プログラミング-言語・基礎・実践- 11－3
(1993. 3. 10)

抽象領域上のMGTPを用いたMGTPの最適化コンパイル

堀内 謙二† 藤田 博

三菱電機 (株) 中央研究所

〒661 兵庫県尼崎市塚口本町8丁目1-1

{horiuchi,fujita}@sys.crl.melco.co.jp

概要 モデル生成型定理証明器 (MGTP) はモデル生成法に基づく推論規則を用いて証明を行なう定理証明器であり，現在KL1やPrologで実装されている．本稿では，抽象領域上のMGTPを用いたMGTPプログラムの解析システムについて述べる．また，この抽象化されたMGTPを用いて，MGTPプログラムのnon-range-restrictedな節の自由変数部分にどのような形の項が現れるかを解析し，その解析結果を用いることによって，non-range-restrictedなMGTPプログラムを特化された効率的なrange-restrictedなMGTPプログラムへ変換できることを示す．また，MGTPプログラムと論理的に等価な対偶プログラムを解析することにより，より精巧な解析結果を得ることができることを示す．

A Specialization of MGTP Programs by Abstract MGTP

Kenji Horiuchi Hiroshi Fujita

Central Research Laboratory, Mitsubishi Electric Corporation

8-1-1, Tsukaguchi-Honmachi, Amagasaki, Hyogo 661 JAPAN

Abstract Model Generation Theorem Prover (MGTP) is an automatic theorem proving system based on the model generation method, which is implemented by KL1 or Prolog. In this paper we present an analysis system of MGTP programs by abstract MGTP. And we show that the system can analyze what terms can be instantiated to free variables appearing in non-range-restricted clauses of MGTP programs, and that non-range-restricted MGTP programs can be transformed into the specialized and efficient (range-restricted) MGTP programs by using the analyzing results. We also show that more precise information can be detected by analyzing programs in contraposition with MGTP programs.

† 平成5年2月16日より，産業システム研究所 (horiuchi@soc.sdl.melco.co.jp) に転任．

1. はじめに

定理証明器は定理(問題)が与えられると推論規則を用いて証明図を構成しようとするシステムである．MGTP[2]はモデル生成法に基づく推論規則を用いて証明を行なう定理証明器であり，KL1やPrologで実装されている．

MGTPでは，問題をrange-restrictedと呼ばれる節の集合(プログラムと呼ぶ)に限定すれば，生成されるモデルは変数を含まない基底アトムの集合になる．この条件により扱える問題の記述は制限を受けるが，OR分岐における場合わけにおいて共有変数の問題がなくなるとか，unification時にoccur checkの必要がないなど多くの利点を持つ．

non-range-restrictedな節は，自由変数を含むアトムを生成するので，これらのアトムの変数部分を出現可能なすべての基底項に置き換えた基底アトムの集合に置き換えるような節に組織的に書き換えることによって，節の意味を変えずにrange-restrictedな節に変換することができる．しかし，この変換は，例えば，$p(X)$というアトムを生成する変わりに，$\{p(X) \mid X \in \mathcal{U}\}$という一般には膨大なアトムの集合を生成する変換であり，例えば無限領域を扱う問題などの場合，実行効率が非常に落ちる場合がある．ここで，$\mathcal{U}$はプログラムに現れるすべての関数記号，定数記号から構成される基底項の集合，すなわち，Herbrand領域である．

本稿では，抽象化したMGTPを用いてMGTPプログラムのnon-range-restrictedな節の自由変数部分に現れる可能性のある項の形を解析することにより，効率的なrange-restrictedな節へ変換する方式を示す．また，MGTPプログラムを論理的に等価な(簡単にいうと対偶をとっている)プログラムに変換し，そのプログラムを解析することにより，より精巧な解析結果を得ることができることを示す．

2. モデル生成型定理証明器(MGTP)

MGTPはモデル生成法に基づき，与えられた節集合(プログラムと呼ばれる)のモデルを生成しようとする．ここでは，モデル生成法とMGTPについて簡単に述べる．

2.1 MGTPの構文

MGTPプログラムは節(*clause*)の集合で，各節は，次のような形をしている．

$$A_1, A_2, \ldots, A_n \rightarrow C_1; C_2; \ldots; C_m.$$

ここで，$A_i (0 \leq i \leq n), C_j (0 \leq j \leq m) \in Atom$である．また，含意(*implication*) $\rightarrow$の左側を前件部(*antecedent*)と呼び，右側を後件部(*consequence*)と呼ぶ．前件部の，は連言(*conjunction*)で，後件部の;は選言(*disjunction*)である．以下では，後件部が$false (m = 0)$の時，負節(*negative clause*)と呼び，そうでない時$(m \leq 1)$を正節(*positive clause*)と呼び，前件部が$true (n = 0)$である正節を特に単位節(*unit clause*)と呼ぶ．また，n個の単位節をまとめて$true \rightarrow C_1, \ldots, C_n$と書くかも知れない．

MGTPではモデルとして基底モデルのみを扱うため，プログラムPのすべての節は，以下の条件(*range-restrictedness*と呼ばれる)を満たしている必要がある．ここで，$var(F)$は式Fに現われる全ての変数の集合である．

$$\forall (A \rightarrow C) \in P (var(A) \supseteq var(C))$$

この条件により，単位節に基底節で，生成されるモデルは変数を含まない基底アトムのみからなる基底モデルとなる．

2.2 モデル生成法

ここでは，MGTPで用いられるrange-restrictedなプログラムのモデル生成法について簡単な例を用いて述べる．

モデル生成法は与えられたプログラムに対するモデルを空集合から始めて，次の規則を用いてモデル候補と呼ばれるアトムの集合を拡張していく方法である．

- ある節$A \rightarrow C$とモデル候補Mに対して，ある代入σが存在し，$M \models \sigma A$でかつ$M \not\models \sigma C$ならば，すなわち，$\sigma A \subseteq M$でかつ$\sigma C \not\subseteq M$ならば，Mは$M \cup \{\sigma C\}$に拡張される．またこの時，$C = C_1; \ldots; C_n (2 \leq n)$ならば，$M$は$n$個のモデル候補$M \cup \{\sigma C_1\}, \ldots, M \cup \{\sigma C_n\}$に場合わけされ拡張される．

もしあるモデル候補Mに対して，全ての節$A \rightarrow C$で，$M \models \sigma A$の時はいつも$M \models \sigma C$であるなら，Mはモデルであり，プログラムは充足可能(*satisfiable*)である．また，プログラムが充足可能ならば，モデルが(いつか)見つかるはずである．また，すべてのモデル候補に対してその前件部を充足するような負節(すなわち，充足されない負節)がプログラムに存在するなら，モデルは構成できないことがわかり，そのプログラムは充足不可能(*unsatisfiable*)である．

例 2.1 プログラムP_1を以下の節集合とする．

$$true \rightarrow p(a); q(b). \qquad C_1$$

$p(X) \rightarrow q(X); q(c).$ C_2
$q(b) \rightarrow p(b).$ C_3
$p(X), q(Y) \rightarrow false.$ C_4

モデル候補 $M_0 = \emptyset$ からモデル候補の拡張が始まる．最初は単位節 C_1 を用いて拡張され，

$M_1 = \{p(a)\}, \quad M_2 = \{q(b)\}$

の2つのモデル候補に場合わけされ拡張される．M_1 は節 C_2 を用いて

$M_{11} = \{p(a), q(a)\}, \quad M_{12} = \{p(a), q(c)\}$

に場合わけ拡張され，M_2 は節 C_3 を用いて

$M_{21} = \{q(b), p(b)\}$

に拡張される．この時，すべてのモデル候補 M_{11}, M_{12}, M_{21} が負節 C_4 の前件部 $p(X), p(Y)$ を充足するので，このプログラム P_1 は充足不可能であることがわかる．

さて次に，負節 C_4 を以下の負節 C_4' に置き換えたプログラム P_1' を考えてみよう．

$p(X), q(X) \rightarrow false.$ C_4'

この時，C_4' の前件部 $p(X), p(X)$ はモデル候補 M_{11} と M_{21} には充足されるが，M_{12} には充足されない．また，M_{12} はもうこれ以上拡張されないので，M_{12} は P_1' のモデルであり，P_1' は充足可能である．

2.3 MGTP

MGTP は前節のモデル生成法に基づき証明を行なう定理証明器である．MGTP にはモデル生成法の拡張規則に加えて高速化のためにいくつかの技法が盛り込まれている．例えば，MGTP は現在 KL1 と Prolog の2つのヴァージョンがあり，処理系に依存した様々な工夫もなされている．ここでは，処理系に依存しないモデル生成法のアルゴリズム上の規則のみを紹介する．

(1) あるモデル候補 M に対してその前件部を充足するようなある負節 (すなわち，充足されないようなある負節) がプログラムに存在するなら，モデル生成の過程でそのモデル候補 M は棄却する．
(2) あるモデル候補 M が $M + \Delta$ に拡張されたとする．次の拡張において，例えば，前件部のアトムが2つならば，$(M + \Delta) \times (M + \Delta)$ に対して照合を行なわなければならないが，この内 $M \times M$ に対しては冗長である．照合の履歴を保持し，$M \times M$ の照合を省略する．

規則 (1) のおかげで，決してモデルになりえないモデル候補を早い段階で棄却することができ，それ以降の不要なモデル候補の拡張を回避することができる．また，規則 (2) のおかげで，前件部における照合の冗長性を大幅に回避できる．

3. Range-restricted なプログラム

3.1 Range-restricted な節への変換

一般に証明したい問題が range-restricted であるとは限らない．MGTP では non-range-restricted な節は後件部の自由変数を強制的に instantiate するように，前件部に $dom/1$ という特殊なアトムを挿入している．$dom/1$ はプログラムに現れる定数，関数記号から構成されるすべての基底項を生成するような節で定義されている．

一般に，range-restricted なプログラムへの変換は以下の手続きで可能である．

- ある正節 $A \rightarrow C$ が non-range-restricted であったとする．すなわち，$var(C) \setminus var(A) \neq \emptyset$ である．
  その時，$A \rightarrow C$ を $\{dom(X) \mid X \in var(C) \setminus var(A)\} \cup A \rightarrow C$ で置き換える．

例 3.1 プログラム P_2 を以下の節集合とする．

$true \rightarrow p(a, b), p(a, c).$
$p(X, Y) \rightarrow p(f(X, Y), Z).$

このプログラムに対する $dom/1$ 述語は以下のように定義される．

$true \rightarrow dom(a), dom(b), dom(c)$
$dom(X), dom(Y) \rightarrow dom(f(X, Y))$

この時，P_2 は $dom/1$ 述語を使って以下のような range-restricted なプログラムに変換できる．

$true \rightarrow p(a, b), p(a, c).$
$dom(Z), p(X, Y) \rightarrow p(f(X, Y), Z)$

上記の例のように無限領域を対象とする問題に対しては，non-range-restricted な節の自由変数へ(無限領域上の)すべての基底項を代入するような range-restricted な節に変換していることになり，著しく効率が悪くなる場合がある．しかし，一般の non-range-restricted な問題を range-restricted な問題に変換できれば，MGTP の高速性を利用できる．

次節ではプログラムを解析することにより問題に依存した *dom* 述語を導入し，冗長な計算を省くことができる場合があることを，例を用いて簡単に説明する．

3.2 問題に依存した dom 述語

例 3.1 で示したプログラム P_2 をもう一度見てみよう．

$$true \rightarrow p(a,b), p(a,c).$$
$$dom(Z), p(X,Y) \rightarrow p(f(X,Y),Z)$$

P_2 のモデルは，

$$\{\ p(a,b), p(f(a,b),\top), p(f(f(a,b),\top),\top), p(f(f(f(a,b),\top),\top),\top),\ldots$$
$$p(a,c), p(f(a,c),\top), p(f(f(a,c),\top),\top), p(f(f(f(a,c),\top),\top),\top),\ldots\ \}$$

である．$\top$ は，*dom* で生成されるすべての基底項を意味している．ここで，上記の $\top$ の部分，例えば述語 p の第 2 引数は任意の基底項でなければならない．しかし，第 1 引数に注目してみると，$\{a, f(a,\_), f(f(a,\_),\_),\ldots\}$ のいづれかの形をしていることがわかる．この集合を生成する述語 dom_1 は以下ように定義できる．ここでの *dom* に関するプログラムは例 3.1 と同じである．

$$true \rightarrow dom_1(a).$$
$$dom_1(X), dom(Y) \rightarrow dom_1(f(X,Y)).$$

そこで，もし負節 c が，

$$p(f(f(f(X,Y),Z),W), f(f(f(X,Y),Z),W)) \rightarrow false.$$

のような節であったなら，プログラム $P_2 \cup \{c\}$ と

$$true \rightarrow p(a,b), p(a,c).$$
$$dom_1(Z), p(X,Y) \rightarrow p(f(X,Y),Z)$$
$$p(f(f(f(X,Y),Z),W), f(f(f(X,Y),Z),W)) \rightarrow false.$$

は，同じ結果(いずれも充足不可能)を返すと思われる．実際実験した結果，Prolog 版の MGTP で 3 倍ぐらいのスピードアップになった．

以下では，この節で説明したような解析を抽象領域上の MGTP で行ない，*dom* 述語を問題に特化し実行効率をあげる方法について述べる．また，ここで述べる抽象化 MGTP は抽象領域上の操作を行なう utility を prolog で記述し，MGTP のプログラムに挿入する以外，MGTP のエンジン部分やコンパイラ部分は通常の MGTP をほとんどそのまま利用できる枠組になっている．そのため，MGTP に実装されている高速化のための手法がほとんどそのまま解析に利用できる．

4. 抽象化モデル生成法

ここでは，2 節で簡単に述べたモデル生成法を不動点計算を用い形式化し，その後，抽象化することにより，抽象化されたモデル生成法を導く．

4.1 モデル生成法の形式化

まず，モデル生成法や MGTP について述べる前に，本稿で使われる基本的な用語について簡単に定義する．プログラムを P とする時，$\mathcal{C}_P$ を P に現れるすべての定数記号の集合，$\mathcal{F}_P$ を P に現れるすべての関数記号の集合，$Pred_P$ を P に現れるすべての述語記号の集合，$\mathcal{U}_P$ を Con_P と Fun_P 上で定義される変数を含まないすべての項の集合，$Base_P$ を $\mathcal{U}_P$ と $Pred_P$ 上で定義されるすべてのアトムの集合とする．また，Den_P を $Base_P$ の巾集合 (2^{Base_P}, すなわち，$Base_P$ の全ての部分集合からなる集合) とする．なお，特にプログラム P を限定しない場合は，下付きの P を省略する．

モデル生成法は，以下で定義されるアトムの集合の集合に対する関数 MG を繰り返し適用することにより，正モデル候補の集合を拡張していく証明方式として形式化できる．以下では，プログラムを P とした時，P^+ を P のすべての正節の集合，P^- を P のすべての負節の集合とする．また，正モデル候補とはプログラム P^+ から拡張生成されるアトムの集合 (モデル候補) である．

プログラム P に対する変換 $MG_P : 2^{Den_P} \to 2^{Den_P}$ は以下のように定義される．

$$MG_P(\mathcal{M}) = \bigcup_{c \in P^+} \bigcup_{M \in \mathcal{M}} ext(c, M)$$

ここで，

$$ext(A \to C_1; \ldots; C_n, M) = \begin{cases} \{M \cup \{\sigma C_i\} \mid 1 \leq i \leq n\}, & \text{if } \exists \sigma(\sigma A \subseteq M \wedge \forall C_i(\sigma C_i \notin M)); \\ \{M\}, & \text{otherwise.} \end{cases}$$

である．

$MG_P \uparrow 0 = \{\emptyset\}$ とした時，変換 MG_P は連続であるので，MG_P は最小不動点 $lfp(MG_P)$ を持ち，かつ，$lfp(MG_P) = MG_P \uparrow \omega$ である．$lfp(MG_P)$ をプログラム P の正モデル集合と呼び，$Model_P^+$ と記述する．この時，任意の $M \in Model_P^+$ に対して，$\sigma A \subseteq M$ なるある負節 $(A \to false) \in P^-$ があるならば，P は充足不可能である．そうでないなら，即ち，ある $M \in Model_P^+$ に対して，すべての負節 $(A \to false) \in P^-$ において $\sigma A \not\subseteq M$ ならば，P は充足可能である．

例 4.1　例 2.1 で用いたプログラム P_1 をもう一度見てみよう．

P_1^+ は以下の節集合である．

$true \to p(a); q(b).$
$p(X) \to q(X); q(c).$
$q(b) \to p(b).$

その時，$\mathcal{M}_i$ を $MG_{P_1} \uparrow i$ とすると，以下ように正モデル候補が生成される．

$\mathcal{M}_0 = \{\emptyset\}$
$\mathcal{M}_1 = \{\{p(a)\}, \{q(b)\}\}$
$\mathcal{M}_2 = \{\{p(a), q(a)\}, \{p(a), q(c)\}, \{q(b), p(b)\}\}$
$\mathcal{M}_2 = \mathcal{M}_3$

したがって，

$$Model_{P_1}^+ = \{\{p(a), q(a)\}, \{p(a), q(c)\}, \{q(b), p(b)\}\}$$

である．ここで，$P_{sat}^- = \{p(X), q(X) \to false\}$ と $P_{unsat}^- = \{p(X), q(Y) \to false\}$ とを考える．この時，モデル $\{p(a), q(c)\} \in Model_{P_1}^+$ が存在するので，$P_1^+ \cup P_{sat}^-$ は充足可能である．しかし，$P_1^+ \cup P_{unsat}^-$ は充足不可能である．

4.2 抽象領域上のモデル生成法

プログラムの意味がある領域上の不動点計算で与えられた場合の一般的な抽象化の方法やその正当性については，ここでは詳しく述べないが，以下の点に注意して領域や意味関数 (ここでは，Den や MG など) を抽象化すれば，正当な抽象化モデル生成法が構成できることがわかっている [3]．

Den の抽象化したものを $\underline{Den}$ とする．ここで，$\underline{Den}$ は有限集合で，$\underline{Den}$ 上に定義された順序関係 $\sqsubseteq$ に関して完備束を形成しなければならない．その最大要素，最小要素をそれぞれ $\top, \bot$ と記述する．

Den はプログラムによって決定されるのに対し，$\underline{Den}$ は，解析したいプログラムの特性 (例えば，タイプであるとか，入出力モードであるとか) に応じて，Den からうまく設計し選ばなければならない．この時，領域 Den と抽象領域 $\underline{Den}$ 間には次の条件を満足する単調な抽象化関数 $\alpha : 2^{Den} \to \underline{Den}$ と具体化関数 $\gamma : \underline{Den} \to 2^{Den}$ が定義できなければならない．

- 抽象領域 $\underline{Den}$ の任意の要素 $\underline{d}$ に対して，$\underline{d} = \alpha(\gamma(\underline{d}))$ である．
- 具体領域 Den の任意の部分集合 d に対して，$d \subseteq \gamma(\alpha(d))$ である．

一般に，$\underline{Den}$ 上の関係 $\sqsubseteq$ を γ によって集合の包含関係 $\subseteq$ へと自然に帰着することができ，その時，$\underline{Den}$ は $\sqsubseteq$ に関して完備束を形成している．

抽象化モデル生成法は，$\underline{Den}$ の設計に加えて MG_P に対応する連続な関数

$$\underline{MG_P} : 2^{\underline{Den}} \to 2^{\underline{Den}}$$

を定義することによって実現される．$\underline{MG_P}$ の最小不動点 $lfp(\underline{MG_P})$ をプログラム P の抽象化正モデル集合と呼び，ここでは $\underline{Model_P^+}$ と記述する．さて，$\underline{Den}$ が有限集合であるため，$\underline{Model_P^+}$ も有限で求まる．そのため，$\underline{Model_P^+}$ を計算することにより，$Model_P^+$ の特性，すなわち，プログラムの特性を知るわけであるが，その結果が「安全」であるためには，

$$Model_P^+ \subseteq \gamma(\underline{Model_P^+})$$

を満たさなければならない．この条件が成り立つためには，$Den, MG_P, \underline{Den}, \underline{MG_P}$ の間に次の条件が成り立っていれば十分であることが分っている．

- $\{\emptyset\} = \gamma(\bot)$，かつ，$\bot = \alpha(\{\emptyset\})$ である．
- $\underline{Den}$ の任意の要素 $\underline{d}$ に対して，$MG_P(\gamma(\underline{d})) \subseteq \gamma(\underline{MG_P}(\underline{d}))$ である．

5. 抽象 MGTP による項パターンの解析

MGTP は ICOT において開発された定理証明器で，処理系に依存した改良も多くなされており，なるべくオリジナルの MGTP システムを用いることが抽象 MGTP の実行速度，開発効率にも有効であると考えた．そのため，今回試作した抽象 MGTP は，オリジナルの MGTP システムをほとんど変更せずに MGTP プログラムを抽象 MGTP プログラムに変換することによって実現した．ここでは，試作した抽象 MGTP を用いた項パターンの解析について述べる．

5.1 領域の抽象化

ここでは項のパターンを解析するために導入した抽象領域について述べる．

まず，基底項の集合を表現する抽象項 ($Aterm$) と抽象項要素 ($Aelm$) を導入する．

$Aterm :=$ [$Aelm${, $Aelm$}*]

$Aelm :=$ 定数シンボル | 関数記号($Aterm${, $Aterm$}*)

ここで，プログラムを P とした時，関数記号は，$\mathcal{F}_P$ のいずれかで，定数シンボルは，$\mathcal{C}_P$ のいずれか，あるいは 'bot'，あるいは 'top'，あるいは関数記号の後ろに '_' を付加した特殊なシンボルである．'bot' は空集合 を表し，'top' は $\mathcal{U}_P$ を表している．また，関数記号が多重に入れ子になっている場合二重以上の入れ子構造を抽象化し '関数記号_' というシンボルで表現している．このシンボルは，すぐ上位の同じ関数記号で始まる抽象項を表している．

簡単にいうならば，項の集合を抽象項要素のリストで表現し，入れ子になっている関数記号に対しては特殊なシンボル "関数記号_" を用いて，2重より深い構造をカットオフした．

具体化関数 γ は以下のように定義される．

$$\gamma([\underline{t}_1, \ldots, \underline{t}_n]) = \bigcup \gamma_{elm}(\underline{t}_i)$$

ここで，

$$\gamma_{elm}(\underline{t}) = \begin{cases} \emptyset & \text{if } \underline{t} = \mathtt{bot}; \\ \mathcal{U} & \text{if } \underline{t} = \mathtt{top}; \\ \underline{t} & \text{if } \underline{t} \in \mathcal{C}; \\ \{\mathtt{f}(t_1, \ldots, t_n) \mid (\underline{t}_i = \mathtt{f\_} \supset t_i \in \gamma(\underline{t})) \vee (\underline{t}_i \neq \mathtt{f\_} \supset t_i \in \gamma(\underline{t}_i))\} & \text{if } \underline{t} = \mathtt{f}(\underline{t}_1, \ldots, \underline{t}_n) \text{ かつ } \mathtt{f} \in \mathcal{F}. \end{cases}$$

である．

例 5.1 抽象項 $\underline{t}$ を `[1,2,t([1,2,t_],[1,2,t_])]` とする．その時，$\gamma(\underline{t})$ は，

$$\gamma(\underline{t}) = \{\mathtt{1},\ \mathtt{2},\ \mathtt{t(1,1)},\ \mathtt{t(1,2)},\ \mathtt{t(2,1)},\ \mathtt{t(2,2)},\ \mathtt{t(t(1,1),1)},\ \mathtt{t(t(1,2),1)},\ \mathtt{t(t(2,1),1)},\ \ldots\}$$

なる基底項の (無限) 集合を表現している．

抽象化関数 α を定義する前に，補助関数 $\Delta: \mathcal{F} \times N \times \mathcal{U} \rightarrow Aterm$ を定義する．

ここで，$\mathcal{U}_{\mathtt{f}_{(n)}} = \{t \in \mathcal{U} \mid t$ は 引数 n の関数記号 $\mathtt{f}$ で始まる項$\}$ とする．

$$\Delta(\mathtt{f}_{(n)}, i, t) = \{\Delta(\mathtt{f}_{(n)}, i, t_j) \mid t = \mathtt{f}(t_1, \ldots, t_n) \wedge (t_j \in \mathcal{U}_{\mathtt{f}_{(n)}} \vee 0 \leq i \neq j \leq n)\} \cup \{\alpha(t_i) \mid t = \mathtt{f}(t_1, \ldots, t_n) \wedge t_i \notin \mathcal{U}_{\mathtt{f}_{(n)}}\}$$

直観的に説明すると，関数 $\Delta(\mathtt{f}_{(n)}, i, t)$ は，項 $t \in \mathcal{U}_{\mathtt{f}_{(n)}}$ の時に，t の $\mathtt{f}/n$ で始まる全ての部分項 $t' \in \mathcal{U}_{\mathtt{f}_{(n)}}$ の i 番目の引数に現れる項 $t_i \notin \mathcal{U}_{\mathtt{f}_{(n)}}$ に対応する抽象項を集合で返す．また，T が項の集合の場合も自然に拡張でき，$\Delta(\mathtt{f}_{(n)}, i, T) = \bigcup_{t \in T} \Delta(\mathtt{f}_{(n)}, i, t)$ である．この時，抽象化関数 α は以下のように定義される．

$$\alpha(T) = (\bigcup_{\mathtt{f}_{(n)} \in \mathcal{F}} \alpha_{\mathtt{f}}(\{t_i \mid \mathtt{f}(t_1, \ldots, t_n) \in T\})) \cup \{t \mid t \in \mathcal{C}\}$$

ここで，$F_i = \mathtt{f}(t_{i_1}, t_{i_2}, \ldots, t_{i_n})$ で，$T = \{F_1, F_2, \ldots, F_m\}$ とすると，

$$\alpha_{\mathtt{f}}(T) = \mathtt{f}(\alpha_{\mathtt{f\_}}(\{t_{1_1}, t_{2_1}, \ldots, t_{m_1}\}) \cup \Delta(\mathtt{f}_{(n)}, 1, T), \ldots, \alpha_{\mathtt{f\_}}(\{t_{1_n}, t_{2_n}, \ldots, t_{m_n}\}) \cup \Delta(\mathtt{f}_{(n)}, n, T))$$

$$\alpha_{\mathtt{f\_}}(\{t_{i_n}, t_{i_n}, \ldots, t_{i_n}\}) = \{\mathtt{f\_} \mid t_{i_j} \in \mathcal{U}_{\mathtt{f}_{(n)}}\} \cup \{\alpha(t_{i_j}) \mid t_{i_j} \notin \mathcal{U}_{\mathtt{f}_{(n)}}\}$$

である．

さて，この抽象関数も簡単に説明すると，定数記号はそのまま，それらの集合もそのまま (実際には [,] によるリストで) 表現される．関数記号 (例えば f) による項も同様にそのまま表現されるが，2重以上の入れ子の場合，それより深いレベルの項は $f\_$ で代用 (抽象化) される．しかし，抽象化によって失われた部分項の情報はそれらの全ての f で始まる部分項の各引数 i 毎に $\Delta(f_{(n)}, i, t)$ を用いて全て集められその和集合として一番上位の f の i 番目の引数に付け加えられる．

例 5.2　項の集合 T を {[1, 2, t(t(1,2),t(2,1))]} とする．その時，$\alpha(T)$ は，

$\alpha(T)$ = [1,2,t([1,2,t_],[1,2,t_])]

なる抽象項である．

抽象項は抽象項要素の集合 (リスト) で表現されており，それらの結び ($join$, $\vee$) や交わり ($meet$, $\wedge$) は集合の和集合や積集合，または，同じ関数記号で始まる抽象項要素に対しては，その引数毎の結びや交わりを用いて自然に定義できる．ここで，$\underline{t}, \underline{s}$ を抽象項要素とすると，$\vee, \wedge$ は以下のように定義できる．

$$\underline{t} \vee \underline{s} = \begin{cases} \mathtt{f}(\underline{t}_1 \vee \underline{s}_1, \ldots \underline{t}_n \vee \underline{s}_n); & \text{if } \underline{t} = \mathtt{f}(\underline{t}_1, \ldots \underline{t}_n) \text{ でかつ, } \underline{s} = \mathtt{f}(\underline{s}_1, \ldots \underline{s}_n); \\ [\underline{t}, \underline{s}]; & \text{otherwise.} \end{cases}$$

$$\underline{t} \wedge \underline{s} = \begin{cases} \mathtt{f}(\underline{t}_1 \wedge \underline{s}_1, \ldots \underline{t}_n \wedge \underline{s}_n); & \text{if } \underline{t} = \mathtt{f}(\underline{t}_1, \ldots \underline{t}_n) \text{ でかつ, } \underline{s} = \mathtt{f}(\underline{s}_1, \ldots \underline{s}_n); \\ [\,] & \text{otherwise.} \end{cases}$$

抽象 MGTP におけるモデルは抽象アトムの集合である．抽象アトムは抽象項を引数にもつアトムとして自然に定義でき，抽象項上の操作がそのまま抽象アトムにも適用できる．

5.2 モデル拡張の抽象化

ここでは，前節の領域の抽象化にしたがって，モデル生成法を抽象化する．4節の変換 MG の定義において用いられている領域の結びや交わりは前節で定義した．ここでは，抽象モデル候補 $\underline{M}$ と正節 $A \rightarrow C$ を用いた操作について述べる．

抽象代入 $\underline{\sigma}$ は抽象項の変数への代入 $X/\underline{t}$ の集合である．

抽象アトム $\underline{A}$ と (通常の) アトム A の unification を以下のように行なわれる．結果は $\underline{\sigma}$ として返される．

(1) A に $m(\geq 2)$ 回現れるすべての変数 X を m 個の異なる変数 $X_1, \ldots, X_m$ に置き換える．A' とする．
(2) $\underline{A}$ の引数は抽象項 $\underline{t}$(リストで表現されている) である．A' の引数は (変数を含む) 項 t である．引数毎に t と unify 可能な抽象項要素を $\underline{t}$ から選ぶ．この時，結果の抽象代入 $\underline{\sigma'}$ の中に現れるすべての定数シンボル $f\_$ は対応する f で始まる抽象項に置き換える．
(3) $\underline{\sigma'}$ 内の全ての代入 $X_i/\underline{t_i}$ を $\underline{\sigma'}$ から取り除き，すべての $\underline{t_i}$ の交わりを X への代入 $X/\wedge_{i=1}^{m} \underline{t_i}$ を $\underline{\sigma'}$ に加える．
(4) A に複数回現れる全て変数 X に対して上記の処理を行ない，結果の抽象代入を $\underline{\sigma}$ とする．

また，抽象代入 $\underline{\sigma}$ の通常の項 C への適用は以下のように行なわれる．

(1) 抽象代入を通常の代入として C に適用し，代入されない全ての変数を [top] で置き換える．C' とする．
(2) 多重に入れ子になっている関数記号は前節の α の定義にならって _ 付きの定数シンボルに置き換える．

例 5.3　抽象モデル候補

$\underline{M}$ = { p([t([1,2,t_],[1,2,t_])], [t([1],[2])]) }

とする．この時，$\underline{M}$ が，例えば，節

p(X,Y),p(Y,Z) --> p(t(X,Y),t(Y,Z))

を用いてどのように拡張されるか考えてみる．まず，前件部 A = p(X,Y),p(Y,Z) を A' = p(X,Y_1),p(Y_2,Z) に変換する．次に，A' の各アトムと $\underline{M}$ の唯一の抽象アトムを unify して，抽象代入 $\underline{\sigma'}$

{X/[t([1,2,t_],[1,2,t_])],Y_1/[t([1],[2])],Y_2/[t([1,2,t_],[1,2,t_])],Z/[t([1],[2])]}

を得る．

[t([1],[2])] $\wedge$ [t([1,2,t_],[1,2,t_])] = [t([1],[2])]

なので，$\underline{\sigma}$ は，

{X/[t([1,2,t_],[1,2,t_])],Y/[t([1],[2])],Z/[t([1],[2])]}

である．この $\underline{\sigma}$ を節の後件部 p(t(X,Y),t(Y,Z)) に単純に適用すると，

p(t([t([1,2,t_],[1,2,t_])], [t([1],[2])]), t([t([1],[2])], [t([1],[2])]))

が得られる．多重の t を取り除くと，

p([t([1,2,t_],[1,2,t_])], [t([1,t_],[2,t_])])

となる．例えばこの時の p の第2引数は，[t([t([1],[2])],[t([1],[2])])] という t の二重構造である．安直に二重以上の t の構造を t_に置き換えるだけでは，[t([t_],[t_])] となり，引数の情報が完全に失われる．そこで，節 5.1 の補

助関数 $\Delta(\mathtt{t}, i, \mathtt{t}(1,2))(i = 1,2)$ を用いて失われた情報を引数の毎に集め上位の t の構造の各引数の位置に加えることにより，引数情報の喪失を防いでいる．すなわち，得られる抽象項は [t([1,t_],[2,t_])] である．この時，実際の構造 (t は二重) を含むような抽象項 (t の n 重の構造が含まれている) を生成している．結局，

$\underline{M} \cup \{$p([t([1,2,t_],[1,2,t_])], [t([1,t_],[2,t_])])$\}$

はもうこれ以上この節では拡張できないことが分る．

5.3 抽象 MGTP の実装

ここでは，抽象 MGTP の実装について述べる．オリジナルの MGTP(Prolog 版) をなるべくそのまま流用することを念頭におき開発したので，抽象項の操作はすべて Prolog で書き，抽象 MGTP utility として MGTP プログラムに挿入した．また，通常，抽象 MGTP では正モデルに対する特性 (ここでは項のパターン) を解析するので負節を解析に用いることはない．

例えば，プログラム

```
true --> p(t(1,2),t(1,2)), p(t(2,1),t(1,2))
p(X,Y),p(Y,Z) --> p(t(X,Y),t(Y,Z))
```

は以下のような抽象 MGTP のためのプログラムに変換される．

```
true --> p([t([1,2],[1,2])],[t([1],[2])])
p(X,Y1),p(Y2,Z), {meet(Y1,Y2,Y3),comp(t(X,Y3),T1),comp(t(Y3,Z),T2)} --> p(T1,T2).
```

ここで，meet/3 は抽象項の交わりの抽象項を返し，comp は抽象項を含んだ項を抽象項に変換する (すなわち，2 重以上の入れ子構造をカットオフする) utility である．これら 2 つの他に，抽象項の結びを計算するプログラム join/2 が基本的な抽象 MGTP utility である．

例 5.4　以下の MGTP プログラム (has-part-t1) は今回の実験に用いたものである [4].

```
hp(jhn,t(2,1),han) --> false.
true       --> in(jhn,boy).
in(X,boy) --> in(X,hum).
in(X,han) --> hp(X,5,fin).
in(X,hum) --> hp(X,2,arm).
in(X,arm) --> hp(X,1,han).
dom(Z), dom(W), hp(X,V,Y)          --> in(sk1(X,Y,Z,V,W),Y) ; hp(X,t(V,W),Z).
hp(X,V,Y), hp(sk1(X,Y,Z,V,W),W,Z) --> hp(X,t(V,W),Z).

true --> dom(boy), dom(hum), dom(fin), dom(arm), dom(han), dom(jhn),
         dom(1), dom(2), dom(5).
dom(X), dom(Y) --> dom(t(X,Y)).
dom(X), dom(Y), dom(Z), dom(V), dom(W) --> dom(sk1(X,Y,Z,V,W)).
```

上記のプログラムを抽象 MGTP utility を用いて書き換える．

```
true --> in([jhn],[boy]).
in(X,Y),{mem(boy,Y),\+mem(hum,Y)} --> in(X,[hum|Y]).
      中略
hp(X,V,Y),{comp([sk1(X,Y,[top],V,[top])],Sk),comp([t(V,[top])],T)}
                    --> in(Sk,Y) ; hp(X,T,Z).
hp(X1,V1,Y1), hp(Sk,W3,Z3),
         {mem(sk1(X2,Y2,Z2,V2,W2),Sk), meet_(X1,X2,X),meet_(V1,V2,V),meet_(Y1,Y2,Y),
          meet_(W2,W3,W),meet_(Z2,Z3,Z), comp([t(V,W)],T)}
                    --> hp(X,T,Z).
```

ここで，meet_/3 は結果が [bot] である時失敗する以外は meet/3 と同じであり，mem/2 は抽象項要素が抽象項に含まれるかどうかをチェックする述語であり，実は，mem(X,Y) は meet_(X,Y,_) と同じである．

5.4 抽象 MGTP の改良

前節の抽象 MGTP の実装は非常に naïve な実装で，抽象 MGTP utility は Prolog で 100 行ほどであるし，MGTP の compiler や core 部分は全く変更をしていない．

このような実装でも一応解析はでき，小さな問題に関しては reasonable な計算時間で解析は終了した．しかし，少し大きな問題対しては(有限領域と言えども)組合せの爆発が起こり，解析に膨大な時間がかかる．ここでは，解析処理の高速化のための改良点について簡単に述べる．

(1) 新しい抽象アトムを生成した時に `in(X,[hum|Y])` のように *cons* で `in(X,Y)` と `in(X,[hum])` の結びを生成するのは論理的には正しいが，MGTP ではアトムの同一性を項の同一性で判定しているので，引数のリストの置換の数だけ同じ意味の抽象アトムを生成するので冗長である．MGTP のエンジン部分はなるべく触りたくなかったので，結びをとる時に先頭に付けるのではなく sorted insert を用いて挿入し，抽象項はいつも sort されているようにした．

(2) 一般に，$A \supset B$ が成り立ちような A が既にモデル候補 M に存在する時は B で拡張する必要はない．このような検査を subsumption テストと呼ぶが，MGTP ではモデルが基底アトムからなるので，B と同一のアトムが M にあるかどうかを検査するだけで十分である．しかし，抽象 MGTP では基底アトム間に論理的な包含関係が存在する (例えば，`in([jhn],[boy])` は `in([jhn],[boy,hum])` に包含されている)．subsumption テストを行なうことは冗長性の除去において有効である．この処理も，抽象 MGTP プログラムレベルで実現した．

(3) (2) の subsumption テストとは逆に新たに生成された抽象アトム B が既に M に存在する抽象アトムのいくつかを包含するかも知れない．このような時，それら包含される抽象アトム $A_1,\ldots,A_n$ の変わりに B をおけば十分であり，$A_1,\ldots,A_n$ がすべて冗長である．このような検査を backward subsumption テストと呼ぶが，MGTP では (2) と同じ理由で必要がない検査である．しかし，抽象 MGTP では一般に既に存在する抽象アトムを包含するような抽象アトムが生成されるのでこの検査は非常に有効である．しかし，この処理は既に登録された抽象アトムを棄却しなければならないのでオリジナルの MGTP のエンジン部分を少し修正した．

例 5.4　上記の改良を施した抽象 MGTP を用いて例 5.3 のプログラム (`has-part-t1`) を解析した結果，以下の抽象モデルが得られる．

```
hp([jhn,sk1([jhn,sk1_],[arm,fin,han],[top],[1,2,5,t([1,2,5,t_],[1,2,5,t_])],[top])],
   [1,2,5,t([1,2,5,t_],[1,2,5,t_])],
   [arm,fin,han]).

in([jhn,sk1([jhn,sk1_],[arm,fin,han],[top],[1,2,5,t([1,2,5,t_],[1,2,5,t_])],[top])],
   [arm,boy,fin,han,hum]).
```

5.5 解析精度の改良

例 5.4 で得られた 2 つの抽象項を $\underline{hp}$ と $\underline{in}$ とすると，それらとプログラム (`has-part-t1`) の正モデル $Model^+$ の間に

$$Model^+ \subseteq \gamma([\underline{hp}, \underline{in}])$$

なる関係がある．この結果を利用してプログラム (`has-part-t1`) の *dom* 述語を特化したい訳であるが，ちょうど *dom* 述語で生成される基底項が代入される `sk1/5` の第 3，5 引数の部分は `[top]` と解析されており，*dom* 述語の特化のための有益な情報が得られていないのが分る．その理由を考えてみると，生成された `hp/3` のアトムは節

```
hp(X,V,Y), hp(sk1(X,Y,Z,V,W),W,Z) --> hp(X,t(V,W),Z).
```

を用いて拡張される．その時，例えば，関数 `sk1/5` の第 3 引数と述語 `hp/3` の第 3 引数とは同じ変数 `Z` なので，unify されようとする．その unification が成功するものだけが拡張に用いられる．すなわち抽象 MGTP では `meet/3` で結びが取られ，その結果が `[bot]` でないものだけが抽象モデル候補の拡張に用いられる．しかし，そのような情報は，この抽象領域での抽象 MGTP では，すでに生成された (抽象) アトムに伝播されない．

このような情報の喪失を避けるために抽象領域の構造を変数の sharing 情報を保持するなどに変更し，より精密な解析を行なうことも可能であるが，ここでは，プログラム P の各節を節をその節の対偶 (*contraposition*) に当たる節に変換し，そのような節の集合 (対偶プログラムと呼び $\overline{P}$ と書く) を抽象 MGTP で解析した．

P の対偶プログラム $\overline{P}$ への変換は以下のように行なう．

(1) $A_1,\ldots,A_n \rightarrow \mathit{false}$ は $\mathit{true} \rightarrow \neg A_1;\ldots;\neg A_n$ に変換される．
(2) $\mathit{true} \rightarrow C_1;\ldots;C_m$ は $\neg C_1,\ldots,\neg C_m \rightarrow \mathit{false}$ に変換される．
(3) $A_1,\ldots,A_n \rightarrow C_1;\ldots;C_m$ は $\neg C_1,\ldots,\neg C_m \rightarrow \neg A_1;\ldots;\neg A_n$ に変換される．

ここで，$\neg A$ は A の述語 (例えば p/n とする) を対応する異なる述語 (例えば not_p/n) に置き換えたアトムである．

対偶プログラムの正節の集合 $\overline{P}^+$ を抽象 MGTP で解析することは，直観的には，プログラム P のモデル生成における前件部の照合における項のパターン (call pattern) を解析することになる．

抽象MGTPでは負節を用いないので，(2)で変換された節は利用しない．プログラムPに含まれる負節から(1)で変換される単位節を用いて解析を行なうと，プログラムPの負節の前件部と照合可能なアトムを生成するためのモデル拡張に用いられたcall patternを解析することになる．しかし，ここで解析したいのはモデルに含まれるすべてのアトムを生成するのに用いられた(すなわち，すべての可能な)call patternを解析したいので，例5.4で示したような，プログラムPの解析で得られたPのモデルを包含するような抽象アトムを単位節として用いる．

例 5.5　抽象MGTPを用いて例5.3のプログラム(has-part-t1)の対偶プログラムを解析した結果，以下のhp/3に関する抽象モデルが得られる．

```
hp([jhn,sk1([jhn,sk1_],[arm,fin,han],[arm,fin,han],
            [1,2,5,t([1,2,5,t_],[1,2,5,t_])],[1,2,5,t([1,2,5,t_],[1,2,5,t_])])],
   [1,2,5,t([1,2,5,t_],[1,2,5,t_])],
   [arm,fin,han]).
```

この結果を用いてプログラム(has-part-t1)の*dom*述語を特化する．今度は，sk1/5の第3，5引数の部分はそれぞれ[arm,fin,han]，[1,2,5,t([1,2,5,t_],[1,2,5,t_])]と解析されており，その結果を用いてプログラム(has-part-t1)のnon-range-restrictedな節

```
dom(Z), dom(W), hp(X,V,Y) --> in(sk1(X,Y,Z,V,W),Y) ; hp(X,t(V,W),Z).
```

を特化し，

```
domZ(Z), domW(W), hp(X,V,Y) --> in(sk1(X,Y,Z,V,W),Y) ; hp(X,t(V,W),Z).
true --> domZ(arm), domZ(fin), armZ(han).
true --> domW(1), domW(2), domW(5).
domW(X), domW(Y) --> domW(t(X,Y)).
```

とする．

6. まとめ

オリジナルのプログラムhas-part-t1ではdom/1が組合せの爆発を起こし，解が得られなかった．しかし，前節で得られた*dom*特化プログラムによってdom/1によって生成されるアトムが大幅に縮小されモデル候補が128,各モデル候補の平均のアトム数が20のモデル候補の集合を生成してプログラムhas-part-t1が充足不可能であることがわかった．

一般に，*dom*述語を導入することによりrange-restrictedなプログラムに変換するれば，non-range-restrictedな問題をMGTPで扱うことが出来る．しかし，安易に*dom*述語を導入すれば膨大な探索空間が生じ，本質的には浅い問題でも解けない場合がある．ここではモデルに含まれるアトムの項パターンを解析することにより，*dom*述語の特化を行なう手法を提案した．ここでの手法の特徴を以下にまとめる．

(1) 抽象MGTPを用いて解析した．抽象MGTPは抽象モデル生成法に基づいており，得られる解析結果の正当性が保証されている．

(2) MGTPプログラムを抽象MGTPプログラムにcompileする方式で実現し，MGTPシステムをほとんど変更せずにそのまま流用しいる．そのため，MGTPの高速化の手法を(処理系依存部分，非依存部分に関わらず)そのまま受け継いでいる．また，開発コストも(新たにinterpreterを書くより)十分少ないと思われる．

(3) 対偶プログラムによる解析をも用いたので，変数共有情報などを残さないような“rough な”抽象領域でも期待した解析結果を得ることが出来た．

参考文献

[1] 太田 好彦，井上 克巳，長谷川 隆三，中島 誠，“ノンホーン・マジックセット”, Technical Memorandum TM-???, ICOT, September, 1992.

[2] Fujita, H. and R. Hasegawa, “A Model-Generation Theorem Prover in KL1 Using Ramified Stack Algorithm”, TR-606, ICOT, 1990. Also in K. Furukawa (ed.), Proc. of the Eighth International Conference on Logic Programming, The MIT Press, 1991.

[3] 堀内 謙二，“論理プログラムの抽象解釈を用いた解析”, Technical Report TR-0807, ICOT, September, 1992. あるいは，関数プログラミング JSSST'91, pp.79–104, 近代科学社， 1992.

[4] Stickel, M. E., “A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler”, J. of Automated Reasoning, Vol.4, No.4, pp.353–380, 1988.