# 1 はじめに

論理和データベースとは, 節の頭部に選言の出現を許す節集合である. これは知識表現の有効な基礎であると考えられており, また定理証明 (特に, 数学的な新しい発見など) やアブダクションのような新しい応用が期待され様々な報告がなされている.

定理証明の分野としてはモデル生成法によりすべてのモデルを生成し, その充足可能性を問う MGTP 処理系が有名である.

一方, 論理和データベースに対する問合わせは, 極小モデル意味論により以下のように定義される.

**定義 1.1 (論理和データベースに対する問合わせ)** プログラム $P$ に対する問合わせ $Q$ の値は以下のように定められる. *($\mathcal{MM}$:$P$ の全ての極小モデルの集合)*

$Q$ が真である. *iff* $\forall M_i \in \mathcal{MM}_P \ M_i \models Q$

$Q$ が偽である. *iff* $\forall M_i \in \mathcal{MM}_P \ M_i \not\models Q$

$Q$ が不定である. *iff* $Q$ は真でも偽でもない.

(**モデル**:$P$ の各式を全て真とする基底アトムの集合. **極小モデル**:$P$ のモデルの中で集合の包含関係に関して最も小さいもの.)

これに対し我々は関連性の概念を用い, 論理和データベースに対する問合わせを行なう手続き TPR[1] を提案し,Prolog 言語により実装, 有効性を示した.

さて, 従来, 定理証明系の分野では Prolog,KL1 といった論理型言語で MGTP[2] の実装が行なわれていたが, 近年 Java-MGTP[3] によって, Java 言語による実装技術を用い効率的な定理証明系が構築できることが示されている. 更にこの実装を利用し, 極小モデルを効率良く生成する MM-MGTP[4] がある.

この Java-MGTP と従来の Prolog,KL1 版の MGTP とを比べると,Java-MGTP は非常に良い実行効率を示している. ここで示された技術を問い合わせ手続き処理系 TPR に用いることで, より実行効率の上昇が期待できると考える.

そこで我々は,Java-MGTP で用いられている実装技術を用い, より高速なデータベースに対する問い合わせ処理系 Java-TPR を実現することを目的とする.

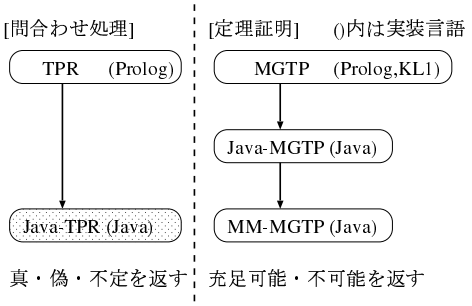



図 1: 問合わせとモデル生成処理系

# 2 関連研究

## 2.1 TPR のアルゴリズム

手続き TPR[1] は関連性の概念を導入し, それを論理和プログラム $P$ の非ホーン節展開基準とすることで問合わせ $Q$ の値を判定するための無駄なモデル拡張を抑えることに成功している.

TPR のアルゴリズムは,TR 部,PR 部とよばれる 2 つの手続きを順に適用することにより,$Q$ に対する答えを求める. 以下に TR 部と PR 部を示す.

[ **TR 部** ]

全関連節を用いたモデル拡張を行ない, 全てのモデルが $Q$ のモデルである時,true を返し終了. モデル拡張ができなくなったら PR 部へ移る.

[ **PR 部** ]

部分関連節を用いたモデル拡張を行ない, 極小モデルで $Q$ のモデルとなっているモデルが存在すれば unknown を返し終了. $Q$ のモデルになっているモデル候補で非極小モデルものは取り除く. モデル拡張ができなくなったら false を返す.

(全/部分) 関連節 (以下, 総称として関連節) は負節, ゴール節に対する SLD 木を構築し, 失敗木である (成功葉をもたない) ときに選択された (木に出現した) リテラルから定義される.

TPR は, **関連節を求める動作は後向き推論であり, モデル拡張は前向き推論という混合型計算**である.

## 2.2 MGTP

MGTP(Model Generation Theorem Prover) は節の頭部に選言の出現を許す.

MGTP は, 初期状態が $\phi$ であるモデル候補を以下の二つのルールにより拡張・棄却し, プログラムのすべてのモデルを計算する.

(i) **モデル拡張ルール:** 以下のような節が存在し, $A_{l+1}, \cdots, A_m \rightarrow$
$A_{1,1}, \cdots, A_{1,k_1} \mid \cdots \mid A_{l,1}, \cdots A_{l,k_l} (l \geq 1)$
$A_{l+1}\sigma, \ldots, A_m\sigma \in S$ かつ, **すべての** $i = 1, \ldots, l$ について, $A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma \in S$ が成り立たないとき, $\mathcal{S}$ から $S$ をとり除き, $\mathcal{S}$ に $i = 1, \ldots, l$ に関して $S \cup \{A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma\}$ を加える.

(ii) **モデル棄却ルール:** $A_1, \ldots, A_m \rightarrow$ なる負節が存在し, $A_1\sigma, \ldots, A_m\sigma \in S$ を満たす代入 $\sigma$ が存在するとき, $\mathcal{S}$ から $S$ を取り除く.

## 2.3 Java-MGTP

真偽値を保持するセル,**A(Active) セル**という概念を導入し, その値を利用したモデル拡張探索を図に示す. モデル生成戦略が深さ優先になっている.

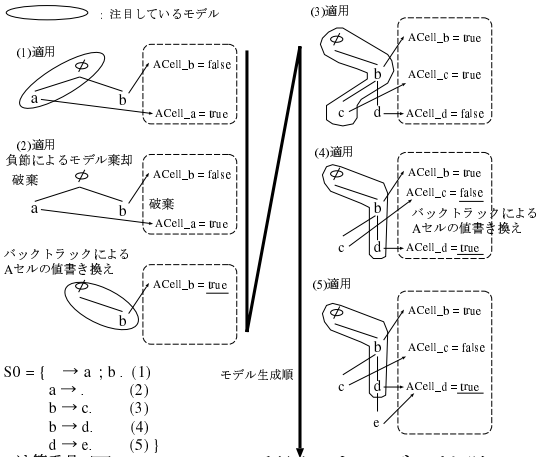



図 2: A セルを利用したモデル拡張

## 2.4 MM-MGTP

MM-MGTP は相補分割という極小モデル生成における枝切りに関する概念について議論を深め, これを強化した分岐補題という概念を示した. これと Java-MGTP を組み合わせることでモデル生成における効率的な枝刈りを行ない, **極小モデルを効率的に生成**する.

## 3 TPR 実装における問題点と提案

### 3.1 実装上の問題点

今回は Java-MGTP,MM-MGTP と実行速度を比較する意味からも, Java 言語により TPR を実装することを考える.

TPR を Java 言語で実装する上では以下の問題点を解決する必要がある.

- 後向き推論機構の必要性

TPR は混合型計算であり, ボトムアップ計算に特化したデータ構造をそのまま用いることができない. 手続き TPR を高速に実装するにあたっては, **前向き, 後向き推論の両方を高速に実行できるデータ構造**を必要とする.

- 文字列生成, 比較の高速化

記号処理系においては文字列の生成, 照合が全体の動作の中で大きな部分を占める.

- リスト, 配列, ベクタの適切なデータ構造選択

これらの構造にはアクセス速度と複製に特徴がある. 適切にデータ構造を選択する必要がある.

- 実装言語特性

仮想マシンによる実行時間にプラットフォームによって違いが見られるがいくつかの実験から共通して以下の点が考察される.

- クラスを用いるよりもプリミティブ型 (基本型) の構造の方が生成, 比較が高速である.
- クラスオブジェクト生成, キャスト操作は, 大変大きな計算コストを必要とする.

### 3.2 実装効率化のための提案

前節の問題点を解決するため以下の提案を行う.

- データ照合方法の変更
  - (i) 読み込み時に全ての記号文字列に対し一意の整数値 (記号識別番号) を割り当て, そのハッシュ値も逆引きできる**記号参照テーブルを生成**.
  - (ii) 項は内部構造として記号識別番号とオブジェクトの識別フラグからなるセルで構成.

  記号識別番号導入により, **オブジェクト生成, 比較の高速化を実現し, 節参照のためのハッシュ値計算の高速化も計る**.

- **生成オブジェクトの再利用**
  (メモリ確保計算の縮約)

  **関連リテラルの探索**で選択されるリテラルは, 毎回, 同じものが出現する場合が多い.

  これを利用し, 生成した関連リテラルに真偽値のフラグを割り当てる. 探索終了時, 生成した関連リテラルを保持したまま, それぞれのフラグを偽とする.

  そして, 新たな探索において再度同じリテラル生成の要求がある場合, 過去に生成されているオブジェクトならば, 過去に生成されたオブジェクトのフラグを真として再利用する.

  **モデル拡張においても, 探索方法を A セルを用いた深さ優先的なモデル拡張探索に変更することにより, リテラル再利用ができる.**

- 節参照テーブルと MG 木

  読み込まれた節や複製されたリテラルは図 3 のデータベースにより管理される. これは TPR アルゴリズムにおける要素参照を高速に実行できる.

- 根拠検査アルゴリズムの変更

  節参照テーブルの構造と A セルの概念を用いると, **根拠検査とよばれる極小モデルであるかを判定する手続きの, モデルに対する所属検査を高速化できる**. また, モデル展開に Java-MGTP の深さ優先戦略を用いると生成されたオブジェクトの再利用が有効に働く.

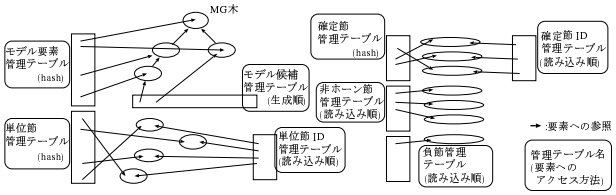
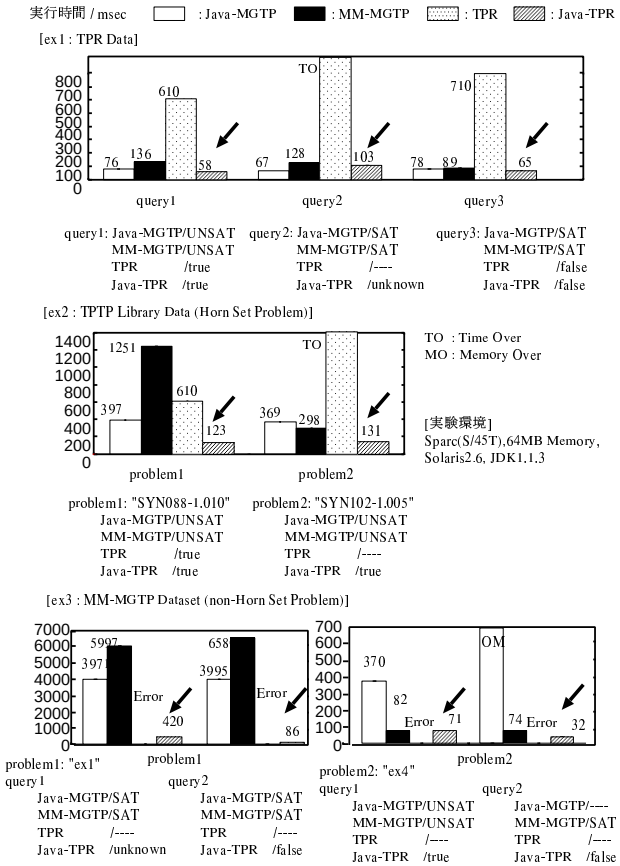



図 3: 節とリテラルの管理

## 4 実験結果

Java-TPR,TPR,Java-MGTP,MM-MGTP における実行時間比較を以下に示す. 実行を比較する際, その意味の違いから, Java-MGTP,MM-MGTP では全てのモデルを生成した時間である.




- 図 4: 処理系実行時間比較

ほとんどの場合において Java-TPR は MM-MGTP,TPR に比べ実行効率が良いことがわかる. なお, 一部の問題において TPR の実行を再現できなかったのは, 元々の実装時の処理系を再現できなかったことに起因する.

## 5 おわりに

提案する論理和データベースに対する効率的な問い合わせ処理システム Java-TPR を実装した. 今後の課題としては, 更に効率改善を行なうため, MM-MGTP で用いられている探索枝切り手法などの導入がある.

## 参考文献


[1] 島尻優香, 世木博久, 伊藤英則: 論理和データベースに対するゴール指向問い合わせ処理, 情報処理学会論文誌, (1995).

[2] Fujita, H. and Hasegawa, R., A Model Generation Theorem Prover in KL1 using Ramified Stack Algorithm, *Proc. of ICLP'91*, pp. 535–548, 1991.

[3] 長谷川 隆三, 藤田 博: Java によるモデル生成型定理証明系 MGTP の開発, 情報処理学会論文誌, Vol.41, No.6, pp.1791-1798, 2000 年 6 月.

[4] 長谷川 隆三, 藤田 博, 越村 三幸: 分岐補題の抽出による極小モデル生成の効率化, 人工知能学会論文誌, 16 巻, 2 号, pp.234-245, 2001 年 3 月.