制約問題を解くためのモデル生成型定理証明系の新実装法

長谷川，隆三
九州大学大学院システム情報科学研究科知能システム学専攻

藤田，博
九州大学大学院システム情報科学研究科知能システム学専攻

https://doi.org/10.15017/1500394

出版情報：九州大学大学院システム情報科学紀要. 4 (1), pp.57-62, 1999-03-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学研究科報告
第4巻 第1号 平成11年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.4, No.1, March 1999

制約問題を解くためのモデル生成型定理証明系の新実装法

長谷川 隆三* ・ 藤 田 博*

A New Implementation Technique for Model Generation Theorem Provers To Solve Constraint Satisfaction Problems

Ryuzo HASEGAWA and Hiroshi FUJITA

(Received December 21, 1998)

Abstract: A new implementation technique for model-generation theorem provers (MGTP) is presented, which is particularly efficient in solving constraint satisfaction problems. We solved the so-called multi-environment problem by using restoration lists for destructively modified data structures. Furthermore, we introduced a new mechanism called Activation-cell which dramatically reduces the complexity of conjunctive matching, subsumption tests, and conflict tests in the proving process of MGTP. Combined with other implementation techniques including term-indexing and clause-indexing, the new MGTP written in Java shows remarkable performance compared to the previous implementations.

Keywords: Automated theorem proving, Constraint satisfaction, Model generation, Java

1. はじめに

我々はモデル生成法に基づく一階述語論理の定理証明システムMGTPを開発している．モデル生成法は一般に値域限定条件を満たす節集合に対して優れた定理証明法であるが，並列処理を行うのに特に有利とされている．実際にMGTPの並列論理型言語での実装と並列計算機上での走行結果により，その有用性が示されている[1]．

その後，MGTPをベースにして制約充足問題を効率良く解くための機能拡張が行われ，CMGTPが開発された．CMGTPでは，入力節の対偶から負情報（制約）を引出し，これを利用して前もって探索空間を狭める（選言縮約と単位反駁）機構を組み入れることにより効率化が図られている[2]．

しかしながら，従来のCMGTPは，並列化を主目的としたMGTPや一部の問題に特化されたMGTPの拡張として実装されており，必ずしも最適かつ汎用のシステムとはいえない．我々は，CMGTPを法的推論等，より広範な応用問題に利用していくことを考えており，汎用計算機上での効率良い実装を必要としている．

CMGTPの効率的実装の最大の妨げとなるのは，選言による多重環境の問題である．(C)MGTPは選言に基づいて場合分け推論を行うが，各場合ごとに保持される情報について，MGTPにおいては場合分け前の共有情報が保たれるのに対し，CMGTPでは書換えが生じて共有情報が保たれない．このようなとき，各場合ごとに共有情報の複製を作るのが安易であるが，メモリと処理時間を著しく消費するのが難点である．実行コストの安い多重環境の実現がまず第一に求められる．

実は，基本MGTP自体についてもまだ改善の余地が残されている．節の前件に対するモデル候補との連言照合や，後件の包摂検査はMGTPの処理時間の大半を占めるが，いずれも構造データの集合に対する所属検査を基本演算としている．安易な実装では，要素数 N の集合を線形リストで実装し，線形探索を行うので，$\mathcal{O}(N)$ の計算量となっていた．適切なインデキシングを用いれば，所属検査は $\mathcal{O}(1)$ で済むはずである．ただし，多重環境下において高速な所属検査を行うためにはまた別の工夫が必要であろう．

我々は，場合分けの逐次実行を前提として，幾つかの新しい実装技法を導入することにより，従来の基本MGTPをも凌ぐ高能率なCMGTPを実現した．実装にはJava言語を用いた．併せて入力節文法も拡張し，問題記述能力の向上を図っている．以上の改善は，CMGTPに限らず一般の知識処理システムの実装にあたっても有用な技術を示唆するものと考える．

2. CMGTP

CMGTPは，入力節中に負リテラルの記述を許し，単位反駁によるモデル候補の早期棄却と，選言縮約によるモデル拡張爆発の抑制を可能としたものである．

たとえば，MGTPの負節

$$p \rightarrow$$

が，CMGTPでは論理的に等価な

$$\rightarrow \neg p$$

でも表現可能である．負リテラル $\neg p$ がモデル拡張候補

平成10年12月21日受付
* 知能システム学専攻

に加えられる時点で，モデル候補 $M \ni p$ は棄却（単位反駁）される．また，モデル拡張候補に選言 $p;q;r$ が含まれるときには，これを $q;r$ に縮約することができる．

2.1 入力節

CMGTPの入力節は次の式で表現される．

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m.$$

ここで，A_i, B_j は単位論理式（アトム，正リテラル）或は否定記号（'¬'）つき単位論理式（負リテラル），'∧'は連言結合子，'→'は含意結合子，'∨'は選言結合子である．上式で $n=0$ を正節，$m=0$ を負節，$m=1$ をホーン節，$m>1$ を非ホーン節という．リテラル L に対する相補リテラルを $\overline{L}$ と表す．A をアトムとするとき，$L=A$ $(L=\neg A)$ に対して $\overline{L}=\neg A$ $(\overline{L}=A)$ である．

2.2 モデル候補拡張／棄却規則

基底リテラルの集合 M をモデル候補とする．与えられた節集合中に節 $A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$ があって，σ を基底置換として，$\forall i.\ A_i\sigma \in M$ のとき，$d = \{B_j\sigma \mid \neg B_j\sigma \notin M\}$ とする．$d=\phi$ なら M を棄却する．$d \neq \phi$ であり，$\forall d_k \in d.\ d_k \notin M$ なら M を $M_k = M \cup \{d_k\}$ に（場合分け）拡張する．

2.3 CMGTP 証明手続き

上のモデル候補拡張／棄却規則に基づきつつ，ホーン節によるモデル候補拡張を優先するようなCMGTP証明手続きの基本形を **Fig.1** に示す．

M をモデル候補，U をリテラルの集合，D を選言の集合とするとき，$\mathbf{M} = \langle M, U, D\rangle$ なる三つ組を広義モデル候補と呼ぶ．U をユニットバッファ，D を選言バッファと呼ぶ．広義モデル候補の初期値を $\mathbf{M_0} = \langle \phi, U_0, D_0\rangle$ とする．U_0 はホーン正節の後件，D_0 は非ホーン正節の後件から得られる．$\mathcal{M}$ を広義モデル候補の集合とし，その初期値を $\{\mathbf{M_0}\}$ とする．

以下，$\mathcal{M}$ の要素 $\mathbf{M} = \langle M, U, D\rangle$ のそれぞれについて，モデル候補拡張／棄却規則の適用を試みる．

- M を $u \in U$ により拡張する．$\overline{u} \in M$ の場合には単位反駁が成立する．
- 基底置換 σ のもとに，連言照合 $\forall i.\ A_i\sigma \in M$ が成立する節があるとき，その後件 $B_j\sigma (1 \le j \le m)$ を得る（CJM(u,M)）．$m=1$（ホーン節）のときは U を，$m>1$（非ホーン節）のときは D を拡張する．$m=0$ のときは負節による棄却が成立する．
- 選言バッファ D 内の選言 $D \ni d = \{d_1, \ldots, d_k\}$ に対し，$\exists d_j \in d.\ d_j \in M$ の場合には d 自体を D から除く（包摂検査）．$\overline{d_j} \in M$ なる d_j はこれを d から除く（選言縮約）．選言縮約の結果，$d=\phi$ となる場合には，選言縮約による棄却が成立する．

```
M0 = ⟨φ, U0, D0⟩; M = {M0};
  /* U0: Unit buffer (ホーン節後件) */
  /* D0: Disj. buffer (非ホーン節後件) */
L1: for each M = ⟨M, U, D⟩ ∈ M do {
  while (U ≠ φ) { U = U \ {u};
    if (u ∉ M) {
      M = M ∪ {u}; ⟨U', D'⟩ = CJM(u, M);
      U = U ∪ U'; D = D ∪ D';
      D = 選言縮約と包摂検査(D, M);
      when ( モデル候補棄却条件の成立 ) {
        M = M \ M; continue L1; } } }
  if (D = φ) {
    output SAT(M); continue L1; }
  else { d = (d1 ∨ ... ∨ dm) ∈ D;
    Mk = ⟨M, U ∪ {dk}, D \ {d}⟩;
    M = (M \ M) ∪ {Mk}; } }
if (M = φ) output UNSAT;
```

Fig.1　CMGTP proof procedure

- 上で単位反駁，負節による棄却，選言縮約による棄却のいずれかが成立したときには，ただちに当該広義モデル候補 $\mathbf{M}$ を棄却し，$\mathcal{M}$ から除く．
- ユニットバッファ $U \neq \phi$ の間，上の手続きを繰り返して $U=\phi$ に至ったとき，選言バッファ $D=\phi$ の場合，M がモデルとして確定し，与えられた節集合は充足可能（SAT）と判定される．
- $D \neq \phi$ の場合，$\mathbf{M}$ を $D \ni d = \{d_k\}$ によって，$\mathbf{M}_k = \langle M, U \cup \{d_k\}, D \setminus \{d\}\rangle$ のように場合分け拡張し，$\mathcal{M}$ 内で置き換える．

上のモデル候補拡張／棄却規則が適用できなくなった時点で，$\mathcal{M}=\phi$ ならモデルは無く，与えられた節集合は充足不能である．

3. CMGTP の実装

新版CMGTPは，Java-MGTP[3]（旧版）をベースにしており，項，リテラルの内部表現や，クラス階層については概ねこれを踏襲している．本稿では，新版で新たに開発された実装法に焦点を絞って述べる．

3.1 Activation-cell

一般に集合 S に対する要素 e の所属検査 $e \in S$ を高速に行うには，例えばハッシュ法を利用すればよい．(C)MGTPでは場合分けの結果，複数のモデル候補 M_i に対する同一リテラル L の所属検査 $L \in M_i$ を行う必要があるので，ハッシュ法だけでは不十分である．

モデル生成法に基づいて構成されるモデル候補は，**Fig.2** の(a)に示すように非ホーン節による場合分けに対応する分岐点を有し，ホーン節拡張によるアトム集合でラベルづけされる枝から成る木構造を形成する．これをMG木という．枝上の黒丸はリテラルを表している．MG

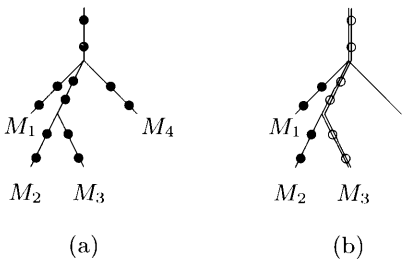


Fig.2 MG-tree of model candidates

木の根から一つの葉に至る枝上の黒丸の列が，一つのモデル候補に対応する．

こうして，複数のモデル候補が同一リテラルを共有する状況がある．逐次実行のもとでは **Fig.2** の(b)に示すように二重線で表された枝に対応する一つのモデル候補のみが対象 (active) となり，白丸で表されたリテラルの所属検査が問題とされる．

現在どのモデル候補が active なのかを明示するために，MG木の分岐点（或は根）から次の分岐点（或は葉）までの枝 S_i のそれぞれに boolean 変数 A_i を割り当てる．この変数を Activation-cell （以下，*A*-セル）という．active なモデル候補に含まれる枝の *A*-セルの値は *true* であり，その状態を A°_S と表す．過去 active であったモデル候補に含まれる枝の *A*-セルの値は *false* であり，その状態を $A^{\bullet}_S$ と表す．各モデル候補 M には，それが含む枝のすべての*A*-セルからなるリスト AL_M を割り当てる．モデル候補 M がMG木の根から順に枝 $S_1,\ldots,S_n$ を含むとき，M の *A*-セルリスト AL_M を

$$AL_M = [A_n,\ldots,A_1]$$

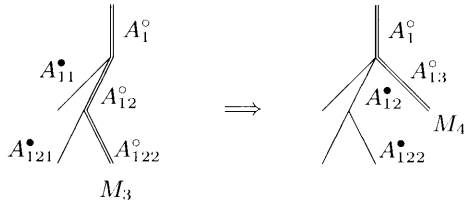
とする．モデル候補 M が active のとき，AL_M を active な *A*-セルリストという．単に“*A*-セルリストの値”というとき，その先頭の *A*-セルの値のことを指すものとする．

アトム A には，*pac* 属性と *nac* 属性を割り当てる．今，active なモデル候補を M，その *A*-セルリストを AL_M とするとき，リテラル $L = A$ 或は $L = \neg A$ の M への登録に際し，$L = A \in M$ なら $A.pac = AL_M$，$L = \neg A \in M$ なら $A.nac = AL_M$ とする．

リテラル L は，MG木において M を構成する枝の中で最も葉に近い枝上に登録されるので，リテラル L の現在のモデル候補への所属は，$A.pac(resp.A.nac)$ の参照する *A*-セルリスト AL_M の先頭の *A*-セルの値によって判定される．

MG木の構成時の模様を**Fig.3**に示す．active なモデル候補が M_3 から M_4 に移るとき，active な *A*-セルリストは AL_{M_3} から AL_{M_4} に代わる．同時に，*A*-セル $A^{\bullet}_{12}, A^{\bullet}_{122}$ が inactive となる．

ここで，ただ二つの *A*-セルの状態変更だけで，枝 S_{12}, S_{122} に属する複数のリテラルを一斉にinactive に変



$AL_{M_3} = [A^{\circ}_{122}, A^{\circ}_{12}, A^{\circ}_1]$ $\quad AL_{M_4} = [A^{\circ}_{13}, A^{\circ}_1]$

Fig.3 Activation-cells for model candidates

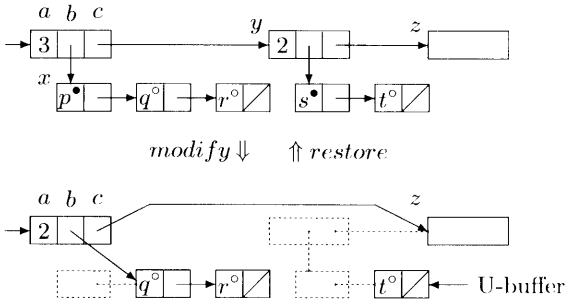


Fig.4 Modifying/Restoring Disj. buffer

更できることに注意．なお，*A*-セルのこのような効能は，後述する拡張MGTP文法を扱うパーザの内部処理においても有効に利用されている．

3.2 選言バッファの書換えと復旧

選言バッファは，選言縮約により変更を受けるので，場合分けが生じるときには，これを多重環境として扱われなければならない．多重環境は複製するのが通常であるが，それには大きなオーバヘッドが伴う．しかしながら，選言バッファの場合，逐次処理を前提とすれば複製せずに破壊的に変更しても復旧が容易であり，オーバヘッドは抑えられる．

選言縮約に伴う選言バッファの書換えと復旧の様子を**Fig.4**に例示する．ここで，選言バッファはリストを用いて実現されている．$p;q;r$ が $q;r$に，$s;t$ が t に縮約されると，選言バッファの a,b,c 属性が変更され，リテラル t は U バッファから新たに参照される．この書換えを復旧するために，復旧リストに$[a \hookrightarrow 3, b \hookrightarrow x, c \hookrightarrow y]$ が追加される．書換えにより切り離されたセル（破線で表示）は復旧リストから参照されているので，廃棄されることなく存続する．後に，復旧リスト上の情報に基づきa,b,c セルの内容を再設定すれば，もとの構造が復旧できる．

3.3 項インデキシング

入力節中の無引数リテラル（命題）或は有引数リテラルで基底のものの多重出現については，読み込み時に同一リテラルごとに固有の内部表現を生成し，これを一意的に参照することが可能である．

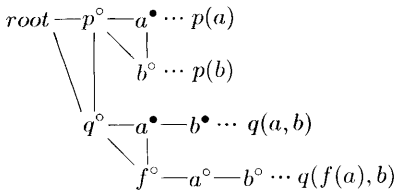


Fig.5　Discrimination tree for term memory

非基底（変数を含む）のリテラルについては，異なる基底代入ごとに固有な内部表現の一意的な生成／参照を行う機構が求められる．すなわち，基底項の一意性を満たす適切な項インデキシングが必要である．

ここでは，**Fig.5**に示すような弁別木に基づく項インデキシングを採用する．*root*ノードを除く弁別木のノードは，述語記号，関数記号，定数記号でラベルづけされる．枝はすべて双方向リンクで実装される．葉ノードには対応する基底項の内部表現の実体への参照が置かれる．

入力節後件の非基底リテラル $p(X)$ が，連言照合の結果X/a なる基底代入によってモデル拡張候補として登録されるとき，弁別木を走査して $p(a)$ の実体が得られる場合には，その参照を用いる．こうして，同一項の複製が排除される．

ノードラベルには，図中の黒白丸で示すように現在 active なモデル候補の *A*-セルリストへの参照が置かれる．図の弁別木は，現在のモデル候補が $p(b), q(f(a),b)$ を含み，$p(a), q(a,b)$ は含まないことを表している．

3.3.1　相補リテラル

相補リテラル $p(a), \neg p(a)$ に共通なアトム部分 $p(a)$ を弁別木上で同一化する．このため，弁別木の葉ノードは**Fig.6**に示すようなデータ構造を有する．

図はこれまでに負リテラル $\neg p(a)$ の出現があり，正リテラル $p(a)$ の出現は無い状況を表している．負リテラル $\neg p(a)$ 自体に固有の内部表現を確保するとともに，その多重出現がある場合の同一性を保証するため，*GNLiteral* なるセルを生成した上でアトム $p(a)$ を間接参照させることにする．この後，$p(a)$ が出現したときには，*pl* セルに共通アトム $p(a)$ の実体（*GCTermF*）への参照が置かれる．

アトム $p(a)$ に関して，正負いずれのリテラルとして activate されているかの判定は，*GCTermF* としての $p(a)$ の実体に付随する *pac*, *nac* セルの内容に従う．*pac* (resp. *nac*) セルが *A*-セルリスト *AL* を参照しており，その（先頭の *A*-セルの）値が *true* である場合に正 (resp. 負) リテラルとして active であると分かる．こうして共通アトムが内部表現で同一化された上，そこに正負リテラルの出現情報が集約されているので，単位反駁等で必要となるリテラル $p(a), \neg p(a)$ の相補性判定は瞬時に行える．

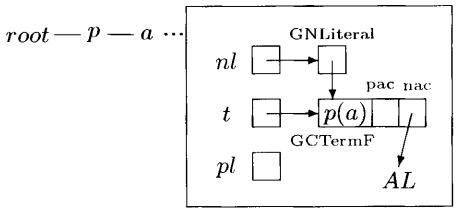


Fig.6　Complementary literals in term memory

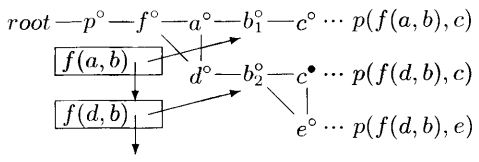


Fig.7　Skip-list for compound terms

3.3.2　Skip-list

弁別木は連言照合の際にも直接的に利用される．すなわち，非基底の前件リテラル $p(X)$ と照合可能な基底リテラルの探索は，弁別木を（active ノードを選択的に）走査して行うことにする．変数 X の可能な基底代入 σ を決定し，前件全体の連言照合が成立したときには，後件を σ で基底化して *U*-, *D*-バッファに投入する．複数の基底代入が得られる場合には，そのすべてについて処理する．

Fig.7には，弁別木上での連言照合を効率良く行うためのデータ構造 Skip-list を示す．図上，前件リテラル $p(X,Y)$ のもとに照合が開始されたとする．p の照合成立の後，変数 X に対する基底代入には $f(a,b)$ と $f(d,b)$ の可能性がある．これは，関数記号 f に付随する Skip-list によって知ることができる．まず，$\sigma = \{X/f(a,b)\}$ としたときの照合の続行点はb_1° ノードであり，引続き $\{Y/c\}$ で照合が成立することが分かる．次に，$\sigma = \{X/f(d,b)\}$ とすれば，b_2° ノードを続行点として $\{Y/e\}$ で照合が成立することが分かる．

Skip-list の生成／追加は，弁別木上で新たな基底項の登録が行われる際に（関数記号ノードにおいてのみ）行われる．

3.4　そのほかの実装技法

3.4.1　節インデキシング

連言照合における節の入口リテラル L_e とモデル候補拡張リテラル（後件リテラルが基底化されたもの）との照合に関して，その組合せを節の読み込み時に解析により限定しておくことができる．すなわち，後件リテラル L_C 毎（本実装では，述語記号 p 毎）に照合可能な入口リテラル L_{L_C} (resp. L_p) を調べ，そのリストを作成して L_C (resp. p) の属性として保持させておけばよい．

3.4.2 命題論理の高速化

例えば，命題的な節

$p, q, r \rightarrow C$

に対し，同じモデル候補に対する連言照合において，リテラル p, q, r につき各1回ずつ，前件全体の成立には計3回（従来の実装では最悪8回！）の照合で済まされるべきである．

非基底リテラルが混在する節

$p(X), r, q(X,Y), s(a), t \rightarrow C$

については，

$(r, s(a), t), (p(X), q(X,Y)) \rightarrow C$

のように命題部分と述語部分にグルーピングし，命題部分に対する冗長な照合を回避するようにしたい．

そのため，

$r, s(a), t \rightarrow C_1$

$C_1 : p(X), q(X,Y) \rightarrow C$

のように（内部的に）節の分離を行う．ここで，C_1 は分離された節間を（計算的に）接続するためのリンクの意味しかなく，通常のリテラルとは区別される．(それゆえ，2本目の節の C_1 : は節を特定する指示子としての役割を明示している．)

3.4.3 Javaコーディング

Java-MGTP[3]において考察されたJava言語に特有のコーディング技法に関し，本実装ではさらに徹底化を試みた．例えば，次節で述べる文法拡張後は，後件のパタンとしてさらに多くの組合せを生じてしまうが，基本的には新たなパタン毎にクラス拡張を行う．if 文と instanceof による条件分岐処理よりも，メソッド呼出しの際に行われる該当クラスの弁別の機構を利用する方が実行効率で優るという判断である．

4. CMGTP 入力節の拡張構文

応用上の表現力強化及び，実行効率上の便宜のため，CMGTP入力節の構文を拡張した．

4.1 後件における連言

節後件として次のような連言が記述できる．

$Ante \rightarrow L_1, \ldots, L_k$

これは，

$(Ante \rightarrow L_1) \wedge \ldots \wedge (Ante \rightarrow L_k)$

と等価である．

一般に，次のように選言中の連言の記述を許す．

$Ante \rightarrow L_{11}, \ldots L_{1k_1}; \ldots; L_{m1}, \ldots, L_{mk_m}$

この構文により，“失敗による否定”

$A, not\ L \rightarrow B$

のMGTPにおける表現

$A \rightarrow \neg KL, B; KL$

が可能となる．

4.2 入口リテラル指定

MGTPでは非冗長な連言照合方式としてMERC法[1]を用いている．MERC法でモデル候補拡張リテラル Δ との照合を行う前件リテラルを入口リテラルという．

前件の入口リテラル L_e を明示的に指定する構文，

$L_e \mid L_2, \ldots, L_n \rightarrow \cdots$

を導入する．L_e は Δ，$L_2, \ldots, L_n$ には $\forall A \in M \cup \{\Delta\}$ が照合される．

例えば，MERC法によれば，

$p(X,Y), p(Y,Z) \rightarrow p(X,Z)$

は内部的に

$p(X,Y) \mid p(Y,Z) \rightarrow p(X,Z)$

$p(Y,Z) \mid p(X,Y) \rightarrow p(X,Z)$

に変換されるが，拡張文法ではシステムの自動変換を介さず，ユーザが直接後者の形を記述できる．

4.3 多重節

（変数名の書換えにより）同一の入口リテラルを持つ節をまとめて記述することができる．これにより，入口リテラルの計算を減らす効果がある．

先の二本の入口リテラル節の例は，等価な

$p(X,Y) \mid [\ p(Y,Z) \rightarrow p(X,Z)$

$\quad \& \ p(W,X) \rightarrow p(W,Y)\ ]$

なる多重節にまとめることができる．

4.4 選択節

多重節形式の特殊な形式として選択節形式を導入する．

例えば，

$p(X,Y) \mid p(Y,Z) \rightarrow p(X,Z)$

$p(X,Y) \mid \neg p(Y,Z) \rightarrow q(X,Z)$

は，選択節構文，

$p(X,Y) \mid (\ p(Y,Z)\ ? \rightarrow p(X,Z)\ : \rightarrow q(X,Z)\ )$

として表現可能である．

上で $p(Y,Z)$ を基軸リテラルという．相補リテラルについてそれぞれ検索を要していたところが，基軸リテラルの検索は1回で済む．

4.5 埋込みコード

整数に関する演算等をMGTPの論理の外部で行うためのコードを，MG節中に埋め込むことができる．

例えば，Fibonacci数列に関し，

$\rightarrow fib(0,1), fib(1,1)$

$fib(N1,F1) \mid \{N = N1 - 1\}, fib(N,F),$

$\quad \{N2 = N + 2, F2 = F + F1\} \rightarrow fib(N2,F2)$

$fib(10,F) \rightarrow$

のようにプログラム的な表現がMGTP上で可能となる．

以上の拡張文法に基づいて，準群問題 QG5(R) をコンパクトに（かつ実行効率良く）記述した例を付録に示す．

Table 1 Results on TPTP problems

Problem	MGTP		Java-CMGTP	
	KLIC	Java	Old	New
SYN009	1.501	0.890	12.688	0.453
SYN015-2	2.228	2.368	2.420	0.433
SYN036-3	2.887	3.720	3.856	0.746
PUZ025-1	0.178	0.138	0.169	0.078
PUZ030-2	0.065	0.076	0.136	0.025

(Time: sec)

Table 2 Results on CS problems

Problem	branches	KLIC	Old	New
8-queens	384	4.301	8.276	2.130
QG5-8	10	17.124	5.141	0.369
QG5-9	15	54.807	10.750	0.754
Channel	52000	96.589	162.436	28.778

(Time: sec)

5. 性 能 評 価

Javaによる新版CMGTPの性能を，いくつかの旧版(C)MGTPと比較した．表中のCPU時間は，すべてSUN Ultra1 Model 170E 上での計測値である．

KLIC-MGTP, Java-MGTP, Java-CMGTPの旧／新版の性能比較を**Table 1** に示す．問題は定理証明のベンチマーク集TPTP[4]から選んだもので，CMGTPの制約機能を用いずに解かれるものである．新版は制約処理の拡張機能を有しながら，従来の(pure-)MGTPに比べてすら高速化されている．これは，*A*-セルという新たな実装技法の導入により，連言照合，包摂検査等，MGTPの基本演算部分において顕著な改善が得られたことを示している．

CMGTPの制約処理が有効に機能するCS(Constraint Satisfaction) 問題については，KLIC-CMGTP とJava-CMGTPの旧／新版を比較した結果を**Table 2** に示す．ここでは，MGTPにおいては障碍とならなかった環境複製問題への対処の差が現れて来る．KLIC-CMGTPでは，リスト構造の高速複製がシステムによりサポートされているので，オーバヘッドは比較的小さい．Java-CMGTP旧版においては，複製(clone)をサポートするユーティリティのjava.util.Vectorクラスを利用し，選言バッファを3重のVector（拡張文法に従い，連言の選言の集合）で実装しているため，大きなオーバヘッドを生じている．新版は，旧版に対し1桁以上のスピードアップを達成している．環境の書換え／復旧方式が複製方式に比べて優位であることを顕著に示している．

6. お わ り に

JavaによるCMGTPの新実装において，従来のCMGTP処理系に比べて大きな性能改善をもたらすこととなった実装技法上の特徴は以下のとおりである．

- 複製によらない多重環境の実現
- Activation-cellによる$\mathcal{O}(1)$時間所属検査
- 多重環境下での効果的な項インデキシング機構

Activation-cellはProlog等の論理型プログラムにおける論理変数に類似している．論理変数の場合，これを駆使した差分リストやショートサーキット法などの“巧い”プログラミング技法が編み出されているが，Activation-cellの場合も，一般に共有部分を有する複数の集合に関する処理に関して，効率的なプログラミング技法として広く応用が期待される．

参 考 文 献

1) 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,” 情報処理学会論文誌, Vol. 37, No. 1, (1996) 1–12.
2) 白井 康之, 長谷川 隆三, “モデル生成型定理証明システムによる制約充足問題の解決とその並列化,” 電子情報通信学会論文誌, Vol. J80-D-II, No. 1, (1997) 224–236.
3) 藤田 博, 長谷川 隆三, “Java言語によるモデル生成型定理証明系MGTPの実装,” 九州大学大学院システム情報科学研究科報告, Vol. 3, No. 1, (1998) 63–68.
4) G. Sutcliffe, C. Suttner and T. Yemenis, “The TPTP Problem Library,” *Proc. CADE-12,* (1994) 252 266.

付) 準群問題 QG5-8 のCMGTP入力節（一部省略）：

```
->r(1),...,r(8).
->p(1,1,1),...,p(8,8,8).
r(M)|
  [r(N),[M!=N,[->p(M,N,1);...;p(M,N,8)&
              ->p(N,M,1);...;p(N,M,8)&
              ->p(M,1,N);...;p(M,8,N)&
              ->p(N,1,M);...;p(N,8,M)&
              ->p(1,M,N);...;p(8,M,N)&
              ->p(1,N,M);...;p(8,N,M)]&
        {N<M-1}-> -p(N,8,M)&
        {M<N-1}-> -p(M,8,N)]].
p(A,B,C)|
  [r(M),[M!=A-> -p(M,B,C)&
        M!=B-> -p(A,M,C)&
        M!=C-> -p(A,B,M)]&
   (p(C,A,M)? ->p(M,A,B): -> -p(M,A,B))&
   (p(M,A,B)? ->p(C,A,M): -> -p(C,A,M))&
   (p(B,M,A)? ->p(C,B,M): -> -p(C,B,M))&
   (p(C,B,M)? ->p(B,M,A): -> -p(B,M,A))&
   (p(B,C,M)? ->p(M,B,A): -> -p(M,B,A))&
   (p(M,B,A)? ->p(B,C,M): -> -p(B,C,M))].
-p(A,B,C)|
  [p(B,M,A)-> -p(C,B,M)& p(C,B,M)-> -p(B,M,A)&
   p(B,C,M)-> -p(M,B,A)& p(M,B,A)-> -p(B,C,M)&
   p(C,A,M)-> -p(M,A,B)& p(M,A,B)-> -p(C,A,M)].
```