Javaによるモデル生成型定理証明系MGTPの開発

長谷川 隆三[†] 藤田 博[†]

モデル生成法に基づく定理証明システム MGTP を Java で実装した．MGTP では単一化が単方向に限られることを利用して，項や変数の効率的な実装が可能である．加えて，Java 版ではデータ構造の破壊的書換えが可能となり，論理型言語による従来の実装に比べ，動的記憶域の使用を抑制できる等のメリットが得られた．さらに本実装では，連言照合，包摂検査，単位反駁，選言縮約等の MGTP の主要演算を効率良く行うため，これらの演算をモデル候補サイズに対し $\mathcal{O}(1)$ で実行できる Active セルと呼ばれる機構を新たに導入した．これにより，命題論理の問題が著しく高速化された．同時に，項インデキシングの機構に工夫を施して非命題論理の高速化も図られている．

Development of a Model Generation Theorem Prover MGTP in Java

Ryuzo Hasegawa[†] and Hiroshi Fujita[†]

A model-generation theorem prover MGTP is implemented in Java. The fact that one-way unification suffices in MGTP enables very efficient implementation for terms and variables. In addition, since destructive assignment is allowed in Java, the new MGTP has an advantage that dynamic memory allocation can be suppressed compared to the previous implementations which are based on logic languages. Furthermore, we introduced a new technique called Active-cell to make it possible to perform conjunctive matching, subsumption tests, unit refutation, and disjunction simplification, in $\mathcal{O}(1)$ with respect to the size of a model candidate. This improves the performance remarkably for propositional cases. We also elaborated on the term-indexing mechanism to speedup predicate cases.

1. はじめに

定理証明の基本方式は，レゾルーションとタブローに大別される．前者は機械化に適しており，優秀なシステムがすでに数多く実装されている．一方，タブローのシステムも近年再び注目されるようになってきた．人間にとって分かりやすい形で論理を展開できるというタブローの特徴が，見直され始めたものと思われる．しかし，その最大の要因はやはり計算機性能の向上と，論理プログラミング分野の進展に寄与した実装技術の蓄積であろう．その典型例が，Manthey らの SATCHMO[1] である．これは，Prolog できわめて簡潔に記述されたシステムで，モデル生成法と呼ばれる一種のタブローに基づいている．この方式は，値域限定性という条件が課せられるが，非ホーン節を含む一階述語論理の問題一般を扱える．

我々は，モデル生成法が並列実装に特に適した方式であると考え，第 5 世代コンピュータプロジェクトの一環として並列定理証明システム MGTP[2] を開発し，有限群論の未解決問題の証明に成功する等，その有用性を示してきた．MGTP には，並列化とは別に，SATCHMO に含まれていた冗長な計算を除去する等，方式上の多くの改良[3] が含まれている☆．そこでは，モデル生成法においては単一化が単方向で十分であるという事実が最大限に活かされている．さらに，我々は制約充足問題を効率良く解くために機能拡張を行った[5]．これは，入力節に対して論理的に等価な対偶節から負リテラルを導けるようにし，単位反駁と選言縮約の推論を新たに組み入れることにより探索空間を狭め，証明の効率化を図ったものである．

これまで，MGTP の開発は論理型言語ベースで行われてきたが，今回，より広範な利用を目指して，プラットホーム独立性の高い Java を用いた実装を行った[6),7]．Java 利用のメリットは他にもいくつかあげられる．その 1 つは，論理型言語と同様に，動的メモリ

† 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

☆ その後，SATCHMO にも同様の改良[4] が一部加えられた．

の割当てとゴミ集めが自動化され，システム開発が容易になっている点である．しかし，我々にとってより実際的なメリットとなったのは，変数への単一代入という足枷がはずされ，データ構造の破壊的書換えが可能になったことである．これにより，場合分けにともなう多重環境の実現の際，複製が不要となり，動的メモリの節約が図られ，高速化につながった．

MGTP☆の主要な演算は，連言照合，包摂検査，単位反駁，選言縮約である．これらは，命題論理に限定すれば，基底項の集合に対して，ある基底項の所属を問う演算に帰着できる．これを高速に実行するには，ハッシュ法等の適用が考えられる．しかし，述語論理では基底項の所属検査のみにとどまらず，非基底複合項に対する単一化操作が必要となるので，単純にハッシュ法は適用できない．そこで今回，我々は新たに *Active-cell*（以下，A セル）と称する機構を考案した．これにより，基底項の所属検査はモデル候補のサイズに対し $\mathcal{O}(1)$ で実行可能となった．さらに，非基底複合項に対する連言照合や包摂検査においても，A セルを利用した項インデキシングの機構を導入することにより，高速な単一化および検索が可能となった．

2. MGTP

MGTP は，$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$ なる含意形式の節の集合に対し，**図 1** のような手続きでモデルの構成を試みる．ここで，A_i，B_j はリテラル，$\rightarrow$ の左辺を前件，右辺を後件という．また，$n = 0$ を正節，$m = 0$ を負節，$m = 1$ をホーン節，$m \geq 2$ を非ホーン節という．

前件に関して，モデル候補 M に対し，ある基礎代入 σ のもとに $\forall i.\ A_i\sigma \in M$ か否かを決定する演算を連言照合という．また，後件に関して，$\forall i.\ B_i\sigma \notin M$ か否かを決定する演算を包摂検査という．

図 1 中，M はモデル候補，U はホーン節後件起源のリテラルの集合，D は非ホーン節後件起源の選言の集合，T は M，U，D の 3 つ組，$\mathcal{M}$ はその 3 つ組の集合である．連言照合関数 $CJM(u, M)$ は，モデル候補 M とモデル拡張リテラル $u \in U$ で充足される前件を持つ非正節のあらゆる具体例を求め，ホーン節の後件の集合と非ホーン節の後件の集合の対を返す．$Simp\&Subsump(D, M)$ では，選言集合 D に対して **図 2** (b) に示す選言縮約と包摂検査を実行する．モデル候補の棄却条件は，1) CJM 関数において負節に対して連言照合が成功する，2) M の拡張（図 1 の★）の際，図 2 (a) の単位反駁が導かれる，3) 図 2 (b) の選言縮約によって D 中のある選言が空となる，の各場合に成立する．U が空で D が空でないとき，D 中の選言に従ってモデル候補の場合分けを行う．

```
M = {T0 = ⟨φ, U0, D0⟩};
/* U0: positive Horn clauses */
/* D0: positive non-Horn clauses */
L1: while (M ≠ φ) {
  M = M \ {T = ⟨M, U, D⟩ ∈ M};
  while (U ≠ φ) {
    U = U \ {u ∈ U};
    if (u ∉ M /* u not subsumed by M */) {
      M = M ∪ {u}; /* ★ */
      ⟨U', D'⟩ = CJM(u, M);
      U = U ∪ U'; D = D ∪ D';
      D = Simp&Subsump(D, M);
      if ( /* M rejected */ ) continue L1; }
  }
  if (D ≠ φ) {
    D = D \ {d = (d1 ∨ ... ∨ dm) ∈ D};
    Tk = ⟨M, U ∪ {dk}, D⟩;
    M = M ∪ {Tk}(1≤k≤m); }
  else output M; /* a model found */
}
if (/* No model found */) output UNSAT;
else output SAT;
```

図 1 MGTP 手続き
Fig. 1 MGTP procedure.

$$\frac{a \in M \quad \neg a \in M}{\bot} \quad \text{(a)}$$

$$\frac{a(\neg a) \in M \quad \neg a(a) \vee \mathcal{C} \in D}{\mathcal{C}} \quad \text{(b)}$$

図 2 (a) 単位反駁と (b) 選言縮約
Fig. 2 (a) Unit refutation and (b) Disj. simplification.

節集合が充足不能のときは，すべてのモデル候補を棄却して有限終了する．すなわち，反駁証明法として完全である．他方，節集合が充足可能な場合，無限個の要素を持つ極小モデルが存在しない限り，すべてのモデルを生成することができる☆☆．

MGTP 手続きに従い構成されるモデル候補集合 $\mathcal{M}$ は，木で表現することができる．これを “MG 木” と呼ぶ．たとえば，**図 3** の節集合 S0 に対する手続き終了後の MG 木は同図右のようになる．MG 木の根から 1 つの葉に至るパス上の節点をラベル付けするリテラルの集合が 1 つのモデル（候補）を表す．⊥ 印の葉

☆ 本論文の MGTP は従来の CMGTP を含んでいる．

☆☆ 任意個のモデルが得られた時点で終了することもできる．

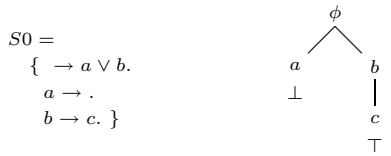


図 3 節集合 S0 とその MG 木
Fig. 3 Clause set S0 and its MG-tree.

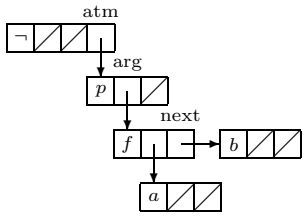


図 4 リテラル $\neg p(f(a), b)$ の内部表現
Fig. 4 Internal structure for a literal $\neg p(f(a), b)$.

は対応するモデル候補が棄却されたことを，⊤ 印の葉は対応するモデル候補がモデルであることを示す．

3. Java-MGTP の基本実装

Java 版の MGTP の特徴は以下のようにまとめられる．

- 項の内部表現に関する，独特な Java クラス階層の定義
- 変数束縛の方法について，単方向単一化の最大限の活用
- 入力節の種別に応じた細かなクラス分け
- 特に命題論理対応に高速化を図った A セル機構の導入
- 基底項の唯一性を保証し，単一化を効率良くサポートする項インデキシング機構の導入

各特徴について，本章ならびに 4，5 章で詳述する．

3.1 項の内部表現

項は最も基本的なデータ構造なので，できる限り “軽い” 実装が望ましいが，ここでは，Java のオブジェクトとして実装する．まず，基本項クラス `Term` を

```
abstract class Term
  { int id; Term arg,next; }
```

のように定義する．ここで，`id` は，述語/関数/定数記号に対して処理系内部で定める識別子である．`arg` には当該項が複合項（または述語）の場合にその最初の引数への参照を置く．`next` には当該項がその外側の項の i 番目の引数である場合に同 $i+1$ 番目の引数への参照を置く．ここで一見奇妙なのは，定数項，変数項に対しても `arg,next` フィールドつきの `Term` を継承させる点である．より自然な考え方によれば，これらのフィールドは，それらが初めて必要となるサブクラスにおいて導入されるべきであろう．ところが，そうすると，`Term` クラスの変数に対し，所望のフィールドを持つ拡張項クラスへの実行時キャストが必要になる場合が頻繁に生じる．我々は，速度を重視して実行時キャストはできるだけ避けたいと考えた．

次に，基底正リテラルのために抽象クラス `GLit` を

```
abstract class GLit extends Term {}
```

のように定義する．さらに，基底負リテラルのために

```
class GNLit extends GLit
  { GLit atm; }
```

を定義する．`atm` には `GLit` オブジェクトへの参照が置かれる．

たとえば，2 引数述語の基底負リテラル $\neg p(f(a), b)$ は**図 4** に示すようになる．

3.2 変　　数

変数については，値の束縛と管理方式がポイントである．前件中の変数 X の最左出現（$?X$ と表示）は，連言照合開始時点でつねに未束縛である．$?X$ がある基底項 t に束縛されると同時に，同じ節内の変数 X の右側の出現（$!X$ と表示）のすべてが t に束縛されなければならない．

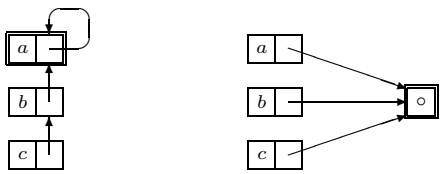
ここで，$?X$，$!X$ をそれぞれ

```
class FVar extends Term { ... }
class BVar extends Term { ... }
```

のように定義する．`BVar` オブジェクト $!X$ の `arg` フィールドには，節の入力時に，`FVar` オブジェクト $?X$ への参照を置いておく．$?X$ の `arg` フィールドには，束縛値が決まるたびにその値への参照を置く．したがって，同じ $?X$ に対して異なる束縛値がある場合，$?X$ の `arg` フィールドに対し破壊的書換えが行われる．一方，$!X$ の値の参照はつねに既定の $?X$ への参照を経て行われる．この方法は，より一般的な方法，たとえば連想リスト等を用いて変数束縛情報を変数と別の実体として管理する方式に比べ，実行時に余計なメモリを必要としない点で優れている．

3.3 クラス分け

MGTP の入力節は，含まれるリテラルの符合に従い正/負/混合に 3 分類され，これに沿って推論における基本的な役割が区別される．さらに，リテラルの基底/非基底，後件のホーン/非ホーン等の属性によって細かく分類することができ，それら小分類ごとに特化された操作が，Java によればメソッドのオーバーライディングによって効果的に実現される．



(a) 直接表現 (b) 間接表現

図 5 文脈 $\{a, b, c\}$ を代表する A セル
Fig. 5 A-cell representing a context $\{a, b, c\}$.

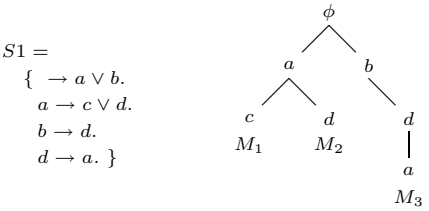


図 6 節集合 S1 とその MG 木
Fig. 6 Clause set S1 and its MG-tree.

4. 基底問題の高速化

MGTP の連言照合，包摂検査，単位反駁，選言縮約のいずれも，その基本はモデル候補 M に対するリテラル e の所属関係 $e \in M$ の検査にほかならず，これを効率良く行うことが高速化の鍵となる．

MGTP 手続きにおいて保持される情報（節，選言集合等）の状態や，それらに対する各処理は，MG 木の “文脈” に依存する．MG 木の文脈とは，MG 木を構成する各分枝 B_i（根〜分岐点，分岐点〜次の分岐点，分岐点〜葉）のことである．文脈 B_i は，モデル候補の拡張（図 1 の★）に用いられたリテラル（u）の集合でラベル付けされる．

現モデル候補が M のとき，MG 木の文脈 B_i のいずれが選ばれているかを判別するのに，**図 5** のような方式が考えられる．図 5 (a) で，文脈 B_{abc} の実体である a，b，c のリストを直接用い，先頭の a をその代表として表現している．図 5 (b) では，B_{abc} のために個別のセルを用意し，これを構成員 a，b，c が参照している．この文脈ごとに用意されたセルを A セルという．

前者では，B_{abc} の各構成員は，リンクが空（null）でないことをもって B_{abc} が M のもとに選ばれていることを知る．したがって，現モデル候補が M' に変わり，文脈 B_{abc} が選ばれなくなったときには，a，b，c すべてのリンクを null に書き換える必要がある．もし構成員 a，b，c 以外に文脈 B_{abc} を参照するものがある場合，この文脈の状態に依存するものすべてにおいて null への書換えが必要である．

これに対し，後者の場合 A セルには boolean 変数を 1 つ用意し，その値が true なら B_{abc} が M で選ばれているものとする．boolean 変数の値はつねに A セルを介した間接参照でしか得られないが，B_{abc} が選ばれなくなったときの処理は，A セル内の boolean 変数に false を代入するだけでよい．もし構成員 a，b，c 以外に文脈 B_{abc} を参照するものがあっても，それらに対する上記のような書換えは不要である．

4.1 A セルによる M への所属検査

MG 木の分岐点で新たな文脈が生じ，その文脈に対応する A セルが生成される．その文脈が現モデル候補 M に含まれるときは，その A セルが持つ値は true となっており，この状態を “活性化されている” という．また，その文脈が現モデル候補 M に含まれなくなったとき，その A セルが持つ値は false となっており，この状態を “非活性化されている” という．

一方，すべての基底リテラル E には pac 属性と nac 属性を持たせる．すなわち，前出の抽象クラス `GLit` に

```
abstract class GLit extends Term
  { ACell pac,nac; . . .}
```

のようにフィールドを加える．リテラル E ($\neg E$) が MG 木の文脈 B_i 上に含まれるとき，その $pac(nac)$ フィールドに，対応する A セル A_{B_i} への参照を置く．E が MG 木のいずれの文脈にも現れないときは，$pac(nac)$ はデフォルトとして恒偽の A セル A_{false} を参照させる．こうして，リテラル E ($\neg E$) の M に対する所属検査 E ($\neg E$) $\in M$ は，E の $pac(nac)$ フィールドが参照する A セルの値の真偽を見ることによりただちに判定できる．

たとえば，**図 6** において，文脈は M_1 と M_2 の両方で活性な B_a，M_1 の B_c，M_2 の B_d，M_3 の B_{bda} の計 4 つ生じる．a に注目すると，1) 文脈 B_a に対応する A セル A_{B_a} が true の状態で a の pac に代入される，2) M_1 と M_2 の処理終了後，A_{B_a} の値は false となる，3) その後，M_3 の文脈 B_{bda} に対応する A セル $A_{B_{bda}}$ が true の状態で a の pac に代入される，という具合に遷移する．

4.2 相 補 分 割

図 1 において，3 つ組 $T = \langle M, \phi, D\rangle$ に対して $d = (d_1 \vee \ldots \vee d_m) \in D\}$ を選び，各 d_i に対してモデル候補を場合分けし拡張するところを，$T_k = \langle M, U \cup \{d_k, \neg d_{k+1}, \ldots, \neg d_m\}, D\rangle$ $(1 \leq k \leq m)$ としても，問題の充足性判定に関して同じ結果が得られ

る．むしろこれら負リテラルの導入により，単位反駁や選言縮約の適用が行われ，冗長なモデル候補探索が回避されて証明時間が短縮される効果を期待できる．この推論方式は相補分割（*complement splitting*）[4] と呼ばれる．

Java-MGTP において，相補分割の実装はきわめて容易である．すなわち，正（負）リテラル d_k の pac(nac) に現モデル候補の最新文脈 B の A セル A_B をセットすると同時に，後続の選言の正（負）リテラル $d_{k+1}, \ldots, d_m$ に対しては nac(pac) に同じ A_B をセットすればよい．

5. 非基底問題の高速化

一般に連言照合は，MERC 法[3] に基づいて実行される．たとえば，$p(X,Y), p(Y,Z) \rightarrow p(X,Z)$ は，

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

の 2 本の “δ 節” に展開される．ここで，$\mid$ の左を δ リテラル，右を後続リテラルという．連言照合 $CJM(u,M)$ は，両 δ 節においてまずモデル拡張候補 u を δ リテラルに照合させ，それに成功したならば $M \cup \{u\}$ の要素を後続リテラルに照合させる．これにより，$p(X,Y)$ と $p(Y,Z)$ の双方を M の要素に照合させる冗長なケースを回避している．

δ リテラルの照合は，`Term` の `match` メソッドとして記述される．たとえば，一般複合項については，

```
boolean match(Term x) {
  return id==x.id && arg.match(x.arg)
         && next.match(x.next); }
```

のようになる．一方，後続リテラルの照合は，以下に述べる項インデキシングに基づいて行われる．

5.1 項インデキシング

基底複合リテラルの唯一性の実現と連言照合の高速化のために，**図 7** 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．弁別木は，*root* ノードおよび述語/関数/定数記号でラベルづけされたノード，ならびにノード間のリンクからなる．葉ノードには基底リテラルの実体への参照が置かれる．ここで，相補リテラル $q(f(d),e), \neg q(f(d),e)$ は弁別木中で同じパスを共有し，正負の区別は葉ノード $(\oplus, \ominus)$ で行われる．

後件の $p(X,Y)$ が，$\{X/f(a), Y/b\}$ なる基底代入によって U または D に登録されるとき，弁別木を走査して，$p(f(a),b)$ の葉ノードに到達する場合にはすでに存在する該当基底項への参照を用い，そうでない場合には新たに葉ノード（および必要な中間ノードならびにリンク）を生成する．

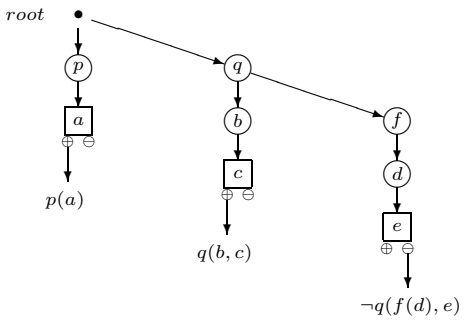


図 7 弁別木による項インデキシング
Fig. 7 Term indexing by a discrimination tree.

異なるモデル候補間で共有されないものも含め，生成されたリテラルをすべて弁別木に保持するか，現モデル候補に関連するリテラルのみを格納するようにするか，の選択がある．前者の場合，リテラル総数がメモリ許容量の範囲内であれば，重複生成の問題や復旧のオーバヘッドがない分，高速性が期待できる．この場合，弁別木の各ノードに A セル参照を置くことにより，現モデル候補に対応する M，U，D に含まれる基底項のみをたどることは容易である．一方，後者の場合，前者に比しメモリ利用効率が飛躍的に改善される．

5.2 弁別木上の連言照合

後続リテラルの連言照合では，たとえば，非基底リテラル $p(?X)$ と弁別木中の基底リテラル $p(f(a))$ や $p(f(b))$ に対する単一化が行われる．このとき，変数 $?X$ に対して基底リテラルの部分項を対応づけ，その部分項に対応するノード列を跳躍する必要が生じる．これは通常，スキップリストを関数ノード f に付随させることにより実現する．スキップリストの役割は，(1) $?X$ に代入するべき部分項（$f(a)$ や $f(b)$）を高速に与える，(2) 部分項に対応する中間ノード列をたどることなく，その終端ノードに直接到達する，の 2 点である．この方法は，結局弁別木に登録された複合項のあらゆる部分項への参照を作るため，膨大な記憶域を消費するのが難点である．そこで我々は，以下で述べるように，分別木中のノードを最大限に利用しつつ，部分項を高速に求めることが可能なスキップチェイン方式を新たに考案した．

5.3 スキップチェインによる変数束縛

項メモリに 3 つのリテラル $p(f(a))$，$p(f(g(b)))$，$p(f(g(c)))$ が登録された状況を**図 8** に示す．関数ノードから出る 2 本の太線のリンクは，その関数ノードの下に登録された，その関数で始まる部分項の範囲を限

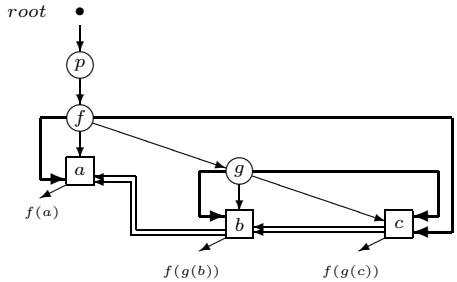


図 8 スキップチェイン
Fig. 8 Skip-chain.

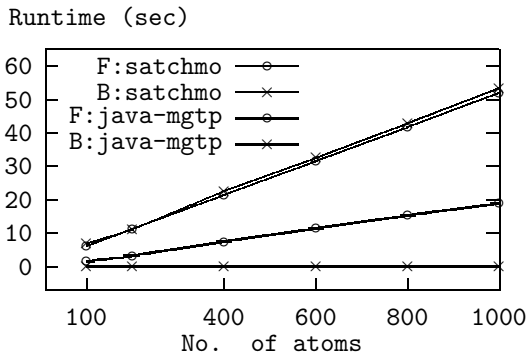


図 9 リテラル検索時間
Fig. 9 Literal retrieval time.

定している．また，その部分項の終端ノード（つねに定数ノード）から左に出て隣の定数ノードを指す二重線のリンクは，同じ関数ノードで始まる部分項をたどるためのものである．この二重線をスキップチェインと呼ぶ．

このスキップチェインにより，f で始まる部分項が $f(a)$，$f(g(b))$，$f(g(c))$ であり，g で始まる部分項が $g(b)$，$g(c)$ であることが分かる．たとえば，図 8 のリテラル集合に対して $p(f(?X))$ なるリテラルを照合させると，$?X$ に対して f のスキップチェインをたどって $g(c)$，$g(b)$，a の 3 つの代入例が得られ，$p(f(g(?Y)))$ なるリテラルの場合は，$?Y$ に対して g のスキップチェインをたどって c，b の 2 つの代入例が得られる．

6. 性能評価

推論システムの主要演算である単一化が，MGTP や SATCHMO のようなモデル生成型では単方向で済むとはいえ，やはり全体の時間性能を支配していることに変わりはない．そこで，SATCHMO と Java 版 MGTP の単一化性能を比較した結果を**図 9** に示す．図中，“F:” は照合パターンが $p(?X, ?Y)$（自由変数）の場合，“B:” は $p(!X, !Y)$（束縛変数）の場合を示している．また，被照合アトムはすべて相異なり，照合時間は 100000 回繰り返した値である．SATCHMO は，assert を用いて登録された Prolog 節に対する頭部単一化を直接用いるため，節数に比例した時間を要する．また，基底/非基底の場合でほとんど差がない．SATCHMO は Prolog 処理系に大きく依存した結果，必ずしも単方向単一化の利点を活かしきっていないと考えられる．

一方，MGTP においては，逐次実行の前提のもとに単方向単一化を最大限に活かす実装方式をとっており，非基底パターンに対する照合では，対象リテラルの総数に比例した時間を要するものの，その単位時間は SATCHMO(ECL^{i}PS^e Prolog) の半分以下である．また，基底パターンに対する照合は，項インデキシング上で行われるため，対象リテラルの総数に関して定数時間（500 nsec/回）で実行される．

Java 版 MGTP と klic による従来版 MGTP，および SATCHMO☆の基本推論性能を比較評価した．Java-MGTP は JDK1.2.1 版（JIT 付き），klic-MGTP は 3.002 版 klic，SATCHMO は ECL^{i}PS^e 3.5.2 版をそれぞれ使用し，すべて SUN UltraSPARC10（333 MHz，128 MB）上で測定した．問題は，定理証明のベンチマーク集 TPTP[8]☆☆v2.1.1 から選んだ．全 3333 問中 433 問は値域限定条件を満たすが，そのうち FLD（体論分野）280 問は “公平性制御” あるいは “非ホーンマジックセット” の導入が必要で，本評価の主旨の範囲を超えるため対象外とした．残り 153 問の内訳は，構文的問題 54 問，群論関係 34 問，パズル関係 27 問，その他となっており，節数 2〜504 本からなる．また，この 153 問中 42 問は充足可能で，これらは全解探索（すべてのモデルを求める）モードで走行させた．なお，証明時間 10 秒以下の問題については，タイマの誤差を軽減するため証明を 10〜1000 回繰り返し，その平均時間をとった．

図 10 は klic-MGTP との証明時間を比較したもので，横軸に Java-MGTP による証明時間の短い順に問題を並べ，縦軸を対数目盛として両システムの証明時間をプロットした．**図 11** は，証明時間の比を示す．ただし，問題 1〜47 は Java-MGTP の証明時間について十分な精度が得られないため，正確な比とはいえない．また，図中歯抜けになっている箇所は，MG 木の分枝数の総数の意味において両者の証明が異なる問題を示しており，比を計算していない．図中，黒く塗り

☆ 節を Prolog プログラムにコンパイルする高速化版[4] を使用．
☆☆ TPTP 問題集は全般的に等号論理を含む数学関係の内容を中心としており，やや偏向して編集されているきらいがある．

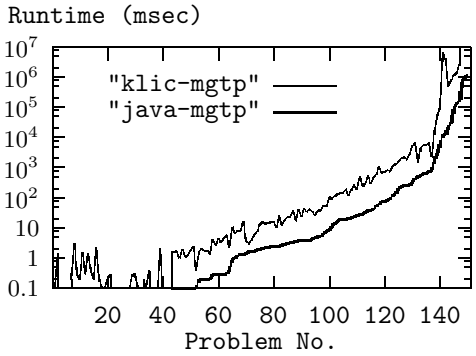


図 10 klic-MGTP との比較（証明時間）
Fig. 10 Comparison with klic-MGTP (time).

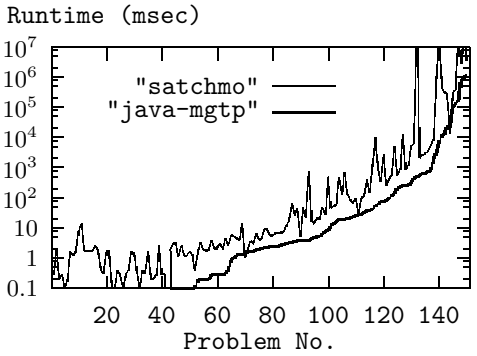


図 12 SATCHMO との比較（証明時間）
Fig. 12 Comparison with SATCHMO (time).

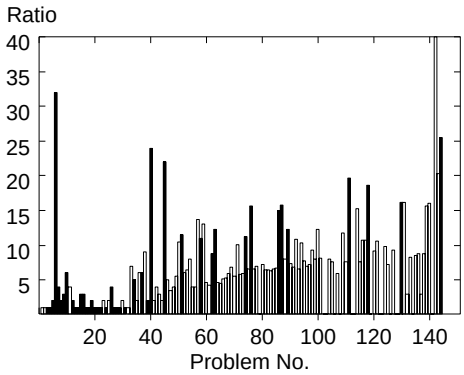


図 11 klic-MGTP との比較（証明時間比）
Fig. 11 Comparison with klic-MGTP (ratio).

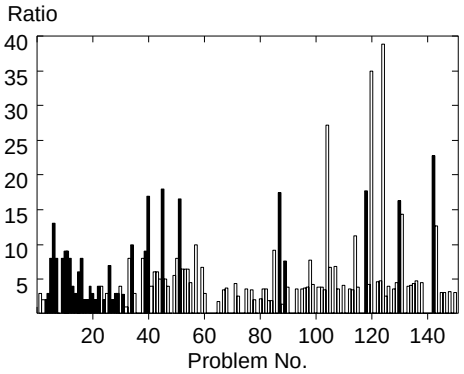


図 13 SATCHMO との比較（証明時間比）
Fig. 13 Comparison with SATCHMO (ratio).

つぶされた棒は，問題が変数を含まず命題論理の範疇に属する問題で，白抜きの棒は変数を含む問題を示す．

命題論理の問題の多くが Java-MGTP で 0.1 ミリ秒以下で解けている．Java-MGTP で 1 ミリ秒以上を要する問題 66 以降においては，klic-MGTP は 12〜25 倍の時間を要している．非命題論理の問題については，5〜15 倍である．また，証明時間のかかる問題ほど実行時間の差が開く傾向が見られる．

図 12 は SATCHMO と Java-MGTP との証明時間を比較したものである．Java-MGTP で解けても SATCHMO ではメモリあふれのため解けなかったり，SATCHMO が Java-MGTP より速く解いた問題が 1 問あったり，問題による変動が klic-MGTP の場合より大きい．これは，一般に Java-MGTP と SATCHMO では証明が異なる☆ためである．具体的には MGTP と SATCHMO で，モデル拡張候補集合 D から選ばれるモデル拡張リテラルの選択順序が異なるのである．

☆ 新 SATCHMO が提供する証明戦略のうち，おおかたの問題に対して平均的に最良となるような戦略を用いた．

ここで，作られた MG 木の枝総数が一致する問題のみに着目した**図 13** を見ると，SATCHMO の場合も，命題論理の問題については実行時間の差異が顕著で，Java-MGTP より 16〜23 倍ほどの時間を要している．非命題論理の問題については，おおむね 3〜5 倍であるが，なかには 27〜38 倍に及ぶものがある．これは，単一化速度の比較において明らかにされたように，自由変数の照合頻度と束縛変数の照合頻度の比率が問題によって大きく異なるためであると考えられる．束縛変数の照合頻度が高いほど SATCHMO は不利であり，Java-MGTP との差が顕著になる．

以上の結果を総合すると，命題論理の範囲においては Java-MGTP の klic-MGTP と SATCHMO に対する優位性が顕著であり，A セルを用いた MGTP の実装の効果がよく現れているといえる．また，変数を含む問題においては，項インデキシングの性能が全体性能に大きく反映するが，Java-MGTP は klic-MGTP に比べ，絶対速度でも問題規模の増大にともなう性能劣化の面でも改善されている．また，基本的に項イン

デキシングが利用されていないSATCHMOに対しては，Java-MGTPの優位性が顕著であり，特に束縛変数の照合頻度が高い問題ほどその差は大きくなる．

7. おわりに

Java-MGTPの開発は，利用面での融通性の改善を主目的として始められたが，破壊的書換え等，Javaの言語機能の活用や，Aセルという新しい機構の導入，ならびに項インデキシング等，方式上の新たな改善により，基本推論性能の面でも従来のMGTPやSATCHMOを凌ぐシステムが実現できた．

また，この開発の経験を通じて，Javaコーディングに関して一般的に有効と考えられる指針をいくつか得ることができた．たとえば，

- いったん生成したオブジェクトのリサイクルに努め，メモリ消費を最小限に抑える．
- 「少ないクラスと一般的メソッド」より「多くのサブクラスと特殊化メソッド」

等があげられる．

モデル生成法は，充足可能な論理式に対し実際にモデルを構成するのが基本である．これは，反例の提示，故障診断における説明の作成等に欠かせない機能である．今回開発したJava-MGTPをベースとして様々な応用（たとえば，文献9)，10)）に向けた拡張が行われている．なお，本システムは，http://ss104.is.kyushu-u.ac.jp/からダウンロード可能である．

参考文献

1) Manthey, R. and Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog, *Proc. 9th Int. Conf. on Automated Deduction* (*CADE*), LNCS, Vol.310, pp.415–434, Springer-Verlag (1988).
2) 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, pp.1–12 (1996).
3) Fujita, H. and Hasegawa, R.: A Model Generation Theorem Prover in KL1 Using A Ramified-Stack Algorithm, *Proc. 8th Int. Conf. on Logic Programming* (*ICLP*), pp.535–548, MIT Press (1991).
4) Schütz, H. and Geisler, T.: Efficient Model Generation through Compilation, *Proc. 13th Int. Conf. on Automated Deduction* (*CADE*), LNAI, Vol.1104, pp.433–447, Springer-Verlag (1996).
5) 白井康之，長谷川隆三，藤田　博：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, pp.224–236 (1997).
6) 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, pp.63–68 (1998).
7) 長谷川隆三，藤田　博：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，九州大学大学院システム情報科学研究科報告，Vol.4, No.1, pp.57–62 (1999).
8) Sutcliffe, G. and Suttner, C.: `http://sunjess-en24.informatik.tu-muenchen.de/~tptp/TPTP.html` (1999).
9) 長谷川隆三，新田克己，白井康之：定理証明技術を応用した高度論証支援システムの開発，`http://www.ipa.go.jp/NBP/10seika/ase9_list.htm/` (1999).
10) 長谷川隆三，藤田　博，越村三幸：分岐補題の導入による極小モデルの効率的生成法，九州大学大学院システム情報科学研究科報告，Vol.4, No.2, pp.145–164 (1999).

(平成11年10月29日受付)
(平成12年 3月 2日採録)

長谷川隆三（正会員）

1949年生．1972年九州大学大学院工学研究科通信工学専攻修士課程修了．同年日本電信電話公社入社．1987〜1995年（財）新世代コンピュータ技術開発機構に出向．現在九州大学大学院教授．工学博士．ポリプロセッサシステム，データフローマシン，関数型言語，論理プログラミングおよび定理証明に関する研究に従事．電子情報通信学会，ソフトウェア科学会，人工知能学会各会員．

藤田　博（正会員）

1955年生．1978年東京大学大学院工学系研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．1984〜1990年（財）新世代コンピュータ技術開発機構に出向．現在九州大学大学院助教授．工学博士．論理プログラミングおよび自動推論システムに関する研究に従事．電子情報通信学会，ソフトウェア科学会，人工知能学会，ACM各会員．