ICOT Technical Report: TR-0885

TR-0885

MGTP: 並列論理型言語KL1による モデル生成型定理証明系

長谷川　隆三、藤田　博（三菱）

August, 1994

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Institute for New Generation Computer Technology

MGTP: 並行論理型言語KL1による モデル生成型定理証明系

長谷川 隆三[†]　　藤田 博[‡]

一階述語論理のための定理証明系を，モデル生成法に基き並行論理型言語KL1で効率良く実現する方法について述べる．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であり，照合操作で十分となる．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．しかしながら，モデル生成法では連言照合の冗長性のために計算量が著しく増大する場合がある．この問題を解決するため，2種の方式を提案する．従来，同様な冗長計算除去方式としてRETEが知られているが，それはホーン節集合に限られていた．これに対し我々の方式は，非ホーン節を含む一般の節集合に対して適用可能である．

MGTP: A Model Generation Theorem Prover in the Concurrent Logic Programing Language KL1

Ryuzo HASEGAWA*　　Hiroshi FUJITA[†]

This paper presents a novel technique to implement model-generation based theorem provers for first-order logic in the concurrent logic programming language KL1. The model generation method makes it possible to use only matching in place of full-unification with occurs-check if the given clause set satisfies the condition called range-restrictedness. As a consequence, a logical variable in the clause set can be represented with a KL1 variable; unification can be executed directly by the head-unification of a KL1 clause; and many of the KL1 system features can be made available for efficient execution of the theorem provers built upon it. In addition, this paper presents two kinds of methods to eliminate redundant computations in conjunctive matching which would be a primary cause of significant speed-down of the model-generation based theorem provers. Although a similar method called RETE has been proposed for eliminating some redundancy in production systems, the method is restricted to Horn clause sets. Our methods, however, can be applied to more general cases including non-Horn clauses.

1　まえがき

ホーン論理を基盤とする論理プログラミングは，元来定理証明分野で派生した技術であるが，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，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るなど，著しい発展を遂げた．本プロジェクトでは同時に「並列性」が重点的に追究され，並行論理型言語KL1[6]とその専用マシンPIMが開発された．

しかし，論理プログラミング分野の最近の動向をみると，対象をホーン論理から非ホーン論理へ拡張する試みが増えてきた．実際ホーン論理の研究はすでに成熟の域に達した感があり，関心が非標準論理を含む多様な領域へ拡がりつつある．これに伴い，一階述語論理をベースに地道な研究が続けられてきた定理証明分野の本流が再び脚光を浴びようとしている．

一方，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．たとえば，StickelはPTTPと呼ばれる定理証明系をPrologをベースに開発した[4]．PTTPは，非ホーン節を含む入力節を，対偶をとることによってホーン節化し，Prologの処理系を

[†]新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology
[‡]三菱電機中央研究所
Mitsubishi Electric Corporation

うまく活用することに成功している．さらに，Manthey と Bry は充足性検査システム SATCHMO[3] を非常に簡潔な Prolog プログラムで実現している．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であり，照合操作で十分となる．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った．

さらに，モデル生成法で顕著な性能劣化の原因となる連言照合における重複計算を除去するため，2 種の方式を考案した．従来，同様な重複計算除去方式として RETE が知られているが，それはホーン節集合に限られていた．これに対し我々の方式は，非ホーン節を含む一般の節集合に対して適用可能である．

本論文では，一階述語論理と KL1 言語の関係にふれた後、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 法を提案し，その効果を定量的に評価する．

2 一階述語論理と KL1

並行論理型言語 KL1 のプログラムは，ガード付きホーン節と呼ばれる節形式 (KL1 節と呼ぶ) の集合によって表現される．また，KL1 の実行系は定理証明法の一つであるレゾリューション法に基づいている．したがって，一階述語論理式が KL1 節として直接表現され，KL1 の実行系が健全かつ完全であるならば，KL1 で直接一階述語論理の定理証明が可能なはずである．

しかしながら，任意の一階述語論理式 (とりわけ非ホーン節) を KL1 節で直接的に表現するのは困難であるし，KL1 の実行系も一階述語論理の証明系としては完全でも健全でもない．健全でないのは，単一化において出現検査を省略しているからであり，完全でないのは，節選択の際，複数の可能性があってもコミッテッドチョイスによってそのうち一つのみが選ばれ，他は捨てられてしまうからである．

このように，KL1 処理系を直接一階述語論理の定理証明系として扱うには無理があるので，メタプログラミング的なアプローチを取らざるを得ないが，KL1 に受け継がれている論理型言語としての特性を可能な限り活用することが望ましい．

入力節中の論理変数については，(1) KL1 の基底項で表現する，あるいは (2) KL1 変数で直接表現する，という二つのアプローチが考えられる．基底項表現によれば，対象レベルの変数と実装レベルの変数が厳密に区別されるので，意味論的な観点から好都合である．しかし，変数管理手続きを KL1 プログラムで実現することになるため，実行時のオーバヘッドが大きくなり，効率上耐え難い方法といえる．

これに対して，KL1 変数による直接表現をとった場合，変数管理の多くを KL1 処理系に委ねることができ，オーバヘッドの問題は軽減される．

しかしながら，並行言語としての仕様上，KL1 変数は同期の機構に関与しているため，ある時点における KL1 変数の束縛状態についてユーザプログラムから任

意に判定／操作することが許されてない．また，その結果，KL1 変数を含む項の複製を作ることもできない[1]．

こうした問題を解決し，KL1 の論理型言語としての特性を活かしつつ，健全かつ完全な一階述語論理の定理証明系を実現するため，我々はモデル生成法と呼ばれる証明方式を採用した．

3 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-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 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 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 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 $consts(S)$，関数記号の集合を $functs(S)$ とするとき，節集合 S に dom 定義節集合

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

  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 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，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 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，$\sigma A_i \ (1 \leq i \leq n)$ がすべて M で充足され，かつ $\sigma B_j \ (1 \leq j \leq m)$ 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 $A_1, \ldots, A_n \rightarrow false$ ならびに，ある基礎代入 σ が存在して，$\sigma A_i \ (1 \leq i \leq n)$ がすべて M で充足されるならば，M を棄却する．

[1] Prolog では var(あるいは nonvar) 組み込み述語によって，変数束縛状態の判定が可能で，項の複製も copy_term という組み込み述語を用いたり，これと同等のものを var 述語を用いてユーザが定義することが可能である．

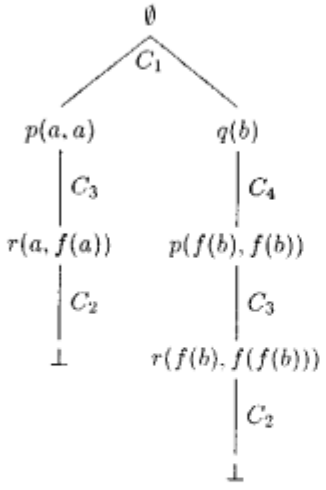

図 1: 問題 S の証明図
Fig. 1 Proof tree for S

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) を考えよう．

$$C_1: \quad true \rightarrow p(a,a);q(b).$$
$$C_2: \quad r(X,f(X)) \rightarrow false.$$
$$C_3: \quad p(X,X),p(X,Y) \rightarrow r(X,f(Y)).$$
$$C_4: \quad q(X) \rightarrow p(f(X),f(X)).$$

モデル生成法による問題 S の証明木を図 1に示す．この証明木において，$\emptyset$ でラベル付けされた根は初期モデル候補 ($M_0=\emptyset$)，$p(a,a)$ などでラベル付けされた節点はモデル候補の拡張に用いられたアトム，枝の分岐はモデル候補の場合分け，$\bot$ でラベル付けされた葉は根からその葉に至る枝上のすべてのアトムを要素とするモデル候補が棄却されたこと，をそれぞれ表している．

この証明木はモデル生成手続きに従い，以下のようにして作られる．C_1 に関してモデル拡張規則を適用することにより，M_0 は $M_1=\{p(a,a)\}$ と $M_2=\{q(b)\}$ に場合分けされる．M_1 は C_3 によって $M_3=\{p(a,a),r(a,f(a))\}$ に拡張される．M_3 は C_2 に関してモデル棄却規則が適用できるので棄却される．一方，M_2 は C_4 によって $M_4=\{q(b),p(f(b),f(b))\}$ に，M_4 は C_3 によって $M_5=\{q(b),p(f(b),f(b)),r(f(b),f(f(b)))\}$ に拡張され，M_5 は C_2 によって棄却される．こうしてモデル候補集合が空となるので，問題 S は充足不能であることが証明される．

4 モデル生成型定理証明系 MGTP

4.1 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が行なわれる．論理変数はモデル候補要素には現れず MG 節にのみ現れるから，この場合出現検査付きの単一化は不要であり，常に前件リテラル側の変数を束縛する一方向の単一化，すなわち照合操作で十分である．

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 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[2]．

```
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．

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[3]．

[2] 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．

[3] 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．

```
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 4: 基本 MGTP 証明系
Fig. 4 Basic MGTP

```
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 5: 基本 MGTP 証明系の連言照合部
Fig. 5 Conjunctive matching part of MGTP

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 $Delta$ を一つ選び，これに対し subsTest/3 により M のもとでの充足性を調べる．もし $Delta$ 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，$Delta$ が選言である場合，caseSplit/6 により場合分けを行なう．$Delta$ 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，負節についてcjm/5 を呼び，モデル棄却規則の適用可能性を調べる．もし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，非負節についてcjm/5 を呼び，モデル拡張規則の前半，すなわち前件の連言照合を適用し，前件が M で充足された非負節の後件からなる集合を得て，これを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．

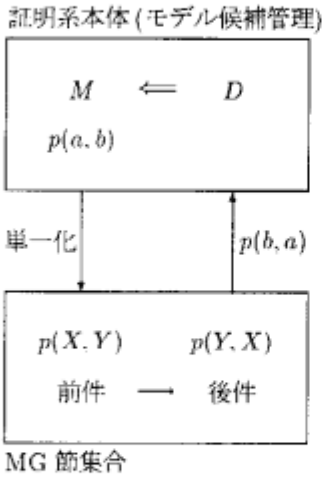

図 2: 証明系の KL1 による実現
Fig. 2 KL1 implementation of the prover

4.2 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，たとえば，　C_n :　$p(X,X), q(X,Z) \rightarrow p(X,Z)$.
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．

この表現法において，連言照合は次のような手順で行なわれる．まず，モデル候補中からアトム A_i を一つ選び，1 本めの c/4 節を呼びだし，第 1 リテラル $p(X,X)$ と照合を行なう．照合が成功したなら，そのときに限り次のアトム A_j を選び，前の照合で束縛された変数 X とともに 2 本めの c/4 節を呼び，第 2 リテラル $q(X,Z)$ の照合を行なう．

この方式に基づき，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 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．

問題 S の MG 節 C_3 が 2 本の KL1 節で表現されていることに注意．C_3 に対する連言照合は以下のように進められる．まず，証明系本体はモデル候補から一つのアトム A_1 を選び，ゴール $c(3, A_1, [], R_1)$ を起動して C_3 の最初のリテラル $p(X,X)$ を A_1 と照合する．もしこの照合が失敗すれば，最後の KL1 節により $R_1 = fail$ という結果が返される．照合が成功すれば，C_3 に対応する 1 本めの KL1 節によって $R_1 = (1:[X])$ という結果が返され，証明系本体は C_3 の次のリテラル $p(X,Y)$ の照合を行なうためにモデル候補から別のアトム A_2 を選び，ゴール $c(3, A_2, 1:[X], R_2)$ を起動する．第 3 引数の $1:[X]$ によって第 1 リテラルで束縛された共有変数 X の値が伝えられる．この X の束縛状態のもとで，$p(X,Y)$ が A_2 と照合するかどうかテストされ，照合に成功すると $R_2 = r(X, f(Y))$ という結果が返される．

```
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 3: 問題 S の KL1 節表現
Fig. 3 KL1 representation of Problem S

4.3 証明系本体の実現

証明系本体は，図 4および 5 に示す KL1 プログラムとして実現することができる．

5 連言照合における冗長性除去

5.1 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．たとえば，前件に2個のリテラルを持つようなMG節$A_1, A_2 \rightarrow B$に対する連言照合を考える．この場合，図5に示すメインループのあるステージにおいてcjm/5が呼ばれる際，リテラルA_1, A_2との照合のため，$[Delta|M]$からアトムの対が取り出される．そのアトム対の総数は，$(1+|M|)^2$となる．ところが，このうち$|M| \times |M|$個のアトム対については1ステージ前のcjmで既に選ばれているはずであり，これらの対に対して連言照合を再び行なう[4]必要はない．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[2]とMERC法について述べる．

5.2 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．

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

[4] これをステージ間冗長性という．

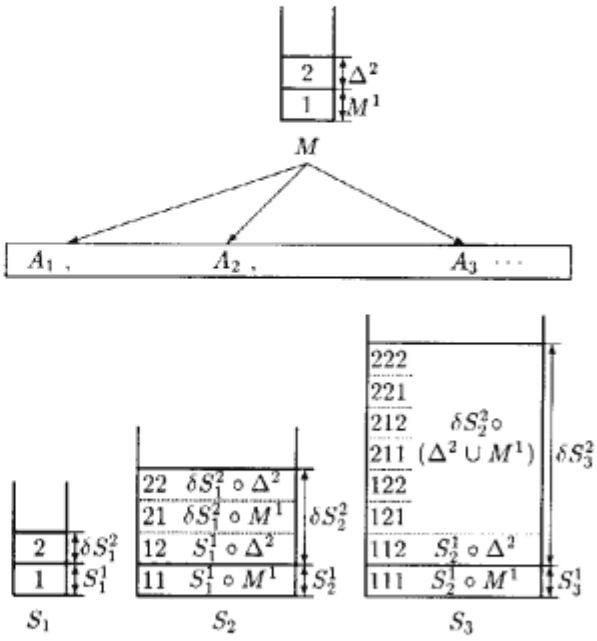

図 6: インスタンススタック
Fig. 6 Instance stacks

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．非ホーン節に適用するには，場合分けを色付きトークンで識別したり，あるいはネットワークの複製が必要となり，実現が困難である．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．それは，モデル候補Mならびに各インスタンススタックS_iは，ボトムからトップまでのアトムの集合で表現

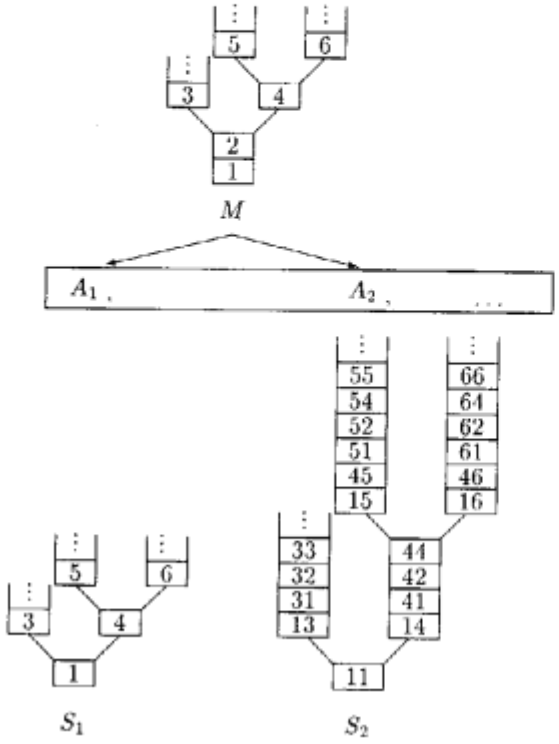

図 7: 分岐スタック
Fig. 7 Ramified stacks

```
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 8: 連言照合部の RAMS 版
Fig. 8 RAMS version of CJM

されており，その拡張はスタックトップにおいてのみ行なわれ，連言照合時にこれを参照する際もスタックトップからのみ行なわれるからである．非ホーン節によって場合分けが起こるときにはスタックトップが複数生じるが，各場合のモデル候補中でアトムを参照する際には，それぞれのスタックトップからボトム方向に辿る道筋が一意であり，場合分けの履歴を調べる必要がない．このようにして，非ホーンの MG 節を含む一般の場合には図 7に示すような分岐するスタックが生成される．本図では，第 1 リテラル A_1 と M 中のアトム 2 の照合は失敗するものと仮定しているので，この結果は S_1 には積まれていない．

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 8に示す．

5.3 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 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 9中 (イ) に属する，入口リテラルが一つのパタン

	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 9: MERC 法による連言照合
Fig. 9 MERC version of CJM

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からである．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．

このようにして，問題SのKL1節表現を作ると図10のようになる．ここで，MG節C_3においてc3_1，c3_2のように二つの入口リテラルごとに別個のKL1節が用意されている．また，c3_1_2は，二つの前件リテラルをファクタリングしたものに対応している．MERC法を採用する場合，基本MGTP証明系の連言照合部は一部図11のように修正される．

```
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 10: 問題SのMERC版KL1表現
Fig. 10 MERC version of KL1 representation for problem S

```
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 11: 連言照合部のMERC版
Fig. 11 MERC version of CJM

6 議論

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．

なお，本論文では2本以上の MG 節で共通の前件リテラルを含む場合に生じる，連言照合の節間冗長性については議論しない．

7 評価

本節では，連言照合における冗長計算の有無が如何に MGTP の全体性能に影響を与えるかを，定理証明系のベンチマーク用例題集 TPTP ライブラリ [5] を用いて検証する．

一般に，得られる証明は推論方式自体や戦略によって変わり，異なる証明系の比較評価は難しい．本実験では，これらの影響をなくし，連言照合の効率特性のみを調べるため，比較対象の各版 (Naive 版 (基本 MGTP)、RAMS 版，MERC 版) とも共通に以下の戦略を用いた．

1. 図 4の`addNew/3` におけるモデル拡張候補集合 D への Newの追加／取り出しはキュー (FIFO) を用いる．
2. `pickup/3` では $Delta$ として単位節を優先して選択する．
3. 項の重み付けによる並び換え (ソーティング) 戦略や削除戦略は用いない．

したがって，証明の違いは連言照合における照合順序の差異のみから生じる．そこで本評価実験では，MERC 版と RAMS 版に対し，Naive 版と証明を一致させるために Newの要素の整列を行なっている．

表 1に，Naive 版，RAMS 版，MERC 版 MGTP の計測結果を示す．本表中，Clauses 欄は MG 節の本数と KL1 節の本数 (括弧内は MERC 版の KL1 節本数) を表し，Proof 欄の Br は反駁証明木の分枝数，Dp は枝の長さ (棄却されたモデル候補の大きさ) の平均値を表す．また，CJM 欄の値は連言照合における KL1 節 `c/4` の呼び出し総数で，RAMS の括弧内はインスタンススタックに要したセルの総数を表す．Time 欄の値は SPARC10-30 上の KL1 処理系 KLIC[1] による CPU 時間を表す．ただし，証明一致のための整列に要した時間は除いてある．また，"TO" は 1 時間以内に解が得られなかったことを，"-" は測定不能を表す。

KL1 節の本数は Naive 版と RAMS 版では常に同数だが，MERC 版の場合 SYN006-1 を除いて 2 割から 3.5 倍位までに増加している．証明木は三者とも完全に一致しており，LAT，PLA，GRP はホーン問題なので，棄却モデル候補数は 1 個 (分枝数 Br が 1) である．その他は非ホーン節を含む問題で，棄却モデル候補数は数十から数十万である．また，棄却モデル候補 1 個あたりの大きさ (アトム数) は，ホーン問題では数十から千数百，非ホーン問題では数十の規模である．

CJM の回数については，Naive 版と RAMS 版では 1 桁から 2 桁以上の開きがあり，RAMS 版による大幅な削減効果が見られる．RAMS 版と MERC 版を比べると，CJM 回数は SYN006-1 で同じ，PLA001-1，PLA003-1，SYN009-1 で RAMS 版の方が 3 割増しから 2 倍弱であるが，その他の問題では逆に MERC 版の方が 1 割から 3 倍近く増加している．全体の傾向としては，RAMS 版の方が概ね CJM 回数は少なくて済んでいる．しかし，RAMS 版では SYN006-1 を除き，インスタンススタックのために他の版より数十から数十万セルのメモリを余分に要している．

CPU 時間で見ると，Naive 版とそれ以外の RAMS 版や MERC 版との間には 10 倍から数千倍以上の開きがあるが，RAMS 版と MERC 版は大差はなく，相互の CPU 時間比は問題により 1 割から 6 割程度の範囲で変動している．RAMS 版では，CJM 回数が MERC 版より少ないにもかかわらず CPU 時間が長くなっている場合があるが，これはインスタンススタック操作に関するオーバヘッドのためと思われる．

以上の結果から，まずステージ間冗長性による無駄な連言照合を除くだけで全体性能を著しく改善できることがわかる．次に，RAMS 版と MERC 版の差異を分析するために，PLA001-1 においては MERC が，PRV009-1 では逆に RAMS の方が，いずれも倍近く速くなっていることに注目する．

PLA001-1 で前件リテラルが 2 個以上の節は，

$$at(A,B,C,D), next\_to(A,E) \rightarrow \ldots$$

という異なる述語の 2 個の前件リテラルを持つ節のみである．ここで，第 2 リテラル $next\_to(A,E)$ に対し，Δとして $at(\_,\_,\_,\_)$ を照合させると失敗する．したがって，RAMS 版ではΔ失敗冗長性のために MERC 版よりも余計な CJM を行う結果，証明時間が長くなると

表 1: MGTP 各版の計測結果
Table 1 Evaluation Results

Problem	Clauses			Proof		CJM				Time (sec)		
	MG	KL1	(merc)	Br	Dp	Naive	RAMS	(stack)	MERC	Naive	RAMS	MERC
LAT005-1	31	52	(182)	1	395	-	8065k	(20386)	11669k	TO	223.2	183.2
LAT005-2	31	46	(135)	1	248	-	1908k	(7664)	2453k	TO	28.2	30.0
MSC006-1	6	10	(17)	612	24	1482k	115k	(2401)	125k	19.8	1.5	1.4
PLA001-1	16	18	(22)	1	1378	-	3789k	(2734)	1922k	TO	41.6	25.8
PLA003-1	11	20	(42)	1	89	513k	14k	(152)	9k	832.3	0.3	0.3
PUZ012-1	18	23	(33)	165	27	606k	31k	(332)	40k	5.5	0.4	0.4
PUZ025-1	24	47	(108)	90	22	424k	34k	(773)	68k	4.4	0.4	0.6
SYN006-1	7	9	(9)	96	84	328k	4k	(0)	4k	41.3	0.2	0.2
SYN009-1	7	9	(22)	19683	12	2918k	246k	(12)	187k	49.6	4.6	3.5
SYN015-2	26	36	(62)	196	43	17057k	687k	(7521)	737k	603.0	8.9	7.8
SYN036-3	36	63	(123)	516	33	15171k	886k	(12548)	1314k	197.1	12.6	12.7
SYN037-1	36	63	(123)	358	33	9053k	535k	(7473)	832k	120.4	7.8	8.0
GRP028-1	4	12	(33)	1	1834	-	64835k	(35345)	83559k	TO	1050.3	971.0
PRV009-1	9	19	(64)	266240	31	-	35307k	(314110)	81122k	TO	498.1	818.2

考えられる.

一方，PRV009-1 においては，

$$le(m,A), lt(A,i), lt(j,B), le(B,n) \rightarrow \ldots$$
$$le(m,A), le(A,B), le(B,j) \rightarrow \ldots$$
$$le(i,A), le(A,B), le(B,n) \rightarrow \ldots$$

という 3 個以上の前件リテラルを持つ節が 3 本ある．このように前件リテラル数 3 以上の節が多く含まれ，かつ 1 本の節の前件に同じ述語が複数出現するような問題では，Δ失敗冗長性よりも M-M 冗長性の影響の方が優勢となるため，MERC 版の方が RAMS 版よりも証明時間が長くなると考えられる[5]．

以上見てきた通り，RAMS 版においてはΔ失敗冗長性とインスタンススタックに伴うオーバヘッド，MERC 版においては M-M 冗長性というそれぞれに固有の性能劣化の原因があり，その効果の度合は問題の特性に応じて異なってくる．

8 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，二つの方式を考案した．両方式は次の特徴を有する．

- RAMS 法では，節の前件リテラルとモデル候補中のアトムとの照合結果をインスタンススタックに格納することにより，ステージ間冗長性と MERC 法における M-M 冗長性に関する重複計算を防ぐ．
- MERC 法では，必ず節の前件リテラルの一つ以上についてモデル拡張アトム (Δ) との照合を優先して行なうことにより，ステージ間冗長性を防ぐとともに，RAMS 法におけるΔ失敗冗長性を防ぐ．

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．

本稿では SPARC10-30 上での性能のみを示したが，MGTP は容易に並列化可能であり，queen 問題や pegeon hole 問題など，分岐数が大で均衡した証明木が得られる非ホーン問題に対しては PIM/m-256 PE 上でほぼ線形の台数効果が得られている．

[5]RAMS 版において，PRV009-1 の cpu 時間は PLA001-1 の場合の約 10 倍であるのに対し，インスタンススタックの量は 100 倍以上になっていることから，PRV009-1 ではΔ失敗の頻度が小さいことが推察される

謝辞 本研究に関して，有益な討論および参考データを頂いたICOTの越村三幸氏，計算機上での計測の労をとって頂いた九州ジェーピーエー・山鹿英樹氏に感謝致します．また，本研究の機会とご支援を頂いたICOTの淵一博前研究所長(現東大教授)，内田俊一研究所長に深く感謝致します．

参考文献

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

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

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

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

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

[6] 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).