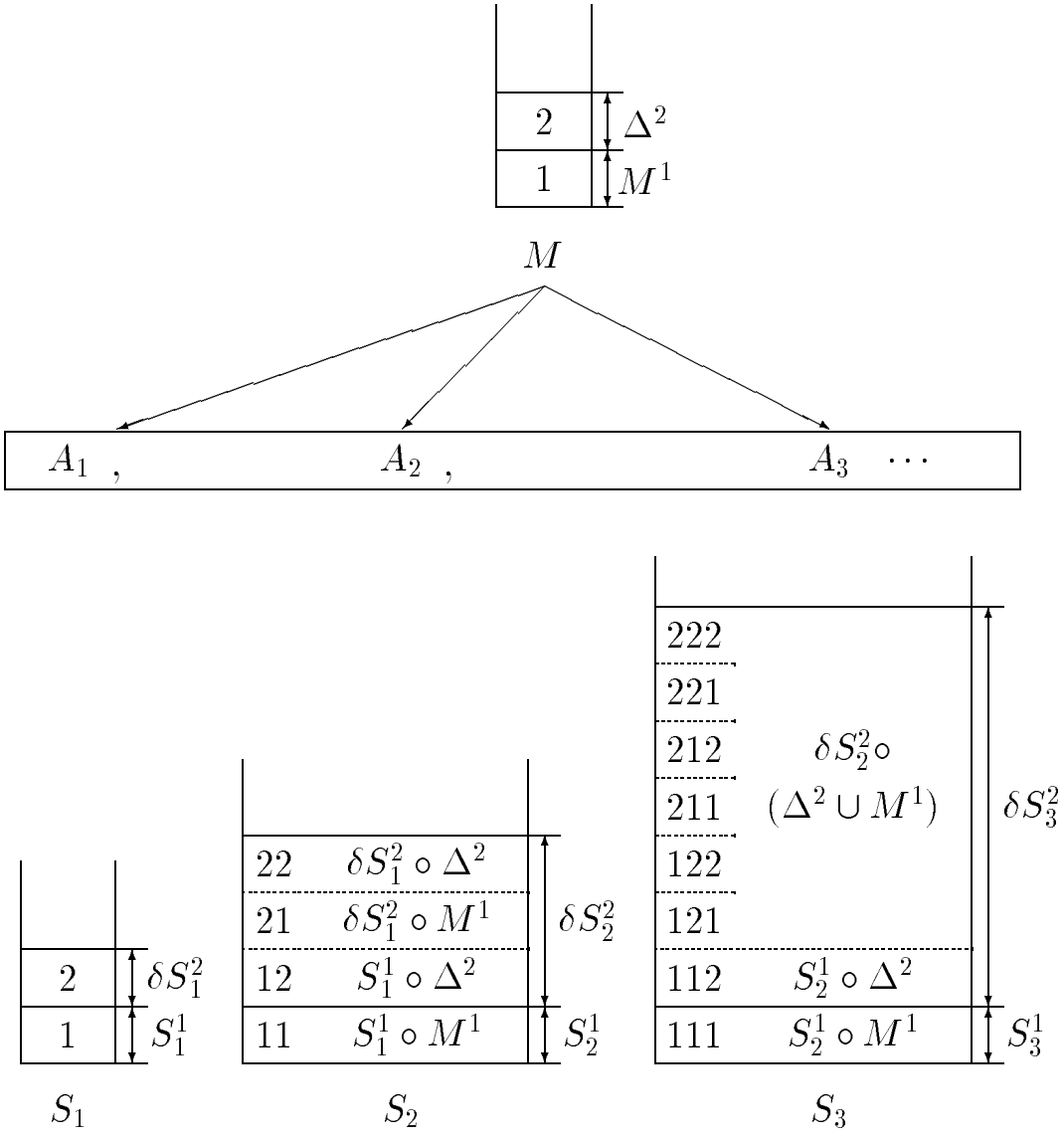

図 1.6: インスタンススタック

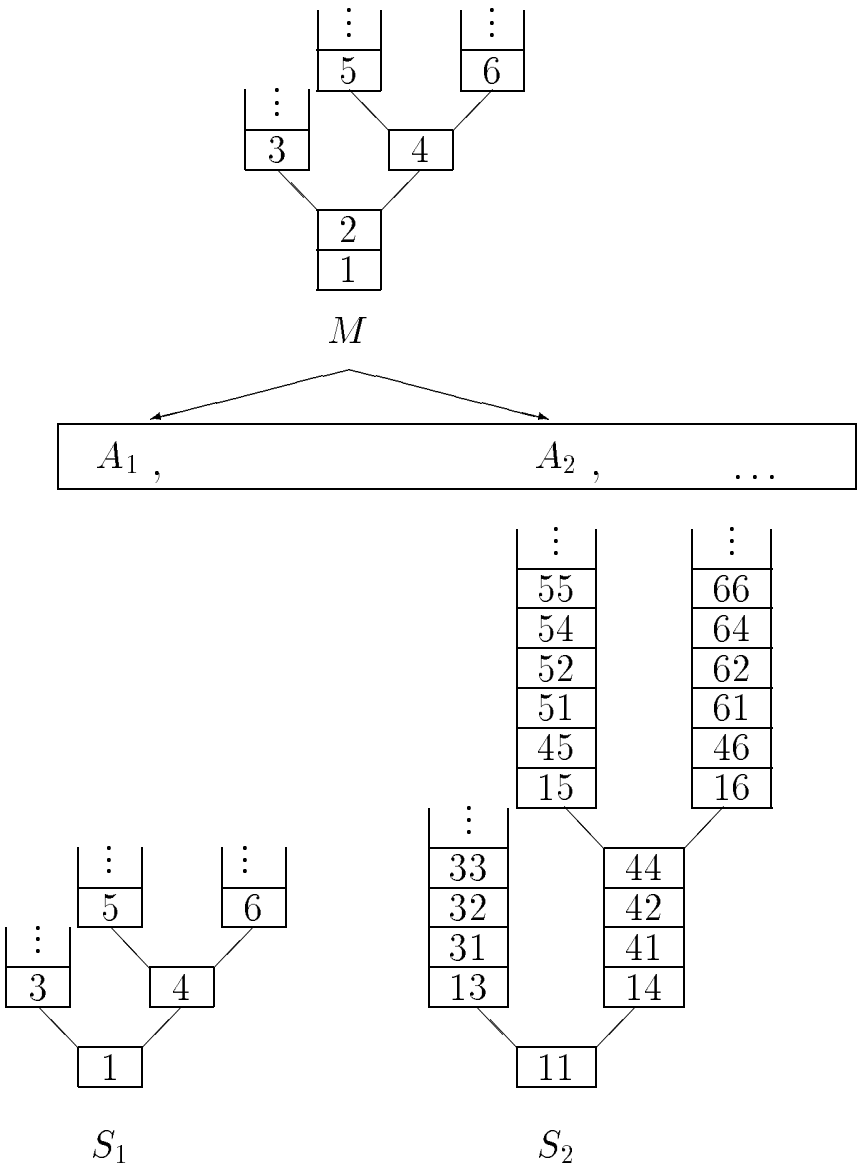
5
6
3
4
2
1
M
A_1 ,
A_2 ,
...
55
54
52
51
45
15
66
64
62
61
46
16
33
32
31
13
44
42
41
14
11
5
6
3
4
1
S_1
S_2

図 1.7: 分岐スタック

```
cjm([N:L|Cs],Cs1,Delta,M,Rh,Rt):-true|
 Cs1=[N:L1|Cs2],
 cjm1(N,L,L1,Delta,M,[[]],[],Rh,R1),
 cjm(Cs,Cs2,Delta,M,R1,Rt).
cjm([],Cs1,_,_,Rh,Rt):-true|Cs1=[],Rh=Rt.

cjm1(N,[Sn|Ls],Ls1,Delta,M,Sp,Dp,Rh,Rt):-true|
 Ls1=[NewSn|Ls2],
 cjm2(N,[Delta|M],Dp,NewSn,S1,Dn,D1),
 cjm2(N,[Delta],Sp,S1,Sn,D1,[]),
 cjm1(N,Ls,Ls2,Delta,M,Sn,Dn,Rh,Rt).
cjm1(N,[],Ls1,Delta,M,Sp,Dp,Rh,Rt):-true|
 Ls1=[],
 cjm2(N,[Delta|M],Dp,Rh,R1,_,_),
 cjm2(N,[Delta],Sp,R1,Rt,_,_).

cjm2(N,As,[V|Vs],Sh,St,Dh,Dt):-true|
 cjm3(N,As,V,Sh,S1,Dh,D1),
 cjm2(N,As,Vs,S1,St,D1,Dt).
cjm2(_,_,[],Sh,St,Dh,Dt):-true|
 Sh=St,Dh=Dt.

cjm3(N,[A|As],V,Sh,St,Dh,Dt):-true|
 problem:c(N,A,V,R),
 (R=fail->Sh=S1,Dh=D1;
  R=(_:_)->Sh=[R|S1],Dh=[R|D1];
  otherwise;true->Sh=[R|S1]),
 cjm3(N,As,V,S1,St,D1,Dt).
cjm3(_,[],_,Sh,St,Dh,Dt):-true|
 Sh=St,Dh=Dt.
```

図 1.8: 連言照合部の RAMS 版

	A_1	A_2	A_3	...	A_n
(イ)	Δ	M	M	...	M
	M	Δ	M	...	M
	M	M	Δ	...	M
			...		
	M	M	M	...	Δ
(ロ)	Δ	Δ	M	...	M
	M	Δ	Δ	...	M
			...		
	Δ	Δ	Δ	...	Δ

図 1.9: MERC 法による連言照合

```
c(c1,true,      [],R):-true|R=(p(a,a);q(b)).
c(c2,r(X,f(X)),[],R):-true|R=false.
c(c3_1,p(X,X),   [],R):-true|R=(1:[X]).
c(c3_1,p(X,Y),1:[X],R):-true|R=r(X,f(Y)).
c(c3_2,p(X,Y),   [],R):-true|R=(1:[X,Y]).
c(c3_2,p(X,X),1:[X,Y],R):-true|R=r(X,f(Y)).
c(c3_1_2,p(X,X), [],R):-true|R=r(X,f(X)).
c(c4,q(X),      [],R):-true|R=p(f(X),f(X)).
otherwise.
c(_,_,_,R):-true|R=fail.
```

図 1.10: 問題 S の MERC 版 KL1 表現

```
cjm([N|Cs],Cs1,[Delta|M],Rh,Rt):-true|
 Cs1=[N|Cs2],
 problem:c(N,Delta,[],R),
 (R=fail->Rh=Rm;
  R=(_:_)->cjm1(N,M,R,M,Rh,Rm);
  otherwise;true->Rh=[R|Rm]),
 cjm(Cs,Cs2,[Delta|M],Rm,Rt).
cjm([],Cs1,_,Rh,Rt):-true|Cs1=[],Rh=Rt.
```

図 1.11: 連言照合部の MERC 版

リテラルの並びの順序によって照合順序が固定されており、 Δ との照合が優先されるということはない。その結果、 MERC 法では Δ による入口リテラルの照合が失敗したときには後続リテラルの照合が省略されるのに対し、 RAMS 法では後続リテラルの照合で成功する組み合わせのすべてについて入口リテラルと Δ との無駄な照合を繰り返す可能性がある。これを Δ 失敗冗長性と呼ぶ。

また、メモリ消費に関しては、 RAMS 法では証明実行時にインスタンススタックの分だけ余計にメモリを消費するのに対し、 MERC 法では入口リテラルの数の分だけ節の複製を必要とする。

このように、ステージ間冗長性、 M-M 冗長性、ならびに Δ 失敗冗長性、さらにメモリ消費特性が、証明系の全体性能にどのように影響するかは、対象の問題に強く依存しており、 RAMS 法と MERC 法の優劣は明確でない。加えて、 RAMS 法と MERC 法では、連言照合で組み合わされるモデル候補中のアトムの組み合わせ順序が異なるので、モデル拡張候補アトムの並べ換えを行なわない限り、一般には Δ の選ばれる順序が異なり証明も異なってくる。

1.6 まとめ

モデル生成型定理証明器 MGTP を並行論理型言語 KL1 で実現した。モデル生成法の採用により、値域限定を満たす節集合に対して出現検査付き単一化が不要となり照合操作で十分となることを利用し、 KL1 の特徴を活かすことによって一階述語論理の効率良い定理証明系を実現することが可能となった。 KL1 処理系を活用するための要点は次のとおりである。

- 入力節中の論理変数が KL1 変数で直接表現される。
- 入力節の単一化は KL1 節の頭部単一化として実行される。
- 入力節の複製時に必要な新変数は、 KL1 節の呼び出しメカニズムから自動的に得られる。

さらに、モデル生成法における連言照合を効率的に行なうため、 RAMS 法と MERC 法の二つの方式を考案した。いずれも連言照合のステージ間冗長性の除去能力については大差なく、 MGTP の実行効率を顕著に改善できる。

なお、 MGTP の並列化の話題については、グランド版 MGTP マニュアル、ノングランド版 MGTP マニュアル、および CMGTP マニュアルの該当部分、あるいはその他の参考文献 ([FHKF92] など) を参照されたい。グランド版 MGTP および CMGTP では OR 並列が、またノングランド版 MGTP では AND 並列がそれぞれ KL1 版で実装されており、いくつかのベンチマーク問題に対してほぼ線形に近い台数効果が達成されている。

参考文献

[HF95] 長谷川隆三, 藤田博, MGTP : 並列論理型言語 KL1 によるモデル生成型定理証明系, ICOT TR 885, 1994.

[CFS94] Chikayama, T., Fujise, T. and Sekita, D.: A Portable and Efficient Implementation of KL1, *Proc. 6th Int. Symp. on Programming Language Implementation and Logic Programming* (1994).

[FH91] Fujita, H. and Hasegawa, R.: A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm. *Proc. 8th ICLP,* pp. 535-548 (1991).

[MB88] Manthey, R. and Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog, *Proc. 9th CADE,* pp. 415-434 (1988).

[Sti88] Stickel, M. E.: A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, *Journal of Automated Reasoning,* 4, pp. 353-380 (1988).

[SSY93] Suttner, C., Sutcliffe, G. and Yemenis, T.: The TPTP Problem Library, *Proc. 12th CADE,* pp. 252-266 (1993).

[UC90] Ueda, K. and Chikayama, T.: Design of the Kernel language for the Parallel Inference Machine. *The Computer Journal,* Vol. 33, No. 6, pp. 494-500 (1990).

[FHKF92] M. Fujita, R. Hasegawa, M.Koshimura and H. Fujita, *Model Generation Theorem Provers on a Parallel Inference Machine*, In Proc. of FGCS92, 1992.

第 2 章

MGTP 入力言語仕様

この章では、 MGTP の入力言語の標準的シンタクスについて説明し、記述例を示す。

2.1 MGTP 入力節シンタクス

MGTP の入力節は、第 1 章の機能説明のところでも説明したように、正節 (Positive clause)、負節 (Negative clause)、混合節 (Neutral clause) の 3 種類があるが、基本的にすべての節は含意 (implicatoin) `-->` によって構成されている。含意の左辺を前件部 (Antecedent)、後件部を (Consequent) と呼ぶ。

正節の場合は、前件部を `true` に、また負節の場合は、後件部を `false` として記述する。値域限定の条件を満足しないような節については、あらかじめ`dom` などの述語を用いることによって条件を満足するような節に変換することが可能である。

また、前件部、後件部においては、各対象言語を直接呼び出すことも許されており、これにより整数間の比較演算などが可能になる。

以下は、 MGTP 入力節のシンタクスについて BNF で記述したものである。

MGTP-Clause ::=	Positive-Clause | Negative-Clause | Neutral-Clause
Positive-Clause ::=	' `true`' ' `-->`' Consequent '.'
Negative-Clause ::=	Antecedent ' `-->`' ' `false`' '.'
Neutral-Clause ::=	Antecedent ' `-->`' Consequent '.'
Antecedent ::=	Antecedent-Atom ',' Antecedent-Atom *
Antecedent-Atom ::=	MGTP-Atom | ' `{{`' [Guards] ' `}}`'
Consequent ::=	['`{{`' [Procedures] '`}}`' ' `->`'] Consequent-Group
Consequent-Group ::=	Consequent-Unit ',' Consequent-Unit *
Consequent-Unit ::=	Atom-Group | Disjunctive-Atom-Group
Disjunctive-Atom-Group ::=	Atom-Group ' ;' Atom-Group *
Atom-Group ::=	MGTP-Atom ',' MGTP-Atom *
MGTP-Atom ::=	Term
Guards ::=	Call ',' Call *
Procedures ::=	Call ',' Call *

```
%%  Problems/MSC/MSC006-1.mg  Ground-MGTP/pl input-clauses
%%  infinite_domain(no).
%%  max_number_of_doms_in_a_clause(2).
%%  total_number_of_doms(2).
%%  number_of_NonHorn_clauses(1).
%%  max_number_of_consequents_in_a_clause(2).
%%  number_of_negated_literals(0).

true-->dom(a),dom(b),dom(c),dom(d).
dom(A),dom(B)-->p(A,B);q(A,B).
p(A,B),p(B,C)-->p(A,C).
q(A,B),q(B,C)-->q(A,C).
q(A,B)-->q(B,A).
p(a,b)-->false.
q(c,d)-->false.

%%  Problems/MSC/MSC006-1.mg  EOF
```

図 2.1: MGTP 入力節の例

ここで、 Term は、 Prolog や KL1 の項を表す。すなわち述語に相当する。変数の表現も Prolog や KL1 のそれに準ずる。また、 Call は該当プログラムの呼び出し文を意味し、例えば、 Prolog では `{{X>Y}}` や `{{X1:=X+1,X1>Y}}` などのような記述が可能である。

なお、細かいシンタクスの相違については各バージョンのマニュアルを参照されたい。

2.2 例題

MGTP 入力節の例を図 2.1に示す。この問題は TPTP problem library[1] の MSC006-1(図 2.2) を MGTP 入力節に書き直したものである。

また、 8 クイーン問題を MGTP で記述した例を図 2.3 に示す。述語 `p(M,N)` は `M` 行 `M` 列にクイーンが置かれることを表している。はじめに正節として、各行における値の候補が選言の形式で記述されている。ここの例では、`{{ }}` で囲まれたプログラム呼び出しが用いられており、これにより、同じ行や同じ列、あるいは対角線上にクイーンが置かれたことを判定し、モデル候補を棄却するようになっている。

```
%--------------------------------------------------------------------------
% File     : MSC006=NonObv-1 : Released v0.0.0, Updated v0.11.5.
% Domain   : Miscellaneous
% Problem  : A ‘‘non-obvious’’ problem
% English  : Suppose there are two relations, P and Q. P is
%            transitive, and Q is both transitive and symmetric.
%            Suppose further the ‘‘squareness’’ of P and Q: any two
%            things are related either in the P manner or the Q
%            manner. Prove that either P is total or Q is total.
% Refs     : Pelletier F.J., and Rudnicki P. (1986), Non-Obviousness,
%            In Wos L. (Ed.), Association for Automated Reasoning
%            Newsletter (6), Association for Automated Reasoning,
%            Argonne, Il, 4-5.
% Source   : [Pelletier & Rudnicki, 1986]
% Names    : nonob.lop [SETHEO]
% Syntax   : Number of clauses           :    6 (   1 non-Horn)(   2 units)
%            Number of literals          :   12 (   0 equality)
%            Number of predicate symbols :    2 (   0 propositions)
%            Number of function symbols  :    4 (   4 constants)
%            Number of variables         :   10 (   0 singletons)
%            Maximal clause size         :    3
%            Maximal term depth          :    1
%--------------------------------------------------------------------------
input_clause(p_transitivity,hypothesis,
    [--p(X,Y),
     --p(Y,Z),
     ++p(X,Z)]).
input_clause(q_transitivity,hypothesis,
    [--q(X,Y),
     --q(Y,Z),
     ++q(X,Z)]).
input_clause(q_symmetry,hypothesis,
    [--q(X,Y),
     ++q(Y,X)]).
input_clause(all_related,hypothesis,
    [++p(X,Y),
     ++q(X,Y)]).
input_clause(p_is_not_total,hypothesis,
    [--p(a,b)]).
input_clause(prove_q_is_total,theorem,
    [--q(c,d)]).
%--------------------------------------------------------------------------
```

図 2.2: MSC006-1(TPTP problem library)

```
true --> p(1,1);p(1,2);p(1,3);p(1,4);p(1,5);p(1,6);p(1,7);p(1,8).
true --> p(2,1);p(2,2);p(2,3);p(2,4);p(2,5);p(2,6);p(2,7);p(2,8).
true --> p(3,1);p(3,2);p(3,3);p(3,4);p(3,5);p(3,6);p(3,7);p(3,8).
true --> p(4,1);p(4,2);p(4,3);p(4,4);p(4,5);p(4,6);p(4,7);p(4,8).
true --> p(5,1);p(5,2);p(5,3);p(5,4);p(5,5);p(5,6);p(5,7);p(5,8).
true --> p(6,1);p(6,2);p(6,3);p(6,4);p(6,5);p(6,6);p(6,7);p(6,8).
true --> p(7,1);p(7,2);p(7,3);p(7,4);p(7,5);p(7,6);p(7,7);p(7,8).
true --> p(8,1);p(8,2);p(8,3);p(8,4);p(8,5);p(8,6);p(8,8);p(8,8).

p(N,M),p(N1,M),{{N \= N1}} --> false.
p(N,M),p(N,M1),{{M \= M1}} --> false.
p(N,M),p(N1,M1),{{N \= N1,M \= M1,A:=N-N1,B:=M-M1,A=B}} --> false.
p(N,M),p(N1,M1),{{N \= N1,M \= M1,A:=N-N1,B:=M1-M,A=B}} --> false.
```

図 2.3: 8 クイーン問題の MGTP での記述例

参考文献

[1] Suttner, C., Sutcliffe, G. and Yemenis, T.: The TPTP Problem Library, *Proc. 12th CADE,* pp. 252-266 (1993).

第 3 章

MGTP/G マニュアル

本章では，MGTP/G のアルゴリズム概要を述べ操作説明を行なう．

MGTP/G は，値域限定性を満たす節集合を対象にしたモデル生成型定理証明器である．値域限定性を満たす節とは，後件部に現れる全ての変数が前件部に現れる節のことをいう．

MGTP/G には，SICStus Prolog 上で稼働する Prolog 版，KLIC 上で稼働する KLIC 版，PIM 上で稼働する KL1 版がある．

3.1 MGTP/G の機能説明

MGTP/G では，値域限定性を満たす節集合を対象としているため，出現検査付き単一化が不要となり，照合操作のみで証明を進めていくことができる．その結果，入力節中の変数を直接 KL1 変数で表現することができ，入力節に対する単一化を KL1 節の頭部単一化として実行できるなど，KL1 言語処理系を活用した効率の良い実装が可能となる．また，出現検査付き単一化が不要であることは，Prolog の組込み単一化を直接利用できることも意味し，Prolog 上でも効率的な実装が可能となる．

いずれの版も，MGTP 節によって与えられた問題は，前処理プログラムによって，Prolog 節集合もしくは KL1 節集合に変換される．そしてこれらは，MGTP/G 本体プログラムとリンクされ，証明が行なわれる．

3.1.1 Prolog 版

Prolog 版はモデル生成定理証明器の基本となる連言照合時の冗長計算を除去する方式 RAMS 法を用いて実装されており，戦略の違いから三つのタイプの処理系がある．

戦略 I 全てのモデル拡張候補によって拡張を行なう．得られるモデル候補にモデル棄却規則を適用する．

戦略 II モデル拡張候補の内一つを選び，拡張を行なう．残りのモデル拡張候補はスタック[1]に積まれ次回のモデル拡張の候補となる．拡張されたモデル候補についてモデル棄却規則

[1]各モデル候補について一つずつスタックを用意する．このスタックは対応するモデル候補が棄却されると破棄される．

を適用する．

戦略 III 選言を含まない全てのモデル拡張候補によって拡張を行なう．選言を含むモデル拡張候補はスタックに積まれる．得られるモデル候補 (一つしかない) にモデル棄却規則を適用する．もし，選言を含むモデル拡張候補しかなかった場合は，そのうちの一つを選び拡張する．残りのモデル拡張候補はスタックに積まれる．得られるモデル候補 (複数) にモデル棄却規則を適用する．

いずれも，MGTP 節より，Prolog 述語の呼出が可能である．

3.1.2 KLIC 版

KLIC 版はモデル生成定理証明器の基本となる NAIVE とモデル生成法の性能劣化要因である連言照合時の冗長計算を除去する方式 RAMS 法と MERC 法を搭載した RAMS, MERC の 3 タイプがある．いずれも基本形は NAIVE である．

RAMS 法は，前件の照合の履歴を記憶する機構を設けることにより，連言照合における冗長計算を避けるものである．しかし，欠点があり，照合の履歴を保持するための必要メモリ量が増大する傾向にある．これに対して，MERC 法は RAMS 法の欠点を解決したものである．MERC 法は入力節の前件リテラルを照合すべき重複組合せのパターンに分解し，そのパターンに従い連言照合を行なうものである．

以上の他に，次のような機能を持つ．

- 各処理方式の比較を公平に行なうための，証明図一致機能を持つ．
- 問題節から KL1 述語を呼び出すことができる．
- オプションにより，証明トレース，証明の統計情報の表示を行なうことができる．

3.1.3 KL1 版

機能は Prolog 版 (3.1.1参照) と， Prolog 呼出の代わりに KL1 呼出が行なえる点を除いて，全く同じ．

前件部では KL1 ガード述語，後件部では KL1 ボディ述語呼出が可能である．

3.2 MGTP/G の操作方法

3.2.1 Prolog 版

Prolog 版 MGTP/G (Prolog 版) はグランド版 MGTP を SICStus-Prolog 処理系 (Prolog 処理系) 上に実装したものである． Prolog 版は与えられた入力節 (MG 節) を Prolog 節に変換する MG 節 –Prolog 節変換, および Prolog 節に変換した MG 節と MGTP/G 本体のコンパイルにより実行可能となる．節変換，コンパイル，実行は全て Prolog 処理系の環境下において行なわれる．

以上の操作手順概略を図?? に示す．まず，(1) トランスレータ mgtp2pl を用いて MG 節を Prolog 節に変換する．次に，(2) Prolog 節を Prolog 処理系上のコンパイラを用いてコンパイルする．最後に，(3) あらかじめコンパイルしておいた本体を用いて実行する．

3.2.1.1 インストール

Prolog 版は Prolog 処理系上で動作するインタプリタである．そのためコンパイル，登録などのインストール作業は必要とせず，トランスレータ，本体などのソース・ファイルをコンパイル時に認識可能な任意のディレクトリ位置に転送すれだけでよい．

表 3.1 にインストールに必要なファイルを示す．ファイル構成は MGTP/G 本体 top.pl および MG 節を Prolog 節に変換するプログラム mgtp2pl.pl，そのユーティリティ群 mgtp2uty.pl からなる．

表 3.1: ファイル構成

ファイル名	ファイル内容	使用言語
topI.pl	MGTP/G 本体 (戦略 I)	Prolog
topII.pl	MGTP/G 本体 (戦略 II)	Prolog
topIII.pl	MGTP/G 本体 (戦略 III)	Prolog
mgtp2pl.pl	MG 節 –Prolog 節トランスレータ	Prolog
mgtp2uty.pl	mgtp2pl.pl のユーティリティ・プログラム	Prolog

3.2.1.2 操作方法

先に述べたように Prolog 版は Prolog 処理系上のインタプリタとして実現されているため，全ての操作は Prolog 処理系上で行なわれる．本節ではトランスレータによる節変換，Prolog 処理系コンパイラによる Prolog 節，本体のコンパイル方法を説明した後，実際に例題を解く過程を示す．以降に示す例および表記法は SICStus-Prolog 処理系によるものであり，他の Prolog 処理系では異なる場合がある．

3.2.1.2.1 準備

top.pl mgtp2pl.pl は Prolog 処理系上のインタプリタである．従って両者を使用するには，Prolog 処理系上のコンパイラを用いて事前にコンパイルする必要がある．[2]

```
compile(['mgtp2pl.pl','top.pl']).
```

mgtp2uty.pl は mgtp2pl.pl 内部において呼び出されコンパイルするため，同一ディレクトリ上に存在する限り，明記する必要はない．

3.2.1.2.2 MG 節 – Prolog 節 変換

[2] consult/1 による登録でもよい． compile/1,consult/1 などのコマンドの使用方法は Prolog 処理系に依存するため，附属のマニュアルを参照のこと．

MG 節 – Prolog 節トランスレータ mgtp2pl.pl を用いて MG 節を Prolog 節に変換する．Prolog 節は本体とともにコンパイルすることにより実行可能となる．

mgtp2pl(<MG ファイル>, <PROLOG ファイル>).

<MG ファイル> に MG 節で記述された問題のファイル名，<PROLOG ファイル> に Prolog 節を出力するファイル名を指定する．<MG ファイル>，<PROLOG ファイル> の拡張子（.mgtp，.pl）は省略することができる．さらに，<PROLOG ファイル> そのものを省略することも可能である．<PROLOG ファイル> を省略した場合 <MG ファイル>.pl を <PROLOG ファイル> として用いる．また，<PROLOG ファイル> に user を指定した場合，変換した Prolog 節は現在使用している端末上に表示される．

mgtp2pl による MG 節 –Prolog 節変換を行なった後，Prolog 処理系による後述のコンパイルを行なう．mgtp2pl.pl には，これら 2 つの過程を 1 命令で行なう仕様がある．

mgtpcomp(<MG ファイル>, <PROLOG ファイル>).

mgtpcomp は内部処理において mgtp2pl による節変換を行なった後，Prolog 処理系のコンパイラを用いて <PROLOG ファイル> をコンパイルする．mgtpcomp の <MG ファイル>，<PROLOG ファイル> も mgtp2pl 同様に拡張子，および <PROLOG ファイル> を省略することができる．

3.2.1.2.3 コンパイル

Prolog 節を Prolog 処理系のコンパイラを用いてコンパイル，登録する．

compile(<PROLOG ファイル>).

<PROLOG ファイル> に ?? の節変換により得られた Prolog 節のファイル名を指定する．<PROLOG ファイル> は，あらかじめコンパイルしてある本体 top.pl から呼び出され実行される．

3.2.1.2.4 実行

先の節変換，コンパイル過程を経て実行環境が整ったところで do(S). を実行することにより推論が開始される．

do(S).

推論を終えると実行結果として変数 S に反駁失敗 (sat) または成功 (unsat) のいずれかが報告される．報告内容を以下に示す．反駁に失敗した場合，検出されたモデル群 model([*AtomM,...,Aton* が変数 S に出力され，成功した場合，棄却されたモデル候補 rejected([*AtomM,...,AtomN*]) が出力される．

sat : S = [model([*Atom1,...,AtomL*]),...,model([*AtomM,...,AtomN*])].
unsat : S = [rejected([*Atom1,...,AtomL*]),...,rejected([*AtomM,...,AtomN*])].

3.2.1.2.5 操作例

以上の準備，節変換，コンパイル，実行の一連の操作例を示す．以下は 図 3.1 に示す例題 example.mgtp を解く例である．(誌面の例では都合上，ホーム，およびソースファイルのディレクトリ位置を $HOME, $HOME/mgtpg として表記している．)

```
p(A,B),p(B,C)-->p(A,C).
q(A,B),q(B,C)-->q(A,C).
q(A,B)-->q(B,A).
dom(A),dom(B)-->p(A,B);q(A,B).
p(a,b)-->false.
q(c,d)-->false.
true-->dom(a),dom(b),dom(c),dom(d).
```

図 3.1: 例題 (example.mgtp)

1. 準備

```
prompt[1]% cd $HOME/mgtpg
prompt[2]% ls -F
 example.mgtp      mgtp2pl.pl        mgtp2uty.pl       top.pl
prompt[3]% sicstus
SICStus 2.1 #8:  Tue May 11 21:04:52 JST 1993
{consulting $HOME/.sicstusrc...  }
{$HOME/.sicstusrc consulted, 0 msec 80 bytes}
| ?- compile(['mgtp2pl.pl','top.pl']).
{compiling $HOME/mgtpg/mgtp2pl.pl...}
{compiling $HOME/mgtpg/mgtp2uty.pl...}
{$HOME/mgtpg/mgtp2uty.pl compiled, 1750 msec 51216 bytes}
{$HOME/mgtpg/mgtp2pl.pl compiled, 3540 msec 99312 bytes}
{compiling $HOME/mgtpg/mgtp2x/top.pl...}
{Warning:  [FLIS,LIS] - singleton variables in do/7 in lines 17-18}
{Warning:  [False] - singleton variables in do1Decide/9 in lines 27-27}
{Warning:  [DM,ID,M] - singleton variables in satisfyClause/9 in lines 41-42}
{$HOME/mgtpg/top.pl compiled, 630 msec 11216 bytes}
yes
| ?-
```

2. MG 節 –KL1 節 変換

```
| ?- mgtp2pl('example.mgtp','example.pl').
[ program saved in "example.pl".  ]
yes
| ?-
```

3. コンパイル

```
| ?- compile('example.pl').
{compiling $HOME/mgtpg/example.pl...}
{$HOME/mgtpg/example.pl compiled, 530 msec 22288 bytes}
yes
| ?-
```

4. 実行

```
| ?- do(S).
S = [rejected([p(a,b),...,dom(a)]),rejected([p(a,b),...,dom(a)]),
···,rejected([q(c,d),...,dom(a)]),rejected([q(c,d),...,dom(a)])] ?
yes
| ?-
```

以上が準備，節変換から実行までの操作例である．ここで，例題の結果は変数 S に棄却されたモデル候補 [rejected([p(a,b),...,dom(a)]),...] を出力していることから例題 example.mgtp は unsat であることが分かる．もし反駁に失敗すれば変数 S にはモデルが出力されるはずである．

3.2.2 KLIC 版

KLIC 版の実行は既に紹介されている MGTP/G と同じく，与えられた入力節 (MG 節) を KL1 節に変換する KL1 コンパイル技術によって行なわれる．KL1 節に変換された MG 節は本体とともにコンパイル，リンクすることにより，実行オブジェクトを得ることができる．
以上の操作手順概略を図?? に示す．まず，(1) MG 節をトランスレータ mg2kl1 により KL1 節に変換する．次に，(2) KL1 節と本体 (NAIVE, RAMS, MERC のいずれか) を KLIC でコンパイル，リンクすることにより実行ファイル (a.out) を得る．

3.2.2.1 インストール

NAIVE, RAMS, MERC 本体ソースファイルの転送，およびトランスレータ mg2kl1 を登録する．

3.2.2.1.1 ファイル構成

表 3.2 にインストールに必要なファイルを示す．MG 節を KL1 節に変換するプログラム mg2kl1.pl および，NAIVE, RAMS, MERC の本体プログラムがある．(merc_alls_st.kl1 は，ある枝が反駁失敗を起こしても証明を中断することなく全てのモデルを求める全解探索型の MERC 本体である．)

各本体ソース・ファイルはコンパイル，リンク時に認識可能な任意のディレクトリ位置に転送すればよい．

3.2.2.1.2 mg2kl1 の登録

mg2kl1.pl は SICStus-Prolog で記述された MG 節 –KL1 節トランスレータである．SICStus-Prolog (または SICStus 相当の Prolog) でコンパイルした後，実行形式ファイル mg2kl1 として登録する．

表 3.2: ファイル構成

ファイル名	ファイル内容	使用言語
mg2kl1.pl	MG 節 –KL1 節トランスレータ	Prolog
naive_a_st.kl1	NAIVE 本体	KL1
rams_a2_st.kl1	RAMS 本体	KL1
merc_a_st.kl1	MERC 本体	KL1
merc_alls_st.kl1	全解探索型 MERC 本体	KL1

以下に登録手順を示す．(誌面の例ではホーム，ソースファイル，パスの通う各ディレクトリを，それぞれ $HOME, $HOME/mgtpg, $HOME/bin としている．)

mg2kl1 の登録例

```
prompt[1]% cd $HOME/mgtpg
prompt[2]% sicstus
SICStus 2.1 #8:  Tue May 11 21:04:52 JST 1993
{ consulting $HOME/.sicstusrc...  }
{$HOME/.sicstusrc consulted, 0 msec 80 bytes}
| ?- compile(mg2kl1).
{compiling $HOME/mgtpg/mg2kl1.pl ... }
{$HOME/mgtpg/mg2kl1.pl compiled, 3450 msec 92016 bytes}
yes
| ?- save.
{SICStus state saved in $HOME/mgtpg/mg2kl1}
prompt[3]% ls -F
 merc_alls_st.kl1   mg2kl1*           naive_a_st.kl1
 merc_a_st.kl1      mg2kl1.pl         rams_a2_st.kl1
prompt[4]% mv mg2kl1 $HOME/bin
prompt[5]%
```

3.2.2.2 操作方法

先に述べたように KLIC 版は mg2kl1 による MG 節 –KL1 節変換，KLIC による MG 節，本体のコンパイル，リンク過程を経て実行オブジェクトを獲得する．本節では変換，コンパイル，リンクおよび実行方法を簡単に説明した後，実際に例題を解く過程を示す．

3.2.2.2.1 MG 節 –KL1 節 変換

MG 節 –KL1 節トランスレータ mg2kl1 を用いて MG 節を KL1 節に変換する．KL1 節は本体とともにコンパイル，リンクすることにより実行オブジェクトとなる．

mg2kl1 <問題名> <実行形式>

<問題名> に MG 節で記述された問題のファイル名 (拡張子無し)，<実行形式> に使用する本体 (naive，rams，merc) のいずれかを指定する．mg2kl1 はファイル <問題名>.mg の MG

節を読み込み KL1 節に変換した後，ファイル <問題名>_<実行形式>.kl1 を出力する．このように mg2kl1 は使用する本体に合わせて MG 節を KL1 節に変換する為，同一 MG 節においても本体が異なる場合，再度変換の必要がなる．

3.2.2.2.2 コンパイル，リンク KL1 節と本体を KLIC を用いてコンパイル，リンクする．

```
klic -O2 <問題名>_<実行形式>.kl1 <実行形式>_a_st.kl1
```

<問題名> に問題のファイル名 (拡張子無し)，<実行形式> に naive，rams，merc のいずれかを指定する．コンパイル時のオプションとして -O2 をつけることにより，実行時間，ファイルサイズを短縮，減少させることができる．KLIC のオプションに関する詳細は klic -h とすることで得られる．その他 KLIC に関する詳細は附属のマニュアルを参照のこと．

3.2.2.2.3 実行

先の変換，コンパイル，リンクによって生成した実行オブジェクト **a.out** (オブジェクト名を指定した場合はそのオブジェクト名) を実行する．

```
a.out -h1m
```

問題により実行時にメモリ不足で中断する場合がある．そのときはオプションとして -h1m を付随させることにより実行時のメモリが増加され実行できる場合がある．(-h1m の付随により副作用として実行時間が若干短縮されるので注意すること．)

3.2.2.2.4 操作例

以上の変換，コンパイル，リンク，実行の一連の操作例を示す．以下は 図 3.2 に示す例題 example.mg を本体に merc_a_st.kl1 を使って解く例である．

```
p(A,B),p(B,C)-->p(A,C).
q(A,B),q(B,C)-->q(A,C).
q(A,B)-->q(B,A).
dom(A),dom(B)-->p(A,B);q(A,B).
p(a,b)-->false.
q(c,d)-->false.
true-->dom(a),dom(b),dom(c),dom(d).
```

図 3.2: 例題 (example.mg)

1. **MG 節 –KL1 節 変換**

```
prompt[1]% cd $HOME/mgtpg
prompt[2]% mg2kl1 example merc
prompt[3]% ls -F
 example.mg         merc_alls_st.kl1   mg2kl1*            naive_a_st.kl1
 example_merc.kl1   merc_a_st.kl1      mg2kl1.pl          rams_a2_st.kl1
prompt[4]%
```

2. コンパイル，リンク

```
prompt[4]% cd klic
prompt[5]% klic -O2 merc_a_st.kl1 example_merc.kl1
prompt[6]% ls -F
 a.out*            example_merc.kl1  merc_a_st.kl1      naive_a_st.kl1
 atom.c            funct.c           merc_alls_st.kl1   predicates.c
 atom.h            funct.h           mg2kl1*            predicates.o
 atom.o            funct.o           mg2kl1.pl          rams_a2_st.kl1
 example.mg        klic.db
prompt[7]%
```

3. 実行

```
prompt[7]% a.out -h1m
unsat

No.  of branches = 384
average length of a branch = 26
        minimum length of a branch = 6
        maximum length of a branch = 30
total cons counts for M = 1902
No.  of CJMs     = 106406
cjm calls        = 25878
cjm1 calls       = 88518
total cjm* calls = 114396

prompt[8]%
```

以上が変換から実行までの操作例である．ここで，例題の結果は unsat となり反駁に成功していることが分かる．もし反駁に失敗すれば結果は sat となる．また同時に得られる統計情報の意味を表 3.3に示す．

3.2.3 KL1 版

Prolog 版 (3.2.1を参照) と処理手順は，ほぼ同じである．

3.2.3.1 インストール

必要なファイルを，表 3.4に示す．

必要なファイルを一つのディレクトリに集め，リスナより take でInstall ファイルを指定する．

```
[17] take("Install").

[18] load["plib","listlib"] .
Load File : icpsi520::>sys>user>miyuki>KL1>MGTPG>plib.sav.1
Load File : icpsi520::>sys>user>miyuki>KL1>MGTPG>listlib.sav.1
```

表 3.3: 統計情報の意味

表示項目	表示内容
No. of branches	反駁証明木の分枝数
average length of a branch	枝の長さ (棄却されたモデル候補の大きさ) の平均値
minimum length of a branch	同 最短
maximum length of a branch	同 最長
total cons counts for M	モデル拡張を行なったアトムの総数
No. of CJMs	連言照合における KL1 節 c/4 の呼び出し総数
cjm calls	連言照合における KL1 節 cjm/5 の呼び出し総数
cjm1 calls	連言照合における KL1 節 cjm1/6 の呼び出し総数
total cjm* calls	cjm, cjm1 の呼び出し総数

表 3.4: ファイル構成

ファイル名	ファイル内容	使用言語
trans.kl1	MG 節 –KL1 節変換プログラム	KL1
Pretrans.kl1	MG 節 –KL1 節変換プログラムの前処理	KL1
mgtpI.kl1	インタプリタ本体 (戦略 I)	KL1
mgtpII.kl1	インタプリタ本体 (戦略 II)	KL1
mgtpIII.kl1	インタプリタ本体 (戦略 I)	KL1
s1.kl1	(ダミーの) 問題節	KL1
mgtpMac.mac	マクロ集	KL1 マクロ
termMemoryG.kl1	弁別木プログラム	KL1
plib.sav	ライブラリ	KL1(コンパイル済)
listlib.sav	ライブラリ	KL1(コンパイル済)
Startup	システム起動用ファイル	リスナ take ファイル
Install	インストール用ファイル	リスナ take ファイル

```
"miyuki::$plib$error" Updated
"miyuki::$plib$window" Updated
"miyuki::$plib$timer" Updated
"miyuki:: listlib" Updated
"miyuki:: plib" Updated
"miyuki::$plib$file" Updated
"miyuki::$plib$string_io" Updated
Load Succeeded

[19] compile["termMemoryG"] .
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>termMemoryG.kl1.1
!WARNING! no with_macro declaraion. assumed `pimos'.
[termMemoryG.kl1@0] Compile Module : tmG
=>  :- module tmG .
Compile Succeeded : tmG

"miyuki:: tmG" Updated
Total Number of Warning : 1
Compilation Time = 17558 [MSEC]

[20] inter([prompt("Input ENGINE Name>> "),getl(L)])';' compile([L,"s1"]) .
Input ENGINE Name>> mgtpI
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>mgtpI.kl1.1
!WARNING! no with_macro declaraion. assumed `pimos'.
[top.kl1@0] Compile Module : mgtp
=>  :- module mgtp .
Compile Succeeded : mgtp

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>s1.kl1.1
!WARNING! no with_macro declaraion. assumed `pimos'.
[s1.kl1@0] Compile Module : mgtp_p
=>  :- module mgtp_p .
Compile Succeeded : mgtp_p

"miyuki:: mgtp" Updated
"miyuki:: mgtp_p" Updated
Total Number of Warning : 2
Compilation Time = 18914 [MSEC]
```

```
[22] compile["trans","preTrans"],compile["mgtpMac"] .
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>mgtpMac.mac.1
[mgtpMac.mac@0] Compile Module : mgtpMac
=>  :- macro_module(mgtpMac) .
Compile Succeeded : mgtpMac

"miyuki:: mgtpMac" Updated
Compilation(s) Succeeded
Compilation Time = 22233 [MSEC]
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>trans.kl1.3
[trans1.kl1@0] Compile Module : mgtp_trans
=>  :- module mgtp_trans .
Compile Succeeded : mgtp_trans

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>preTrans.kl1.1
[preTrans.kl1@0] Compile Module : mgtp_pre_trans
=>  :- module mgtp_pre_trans .
Compile Succeeded : mgtp_pre_trans

"miyuki:: mgtp_pre_trans" Updated
"miyuki:: mgtp_trans" Updated
Compilation(s) Succeeded
Compilation Time = 57689 [MSEC]

[23] unload([mgtp_p,mgtp,mgtp_trans,mgtp_pre_trans,mgtpMac,tmG],L) .
Unload File : icpsi520::>sys>user>miyuki>KL1>MGTPG>mgtpI.sav.4
  miyuki::mgtp_p unloaded
  miyuki::mgtp unloaded
  miyuki::mgtp_trans unloaded
  miyuki::mgtp_pre_trans unloaded
  miyuki::mgtpMac unloaded
  miyuki::tmG unloaded
Unload Succeeded

[24]
```

3.2.3.2 操作方法

インストールが完了していれば，Startup ファイルをリスナよりtake すれば，必要なファイルがコンパイル又は，ロードされた後，新たなウィンドウ (MGTP ウィンドウ) が開かれる．

以下では，MGTPG というディレクトリに全てのファイルが既に格納されている．

3.2.3.2.1 起動

```
[5] cd("MGTPG").

[6] take("Startup").          : 起動の開始

[7] load["plib","listlib"] . : ライブラリのロード
Load File : icpsi520::>sys>user>miyuki>KL1>MGTPG>plib.sav.1
Load File : icpsi520::>sys>user>miyuki>KL1>MGTPG>listlib.sav.1
"miyuki::$plib$error" Loaded
"miyuki::$plib$window" Loaded
"miyuki::$plib$timer" Loaded
"miyuki:: listlib" Loaded
"miyuki:: plib" Loaded
"miyuki::$plib$file" Loaded
"miyuki::$plib$string_io" Loaded
Load Succeeded

[8] inter([prompt("Input ENGINE Name>> "),getl(L)])';' load[L] .
Input ENGINE Name>> mgtpI
Load File : icpsi520::>sys>user>miyuki>KL1>MGTPG>mgtpI.sav.4
"miyuki:: tmG" Loaded
"miyuki:: mgtp" Loaded
"miyuki:: mgtp_p" Loaded
"miyuki:: mgtpMac" Loaded
"miyuki:: mgtp_pre_trans" Loaded
"miyuki:: mgtp_trans" Loaded
Load Succeeded
 L ="mgtpI"

[9] pimos:: listener: go(at(109,199),char(70,30),"font:test_11") .
                              : ＭＧＴＰウィンドウの起動
```

3.2.3.2.2 MGTP 節のコンパイル

```
[6] compile("s2.mgtp").       : Ｓ２問題をコンパイル
** KL1 Compiler **
```

```
Parameter File : "icpsi520::>sys>user>miyuki>KL1>MGTPG>compile.param.2"

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPG>s2.mgtp.1
!WARNING! no with_macro declaraion. assumed ‘pimos’.
[s2.mgtp] Compile Module : mgtp_p
Compile Succeeded : mgtp_p

"miyuki:: mgtp_p" Updated
Total Number of Warning : 1
"miyuki:: mgtp" Updated
Relink Succeeded
Compilation Time = 14038 [MSEC]
```

3.2.3.2.3 実行

```
[7] mgtp:do(U,S).      : 証明開始 (充足不能なら U が unsat に, 充足可能なら
 S = S                 : S が sat に束縛される.
 U = unsat

[8]
```

第 4 章

MGTP/N マニュアル

本章では，MGTP/N のアルゴリズム概要を述べ操作説明を行なう．

MGTP/N とは，証明対象領域をホーン問題のみに限定したモデル生成型定理証明器である．ホーン問題とは，ホーン節[1]のみからなる節集合のことである．MGTP/G にあった値域限定性は，特に課せられない．

MGTP/N には，SICStus Prolog 上で稼働する Prolog 版と PIM 上で稼働する KL1 版がある．KL1 版には，内部記憶機構の違いから，モデル共有型とモデル分散型の二つのタイプがある．

4.1 MGTP/N の機能説明

4.1.1 Prolog 版

Prolog 版は，システム核部分がコンパクト (A4 1 ページ) であるのでユーザが容易に全貌を理解することができ，手を入れやすいという特徴がある．モデル候補やモデル拡張候補の格納は，全て Prolog の assert， retract 機能を使っているため，今後 Prolog の内部データベースへの参照・更新速度が向上すれば (例えば，インデックスアルゴリズムの改良により)，さらに推論速度が向上することが期待できる．

また，前向き及び後向き包摂検査，ソーティング及び削除戦略の組込み，連言照合順の切替え，を指定することができる．

4.1.2 KL1 版

KL1 版は，PIM の能力を最大限に引き出すことを目指している．モデル候補及びモデル拡張候補の保持の違いにより，モデル共有型とモデル分散型の二つのタイプがある．モデル共有型では，モデル候補とモデル拡張候補の全体を各プロセッサが保持している．一方モデル分散型では，モデル候補とモデル拡張候補は，全プロセッサに分散して保持されており，各プロセッサは，その一部を保持しているのに過ぎない．

[1]後件アトムの数が高々 1

```
(0)  Init: MD[i] := C_i for i = 1,...,k where {C_1,C_2,...,C_k} = {C | (true → C) ∈ S}
(1)  Init: g := 1; s := k;
(2)  while g ≤ s do begin
(3)      foreach A_1,...,A_n → C
(4)          foreach (i_1,...,i_n) s.t. ∀j(1 ≤ i_j ≤ g) and ∃j(i_j = g)
(5)              if ∃σ(置換)∀j(MD[i_j]σ = A_jσ and Cσ is new) then begin
(6)                  s := s + 1; MD[s] := Cσ; end; /* モデル拡張 */
(7)                  foreach A_1,...,A_m → false do
(8)                      foreach (i_1,...,i_m) s.t. ∀j(1 ≤ i_j ≤ s) and ∃j(i_j = s) do
(9)                          if ∃σ'(置換)∀j(MD[i_j]σ' = A_jσ') then ret urn (unsat);
                                 /* モデル棄却 */
(10)             end;
(11)     g := g + 1;
(12) end ;
(13) return (sat);
```

図 4.1: MGTP/N の逐次アルゴリズム

モデル共有型では，包摂検査時の通信コストを低く抑えられ，結果として高い (計算) 台数効果得ることができる．一方，モデル分散型では，包摂検査時の通信コストは高くなるものの，メモリ・スケーラビリティ (領域台数効果) 得ることができる．

いずれも，前向き包摂検査を内蔵しており，削除戦略を組込むことができる．

4.2 MGTP/N のアルゴリズム

本節では，逐次アルゴリズム，モデル共有型並列アルゴリズム，モデル分散型並列アルゴリズムについて述べる．なお，モデル共有型並列アルゴリズムについてのさらに詳しい説明は，[越村 95] にある．

4.2.1 逐次アルゴリズム

図 4.1に逐次アルゴリズムを示す．ホーン節のみを扱うので，場合分けによる分岐は考慮されていない．本節で示すいずれの版もこのアルゴリズムを元にしている．

ここで，MD はモデル拡張の適用によって生成された (後件) アトムを保持するための配列であり，$MD[1]$ から $MD[g-1]$ までがモデル候補を，$MD[g]$ から $MD[s]$ までがモデル拡張候補 (モデル候補に付加されるべき生成アトムの集合) を表す．モデル候補は空集合，モデル拡張候補は正節の後件アトムの集合で初期化される ((0)，(1))．

(3) から (6) までがモデル拡張に対応している．各混合節の前件部がモデル候補で充足されているか否かを判定し ((4) と (5) の条件の前半)，充足可能であれば後件部の包摂テストを

行ない ((5) の条件の後半)，包摂されなければ MD に加える ((6))．

(5) の条件部中の $C\sigma$ is new は，生成アトム $C\sigma$ が MD のどの要素にも包摂されない ($\not\exists\sigma'$(置換)$C\sigma = MD[i]\sigma'$ for $\forall i(1 \leq i \leq s)$) ことを表す．包摂テストについては，本論文ではこの前方包摂テストのみを扱い，$C\sigma$ が MD の要素を包摂している ($\exists i(1 \leq i \leq s)\exists\sigma'$(置換)$C\sigma\sigma' = MD[i]$) かを調べる後方包摂テストは扱わない[2]．

その後の (7) から (9) がモデル棄却に対応する．各負節の前件部がモデル候補またはモデル拡張候補で充足されているかを判定し，充足されていれば証明手続きは終了する．

4.2.2 モデル共有型並列アルゴリズム

4.2.2.1 ロックとアンロックを用いたアルゴリズム

図 4.1 の逐次アルゴリズムの並列化で最も簡単な方法は **while** ループの一つずつを並列に実行することである．ループ内部では，先ず混合節の連言照合 ((4) と (5) の前半)，包摂テスト ((5) の後半)，MD の更新 (6)，そして負節の連言照合 ((7) から (9)) が行なわれる．

我々は，これを混合節の連言照合を行なう生成プロセス (G プロセス) と負節の連言照合を行なう棄却テストプロセス (T プロセス) の二種のプロセス連結している generate-and-test 型計算と捉えてループ内部の並列アルゴリズムを考案した．

ここで，問題となる点は，並列に行なわれる包摂テストと MD の更新である．包摂テストおよび MD の更新は競合的に行なわれるので，ある種のロック機構が必要となる．

そこで，G プロセスが新たに生成したアトムを一旦蓄えておくバッファを設けることにした．図 4.2 にそのバッファ (Buf で表現) を用いた並列アルゴリズムを示す．Buf を介して G プロセス群が生成したアトムは T プロセス群に送られる．G プロセスは (G0) から (G6) までを，T プロセスは (T0) から (T8) までを繰り返す．G プロセスによって Buf に生成アトムが入れられ ((G5))，T プロセスによって Buf からそれらが取り出される ((T0))．

複数の G プロセスの排他制御は g に対するロックにより行なわれる ((G0))．(G1) から (G3) までの手続きは，逐次と変わらない．同様に (T4) から (T6) の処理も逐次と全く同じである．

包摂テストは，T プロセスで行なわれる ((T2))．MD は時々刻々更新されていくので，包摂テストを完全に行なうためには，MD をロックする必要がある．MD をロックした ((T1)) 後，Buf から取り出したアトム δ の包摂テストを行ない ((T2))，包摂されなければこれを MD に登録し，ロックを解除する ((T3))．

4.2.2.2 マスタープロセスを用いたアルゴリズム

このアルゴリズムに種々の検討結果を反映させたものが図 4.3である．このアルゴリズムは，G プロセス，T プロセスに加えてマスター (M) プロセスの三種からなっている．これらのプロセスの論理的結合関係を示したのが図 4.4である．中央の M プロセスを介して上部の G プロセス群と下部の T プロセス群が連結している．

図 4.3 では，プロセス間通信を表現するために，Occam 風チャネル記法を用いた．$channel!X$

[2] $C\sigma$ に包摂される $MD[i]$ を消去しなければならないので，データの一貫性を保つ必要が生じ，前方包摂テストより並列化は難しい．また，破壊代入を許していない KL1 では，消去の際にコピーが必要になる．

```
(0)(1)  図 4.1(0)(1) と同じ
(1)     (2)Init: Buf := ∅; b := 1;

process G
(G0) lock g; g' := g; g := g + 1; unlock g;
(G1) foreach A_1, ..., A_n → C do
(G2)     foreach (i_1, ..., i_n) s.t. ∀j(1 ≤ i_j ≤ g') ∧ ∃j(i_j = g') do
(G3)         if ∃σ∀j(MD[i_j]σ = A_jσ)
(G4)             then begin lock Buf;
(G5)             Buf[b] := Cσ; b := b + 1;
(G6)             unlock Buf; end;

process T
(T0) lock Buf; δ := Buf[b]; b := b − 1; unlock Buf;
(T1) lock MD;
(T2) if δ is new then begin
(T3)     s := s + 1; MD[s] := δ; s' := s; unlock MD;
(T4)     foreach A_1, ..., A_m → false do
(T5)         foreach (i_1, ..., i_m) s.t. ∀j(1 ≤ i_j ≤ s') ∧ ∃j(i_j = s') do
(T6)             if ∃σ'∀j(MD[i_j]σ' = A_jσ') then
(T7)                 return (unsat);
(T8) end else unlock MD
```

図 4.2: MGTP/N の並列アルゴリズム (lock&unlock)

```
(0)  図 4.1(0) と同じ
(1)  Init: s := k;

process M
(M0) Init: g := 1; g' := 1; b' := 1
(M1) MGC ! {g, New_g}; g := g + 1; goto (M1);
(M2) while New_g'[b'] = close then begin
(M3)     g' := g' + 1; b' := 1 end
(M4) MTC ! {New_g'[b'], s}; TMC ? s; goto (M2);

process G
(G0) MGC ? {g, New_g}; b := 1;
(G1) foreach A_1, ..., A_n → C do
(G2)     foreach (i_1, ..., i_n) s.t. ∀j(i_j ≤ g) ∧ ∃j(i_j = g) do
(G3)         if ∃σ∀j(MD[i_j]σ = A_jσ ∧ Cσ is new to MD[1, ..., m]) then begin
(G4)             New_g[b] := {Cσ, m}; b := b + 1 end;
(G5)     New_g[b] := close;

process T
(T0) MTC ? {{δ, m}, s'};
(T1) if δ is new to MD[m + 1, ..., s'] then begin
(T2)     MD[s' + 1] := δ; TMC ! s' + 1;
(T3)     foreach A_1, ..., A_m → false do
(T4)         foreach (i_1, ..., i_m) s.t. ∀j(i_j ≤ s') ∧ ∃j(i_j = s') do
(T5)             if ∃σ'∀j(MD[i_j]σ' = A_jσ') then
(T6)                 return (unsat);
(T7) end else TMC ! s';
```

図 4.3: モデル共有型 MGTP/N の並列アルゴリズム

で変数 X の値をチャネル $channel$ に出力し，$channel?X$ でチャネル $channel$ から変数 X に値を入力することを表す．MGC は M プロセスから G プロセスへの，MTC は M プロセスから T プロセスへの，TMC は T プロセスから M プロセスへの通信に用いられるチャネルである．これらの通信を利用することによって，図 4.2 で示されていたロック機能と等価な働きを実現することができる[3]．

M プロセスでは初期化の後，(M1) と (M2)〜(M4) が並行実行される．(M1) では各 G プロセスに連言照合の受け持ち範囲 (g で示される) を指示すると共に，生成されたアトムを蓄えるためのバッファ New_g を用意しこれも送付する．New_g は，各 G プロセス毎に用意されるので，この更新にロックは必要ではない ((G4))．

包摂テストは，G プロセスと T プロセスで分離して行なわれる ((G3) と (T1) の条件部)．包摂テストの内，G プロセスで行なわれるものを局所的包摂テスト (LS)，T プロセスで行なわれるものを大域的包摂テスト (GS) と呼ぶことにする．ここでは，LS を行なう時点でその PE に保持されている MD の要素数を m と仮定し，包摂テストの範囲を明示した．例えば，$C\sigma$ is new to $MD[1,\ldots,m]$ で，$\forall i(1 \leq i \leq m)\ \not\exists\sigma'(C\sigma = MD[i]\sigma')$ を表している．

生成され LS を通過したアトムは New_g に蓄えられるが，同時に LS の適用範囲を示す m も蓄えられる ((G4))．

T プロセスは，先ず G プロセスが生成したアトム δ と GS の範囲 $(m,\ldots,s')$ の組を M プロセスから受けとる ((T0))．そして δ が包摂されなければ，$MD[s'+1]$ に δ を代入し，$s'+1$ を M プロセスに送る．δ が包摂されれば，s' を M プロセスに送る．M プロセスに送った値 (s' または $s'+1$) が δ の次のアトムに必要な GS の範囲を与える．

4.2.2.3 逐次性低減のための幾つかの改良

本並列化方式には原理的に二つの逐次性があり，並列性能を引き出す上での阻害要因となっている．一つは包摂テストの逐次性であり，もう一つは証明不変を保証するための逐次性である．

1. 包摂テスト

   新たに生成されたアトムの前方包摂テストを完全に行なうには，その生成順を保持して，以前に生成されたアトム集合に対して包摂テストを行なう必要があるため逐次性が生じる．この逐次性の低減を図るため，包摂テストを LS と GS に分離した．

   モデル共有方式を採用しているので，各生成アトムに対する LS は他の PE に影響を与えることなく，各 PE 内でそれぞれ独立に行なうことができる．但し，LS の範囲は広ければ広いほど (図 4.3の (G3) 中の m が大きければ大きいほど)，アトムが包摂される可能性が高まり，結果として通信と GS の負荷が減るので好ましい．従って，ある PE での MD の更新結果は，直ちに他の全 PE に伝搬するような実装が必要である．

   各 PE で独立に行なうことのできる LS に対し，GS には逐次性が残っている．これは，直前のアトムの包摂テストが終了しないと，次のアトムの包摂テストが終了できないことに起因する．図 4.3 でいうと，(T1) での δ is new to $MD[m+1,\ldots,s']$ の判定に

[3]実際の実装は KL1 の未定義変数の送受信を利用して行なわれる．

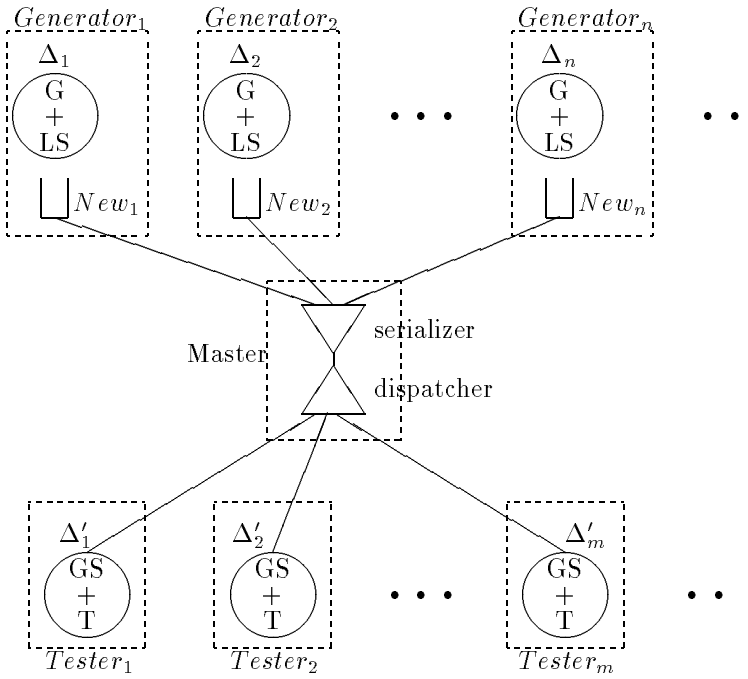

図 4.4: モデル共有型 MGTP/N のプロセス結合図

は，$MD[m+1],\ldots,MD[s']$ の値が全て確定，つまりそれらの包摂テストが全て終了している必要がある．

この GS の逐次性を低減するために，LS 完了後のアトムを保持するための大域的な配列 TMD (仮積みリストと呼ぶ) を用意する．これは，図 4.2 で使用している Buf と実際には全く同じである．TMD の要素の内 GS で包摂されなかったアトムが，MD の要素となるので，MD は TMD の部分集合となる．

TMD を使った場合の手続きは，図 4.3 を，次のように変更すればよい．TMD は，MD と同様な初期化手続きがなされ，$ts := k+1$ と初期化がなされているとする．

(1) (M4) を次のように変更する：MTC ! $\{New_{g'}[b'], s, ts\}; ts := ts+1; TMC$? s;

(2) (T0) を次のように変更する：MTC ? $\{\{\delta, m\}, s', ts'\}$;

(3) (T0) と (T1) の間に以下を挿入する：$TMD[ts'] := \delta$

(4) (T1) の条件部を次のように変更する：δ is new to $TMD[(m+1)^\flat, \ldots, ts'-1]$
ここで，$()^\flat$ は，MD のインデックスから TMD のインデックスへの関数で $MD[m] = TMD[(m)^\flat]$ を満たす．

(3) によって，GS を始める前にアトムを仮積みリスト (TMD) に積んでおくので，このアトムの GS の終了を，他のアトムの GS が待つことはない．但し，仮積みリストには，包摂されるかも知れないアトムも置かれる可能性があり，このようなアトムに対する包摂テストは，仮積みリストを設けない場合には行なわれないので，冗長である．仮積みリストによる逐次性の削減は，この冗長計算と引き替えに得られるものである．

2. 証明不変

   証明を不変にするためには，生成されたアトムの取り出し順序を固定しなければならない．この順序を固定することは，逐次性が発生することに他ならない．例えば，二つの G プロセス G_1 と G_2 が並列に動いている時，G_1 の生成アトム集合 New_1 と G_2 の生成アトム集合 New_2 は並列に生成されるが，早くできたアトムから取り出すわけにはいかない．予め決められた順序で取り出さなければ，実行のたびに証明が変わり得る．さて，予め New_1 から取り出すことになっていたとすると，New_1 の生成が何らかの理由で遅れた場合，New_2 がすでに生成されていたとしても，次のアトムを取り出すことができないので，T プロセスが動けない状態になってしまう．G_1 の実行が滞っていたとしても，G_2，$G_3 \ldots$ と次々に G プロセスを並列実行させれば見かけ上暇な PE をなくすことはできるが，これは無駄なアトムの生成を引き起こす可能性があり，遅延生成の考え方に反する．

   そこで上記の逐次性を低減するために，G プロセスの粒度を細分化することにした．具体的には，これまで G プロセスが行なっていた仕事の単位 (図 4.3において，一つのアトム $MD[g]$ に対する (`G1`) のループの処理) をさらに $1/k$ に均等分割し，これを仕事の単位とする．このようにすることにより，G プロセスの粒度がより細かくなり，かつ不均一さも顕著ではなくなった．

4.2.3 モデル分散型並列アルゴリズム

モデル分散型並列アルゴリズムは，マスタースレーブ方式で実装され，逐次アルゴリズム (図 4.1) の (3) から (9) までを単位ステップとし，生成過程 (*(4)* から *(5)* の条件部の前半)，包摂テスト (*(5)* の条件部の後半)，棄却テスト (*(8)* と *(9)*) をステップ内で並列化を行なうものでモデル候補とモデル拡張候補を分散管理している．

単位ステップ内の処理の流れを図?? に示す．

1. モデル拡張候補の選択

   あるプロセッサ j から当該ステップのモデル拡張候補 d が選択され[4] 他の全プロセッサに配布される．

2. 連言照合

   各ノードはモデル拡張候補 d を入力とし，

   (a) 混合節と，当該ノードのモデル候補集合 M と d，および d 同士の組合せによる連言照合 (単一化)[5]を行なう．

   (b) 充足されたら置換ずみの混合節後件アトム集合 CS を求め，ノード固有の番号 key を生成する．

[4]連言照合終了後当該プロセッサに新たなモデル候補として登録される．

[5]d 同士の組合せによる連言照合はすべてのノードで重複して行なわれる．

3. 連言照合結果の編集

   マスタプロセスでは連言照合結果 C の到着順に当該の置換済み後件アトムが，並列に実行される包摂テストにおいて, 包摂された場合の情報を他のノードにブロードキャストするためのスイッチを付加し D とする．[6]

4. 包摂テスト

   D の要素とノードが分散管理しているモデル候補 (+ モデル拡張候補) 集合 MD[7] および D 同士とのパターンマッチ[8]を行ない， MD または D に包摂された場合は当該の要素を削除する．

5. 負節との連言照合テスト

   負節と，当該プロセッサのモデル候補 (+ モデル拡張候補) 集合 MD および包摂されなかった D の要素との組合せによる連言照合 (単一化) を行ない充足されたら処理を終了する．

6. 内部データの更新

   マスタで指定されたモデル拡張候補[9]を当該ノードのモデル拡張候補集合 MD に追加する．また，次のステップへのフィードバック情報である内部データ状況 Fd を出力する．[10]

4.3 操作方法

MGTP 入力節は， Prolog 版では Prolog 節に， KL1 版では KL1 節に変換されるが，今のところ，いずれの版もこの変換系は用意されていないが， [McCune] に掲載されている問題については，例題としてファイルが附属している．

4.3.1 Prolog 版操作法

4.3.1.1 必要ファイル

ファイル名	ファイル内容
`mgtp.pl`	インタプリタ本体
`mgtpS.pl`	インタプリタ本体 (ソート機能付)
`unify.pl`	出現検査付単一化プログラム
`pro*.pl`	(変換済み) 正節と混合節
`pro*t.pl`	(変換済み) 棄却節

4.3.1.2 コンパイル，実行例

[6] プロセッサプロセス後半部への入力データとなる

[7] MD に対する後向き包摂テストは行なわない

[8] d 同士の組合せによるパターンマッチはすべてのノードで重複して行なわれる．

[9] 包摂されていない場合

[10] この情報をもとにモデル拡張候補が選択されるべきノード j が決定される

1. コンパイル：

```
koshi@ss130[3]% sicstus    : SISCtus Prolog の起動
SICStus 2.1 #8: Tue May 11 21:04:52 JST 1993
| ?- compile([mgtp,unify]).     : インタプリタ, 単一化プログラムのコンパイル
{compiling /home2/koshi/prolog/MGTPN/mgtp.pl...}
{Warning: [X] - singleton variables in do1/1 in lines 7-8}
{Warning: [F] - singleton variables in weight/3 in lines 35-36}
{Warning: [T] - singleton variables in weightArg/4 in lines 36-38}
{Warning: [X] - singleton variables in init/0 in lines 70-72}
{Warning: [X] - singleton variables in init/0 in lines 72-73}
{Warning: [X] - singleton variables in init/0 in lines 73-74}
{Warning: [F,N,Ref] - singleton variables in countData/2 in lines 74-76}
{Warning: [N,X] - singleton variables in countData/2 in lines 76-80}
{Warning: [F] - singleton variables in countData/2 in lines 80-81}
{compiled /home2/koshi/prolog/MGTPN/mgtp.pl in module mgtp, 340 msec 42976 byte
{compiling /home2/koshi/prolog/MGTPN/unify.pl...}
{Warning: [X,Y] - singleton variables in unifyArg/3 in lines 6-8}
{Warning: [X] - singleton variables in notOccursIn/2 in lines 17-18}
{Warning: [F] - singleton variables in notOccursIn/2 in lines 18-19}
{Warning: [X,Y] - singleton variables in notOccursInArg/3 in lines 19-21}
{compiled /home2/koshi/prolog/MGTPN/unify.pl in module unify, 160 msec 9248 byt

yes
| ?-
```

2. 実行例

```
| ?- compile([pro2,pro2t]).     : 問題節のコンパイル
{compiling /home2/koshi/prolog/MGTPN/pro2.pl...}
{compiling /home2/koshi/prolog/MGTPN/pro1.pl...}
{Warning: [Y] - singleton variables in delta/1 in lines 6-8}
{Warning: [X1,Y] - singleton variables in '$p/1'/3 in lines 25-26}
{Warning: [F] - singleton variables in useless1/1 in lines 50-51}
{Warning: [A,X] - singleton variables in useless1Arg/3 in lines 54-56}
{Warning: [Fa] - singleton variables in useless1Arg/3 in lines 56-57}
{compiled /home2/koshi/prolog/MGTPN/pro1.pl in module P, 170 msec 15232 bytes}
The procedure delta/1 is being redefined.
    Old file: /home2/koshi/prolog/MGTPN/pro1.pl
    New file: /home2/koshi/prolog/MGTPN/pro2.pl
```

```
Do you really want to redefine it? (y, n, p, or ?) p
{Warning: [Y] - singleton variables in delta/1 in lines 4-6}
{Warning: [Y] - singleton variables in delta/1 in lines 6-7}
{/home2/koshi/prolog/MGTPN/pro2.pl compiled, 220 msec 15408 bytes}
{compiling /home2/koshi/prolog/MGTPN/pro2t.pl...}
{compiling /home2/koshi/prolog/MGTPN/pro1t.pl...}
{compiled /home2/koshi/prolog/MGTPN/pro1t.pl in module Pt, 90 msec 2256 bytes}
The procedure p/1 is being redefined.
    Old file: /home2/koshi/prolog/MGTPN/pro1t.pl
    New file: /home2/koshi/prolog/MGTPN/pro2t.pl
Do you really want to redefine it? (y, n, p, or ?) p
{/home2/koshi/prolog/MGTPN/pro2t.pl compiled, 130 msec 2624 bytes}

yes
| ?- do.      : 証明開始
Succeed       : 証明成功

yes
| ?-
```

【注意】引続き問題を解く場合：内部データベースを init 命令で初期化した後，新たに問題節をコンパイルする必要がある．

```
| ?- init.  : 内部データベース初期化

yes
| ?- compile([pro3,pro3t]).        : 問題節を新たにコンパイル
{compiling /home2/koshi/prolog/MGTPN/pro3.pl...}
{compiling /home2/koshi/prolog/MGTPN/pro2.pl...}
{compiling /home2/koshi/prolog/MGTPN/pro1.pl...}
{Warning: [Y] - singleton variables in delta/1 in lines 6-8}
{Warning: [X1,Y] - singleton variables in '$p/1'/3 in lines 25-26}
{Warning: [F] - singleton variables in useless1/1 in lines 50-51}
{Warning: [A,X] - singleton variables in useless1Arg/3 in lines 54-56}
{Warning: [Fa] - singleton variables in useless1Arg/3 in lines 56-57}
{compiled /home2/koshi/prolog/MGTPN/pro1.pl in module P, 210 msec 3440 bytes}
The procedure delta/1 is being redefined.
    Old file: /home2/koshi/prolog/MGTPN/pro1.pl
    New file: /home2/koshi/prolog/MGTPN/pro2.pl
Do you really want to redefine it? (y, n, p, or ?) p
```

```
{Warning: [Y] - singleton variables in delta/1 in lines 4-6}
{Warning: [Y] - singleton variables in delta/1 in lines 6-7}
{/home2/koshi/prolog/MGTPN/pro2.pl compiled, 270 msec 3504 bytes}
{/home2/koshi/prolog/MGTPN/pro3.pl compiled, 290 msec 3584 bytes}
{compiling /home2/koshi/prolog/MGTPN/pro3t.pl...}
{compiling /home2/koshi/prolog/MGTPN/pro1t.pl...}
{compiled /home2/koshi/prolog/MGTPN/pro1t.pl in module Pt, 90 msec 1744 bytes}
The procedure p/1 is being redefined.
    Old file: /home2/koshi/prolog/MGTPN/pro1t.pl
    New file: /home2/koshi/prolog/MGTPN/pro3t.pl
Do you really want to redefine it? (y, n, p, or ?) p
{/home2/koshi/prolog/MGTPN/pro3t.pl compiled, 119 msec 2112 bytes}

yes
| ?-
```

4.3.1.3 MGTP 節変換例

1. MGTP 入力節 (問題 1)

   a) 正節

   $true \rightarrow p(i(X, i(Y, X))).$
   $true \rightarrow p(i(i(X, i(Y, Z)), i(i(X, Y), i(X, Z)))).$
   $true \rightarrow p(i(n(n(X)), X)).$
   $true \rightarrow p(i(X, n(n(X)))).$
   $true \rightarrow p(i(i(X, Y), i(n(Y), n(X)))).$

   b) 負節

   $p(i(i(a, i(b, c)), i(b, i(a, c)))) \rightarrow false.$

   c) 混合節

   $p(X), p(i(X, Y)) -> p(Y).$

2. 変換後の Prolog 節

a) 正節 (モジュールは 'P')

```
delta(p(i(X,i(Y,X)))).
delta(p(i(i(X,i(Y,Z)),i(i(X,Y),i(X,Z))))).
delta(p(i(n(n(X)),X))).
delta(p(i(X,n(n(X))))).
delta(p(i(i(X,Y),i(n(Y),n(X))))).
```

b) 負節 (モジュールは 'Pt')

```
p(i(i(a,i(b,c)),i(b,i(a,c)))) :- mgtp:proofEnd.
```

c) 混合節 (モジュールは 'P')

```
p(X) :- copy_term(X,Xc), '$p/1'(3, X,Xc).
p(i(X1,Y)) :- '$p/1'(2, X, v(X,X1,Y)).
p(X) :- '$p/1'(1, i(X1,Y), v(X,X1,Y)).

'$p/1'(1, A2, Vars) :- model(p(A2)), '$p/2'(1,Vars).
'$p/1'(2, A2, Vars) :- model(p(A2)), '$p/2'(2,Vars).
'$p/1'(3, X, Xc) :- '$p/2'(3, Xc, v(X,X1,Y)).

'$p/2'(1, v(X,X1,Y)) :- unify:unify(X,X1), mgtp:consq(p(Y)).
'$p/2'(2, v(X,X1,Y)) :- unify:unify(X,X1), mgtp:consq(p(Y)).
'$p/2'(3, i(X1,Y), v(X,X1,Y)) :- unify:unify(X,X1), mgtp:consq(p(Y)).
```

【注意】 MGTP 負節が $p(X), p(i(X, i(i(a, i(b, c)), i(b, i(a, c))))) \rightarrow false.$ の時は,

```
p(X) :- '$p/1'(1, i(X1,i(i(a,i(b,c)),i(b,i(a,c)))), v(X,X1)).
p(i(X1,i(i(a,i(b,c)),i(b,i(a,c))))) :- '$p/1'(2, X, v(X,X1)).
%p(i(i(a,i(b,c)),i(b,i(a,c)))) :- mgtp:proofEnd.

'$p/1'(1, A2, Vars) :- 'P':model(p(A2)), '$p/2'(1,Vars).
'$p/1'(1, A2, Vars) :- 'P':current(p(A2)), '$p/2'(1,Vars).
'$p/1'(1, A2, Vars) :- 'P':delta(p(A2)), '$p/2'(1,Vars).
'$p/1'(2, A2, Vars) :- 'P':model(p(A2)), '$p/2'(2,Vars).
'$p/1'(2, A2, Vars) :- 'P':current(p(A2)), '$p/2'(2,Vars).
'$p/1'(2, A2, Vars) :- 'P':delta(p(A2)), '$p/2'(2,Vars).

'$p/2'(1, v(X,X1)) :- unify:unify(X,X1), mgtp:proofEnd.
'$p/2'(2, v(X,X1)) :- unify:unify(X,X1), mgtp:proofEnd.
```

と変換される.

4.3.2 KL1 版 (モデル共有型) 操作法

4.3.2.1 必要ファイル

ファイル名	ファイル内容
mgtpN.kl1	インタプリタ本体
print.kl1	出力関係
meta.kl1	単一化，照合手続き
tm.kl1	弁別木プログラム
tmLib.kl1	弁別木プログラムのインターフェイス
pro*.kl1	(変換済み) 問題節

4.3.2.2 コンパイル，実行，演算例

1. コンパイル：

```
[1] compile["mgtpN","print","meta","tm","tmLib","pro3"].  :  必要ファイルの
                                                          :  コンパイル
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>print.kl1.1
[dbg.kl1@0] Compile Module : mgtp_d
Compile Succeeded : mgtp_d

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>meta.kl1.1
[metaOm1.kl1@0] Compile Module : meta
Compile Succeeded : meta

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>tmLib.kl1.1
!WARNING! no with_macro declaraion. assumed ‘pimos’.
[mgtpLibne.kl1@1] Compile Module : mgtp_lib
Compile Succeeded : mgtp_lib

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>tm.kl1.1
[tm_ne.kl1@0] Compile Module : tm
Compile Succeeded : tm

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>pro3.kl1.1
[pro3Om2.kl1@1] Compile Module : mgtp_p
Compile Succeeded : mgtp_p

Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>lmgtpN.kl1.23
[lmgtpNp_mlvns1Debug2_6.kl1@1] Compile Module : mgtp
Compile Succeeded : mgtp

!WARNING! register shortage in the guard 2 time(s) : master/ 19
```

```
!WARNING![lmgtpNp_mlvns1Debug2_6.kl1@343] unused predicate found : min/ 3
!WARNING![lmgtpNp_mlvns1Debug2_6.kl1@345] unused predicate found : max/ 3
"mgtp:: meta" Loaded
"mgtp:: mgtp_lib" Loaded
"mgtp:: mgtp_d" Loaded
"mgtp:: mgtp_p" Loaded
"mgtp:: tm" Loaded
"mgtp:: mgtp" Loaded
Compilation Time = 121325 [MSEC]
 6664355 reductions
 122714 msec


[2]
```

2. 実行例： (1 PE 構成の場合)

```
[2] mgtp: do([{0,0}],[{0,0}],1000,10000,50,10,10,10,1,{1,1},4096,
                          {3960,3980,2600},{4000,3900,4010,2500},S,0)@node(0).
distribute(500,buf:7,store:473,tested:473,eqlast:0,ls_IML:486,
           ls_ML:461,distrG:89)/0.
store(500,21,[{0,{47,47,16}},{1,{47,47,16}}]) / 0.
reach(500) / 0.
distribute(1000,buf:6,store:973,tested:973,eqlast:0,ls_IML:982,
           ls_ML:956,distrG:128)/0.
store(1000,20,[{0,{51,51,18}},{1,{67,67,13}}]) / 0.
reach(1000) / 0.
distribute(1500,buf:26,store:1464,tested:1464,eqlast:0,ls_IML:1457,
           ls_ML:1423,distrG:165)/0.
store(1500,20,[{1,{71,71,17}},{0,{84,84,10}}]) / 0.
reach(1500) / 0.
goal([{1,{138,138,19}},{0,{1797,1758,13}}],i(i(a,i(b,c)),i(i(a,b),i(a,c)))).
 S =[otterOrder,'unify+unifyOc',rejected,input: 5,model: 1758,
     kept: 1753,tested: 1757,keptG: 61,distr: 1797,distrG: 183,eqlast: 0]
 65247574 reductions
 729807 msec


[3]
```

【注意】

- PIM/m 64 PE 構成の場合

```
mgtp:do([{0,62}],[{0,62}],1000,40,50,10,100,100,1,{2,2},4096,
        {3960,3980,2600},{4000,3900,4010,2500},S,63)@node(63).
```

- PIM/m 256 PE 構成の場合

```
mgtp:do([{1,252}],[{1,252}],1000,80,50,10,400,400,1,{2,2},4096,
        {3960,3980,2600},{4000,3900,4010,2500},S,253)@node(254).
```

- PIM/p 32 PE(4 cluster) 構成の場合

```
mgtp:do([{0,3}],[{0,3}],1000,160,50,10,50,50,8,{16,16},4096,
        {3960,3980,2600},{4000,3900,4010,2500},S,3)@node(3).
```

- PIM/p 64 PE(8 cluster) 構成の場合

```
mgtp:do([{0,7}],[{0,7}],1000,200,50,10,100,100,8,{16,16},4096,
        {3960,3980,2600},{4000,3900,4010,2500},S,7)@node(7).
```

- PIM/p 128 PE(16 cluster) 構成の場合

```
mgtp:do([{0,15}],[{0,15}],1000,200,50,10,200,200,8,{16,16},4096,
        {3960,3980,2600},{4000,3900,4010,2500},S,15)@node(15).
```

【注意】 他の問題を引続き証明したい場合は，当該問題ファイルをコンパイルした後，再リンクする必要がある．

```
[16] compile(["pro28Om2"]).        : 次の問題のコンパイル
** KL1 Compiler **
Compile File : icpsi520::>sys>user>miyuki>KL1>MGTPN>pro28Om2.kl1.1
[pro28Om2.kl1@1] Compile Module : mgtp_p
=>  :- module mgtp_p .
Compile Succeeded : mgtp_p

"mgtp:: mgtp_p" Updated
Compilation(s) Succeeded
Compilation Time = 16192 [MSEC]
 244554 reductions
 17024 msec

[17] relink(mgtp,[mgtp_p]) .       : 再リンク
"mgtp:: mgtp" Updated
Relink Succeeded
 22115 reductions
 2159 msec

[18]
```

4.3.3 KL1 版 (モデル分散型) 操作法

4.3.3.1 必要ファイル

ファイル名	ファイル内容
mgtpv0.kl1	インタプリタ本体
util.kl1	ユーテリティ

4.3.3.2 コンパイル，実行，演算例

PIM/m 上でのモデル分散型 MGTP/N の起動および実行方法について述べる．

1. 起動:

```
[1] compile(["mgtpv0","util"]).    : ディレクトリの下のソースファイルを
                                     コンパイルする．

[2] st.                            : タイマーの設定

[3] pl.                            : 印刷時の長さ
```

```
 Print length: 10 => 100
```

```
[4] pd.                                  : 印刷時の深さ
 Print depth: 4 => 20
```

2. 実行:

述語と引数の定義は

mgtpv0: main/10

- Cnd : 負荷分散開始ノード (>=1)
- Tnd : 負荷分散終了ノード (>=Cnd)
- Ofset : コードでは変数を {1} から始まる {整数} で定義する.
  たとえば
  i(X,Y)->Y を i({1},{2})->{2} のように表す.
  混合節および負節の定義に使用した整数の最大値を指定する
- Axiom : 正節後件アトムのリストである. 一つのアトムのフォーマットは
  ((0,0,0),Ofset 値以上の偶数で変数定義した後件アトム,
  同じく Ofset 値以上の奇数で変数定義した後件アトム)

  また, (0,0,0) は初期値におけるダミー情報でありコード内では
  連言照合による生成番号を意味する. すなわち
  (ノード番号, ノード内生成番号, 予備)
- Gen : 混合節のリストである. ひとつの混合節のフォーマットは
  gen(前件アトム , 後件アトム) または
  gen(前件アトム 1, 前件アトム 2, 後件アトム)
- Tst : 負節前件アトムのリスト. 一つの負節のフォーマットは
  tst(前件アトム) または
  tst(前件アトム 1, 前件アトム 2)
- Jinit : 各ノードが参照するステップ情報の初期値 (Js,Ja,Jn,YN)
  Js: モデル拡張候補 d が選択されるノードの番号
  Ja:d がモデル候補として登録されるノードの番号
  Jn: 次ステップで d が選択される予定のノードの番号
  YN: コード内で Jn にモデル拡張候補が存在すれば yes,
  しなければ no がセットされる. 初期値は未定義変数.

以降の引数は証明結果である.

- Result : sat/unsat= 節集合が充足可能 / 充足不能
- Count : 証明終了時までに登録されたモデル候補数
- Delta : 証明終了時までに登録されたモデル候補のリスト

3. 演算例:

下記の問題を例にとってデータ入力方法および結果の見方を述べる.

サンプル問題 (問題 2)
a) 正節
```
true -> (p(i(X,i(n(X),Y)))).
true -> (p(i(X,i(Y,X)))).
true -> (p(i(i(X,i(Y,Z)),i(Y,i(X,Z))))).
true -> (p(i(i(X,Y),i(i(Z,X),i(Z,Y))))).
true -> (p(i(i(X,Y),i(i(n(X),Y),Y)))).
```
b) 負節
```
p(i(i(a,i(a,b)),i(a,b))) -> false.
```
c) 混合節
```
p(X),p(i(X,Y)) -> p(Y).
```

実行定義は
```
[5] mgtpv0: main(1,5,3,
[((0,0,0),p(i({4},i(n({4}),{6}))),
          p(i({5},i(n({5}),{7})))),
((0,0,0),p(i({4},i({6},{4}))),
         p(i({5},i({7},{5})))),
((0,0,0),p(i(i({4},i({6},{8})),i({6},i({4},{8})))),
         p(i(i({5},i({7},{9})),i({7},i({5},{9}))))),
((0,0,0),p(i(i({4},{6}),i(i({8},{4}),i({8},{6})))),
         p(i(i({5},{7}),i(i({9},{5}),i({9},{7}))))),
((0,0,0),p(i(i({4},{6}),i(i(n({4}),{6}),{6}))),
         p(i(i({5},{7}),i(i(n({5}),{7}),{7}))))],
[gen(p({1}),p(i({1},{3})),p({3}))],
[tst(p(i(i(a,i(a,b)),i(a,b))))],
(1,1,2,YYNN),
Result,Ncalc,Delta) .
```

5 台のノードを使用したこのケースの結果は
```
 Result=unsat
 Ncalc= 18
 time=385832 reductions
      2367 msec
 Delta=
((0,0,0),p(i({4},i(n({4}),{6}))),(奇数で変数定義した情報は省略)
((0,0,0),p(i({4},i({6},{4}))),
```

```
((0,0,0),p(i(i({4},i({6},{8})),i({6},i({4},{8})))),
((0,0,0),p(i(i({4},{6}),i(i({8},{4}),i({8},{6})))),
((0,0,0),p(i(i({4},{6}),i(i(n({4}),{6}),{6}))),
((1,3,dummy),p(i(n(i({4},i({6},{4}))),{8})),(ノード 1 で 3 番目に
                                                発生したことを示す)
((2,4,dummy),p(i({4},i(i({6},i({8},{10})),i({8},i({6},{10}))))),
((3,5,dummy),p(i(i({4},{6}),i(i({6},{8}),i({4},{8})))),
((4,1,dummy),p(i(n(i({4},i(n({4}),{6}))),{8})),
((5,3,dummy),p(i({4},i(i({6},i({4},{8})),i({6},{8})))),
((1,4,dummy),p(i({4},i({6},i(n({6}),{8})))),
((2,5,dummy),p(i({4},i({6},{6}))),
((3,6,dummy),p(i(i({4},i({6},i({8},{10}))),i({4},i({8},i({6},{10}))))),
((4,2,dummy),p(i({4},i({6},i({8},{6})))),
((5,6,dummy),p(i(i(n(n(i({4},i({6},{4})))),{8}),{8})),
((1,6,dummy),p(i(n({4}),i({4},{6}))),
((2,6,dummy),p(i(i({4},{6}),i({4},i({8},{6})))),
((3,8,dummy),p(i(i(n({4}),{6}),i(i({4},{6}),{6}))),
 であった.
```

参考文献

[越村 95] 越村 三幸，長谷川 隆三：モデル生成型定理証明系の *AND* 並列化方式，電子情報通信学会論文誌， VOL. J78-D-1 No. 2, 1995

[McCune] McCune, W. W. and Wos, L.: Experiments in Automated Deduction with Condensed Detachment. In *Proc. 11th Int. Conf. on Automated Deduction*, pages 209–223, Saratoga Springs, NY, 1992.

第 5 章

CMGTP マニュアル

本資料は、モデル生成型の定理証明システム MGTP を有限領域の制約充足問題を扱えるように拡張した制約 MGTP (以下 CMGTP) に関して、その機能、および操作方法を記述したものです。

CMGTP は、入力節に負アトムを用いた表現を許し、また負アトムによる候補の枝刈り機能を備えたシステムで、制約伝搬ルールが負アトムを用いてダイレクトに記述できるという利点を持っています。

CMGTP の有効なアプリケーション領域として、代数分野における準群問題 (quasigroup problems) があり、 ICOT では、 CMGTP を用いてこの問題にチャレンジしてきました。

CMGTP を用いることにより従来の MGTP や制約論理型言語ではフォローできない制約伝搬を行なうことが可能になり、これにより探索空間を大幅に縮小させることができます。また、並列マシン上では、 PIM/m 上および SparcCenter 上での利用が可能で、特に PIM/m 上においては、並列化方式等の工夫により、ほぼ線形に近い台数効果が達成されています。

また、準群問題に限らず、一般の制約充足問題も CMGTP で記述することができ、制約伝搬に関する実験を行なうことが可能です。

なお、本ソフトウェアに関して、コメント、不備等がある場合には、下記まで御照会頂ければ幸いです。

平成 7 年 3 月

(財) 新世代コンピュータ技術開発機構

白井康之　(shirai@icot.or.jp)
長谷川隆三　(hasegawa@icot.or.jp)

5.1 CMGTP の機能説明

5.1.1 概要

MGTP は一階述語論理に基づくボトムアップ型の定理証明システムで、連言照合における冗長性を極力排除し、また実装上の工夫などにより並列マシン上で極めて効率的に動作する問題解決器である。 MGTP はその問題記述の簡便性から、一階述語論理による知識表現言語と見ることもでき、 Prolog ではフォローできない Non-Horn 論理式を取り扱うことができるほか、並行論理型言語 KL1 では失われている探索の完全性が保証されているなどの利点を持っている。

MGTP は、有限領域の制約充足問題に対しても容易に記述することができ、また並列実行などを通じて、有限代数における準群の存在問題においていくつかの未解決問題を解くなどの成果をおさめている。

MGTP のこの成果をきっかけとして世界中で準群問題に対してさまざまなアプローチから研究が行なわれるようになったが、この結果として明らかになったことは、 MGTP では問題の記述は容易であるものの、枝刈りを行なうのに重要な情報の伝搬を行なうことができず、結果として冗長な枝を大量に探索していたということである。これらの結果から我々は、制約充足問題を解くために MGTP に改良を加えて、制約 MGTP (CMGTP) を開発した。

CMGTP は MGTP に負のアトムによる表現を許すことによって負の制約伝搬ルールを記述できるようにしたもので、単位簡約化 (unit simplification) ルールなどによって候補をあらかじめ削っておくことができる。これにより CMGTP は、準群問題における冗長な枝の探索を極力減らし、また効率の上でも並列化を行なうことによってきわめて良い結果をおさめることができた。

5.1.2 準群問題

CMGTP の最も有用なアプリケーションとして準群問題があり、また CMGTP における制約伝搬機能を説明する都合上からもまず準群問題 [B89] について簡単に説明する。なお、準群問題に関する詳しい説明や今までにとられてきたアプローチについては [SH95] などを参照されたい。

定義 (準群問題)

準群 (Quasigroup)

有限集合 Q と Q 上の二項演算 $\circ$ に対し、 Q の任意の元 a, b, c に関して以下が成り立つとき、 $\langle Q, \circ \rangle$ は準群であるという。

$$a \circ b = a \circ c \Rightarrow b = c$$

$$a \circ c = b \circ c \Rightarrow a = b.$$

この二項演算 $\circ$ による掛算テーブルはよく知られたラテン方陣を構成する (図 5.1)。

$\circ$	1	2	3	4	5
1	1	3	2	5	4
2	5	2	4	3	1
3	4	5	3	1	2
4	2	1	5	4	3
5	3	4	1	2	5

図 5.1: オーダ 5 のラテン方陣

準群の上の定義は、「各行各列には同じ元が 2 度以上現れない方陣」、あるいは「各行各列がそれぞれ Q の元の置換 (permutation) によって構成される方陣」と言い換えることができる。

また、準群 $\langle Q, \circ \rangle$ が条件

$$\forall x \in Q, x \circ x = x,$$

を満足する時、その準群はベキ等であるといわれる。図 5.1 に示されるラテン方陣はベキ等準群である。

準群問題 (Quasigroup Problems)

準群に対してある制約を付加し、それらを満たす準群、すなわちラテン方陣が存在するかという問題を準群問題という。準群問題には、追加する制約によっていくつかの種類があるが、例えば、 QG5 と呼ばれる問題では、以下のような制約が新たに準群の制約に対して追加される。

QG5： $\forall ab \in Q.\ ((ba)b)b = a$

図 5.1 に示したラテン方陣は QG5 の条件を満足するので、少なくとも QG5 はオーダ 5 において 1 つのベキ等モデルを有することがわかる。現在多くの準群問題が未解決問題として残されており、例えば、 QG5 においては、オーダ n $(n \geq 17)$ はすべて未解決問題となっている。

準群問題における制約伝搬

準群問題において必要とされるべき制約伝搬についてまとめよう。

図 5.2 は、 QG5 において必要とされる制約伝搬ルールを前向き推論形式で記述したものである。ルール (a) は、任意の x, y, a, b に対して、 $yx = a, ay = b$ が成り立つならば、 by は x でなければならないことを表しており、 (b) (c) はルール (a) の対偶をとることによって得られる。これらのルールは負の情報からの制約伝搬を行なう際に必要とされるが、その意味するところは直観的に明らかであろう。また、ルール (d)(g) は、準群の定義によりルール (a) より

$$
\begin{array}{ll}
yx = a, ay = b \rightarrow by = x. & \text{(a)} \\
yx = a, by \neq x \rightarrow ay \neq b. & \text{(b)} \\
ay = b, by \neq x \rightarrow yx \neq a. & \text{(c)} \\
yx = a, ay \neq b \rightarrow by \neq x. & \text{(d)} \\
yx = a, by = x \rightarrow ay = b. & \text{(e)} \\
ay \neq b, by = x \rightarrow yx \neq a. & \text{(f)} \\
yx \neq a, ay = b \rightarrow by \neq x. & \text{(g)} \\
yx \neq a, by = x \rightarrow ay \neq b. & \text{(h)} \\
ay = b, by = x \rightarrow yx = a. & \text{(i)}
\end{array}
$$

図 5.2: QG5 において必要とされる制約伝搬ルール

導かれるものである。ルール (e) (f) は ルール (d) の、またルール (h) (i) はルール (g) の対偶としてそれぞれ導かれるものである。

オリジナルの MGTP では、ある枝における探索の終了は、

$$p(Y, X, A), p(A, Y, B), p(B, Y, C), C \neq X \rightarrow false.$$

という棄却ルールによって検出していたので、負の情報を用いた候補の枝刈りを事前に行なうことができず、結果として、過剰に不必要な枝を探索していた。しかし、実際には負の情報から制約を伝搬させることによって、あらかじめ有り得ない候補を枝刈りしておくことができる。

5.1.3 制約 MGTP

制約 MGTP の特徴

MGTP は、モデル生成法 [MB88] に基づく一階述語論理定理証明システムである。入力節は前向き推論形式で与えられ、 MGTP は入力節を充足 (satisfy) するモデルがあるかどうかを推論し、もしあればそのモデルを、なければ $unsat$ を返す。

MGTP によって準群問題を解くことの利点は、問題が一階述語形式で非常に簡潔に書けることである。例えば QG5 の場合は 10 個の MGTP の入力節として表現することが可能である。図 5.3 に QG5 の MGTP での記述例を示す。

一方で、先にも述べたように、 MGTP は前向き推論に基づくこと、また正アトムのみがモデルとして用いられることから、負の制約伝搬をすることができないという欠点も持っている。 CMGTP ではこれらの欠点を補うため、負アトムによる表現を許し、それらによる候補の枝刈り機能を備えている。

アルゴリズム

図 5.4 に CMGTP のモデル生成プロセスを示す。ここで、 D はモデル拡張候補と呼ばれ、 M はモデル候補と呼ばれる。これらは、 OTTER における “to be given list” と “has been

$$true \rightarrow dom(1), dom(2), dom(3), dom(4), dom(5). \quad (M1)$$
$$true \rightarrow p(1,1,1), p(2,2,2), p(3,3,3), p(4,4,4), p(5,5,5). \quad (M2)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(M,N,1); p(M,N,2); p(M,N,3); p(M,N,4); p(M,N,5). \quad (M3)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(M,1,N); p(M,2,N); p(M,3,N); p(M,4,N); p(M,5,N). \quad (M4)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(1,M,N); p(2,M,N); p(3,M,N); p(4,M,N); p(5,M,N). \quad (M5)$$
$$p(M,N,X), p(M1,N,X), \{M1 \neq M\} \rightarrow false. \quad (M6)$$
$$p(M,N,X), p(M,N1,X), \{N1 \neq N\} \rightarrow false. \quad (M7)$$
$$p(M,N,X), p(M,N,X1), \{X1 \neq X\} \rightarrow false. \quad (M8)$$
$$p(X,5,Y), \{X1 := X-1, Y < X1\} \rightarrow false. \quad (M9)$$
$$p(Y,X,A), p(A,Y,B), p(B,Y,C), \{C \neq X\} \rightarrow false. \quad (M10)$$

図 5.3: QG5 (オーダ 5) に対する MGTP 入力節

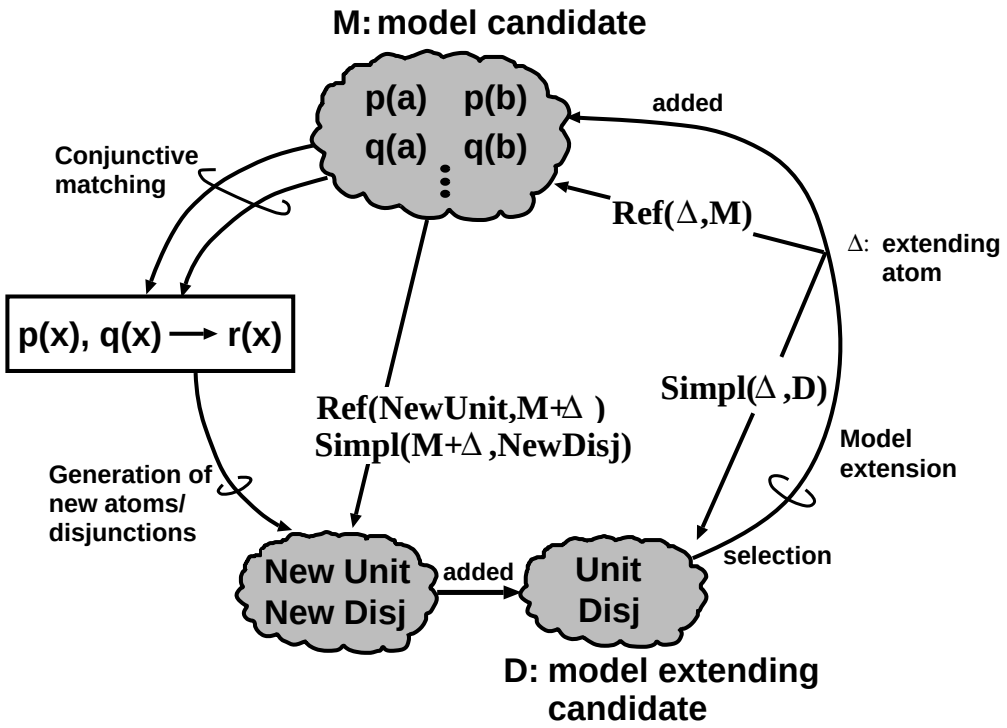

図 5.4: CMGTP におけるモデル生成プロセス

given list" にそれぞれ対応するものである。

CMGTP のモデル生成プロセスは、MGTP のモデル生成プロセスに 2 つの Ref プロセスと 2 つの $Simpl$ プロセスを新たに付け加えたものである。MGTP のアルゴリズムについては [FH91][FHKF92] などを参照されたい。

CMGTP と MGTP の相違は、単位反駁 (unit refutation) プロセス と単位簡約化 (unit simplification) プロセスにある。単位反駁メカニズムにより、もし、P と $\neg P$ が共に M にあれば、以下のように $false$ が導出される。

$$\frac{\begin{array}{cc}(\mathrm{M}) & (\mathrm{M}) \\ \vdots & \vdots \\ \neg P & P\end{array}}{false}$$

また、もし M の中に $\neg P_i$ (P_i) があり、また D に $P_i(\neg P_i)$ を含む選言が存在するならば、以下に示すような単位簡約化メカニズムによって $P_i(\neg P_i)$ はその選言から削除される。

$$\frac{\begin{array}{cc}(\mathrm{M}) & (\mathrm{D}) \\ \vdots & \vdots \\ \neg P_i & P_1 \vee \ldots \vee P_{i-1} \vee P_i \vee P_{i+1} \vee \ldots P_n\end{array}}{P_1 \vee \ldots \vee P_{i-1} \vee P_{i+1} \vee \ldots \vee P_n} \qquad \frac{\begin{array}{cc}(\mathrm{M}) & (\mathrm{D}) \\ \vdots & \vdots \\ P_i & P_1 \vee \ldots \vee P_{i-1} \vee \neg P_i \vee P_{i+1} \vee \ldots P_n\end{array}}{P_1 \vee \ldots \vee P_{i-1} \vee P_{i+1} \vee \ldots \vee P_n}$$

単位簡約化の結果として nil (長さ 0 の選言) が生成されるときは、$false$ が導出される。

図 5.4 における $Ref/2$ と $Simpl/2$ はそれぞれ上で述べたような単位反駁と単位簡約化を実行する関数である。すなわち $Ref(U_1, U_2)$ は、アトムの集合 U_1, U_2 に対してあるアトム $P \in U_1, Q \in U_2$ があって、このとき $P = \neg Q$ もしくは $\neg P = Q$ が成り立つ時 $false$ を返す。また、$Simpl(U, D)$ は単位節の集合 U に含まれるアトムによって選言の集合 D を simplify した結果を返すが、もしこの結果として nil が導出された時は、$Simpl$ の結果として $false$ を返す。

Ref や $Simpl$ が $false$ を返した時には、現在探索中の枝は $unsat$ として終了 (terminate) する。容易にわかるように、アトムは長さ 1 の選言と見ることができるから、Ref は $Simpl$ の特殊な形ともいえる。

オリジナルの MGTP に追加された単位反駁と単位簡約化のプロセスは以下の 4 つである。

- $Ref(\{\Delta\}, M)$
- $Ref(NewUnit, M \cup \{\Delta\})$
- $Simpl(\{\Delta\}, D)$
- $Simpl(M \cup \{\Delta\}, NewDisj)$

CMGTP がとる単位節優先戦略とこれらの関数により、任意のアトム $P \in M$ に対して、P と $\neg P$ が現在の M の中に共存しないこと、また、現在の D に含まれるすべての選言はすでに M 中のすべての単位節によって simplify されていることが保証される。

$$true \rightarrow dom(1), dom(2), dom(3), dom(4), dom(5). \quad (CM1)$$
$$true \rightarrow p(1,1,1), p(2,2,2), p(3,3,3), p(4,4,4), p(5,5,5). \quad (CM2)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(M,N,1); p(M,N,2); p(M,N,3); p(M,N,4); p(M,N,5). \quad (CM3)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(M,1,N); p(M,2,N); p(M,3,N); p(M,4,N); p(M,5,N). \quad (CM4)$$
$$dom(M), dom(N), \{M \neq N\} \rightarrow p(1,M,N); p(2,M,N); p(3,M,N); p(4,M,N); p(5,M,N). \quad (CM5)$$
$$p(M,N,X), dom(M1), \{M1 \neq M\} \rightarrow \neg p(M1,N,X). \quad (CM6)$$
$$p(M,N,X), dom(N1), \{N1 \neq N\} \rightarrow \neg p(M,N1,X). \quad (CM7)$$
$$p(M,N,X), dom(X1), \{X1 \neq X\} \rightarrow \neg p(M,N,X1). \quad (CM8)$$
$$dom(X), dom(Y), \{X1 := X - 1, Y < X1\} \rightarrow \neg p(X,5,Y). \quad (CM9)$$
$$p(Y,X,A), p(A,Y,B) \rightarrow p(B,Y,X). \quad (CM10)$$
$$p(Y,X,A), p(B,Y,X) \rightarrow p(A,Y,B). \quad (CM11)$$
$$p(A,Y,B), p(B,Y,X) \rightarrow p(Y,X,A). \quad (CM12)$$
$$p(Y,X,A), \neg p(B,Y,X) \rightarrow \neg p(A,Y,B). \quad (CM13)$$
$$p(Y,X,A), \neg p(A,Y,B) \rightarrow \neg p(B,Y,X). \quad (CM14)$$
$$\neg p(Y,X,A), p(B,Y,X) \rightarrow \neg p(A,Y,B). \quad (CM15)$$
$$\neg p(B,Y,X), p(A,Y,B) \rightarrow \neg p(Y,X,A). \quad (CM16)$$
$$\neg p(A,Y,B), p(B,Y,X) \rightarrow \neg p(Y,X,A). \quad (CM17)$$
$$p(A,Y,B), \neg p(Y,X,A) \rightarrow \neg p(B,Y,X). \quad (CM18)$$

図 5.5: QG5 (オーダ 5) に対する CMGTP 入力節

QG5.5 に対する CMGTP のルール

図 5.5 に QG5 のオーダ 5 に対する CMGTP の入力節を示す。(CM1) から (CM5) はオリジナルの MGTP でのルールと同じである。(CM6)-(CM9) も同様に準群の定義と同型排除のヒューリスティックスを表しているが、条件にあわない候補をあらかじめ否定モデルとして生成するルールに置き換えられていることに注意されたい。

そして、(CM10)-(CM18) は QG5 での制約条件を表現したものとなっているが、これは、図 5.2 で示した制約伝搬ルールそのものになっている。すなわち、図 5.2 に示されるような制約伝搬ルールは CMGTP の入力節としてダイレクトに記述することが可能であり、この意味で、CMGTP は制約伝搬を記述するメタ言語として考えることが可能である。

5.1.4 並列化方式

MGTP は並列化に非常に適した構造を有しており、実際 AND 並列、OR 並列のいずれにおいても望ましい台数効果が達成されている [FHKF92]。CMGTP は基本的に MGTP のアルゴリズムに従っているので、CMGTP もまた並列化効果が高いことが期待される。特に準群問題は、Non-Horn 問題であり、場合分けが多数発生することなどから OR 並列効果の高いことが期待できる。以下では、CMGTP の並列化の方法について説明する。

CMGTP における証明木は図 5.6 に示すように、OR 分岐による木構造となっている。ただし、準群問題の場合は、枝の長さが予期できない上、場合分けの個数もまちまちであるので、pegion hole 問題などのような balanced tree ではなく inbalanced tree である。

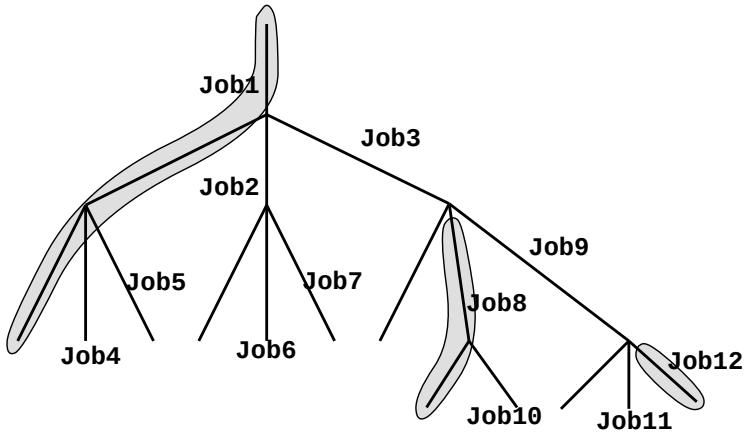

図 5.6: CMGTP の証明木

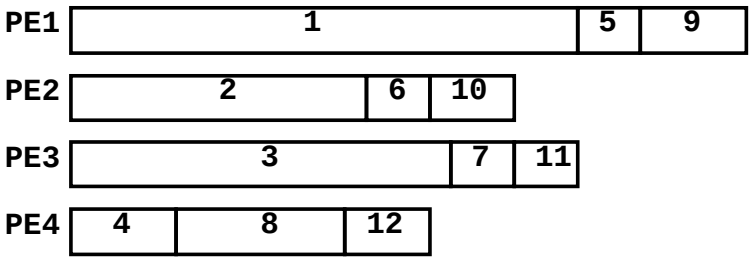

図 5.7: サイクリック割り当て方式

並列実行においては、木のあるノードで場合分けが生じた時、一本の枝は自分自身のプロセッサで処理し、その他の枝は他のプロセッサに投げるものとする。このとき、親ノードから受け継いだ枝を一つのジョブと考えると、図 5.6 の例では、合計 12 個のジョブが存在することになる。一般にジョブの長さはまちまちである。例えば、Job1 は非常に浅いレベルで生成された枝であり、非常に長いジョブになっているが、Job8 や Job12 はそれよりも深いレベルで生成された枝であり、Job1 に比べると短い枝となっている。

並列化の方法は、このようなジョブをどのようにプロセッサに割り当てるか、という問題に置き換えることができる。我々は以下に示すような 3 通りの方法を用いて実験を行なった。

- サイクリック割り当て方式
- 確率的割り当て方式
- 枝数制限方式

ただし、いづれの方法においても、浅いレベルで生成された枝は、自分自身が長くなる可能性が高いと共に、多くの分岐を生ずる可能性も高いので、各プロセッサ内では浅いレベルで

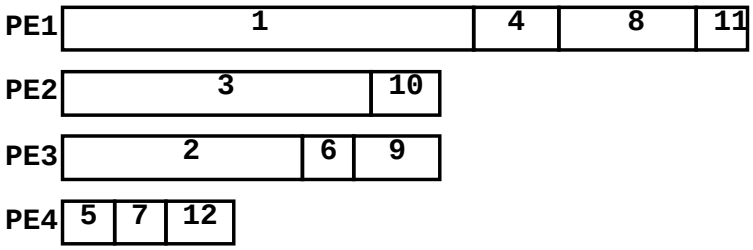

図 5.8: 確率的割り当て方式

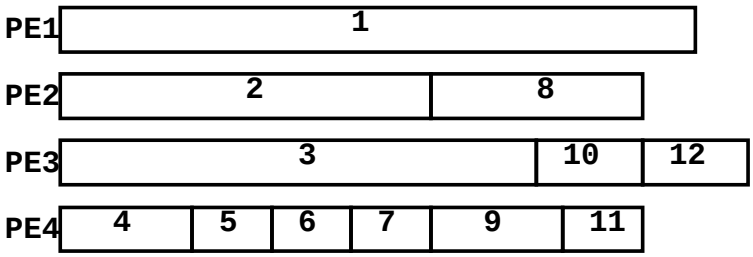

図 5.9: 枝数制限方式 $(Limit = 1)$

生成された枝を深いレベルで生成された枝に対して優先的に処理するようにしている。以下それぞれの方法について説明する。

サイクリック割り当て方式

この方式では、新しく生成されたジョブはすべてサイクリックに割り当てられる。つまり、あるプロセッサ n で分岐が生じた時、一本の枝は自分自身で担当し、それ以外の枝、すなわち新しいジョブはプロセッサ $n+1$ から順に割り当てていく。

結果として、実行が終了した時点ではそれぞれのプロセッサが担当したジョブの数はほぼ等しくなる。図 5.7 に図 5.6 で示した証明木のサイクリック方式によるプロセッサ割り当ての例を示す。今、プロセッサが 4 つ (PE1,PE2,PE3,PE4) あるとし、Job1 を PE1 が担当していたとすると、Job2，Job3，... は順に PE2，PE3，... に割り当てられる。

図 5.6 の証明木ではジョブは全部で 12 個生成されるので、4 つあるプロセッサは最終的にそれぞれ 3 本の枝を処理することになる。ただし、実行中の各時点に着目してみると各プロセッサがかかえている枝数は一定ではない。例えば、Job10 が PE2 で終了した時点では、PE2 は仕事がないアイドル状態になっているが、PE1 はいまだ 3 本の枝を抱えている。この例では Job1 が非常に長い枝であるために結果的に PE1 の実行時間が全体の実行時間を決定することになる。

確率的割り当て方式

この方式では、新しく生成されたジョブの割り当て先プロセッサは確率的に決定される。図 5.8 に図 5.6 で示した証明木の確率的方式によるプロセッサ割り当ての例を示す。

実際、この方式も上のサイクリック割り当てと全く同じ問題点を持つ。すなわち、あるプロセッサに長い枝が割り当てられるとそのプロセッサは結果として他のプロセッサに比べ長い実行時間を要し、全体の実行時間を長くする。サイクリック方式と確率的方式に共通するこの問題点は、各プロセッサの負荷を実行中にモニタリングしていないことに起因している。次に述べる枝数制限方式では、各プロセッサが各時点で抱えているジョブの数をマスタプロセッサがモニタリングして最も暇なプロセッサに割り当てる方式である。

枝数制限方式

この方式では、新しく仕事が生成された時点でそれぞれのプロセッサが抱えるジョブの数が同じになるように新しいプロセッサが選択される。すなわちこの方式では、マスタプロセッサが各プロセッサが現在何本のジョブを抱えているかを管理しており、最も少ないジョブを抱えているプロセッサに新しい仕事を投げる。もし長い枝を抱えこんだプロセッサがあれば、そのプロセッサにおいては処理中の枝数はなかなか減らないので結果として他から新しい仕事を割り当てられることはなくなる。

この方式においては $Limit$ というパラメータが用意されており、この値を変えて実験を行なうことができる。このパラメータは各プロセッサが同時に何本の枝を抱えることができるかを表している。例えば、$Limit = 1$ の場合には、各プロセッサは 2 本以上のジョブを並行して処理することができないので、新しい枝が生成された時には必ずその時にアイドル状態にあるプロセッサに割り当てられることになる。もしすべてのプロセッサが忙しい状態 (すなわち、$Limit$ の値だけのジョブを抱え込んでいる状態) であれば、暇なプロセッサが出現するまでその新しい仕事はサスペンドしている。

図 5.9 に $Limit = 1$ にした場合のプロセッサ割り当ての例を示す。一般に $Limit$ を小さい値にセットすることは、結果的に、各プロセッサが同時に抱えるジョブの数を減少させるので、メモリーの節約に役立つほか、より最適なジョブの割り当てを行なうことが可能になる。

ただ逆に $Limit$ が小さい時は、ジョブの粒度が比較的小さく、分岐が頻繁に起こる場合には、サスペンドしたジョブが大量に発生してしまい、マスタープロセッサにおけるジョブの分配が間に合わずに (マスターボトルネック状態)、結果的にジョブ待ち状態のプロセッサが多数発生することが予想される。

5.1.5 並列実行に関する考察

ここでは、CMGTP の並列化効果について ICOT で行なわれた実験を通じて簡単に説明する。

上で述べた 3 つの方式の中では、枝数制限方式が最も成績が良く、台数効果も高い。図 5.10、5.11、5.12 に各方式による実行終了時の実行モニタを示す。ここで見るように、サイクリック割り当てや確率的割り当てでは特に後半、プロセッサ間での負荷の差が激しく、結果的に全体の実行時間が長くなっている。これは、すでに述べたように各プロセッサの抱えるジョブの負

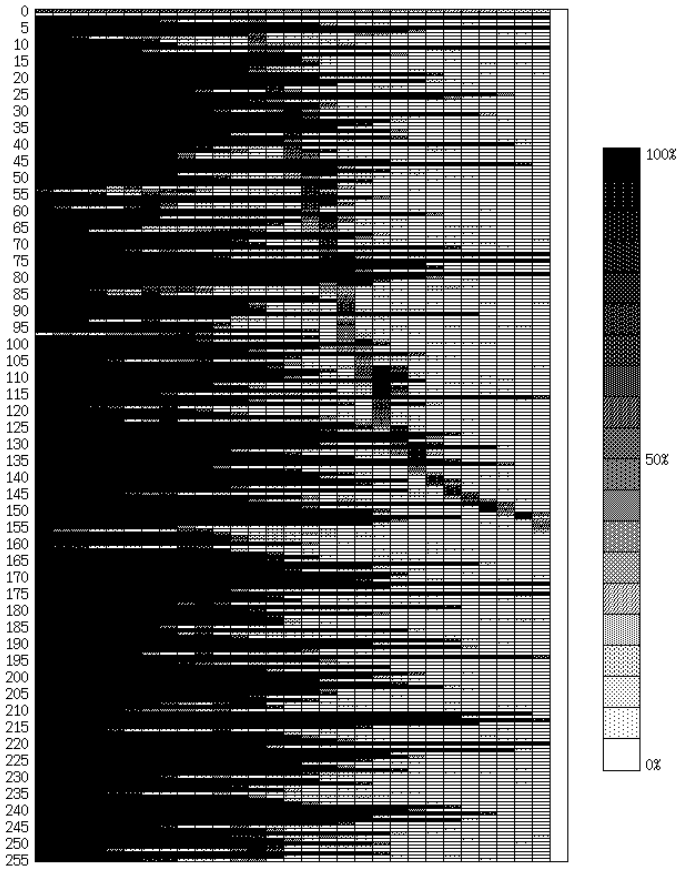

図 5.10: サイクリック割り付けにおける実行モニタ (終了時)

荷を考慮して割り当てを行なっていないので、結果として長いジョブを担当したプロセッサが全体の実行時間を決定することになる。これに対して、枝数制限方式では、各プロセッサに対して負荷はほとんど均等に配分され、各プロセッサはほとんど同時に実行を終了させることができる。

また、図 5.13 に各方式による台数効果のグラフを示す。ここでは 62 台での実行時間を基準としてスピードアップの割合を示した。ここで示されているように枝数制限方式は 62 台での実行に比べ 256 台ではプロセッサ台数が 4 倍なのに対して実行時間では 3.97 倍 ($Limit = 1$ の場合) とほとんど線形の台数効果が達成されている。一方、サイクリック方式や確率的方式はプロセッサ台数を増やしても台数分のスピードアップは望めない。

5.1.6 まとめ

一階述語論理による問題表現能力と高速な推論機能を持ったボトムアップ型定理証明システム MGTP は、準群問題のような有限領域の制約伝搬問題に対しても適用可能であることがわかったが、反面、負の制約伝搬が行なえないため、枝刈りが十分ではなかった。

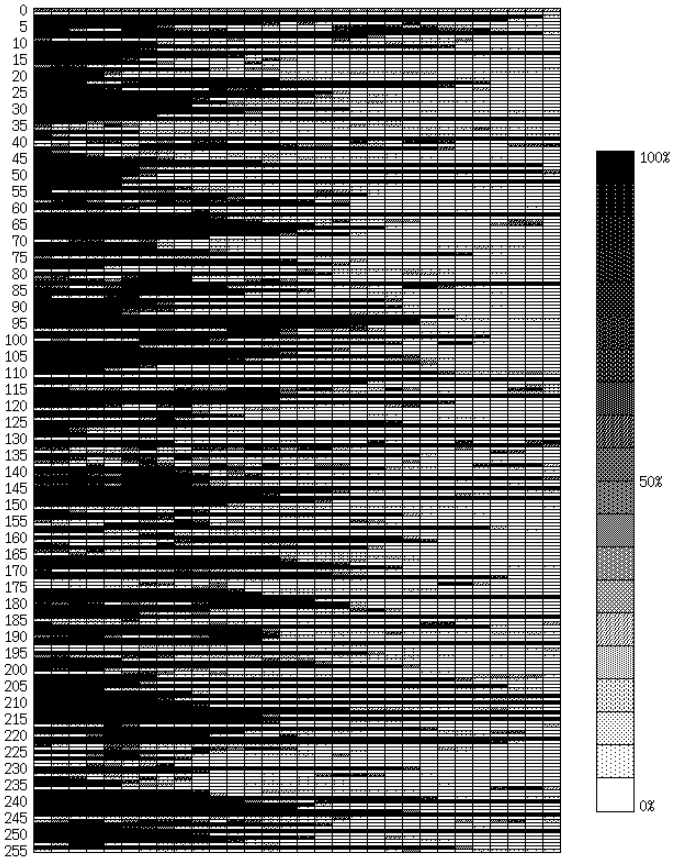

図 5.11: 確率的割り付けにおける実行モニタ (終了時)

我々は、一般の制約充足問題へ適用する際の MGTP の持つこのような弱点は、負のアトムが扱えないことによるものであると考え、負のアトムも扱え、また単位簡約化メカニズムや単位反駁メカニズムを兼ね備えた制約 MGTP (CMGTP) システムを SICStus Prolog 上で開発した。 CMGTP は、 DDPP や制約論理型言語に基づいて我々が開発した CP と同等の枝刈り能力を持ち、かつ MGTP の持つ一階述語論理形式による問題記述の簡便性をそのまま継承しているシステムである。 CMGTP では、直観的な制約伝搬のルールが CMGTP の入力節としてダイレクトに記述できるようになっているが、これは、 DDPP において QG5 のオーダ 10 で約 100 万節が入力節として必要になる (そしてそれらを生成するためのトランスレータが別に必要になる) のとは対照的である。

また並列化に関しては、サイクリック割り当て、確率的割り当て、枝数制限方式などの各方式で実験が行なわれ、準群問題に関しては枝数制限方式が最も優れた方式であることが確認できている。台数効果という点でも、ほとんど線形に近い効果を達成することができた。

CMGTP は制約伝搬に関しては非常に優れた記述能力を持っており、また並列化を行なうことによって優れた処理効率を示すことから、準群問題に限らず一般の有限領域制約充足問題のソルバーとして、広範な応用領域を持つと思われる。

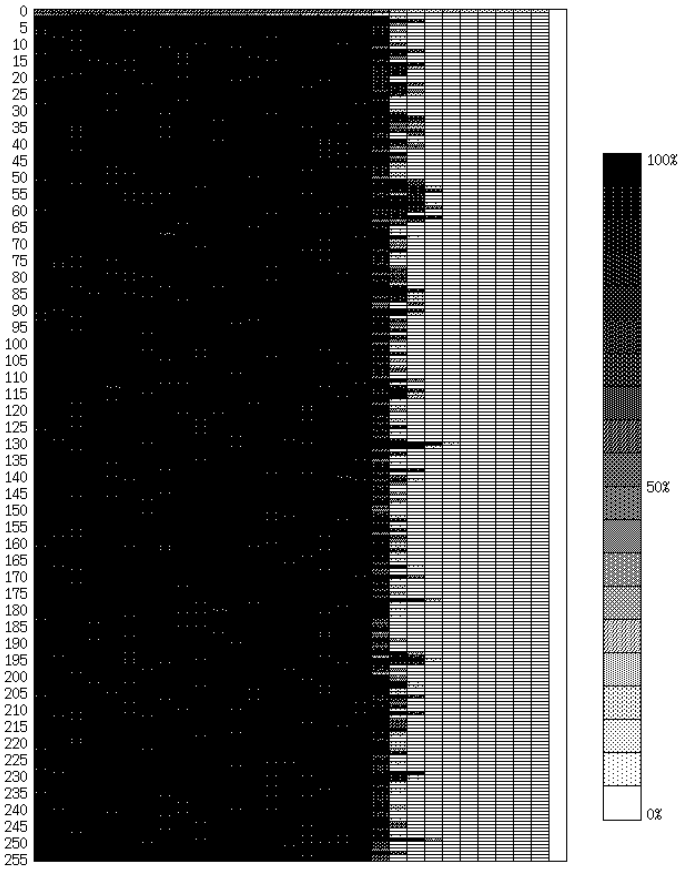

図 5.12: 枝数制限方式 ($Limit = 1$) における実行モニタ (終了時)

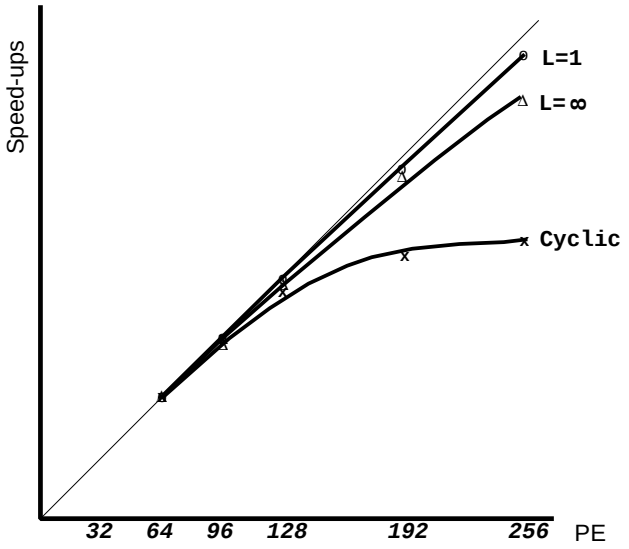

図 5.13: 台数効果

参考文献

[FH91] H. Fujita and R. Hasegawa, *A Model-Generation Theorem Prover in KL1 Using Ramified Stack Algorithm*, In Proc. of the Eighth International Conference on Logic Programming, The MIT Press, 1991.

[FHKF92] M. Fujita, R. Hasegawa, M.Koshimura and H. Fujita, *Model Generation Theorem Provers on a Parallel Inference Machine*, In Proc. of FGCS92, 1992.

[B89] F. Bennett, *Quasigroup Identities and Mendelsohn Designs*, In Canadian Journal of Mathematics 41, pp.341-368, 1989.

[CS89] P. V. Hentenryck, *Constraint Satisfaction in Logic Programming*, The MIT Press,1989.

[S92] J. Slaney, *FINDER, Finite Domain Enumerator: Version 2.0 Notes and Guides*, Technical report TR-ARP-1/92, Automated Reasoning Project, Australian National University, 1992.

[S94] J. Slaney, *FINDER, Finite Domain Enumerator System Description*, In Pro. of CADE-12, Nancy, France, 1994.

[MB88] R. Manthey and F. Bry, *SATCHMO: a theorem prover implemented in Prolog*, In Proc. of CADE-9, Argonne, Illinois, 1988.

[SFS93] J.Slaney, M.Fujita, and M.Stickel, *Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems*, To appear in Computers and Mathematics with Applications.

[FSB93] M. Fujita, J. Slaney, and F. Bennett, *Automatic Generation of Some Results in Finite Algebra*, In Proc. of International Joint Conference on Artificial Intelligence, 1993.

[H94] R. Hähnle, *MGTP with an extended constraint language*, private communication, 1994.

[FK94] 藤田正幸, 粂野文洋, MGTP による有限代数の新事実の発見, 情報処理学会論文誌 Vol 35, No 7 (pp1282-1292), 1994.

[HS94] R. Hasegawa and Y. Shirai, *Constraint Propagation of CP and CMGTP: Experiments on Quasigroup Problems*, Workshop 1C (Automated Reasoning in Algebra), CADE-12, 1994.

[SH95] Y. Shirai and R. Hasegawa, *Two Approaches for Finite-Domain Constraint Satisfaction Problems*, In Proc. of 12th International Conference on Logic Programming, 1995. (to appear)

5.2 Prolog 版操作マニュアル

5.2.1 インストール

CMGTP Prolog 版の tar ファイルには以下のファイルが含まれている。

mgtp4.pl	CMGTP エンジン
mgpl4.pl	トランスレータ
QG5-7.mg4	問題サンプルファイル

mgtp4.pl は CMGTP のエンジンであり、与えられた入力節を読み込んでモデル生成プロセスを繰り返すプログラムである。

mgpl4.pl はユーザが記述した入力節を Prolog プログラムに変換するトランスレータである。トランスレータの詳細については、トランスレータの機能説明書を参照されたい。 CMGTP Prolog 版におけるトランスレータは、他のバージョンのトランスレータと同様に、項メモリのプログラムも生成するようになっている。

QG5-7.mg4 は、サンプル問題の記述例で、準群問題 QG5 のオーダ 7 の例である。準群問題については、第 1 章の CMGTP の機能説明を参照のこと。

なお、本プログラムは SICStus Prolog 上で開発されたもので、 SICStus Prolog 処理系の上で動作する。

5.2.2 トランスレータのコンパイル

トランスレータは SICStus Prolog で書かれているので、実行形式を生成するためには、まず SICStus Prolog を起動させる必要がある。

```
$ sicstus
SICStus 2.1 #8: Tue May 11 21:04:52 JST 1993
| ?-
```

次に、mgpl4 をコンパイルし、セーブする。

```
| ?- compile(mgpl4).
| ?- save_mgpl(mgpl4).
```

これで、コンパイルされた実行形式 mgpl4 が生成される。

5.2.3 問題の KL1 プログラムへの変換

問題ファイルを CMGTP エンジンが参照できるような Prolog プログラムに変換するには以下のようにする。

```
$ mgpl4 <problem file>
```

問題は <problem file>.mg というファイル名になっている必要がある。上記の <problem file> の指定では、 extension は不要である。

トランスレータを用いて、 Prolog プログラムへの変換を行なうことにより、 <problem file>_mg4.pl という Prolog ファイルが生成される。

5.2.4 CMGTP 実行オブジェクトの作成

次に CMGTP の実行オブジェクトを作成する。実行オブジェクトの作成は、プログラム mgtp4.pl を SICStus Prolog 上でコンパイルし、セーブすることによって得られる。

```
$ sicstus
SICStus 2.1 #8: Tue May 11 21:04:52 JST 1993
| ?- compile(mgtp4).
| ?- save_mgtp(mgtp4).
```

これで、 CMGTP の実行オブジェクト mgtp4 が作成される。

5.2.5 実行

実行は以下のように行なう。

```
$ mgtp4 <problem file>
```

例として、 QG5-7.mg をトランスレートして、実行させる場合は以下のように行なう。

```
$ mgpl4 QG5-7

$ mgtp4 QG5-7
```

5.2.6 実行結果

実行途中には、場合わけが生じた時やモデルが見つかった時、また、 branch が fail した時にコメントがでる。実行が終了した時点で、モデルの総数、 branch の総数、実行時間などについて表示される。

QG5-7.mg に対して、 CMGTP を実行した結果を以下に示す。

```
$ mgtp4 QG5-7


{compiling QG5-7_mg4.pl...}
{QG5-7_mg4.pl compiled, 1730 msec 45904 bytes}
:(1,dom(1))
:(2,dom(2))
```

```
:(3,dom(3))
:(4,dom(4))
:(5,dom(5))
:(6,dom(6))
:(7,dom(7))
:(8,p(1,1,1))
:(9,p(2,2,2))
:(10,p(3,3,3))
:(11,p(4,4,4))
  ......

:(89,not(p(7,4,4)))
:(215,not(p(6,4,7)))
:(216,not(p(4,6,7)))
--> p(1,7,6)
:(217,p(1,7,6))
:(218,not(p(4,7,6)))
  ......

:(266,not(p(6,4,2)))
--> p(1,6,7)
:(267,p(1,6,7))
  ......

:(317,p(2,4,3))
:(318,not(p(2,5,3)))
Failed Branch := 1
--> p(3,6,7)
:(267,p(3,6,7))
  ......

:(303,not(p(3,2,6)))
Failed Branch := 2
--> p(4,7,6)
:(217,p(4,7,6))
  ......

:(260,not(p(6,2,1)))
--> p(1,6,7)
:(261,p(1,6,7))
  ......

:(350,p(1,2,4))
model case:2.1:size(350).
--> p(2,6,7)
:(261,p(2,6,7))
  ......
```

```
:(350,p(2,3,4))
model case:2.2:size(350).
--> p(3,6,7)
:(261,p(3,6,7))
  ......

:(350,p(3,1,4))
model case:2.3:size(350).
====================================
satisfiable.

- - - - - - - - - - - - - - - - - -
Number of Models  : 3
Failed Branches   : 2
Execution Runtime : 9.541 seconds.
- - - - - - - - - - - - - - - - - -

$
```

公開したバージョンでは、生成されたモデルをすべて表示しており、特に CMGTP では否定モデルが多数生成されていることがわかる。もし、このようなモデル出力が不要であれば、プログラム中でモデルの表示を行なっている部分をコメントアウトすれば良い。

`-->` は場合わけが生じたことを示しており、`Failed Branch` は枝が fail したときに表示される。

また、モデルが発見された時には、`model case:2.1:size(350)` などの表示がある。具体的に、モデルの中身を知りたい場合、あるいは、それを表の形で出力したい場合には、ユーザがプログラム中で、モデル出力ルーチンを記述することは容易である。

実行が終った段階で、生成されたモデルの個数や失敗した枝の個数、また実行時間が表示される。

5.3 KLIC 版操作マニュアル

5.3.1 インストール

CMGTP の tar ファイルには以下のファイルが含まれている。

cmgtp.kl1	CMGTP エンジン
mg2kl1.pl	トランスレータ
QG5-8.mg	問題サンプルファイル

cmgtp.kl1 は CMGTP のエンジンであり、与えられた入力節を読み込んでモデル生成プロセスを繰り返すプログラムである。

mg2kl1.pl はユーザが記述した入力節を KL1 プログラムに変換するトランスレータである。トランスレータの詳細については、トランスレータの機能説明書を参照されたい。 CMGTP KLIC 版におけるトランスレータは、他のバージョンのトランスレータと同様に、項メモリのプログラムも生成するようになっている。

また、 KL1 版のところでも述べているように、現在 KL1 版についてはトランスレータが準備されていないので、この KLIC 版のトランスレータを使用することになる。

QG5-8.mg は、サンプル問題の記述例で、準群問題 QG5 のオーダ 8 の例である。準群問題については、第 1 章の CMGTP の機能説明を参照のこと。

5.3.2 トランスレータのコンパイル

トランスレータは SICStus Prolog で書かれているので、実行形式を生成するためには、まず SICStus Prolog を起動させる必要がある。

```
$ sicstus
SICStus 2.1 #8: Tue May 11 21:04:52 JST 1993
| ?-
```

次に、mg2klic をコンパイルし、セーブする。

```
| ?- compile(mg2kl1).
| ?- save.
```

これで、コンパイルされた実行形式 mg2kl1 が生成される。

5.3.3 問題の KL1 プログラムへの変換

問題ファイルを CMGTP エンジンが参照できるような KL1 プログラムに変換するには以下のようにする。

```
$ mg2kl1 <problem file> merc
```

ここで、merc はバージョンの指定であり、今回のリリースバージョンでは、ここでは常に merc を指定する。また、問題は <problem file>.mg というファイル名になっている必要がある。上記の <problem file> の指定では、 extension は不要である。

トランスレータを用いて、 KL1 プログラムへの変換を行なうことにより、
<problem file>_merc.kl1 という KL1 ファイルが生成される。

5.3.4 コンパイル

次に、 CMGTP のエンジンと問題を KLIC でコンパイルし、リンクする。

```
$ klic cmgtp.kl1 <problem file>_merc.kl1
```

コンパイルオプションは必要に応じてつけること。詳しくは KLIC のマニュアルを参照されたい。

コンパイルされた結果として、実行オブジェクト a.out が生成される。

5.3.5 実行

実行は以下のように行なう。

```
$ a.out
```

実行時のオプションについては、 KLIC のマニュアルを参照されたい。

5.3.6 実行結果

実行途中には、場合わけが生じた時やモデルが見つかった時、また、 branch が fail した時にコメントがでる。実行が終了した時点で、モデルの総数、 branch の総数、実行時間などについて表示される。

QG5-8.mg に対して、 CMGTP を実行した結果を以下に示す。

fail した枝で、failed_branch_by_Fsimpl になっているものは、 unit simplification によってある選言が空になって fail したもの、また、failed_branch_by_Fref になっているものは、 unit refutation によって fail したものを表す。

```
[p(1,8,3),p(4,8,3)]
--> p(1,8,3)
[p(4,1,8),p(5,1,8),p(6,1,8),p(7,1,8)]
--> p(4,1,8)
failed_branch_by_Fref
--> p(5,1,8)
failed_branch_by_Fref
--> p(6,1,8)
failed_branch_by_Fref
--> p(7,1,8)
```

```
failed_branch_by_Fref
--> p(4,8,3)
[p(1,8,4),p(5,8,4)]
--> p(1,8,4)
[p(2,1,8),p(5,1,8),p(6,1,8),p(7,1,8)]
--> p(2,1,8)
failed_branch_by_Fsimpl
--> p(5,1,8)
failed_branch_by_Fref
--> p(6,1,8)
failed_branch_by_Fref
--> p(7,1,8)
failed_branch_by_Fref
--> p(5,8,4)
[p(1,8,5),p(6,8,5)]
--> p(1,8,5)
failed_branch_by_Fsimpl
--> p(6,8,5)
[p(2,1,8),p(4,1,8)]
--> p(2,1,8)
failed_branch_by_Fsimpl
--> p(4,1,8)
model found

heap size = 1048576 words
43950 ms total; 43570 user; 380 system
  0 swaps; 34 minor page faults; 0 major page faults
  0 block inputs; 0 block outputs
  1226 context switches (0 voluntary)
  122 GC
  251 suspensions; 250 resumptions
sat
**********************************
No. of Failed Branches  = 10
No. of Models           = 1
**********************************
```

5.4 KL1 版操作マニュアル

5.4.1 KL1 版の特徴

KL1 版においては、PIM/m 上での並列実行を行なうことを前提とし、並列版の準群問題用のプログラムのみをリリースする。

また、トランスレータは KLIC のものを併用しているが、KL1 トランスレータは CMGTP においては用意されていないので、生成されたプログラムにおける KLIC と KL1 の組み込み関数などの相違は手入力で修正する必要がある。

CMGTP の利用に関しては、今後は、PIM での利用から汎用の並列 UNIX マシン上での利用にシフトしていくことが予想されるため、KL1 版に関して今後サポートが行なわれる予定はない。

今回のリリースに含まれるファイルを図 5.14 に示す。

問題ファイルとしては、QG1 オーダ 6 から 9 まで、QG2 オーダ 7 から 9 まで、QG5 オーダ 7 から 16 まで、また QG7 はオーダ 11 から 14 までとなっている。

また CMGTP のエンジンの方では、並列化方式の違いに従ってサイクリック方式版、確率的方式版、また枝数制限方式版では $Limit = 1, 2, 3, 5, 10, \infty$ を用意してある。なお枝数制限方式版におけるパラメータ $Limit$ の設定箇所は容易にわかるので、その部分を任意の数字に置き換えることにより好みのパラメータで実行させることが可能である。

5.4.2 PIM/m 上での実行方法

CMGTP での準群問題を実際に PIM 上で動作させる方法について説明する。

ユーザはまずリスナを開き、モニタの表示、統計情報出力などの環境設定を好みに応じて行なう。以下の例は、xrmonitor を画面上に表示させ、統計情報を出力させるモードを指定している。

```
Shell> listener

Listener> setenv pmeter:display_node = ss145 .
Listener> xrmonitor & .
Listener> st.
Listener>
```

次に、MGTP 用の GC プログラムをロードし、レベルで 70 を指定する。これは、あるプロセッサのメモリ消費量が 70 % を越えた時、一斉に GC をかけることを意味している。

```
Listener> load("gc.sav").
Listener> mgtp_gc:allocate(70) & .
Listener>
```

問題ファイル：

QG1-6_index.kl1	QG1 オーダ 6
QG1-7_index.kl1	QG1 オーダ 7
QG1-8_index.kl1	QG1 オーダ 8
QG1-9_index.kl1	QG1 オーダ 9
QG2-7_index.kl1	QG2 オーダ 7
QG2-8_index.kl1	QG2 オーダ 8
QG2-9_index.kl1	QG2 オーダ 9
QG5-7_index.kl1	QG5 オーダ 7
QG5-8_index.kl1	QG5 オーダ 8
QG5-9_index.kl1	QG5 オーダ 9
QG5-10_index.kl1	QG5 オーダ 10
QG5-11_index.kl1	QG5 オーダ 11
QG5-12_index.kl1	QG5 オーダ 12
QG5-13_index.kl1	QG5 オーダ 13
QG5-14_index.kl1	QG5 オーダ 14
QG5-15_index.kl1	QG5 オーダ 15
QG5-16_index.kl1	QG5 オーダ 16
QG7-11_index.kl1	QG7 オーダ 11
QG7-12_index.kl1	QG7 オーダ 12
QG7-13_index.kl1	QG7 オーダ 13
QG7-14_index.kl1	QG7 オーダ 14

CMGTP エンジン：

cmgtpCycle.kl1	サイクリック割り当て方式
cmgtpProb.kl1	確率的割り当て方式
cmgtpLimit1.kl1	枝数制限方式 ($Limit = 1$)
cmgtpLimit2.kl1	枝数制限方式 ($Limit = 2$)
cmgtpLimit3.kl1	枝数制限方式 ($Limit = 3$)
cmgtpLimit5.kl1	枝数制限方式 ($Limit = 5$)
cmgtpLimit10.kl1	枝数制限方式 ($Limit = 10$)
cmgtpLimitInf.kl1	枝数制限方式 ($Limit = \infty$)

ユーティリティ：

gc.sav	MGTP 用 GC ユーティリティ

図 5.14: 今回のリリースに含まれるファイル

プログラムのコンパイルは、問題と CMGTP のエンジンを同時にコンパイルする。あるいは、それぞれコンパイルしたものをリンクする。

```
Listener> compile(['QG5-13_index.kl1','cmgtpLimit5.kl1']).
Listener>
```

実行は、モジュール名 mgtp、述語名 do で、引数にはラテン方陣のサイズと使用するプロセッサ数を指定する。以下は、上の例に沿って、 QG5 のオーダ 13 を 256 プロセッサで解かせようとしたものである。

```
Listener> mgtp:do(13,256).
Listener>
```

実行が終了すると、枝数、モデル数、また実行時間などが表示される。

```
Listener> mgtp: do(13,256) .

Number of Failed Branches = 15173
Number of Models = 0
 5498685340 reductions
 229369 msec
```

この実行例では、モデルの数が 0、失敗した枝数が 15,173 個で、実行時間が約 229 秒であったことを表している。