LLPTTP: 線形論理型言語コンパイラ処理系を用いた定理証明システム

田村 直之　番原 睦則

一階述語論理の節形式を Prolog プログラムに変換し，Prolog コンパイラ処理系を用いて定理証明を行うシステムとして PTTP (Prolog Technology Theorem Prover) が知られている．本論文では，節形式を線形論理型言語 LLP のプログラムに変換し，LLP コンパイラ処理系を用いることで，より効率的な証明探索が可能になることを示す．特に，証明中のリテラルをリソースとして追加することにより，ME (model elimination) 処理を高速化している点に特徴がある．

1 はじめに

一階述語論理の節形式を Prolog プログラムに変換し，Prolog コンパイラ処理系を用いて定理証明を行うシステムとして PTTP が知られている [9] [10].

PTTP で用いている ME (Model Elimination) 証明手続き [6] [5] では，パスと呼ばれるリテラルの列を保持し，パス中に単一化可能なリテラルが存在するかどうかをチェックする必要がある．PTTP では，パスはリテラルのリストで表現されているため，Stickel 自身も述べているように効率が良くない [10].

一方，Lolli [3] や LLP [14] [13][†1] などの線形論理型言語では，事実や規則に相当する論理式を動的に追加・削除することができ，リテラルの列であるパスを自然かつ効率良く表現することが可能である．

そこで本論文では，節形式を線形論理型言語 LLP のプログラムに変換し，LLP コンパイラ処理系を用いて定理証明を行う LLPTTP (LLP Technology Theorem Prover) について述べ，TPTP 問題セット [11] に対する実行結果を基にその有効性について論じる．なお，線形論理および線形論理型言語については，[12], [7] などを参考にしていただきたい．

2 体系 ME

まず，項，原子論理式，リテラル (正リテラル，負リテラル) などは通常のように定義し，リテラル L の補リテラルを $\overline{L}$ で表す．また，偽を表す特別な原子論理式として $\bot$ を用いる．

節はリテラルのマルチ集合 (重複要素を許す集合)[†2] で，少なくとも一つの正リテラルを含むものとする (負リテラルしか含まない節には $\bot$ を追加する)．また，全変数が $\vec{x}$ である節 C を，$C[\vec{x}]$ と表し，変数 $\vec{x}$ に項 $\vec{t}$ を代入した結果を $C[\vec{t}/\vec{x}]$ と表す．

節 C が $\{L_1, L_2, \ldots, L_n\}$ の時，各 $i = 1, 2, \ldots, n$ について $(\overline{L_1} \wedge \cdots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \cdots \wedge \overline{L_n}) \supset L_i$ を C の i 番目の contrapositive と呼ぶことにする．

LLPTTP: Theorem Prover using Compiler of a Linear Logic Programming Language.

Naoyuki TAMURA, 神戸大学学術情報基盤センター, Information Science and Technology Center, Kobe University.

Mutsunori BANBARA, 神戸大学学術情報基盤センター, Information Science and Technology Center, Kobe University.

コンピュータソフトウェア, Vol.?, No.?(2003), pp.?–?.
2002 年?月?日受付.

†1 http://bach.scitec.kobe-u.ac.jp/llp/

†2 Model Elimination では factoring を必要としないので，節はリテラルの集合ではなく，マルチ集合で良い．

$$\frac{}{\Gamma\ ;\ \Delta\ ;\ L, \Pi \longrightarrow L}\ (\mathrm{r})$$

$$\frac{C, \Gamma\ ;\ \Delta\ ;\ \overline{L}, \Pi \longrightarrow \overline{L_1} \quad \cdots \quad C, \Gamma\ ;\ \Delta\ ;\ \overline{L}, \Pi \longrightarrow \overline{L_n}}{C, \Gamma\ ;\ \Delta\ ;\ \Pi \longrightarrow L}\ (\mathrm{e}_1)$$
(provided $n \geq 0$, $C[\vec{t}/\vec{x}] = \{L, L_1, \ldots, L_n\}$ for some $\vec{t}$)

$$\frac{\Gamma\ ;\ \Delta\ ;\ \overline{L}, \Pi \longrightarrow \overline{L_1} \quad \cdots \quad \Gamma\ ;\ \Delta\ ;\ \overline{L}, \Pi \longrightarrow \overline{L_n}}{\Gamma\ ;\ C, \Delta\ ;\ \Pi \longrightarrow L}\ (\mathrm{e}_2)$$
(provided $n \geq 0$, $C = \{L, L_1, \ldots, L_n\}$)

図 1 体系 ME

$$\frac{}{\Gamma; \longrightarrow 1}\ (\mathrm{R1}) \qquad \frac{}{\Gamma; \Delta \longrightarrow \top}\ (\mathrm{R}\top)$$

$$\frac{\Gamma; \Delta_1 \longrightarrow B_1 \quad \Gamma; \Delta_2 \longrightarrow B_2}{\Gamma; \Delta_1, \Delta_2 \longrightarrow B_1 \otimes B_2}\ (\mathrm{R}\otimes) \qquad \frac{\Gamma; \Delta \longrightarrow B_1 \quad \Gamma; \Delta \longrightarrow B_2}{\Gamma; \Delta \longrightarrow B_1 \,\&\, B_2}\ (\mathrm{R}\,\&)$$

$$\frac{\Gamma; \Delta \longrightarrow B_i}{\Gamma; \Delta \longrightarrow B_1 \oplus B_2}\ (\mathrm{R}\oplus_i)\ (i = 1, 2) \qquad \frac{\Gamma; \longrightarrow B}{\Gamma; \longrightarrow !\,B}\ (\mathrm{R}\,!)$$

$$\frac{\Gamma; \Delta, B \longrightarrow C}{\Gamma; \Delta \longrightarrow B \multimap C}\ (\mathrm{R}\multimap) \qquad \frac{\Gamma, B; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \Rightarrow C}\ (\mathrm{R}\Rightarrow)$$

$$\frac{\Gamma; \Delta \longrightarrow B[y/x]}{\Gamma; \Delta \longrightarrow \forall x.B}\ (\mathrm{R}\forall)\ (y \text{ is not free in the lower sequent}) \qquad \frac{\Gamma; \Delta \longrightarrow B[t/x]}{\Gamma; \Delta \longrightarrow \exists x.B}\ (\mathrm{R}\exists)$$

$$\frac{\Gamma; \longrightarrow B_1 \quad \cdots \quad \Gamma; \longrightarrow B_n \quad \Gamma; \Delta_1 \longrightarrow C_1 \quad \cdots \quad \Gamma; \Delta_m \longrightarrow C_m}{\Gamma; \Delta_1, \ldots, \Delta_m, B \longrightarrow A}\ (\mathrm{BC})$$
(provided $n, m \geq 0$, A is atomic, and $\langle \{B_1, \ldots, B_n\}, \{C_1, \ldots, C_m\}, A\rangle \in \|B\|$)

$$\frac{\Gamma, B; \longrightarrow B_1 \quad \cdots \quad \Gamma, B; \longrightarrow B_n \quad \Gamma, B; \Delta_1 \longrightarrow C_1 \quad \cdots \quad \Gamma, B; \Delta_m \longrightarrow C_m}{\Gamma, B; \Delta_1, \ldots, \Delta_m \longrightarrow A}\ (\mathrm{BC}\,!)$$
(provided $n, m \geq 0$, A is atomic, and $\langle \{B_1, \ldots, B_n\}, \{C_1, \ldots, C_m\}, A\rangle \in \|B\|$)

図 2 体系 HLL

次に，Model Elimination に基づく体系 ME を示す．ここで定義する体系 ME は，factoring や lemma 化を含まない単純なものを考えるが [6]，Connection Calculus [2] のアイデアを採り入れて，基底節 (変数を含まない節) と非基底節を区別することにする．これは，定理証明系の leanCoP [8] や lolliCoP [4] でも採用されている方法である．

体系 ME のシーケントは以下の形式で表される．

$$\Gamma\ ;\ \Delta\ ;\ \Pi \longrightarrow L$$

ここで，Γ は非基底節の集合，Δ は基底節のマルチ集合，Π はリテラルのマルチ集合 (パスと呼ばれる)，L はリテラル (ゴールと呼ばれる) である．なお，Γ の各節は全称束縛された閉じた論理式であるが，Π 中の論理式および L は自由変数を含んでいる．

図 1 に体系 ME の推論規則を示す．

- reduction 規則 (r): ゴール L と単一化可能なリテラルがパス Π 中にあれば，その部分に対する証明は成功する．なお，この単一化により L お

よびパス Π は変化することがある．

- extension 規則 (e_1): ある $\vec{t}$ について，$C[\vec{t}/\vec{x}] = \{L, L_1, \ldots, L_n\}$ となる非基底節 $C[\vec{x}]$ が Γ 中にあれば，パス Π に $\overline{L}$ を追加し，各 $\overline{L_i}$ をゴールとしてそれぞれ証明探索を行う．特に $n = 0$ の時，その部分に対する証明は成功する．
- extension 規則 (e_2): $C = \{L, L_1, \ldots, L_n\}$ となる基底節 C が Δ 中にあれば，Δ から C を取り除いた後，規則 (e_1) と同様の処理を行う．

なお，元々の ME では extension 規則でリテラル L をパスに追加し，reduction 規則で補リテラル $\overline{L}$ を探索している．ここではパス中のリテラルの正負が逆になっているが，本質的な違いはない．

体系 ME は一階述語論理の節形式に対して健全かつ完全である†3．すなわち，$\Gamma\,;\,\Delta\,;\,\longrightarrow \bot$ が ME で証明可能の時，かつその時に限り，節集合 $\Gamma \cup \Delta$ (ただしリテラル $\bot$ を削除する) は通常の導出原理で反駁可能である．また命題論理については，ME は決定手続きを与える (規則 (e_2) により，基底節が高々一度しか利用できないため)．

3 体系 HLL

体系 HLL は，線形論理型言語 Lolli [3] および LLP [14] [13] に対する体系であり，以下に定義される R-論理式 (リソース論理式) および G-論理式 (ゴール論理式) を用いる．ただし，A は原子論理式，$B \Rightarrow C$ は $!B \multimap C$ を意味する．

$$R ::= \top \mid A \mid R_1 \,\&\, R_2 \mid G \multimap R \mid G \Rightarrow R \mid \forall x.R$$

$$G ::= \top \mid A \mid G_1 \,\&\, G_2 \mid R \multimap G \mid R \Rightarrow G \mid \forall x.G \mid G_1 \oplus G_2 \mid 1 \mid G_1 \otimes G_2 \mid\, !G \mid \exists x.G$$

体系 HLL のシーケントは以下の形式で表される．

$$\Gamma\,;\,\Delta \longrightarrow G$$

ここで，Γ は R-論理式の集合 (無限コンテキスト)，Δ は R-論理式のマルチ集合 (有界コンテキスト)，G は G-論理式 (ゴール) である．

HLL のシーケント $\Gamma\,;\,\Delta \longrightarrow G$ は，直観主義線形論理のシーケント $!\Gamma, \Delta \longrightarrow G$ に対応している．したがって，無限コンテキスト中の事実や規則は何度でも利用できるが，有界コンテキスト中の事実や規則は一度しか利用できない．なお，無限および有界コンテキストには，それぞれ $\Rightarrow$, $\multimap$ 演算子により，事実や規則を動的に追加できる．

図 2 に体系 HLL の推論規則を示す．図中のバックチェーン規則 (BC), (BC!) で用いられている $\|R\|$ は，以下のように定義される 3 つ組を表す†4．

1. $\langle \emptyset, \emptyset, R \rangle \in \|R\|$
2. $\langle \Gamma, \Delta, R_1 \,\&\, R_2 \rangle \in \|R\|$ の時，$\langle \Gamma, \Delta, R_1 \rangle \in \|R\|$, $\langle \Gamma, \Delta, R_2 \rangle \in \|R\|$
3. $\langle \Gamma, \Delta, \forall x.R' \rangle \in \|R\|$ の時，すべての基底項 t について $\langle \Gamma, \Delta, R'[t/x] \rangle \in \|R\|$,
4. $\langle \Gamma, \Delta, G \Rightarrow R' \rangle \in \|R\|$ の時，$\langle \Gamma \cup \{G\}, \Delta, R' \rangle \in \|R\|$
5. $\langle \Gamma, \Delta, G \multimap R' \rangle \in \|R\|$ の時，$\langle \Gamma, \Delta \uplus \{G\}, R' \rangle \in \|R\|$

体系 HLL は，上記のように論理式を制限した直観主義線形論理に対して健全かつ完全である．すなわち，$\Gamma\,;\,\Delta \longrightarrow G$ が HLL で証明可能の時，かつその時に限り，$!\Gamma, \Delta \longrightarrow G$ は直観主義線形論理で証明可能である．

4 節形式の線形論理式への変換

4.1 PTTP での変換

PTTP では，正リテラル，負リテラルそれぞれに別の述語名を対応させ，節 C の各 contrapositive を一つのホーン節に対応させている．したがって，規則 (e_1) は Prolog での述語呼出しで実現できる．また，各述語にパスを表すリストを引数として追加し，リストを探索することで規則 (r) を実現している．

たとえば，節 $\{p(X), q(X)\}$ の 1 番目の contrapositive $\overline{\mathtt{q(X)}} \supset \mathtt{p(X)}$ は以下のホーン節に変換される．

```
p(X,Path) :- not_q(X,[not_p(X)|Path]).
```

また，節集合 $\{\{\mathtt{p(X)}, \mathtt{q(X)}\}, \{\neg\mathtt{p(X)}, \mathtt{q(X)}\}, \{\mathtt{p(X)}, \neg\mathtt{q(X)}\}, \{\neg\mathtt{p(X)}, \neg\mathtt{q(X)}\}\}$ 全体は以下のように変換される ($\bot$ については特別扱いしている)．ここで，最初の 4 行が規則 (r) に，次の 8 行が各 contrapositive に，最後の行が証明すべきゴールに対応する．

†3 基底節と非基底節を区別しない場合，規則 (r), (e_1) だけで完全である．

†4 定義中の $\uplus$ はマルチ集合の和集合を表す．

```
p(X,Path) :- member(p(X),Path).
not_p(X,Path) :- member(not_p(X),Path).
q(X,Path) :- member(q(X),Path).
not_q(X,Path) :- member(not_q(X),Path).

p(X,Path) :- not_q(X,[not_p(X)|Path]).
q(X,Path) :- not_p(X,[not_q(X)|Path]).
not_p(X,Path) :- not_q(X,[p(X)|Path]).
q(X,Path) :- p(X,[not_q(X)|Path]).
p(X,Path) :- q(X,[not_p(X)|Path]).
not_q(X,Path) :- not_p(X,[q(X)|Path]).
not_p(X,Path) :- q(X,[p(X)|Path]).
not_q(X,Path) :- p(X,[q(X)|Path]).

bot(Path) :- p(X,Path), q(X,Path).
```

ただし，パス中のリテラルの正負は元の PTTP の逆になっており，また簡単のため後述の Iterative Deepening のための引数などは含めていない．

さらに，PTTP では，ゴールと identical な補リテラルがパス中にあれば失敗する規則を追加し，探索空間の枝刈りを行っている (Identical Goal Pruning)．これを用いれば，規則 (e_2) を導入しなくても命題論理に対する決定手続きを実現することが可能であるが，PTTP では実現されていない．

4.2 線形論理式への変換の方針

線形論理型言語 LLP は Prolog のスーパーセットであるから，PTTP と同様の変換は可能だが，さらに線形論理の特徴を生かした方法を考える．

PTTP では，パスはリテラルのリストで表現されており，Stickel 自身も述べているように効率が良くない [10]．一方，LLP では，事実や規則をリソースとして追加可能である．すなわち $R \Rightarrow G$ を実行すると，ゴール G 中で R をプログラム節と同様に使用できる．LLP コンパイラでは，リソースはクロージャとしてコンパイルされ，ハッシュ表を通して検索されるので，効率の高い実現が可能になる．なお，リソースが自由変数を含んでいる場合，すなわちハッシュのキーが後から具体化されうる場合も正しく動作する．

そこで，パスを無限コンテキスト中のリソースとして表現する．たとえば，contrapositive $\overline{\texttt{q(X)}} \supset \texttt{p(X)}$ を以下のように変換する (top の役割については後述)．

```
p(X) :- (not_p(X) :- top) => not_q(X).
```

not_p(X) :- top，すなわち $\top \multimap$ not_p(X) が規則 (r) そのものであり，それを $\Rightarrow$ により追加している．

また基底節は，規則 (e_2) により証明図の各枝で高々一度しか利用できない．そこで，基底節に対応する論理式を有界コンテキスト中に置くことにし，contrapositive 中の $\wedge$ に & を対応させる．また，証明図の各葉ではゴール $\top$ を実行し，残ったリソース (利用しなかった基底節) を削除する．これにより，基底節に対応する論理式は証明図の各枝で高々一度しか利用できないことを自然に表現できる．

4.3 線形論理式への変換の方法

まず，リテラル L に対する変換 L^* を次のように定める．L が正リテラル $P(\vec{x})$ の場合は $L^* = P(\vec{x})$，L が負リテラル $\neg P(\vec{x})$ の場合は $L^* = not\_P(\vec{x})$，ただし not_P は新しい述語記号である．また $L^+ = \top \multimap L^*$ とする．これは，上で述べたように利用しなかった基底節を削除するためである．

節 $C[\vec{x}] = \{L_1, \ldots, L_n\}$ の i 番目の contrapositive を $H_i[\vec{x}] = (\overline{L_1} \wedge \cdots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \cdots \wedge \overline{L_n}) \supset L_i$ として，$H_i[\vec{x}]$ に対する変換 $H_i[\vec{x}]^*$，および節 $C[\vec{x}]$ に対する変換 $C[\vec{x}]^*$ を次のように定義する．

$$H_i[\vec{x}]^* = \forall\vec{x}.((\overline{L_i}^+ \Rightarrow (\overline{L_1}^* \,\&\, \cdots \,\&\, \overline{L_{i-1}}^* \,\&\, \overline{L_{i+1}}^* \,\&\, \cdots \,\&\, \overline{L_n}^*)) \multimap L_i^*)$$

$$C[\vec{x}]^* = H_1[\vec{x}]^* \,\&\, \cdots \,\&\, H_n[\vec{x}]^*$$

特に $n = 1$ の場合，$C[\vec{x}]^* = \forall\vec{x}.((\overline{L_1}^+ \Rightarrow \top) \multimap L_1^*)$ である．

例えば，節集合 $\{\{\texttt{p(X)}, \texttt{q(X)}\}, \{\neg\texttt{p(X)}, \texttt{q(X)}\}, \{\texttt{p(X)}, \neg\texttt{q(X)}\}, \{\neg\texttt{p(X)}, \neg\texttt{q(X)}\}\}$ は以下のように変換され，PTTP の変換での最初の 4 行は不要となる ($\bot$ については特別扱いしている)．

```
p(X) :- (not_p(X) :- top) => not_q(X).
q(X) :- (not_q(X) :- top) => not_p(X).
not_p(X) :- (p(X) :- top) => not_q(X).
q(X) :- (not_q(X) :- top) => p(X).
p(X) :- (not_p(X) :- top) => q(X).
not_q(X) :- (q(X) :- top) => not_p(X).
not_p(X) :- (p(X) :- top) => q(X).
not_q(X) :- (q(X) :- top) => p(X).

bot :- p(X) & q(X).
```

この変換結果である R-論理式の集合を Γ とすると，$\Gamma\,;\longrightarrow \bot$ に対する HLL での証明図は図 3 のように

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \top \multimap \mathtt{not\_p}, \top \multimap \mathtt{q}\ ;\ \longrightarrow \top}\,(\mathrm{R}\top)}{\Gamma, \top \multimap \mathtt{not\_p}, \top \multimap \mathtt{q}\ ;\ \longrightarrow \mathtt{not\_p}}\,(\mathrm{BC!})}{\Gamma, \top \multimap \mathtt{not\_p}\ ;\ \longrightarrow (\top \multimap \mathtt{q}) \Rightarrow \mathtt{not\_p}}\,(\mathrm{R}\Rightarrow)}{\Gamma, \top \multimap \mathtt{not\_p}\ ;\ \longrightarrow \mathtt{not\_q}}\,(\mathrm{BC!})}{\Gamma\ ;\ \longrightarrow (\top \multimap \mathtt{not\_p}) \Rightarrow \mathtt{not\_q}}\,(\mathrm{R}\Rightarrow)}{\Gamma\ ;\ \longrightarrow \mathtt{p}}\,(\mathrm{BC!})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \top \multimap \mathtt{not\_q}, \top \multimap \mathtt{p}\ ;\ \longrightarrow \top}\,(\mathrm{R}\top)}{\Gamma, \top \multimap \mathtt{not\_q}, \top \multimap \mathtt{p}\ ;\ \longrightarrow \mathtt{not\_q}}\,(\mathrm{BC!})}{\Gamma, \top \multimap \mathtt{not\_q}\ ;\ \longrightarrow (\top \multimap \mathtt{p}) \Rightarrow \mathtt{not\_q}}\,(\mathrm{R}\Rightarrow)}{\Gamma, \top \multimap \mathtt{not\_q}\ ;\ \longrightarrow \mathtt{not\_p}}\,(\mathrm{BC!})}{\Gamma\ ;\ \longrightarrow (\top \multimap \mathtt{not\_q}) \Rightarrow \mathtt{not\_p}}\,(\mathrm{R}\Rightarrow)}{\Gamma\ ;\ \longrightarrow \mathtt{q}}\,(\mathrm{BC!})$$

$$\dfrac{\Gamma\ ;\ \longrightarrow \mathtt{p} \,\&\, \mathtt{q}}{\Gamma\ ;\ \longrightarrow \bot}\,(\mathrm{R}\,\&)\,(\mathrm{BC!})$$

図 3 LLPTTP による変換結果の証明例

なる (簡単のため引数の部分は省略してある).

以上の変換について，以下の命題が成り立つ.

命題 4.1

$$\Gamma\ ;\ \Delta\ ;\ \Pi \longrightarrow L \text{ が ME で証明可能}$$
$$\iff \quad \Gamma^*, \Pi^+\ ;\ \Delta^* \longrightarrow L^* \text{が HLL で証明可能}$$

5 LLPTTP

定理証明システム LLPTTP は，基本的には前節での変換方法に従って，節形式を LLP のプログラムに変換し，それをコンパイル・実行するが，さらに PTTP と同様に以下の手法を導入する.

LLP での実行は depth-first で行われるため，HLL に対する完全な証明探索戦略ではない．そこで PTTP と同様に Iterative Deepening の手法を用い，完全な証明探索を実現する．すなわち，各述語に探索の深さを表す引数を追加し，証明図の各枝で規則 (e_1) の利用される最大回数を制限する．証明探索はまず深さ 0 に対して行い，証明が見つからなければ順に深さをインクリメントしながら探索を進める．ただし，節のすべてが基底節の場合 (命題論理の場合) は，深さ 0 で証明が見つからなければ証明不可能とする.

また，PTTP でも行われている Identical Goal Pruning を導入し探索空間の枝刈りを行う．その他，証明出力のための引数も追加する.

付録に，TPTP 問題セット中の SYN079-1 に対する変換結果を示す．ただし，本質的でない部分については変換結果を一部省略している.

6 LLPTTP の性能評価

LLPTTP の性能評価のため，証明探索問題セットである TPTP v2.4.1 [11] を用い，証明探索時間を計測した．用いた問題は，節形式の充足不可能な全問題 2752 問 (うち命題論理 35 問) と，節形式の充足可能な命題論理の全問題 7 問の計 2759 問である．TPTP 中の各問題には証明の困難さを表す 0.00–1.00 の指標が付けられており，指標が大きいほど困難な問題であることを表している.

性能比較のため PTTP [9] [10] および lolliCoP [4] についても計測を行った．計測は Pentium III 550MHz の Linux マシン上で行い，変換時間およびコンパイル時間を含めて CPU 時間が 3 分以上の場合はタイムアウトとして実行を打ち切った.

PTTP は，節形式を Prolog プログラムに変換し，抽象機械 WAM の命令列にコンパイルした後，エミュレータが WAM 命令列を実行することで証明探索を行う．Prolog コンパイラ処理系としては，高速な商用 Prolog コンパイラの一つとして知られている SICStus Prolog 3.7.1 を用いた．また，オリジナルの PTTP では occur check 付きの単一化を行うためのプログラムを独自に用意しているが，ここではより高速な SICStus Prolog のライブラリ述語 `unify_with_occurs_check` を利用した．これが，オリジナルの PTTP からの唯一の変更点である.

lolliCoP は，読み込んだ節形式に対して，体系 ME の規則に基づいて証明探索を行う LLP プログラムである．LLPTTP と同様に，パス中のリテラルはリ

表 1 LLPTTP, PTTP, lolliCoP の TPTP 問題セットに対する実行結果

指標	問題数	LLPTTP	PTTP	lolliCoP
0.00	1457	903 (62%)	896 (61%)	757 (52%)
0.01∼	60	42 (70%)	47 (78%)	23 (38%)
0.10∼	106	42 (40%)	43 (41%)	19 (18%)
0.20∼	329	56 (17%)	55 (17%)	19 (6%)
0.30∼	79	19 (24%)	10 (13%)	8 (10%)
0.40∼	176	54 (31%)	26 (15%)	28 (16%)
0.50∼	552	12 (2%)	8 (1%)	1 (0%)
Total	2759	1128 (41%)	1085 (39%)	855 (31%)

ソースとして表現されているが，節形式を LLP プログラムに変換するのではなく，そのままインタプリタ的に証明探索を行う点が異なっている．

LLPTTP のための LLP コンパイラ処理系としては LLP 0.51 を使用した．LLP のプログラムは，WAM を拡張した抽象機械 LLPAM の命令列にコンパイルされた後，LLPAM エミュレータにより実行される．LLP では実行時フラグをセットすれば，すべての単一化で occur check が行われる．

なお，LLP 0.51 の Prolog コンパイラ処理系としての性能は，SICStus Prolog と比較した場合は 2 倍程度遅く，フリーの Prolog コンパイラ処理系の SWI Prolog と比較した場合は 2 倍程度速くなっている [13].

表 1 に，2759 問に対する LLPTTP, PTTP, lolliCoP での実行結果を示す．LLPTTP は最大の 1128 問を解くことができた．また，指標が大きい (すなわちより困難な) 問題ほど，LLPTTP の解いた問題数の割合は大きくなっており，LLPTTP の手法が有効であることが確認できる．

LLPTTP, PTTP, lolliCoP のすべてが解いた問題は 790 問あったが，それらについての CPU 時間の平均は，LLPTTP が約 20.1 秒，PTTP が約 4.8 秒，lolliCoP が約 8.2 秒だった．LLPTTP の CPU 時間が長くなっているのは，LLP コンパイラが比較的遅く，コンパイルに平均 15 秒近くかかっていることが原因である．コンパイル時間の短縮は今後の改善課題といえる．

7 おわりに

本論文では，一階述語論理の節形式を線形論理型言語 LLP のプログラムに変換し，LLP コンパイラ処理系を用いて定理証明を行うシステム LLPTTP について述べた．

LLPTTP の証明探索は ME (Model Elimination) 証明手続きに基づいており，ME 中のパス (リテラルの列) を，線形論理型言語のリソースとして表現している点に特徴がある．また，命題論理に対しては決定手続きとなっている．

TPTP 問題セットに対して，既存の PTTP および lolliCoP と比較した結果，より良い結果を得られることが確認できた．これは，線形論理型言語での特徴であるリソースを用いたプログラミングが有効に働いた結果であると考えられる．

もちろん，ここで述べた LLPTTP の結果は 1980 年代後半の仕事である PTTP に基づいたものであり，最新の定理証明系に匹敵する性能を得ているとはいえない．今後は，論文 [5] に述べられているさまざまな戦略の導入や，論文 [1] の restart model eliniation の手法を導入することなどが考えられる．

参考文献

[1] Baumgartner, P. and Furbach, U.: Model Elimination without Contrapositives and its Application to PTTP, *Journal of Automated Reasoning*, Vol. 13(1994), pp. 339–359.

[2] Bibel, W.: Mating in matrices, *Communications of the ACM*, Vol. 26,No. 11(1983), pp. 844–852.

[3] Hodas, J. S. and Miller, D.: Logic Programming in a Fragment of Intuitionistic Linear Logic, *Information and Computation*, Vol. 110,No. 2(1994), pp. 327–365.

[4] Hodas, J. S. and Tamura, N.: LolliCoP – a linear logic implementation of a lean connection-method theorem prover for first-order classical logic, *Proceedings of the International Joint Conference on Automated Reasoning 2001*, Springer-Verlag LNCS

2083, June 2001, pp. 670–684.
[5] Letz, R. and Stenz, G.: Model Elimination and Connection Tableau Procedures, *Handbook of Automated Reasoning*, Vol. II, North-Holland, 2001, pp. 2015–2114.
[6] Loveland, D. W.: A simplified format for the model elimination procedure, *Journal of the ACM*, Vol. 16,No. 3(1969), pp. 349–363.
[7] Miller, D.: Overview of linear logic programming. Submitted as a chapter for a book on linear logic, ed. Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Phil Scott, Cambridge University Press.
[8] Otten, J. and Bibel, W.: leanCoP: lean Connection-Based Theorem Proving, *Proceedings of the Third International Workshop on First-Order Theorem Proving*, 2000, pp. 152–157.
[9] Stickel, M.: A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, *Journal of Automated Reasoning*, Vol. 4(1988), pp. 353–380.
[10] Stickel, M.: A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog, Technical note 464, SRI International, 1989.
[11] Sutcliffe, G. and Suttner, C.: The TPTP problem library—CNF release v1.2.1, *Journal of Automated Reasoning*, Vol. 21(1998), pp. 177–203.
[12] 田村直之: [特別招待論文] 線形論理と論理プログラミング, **電子情報通信学会技術研究報告**, Vol. 102, No. 91, 5 月 2002, pp. 37–42.
[13] 番原睦則, 姜京順, 田村直之: 線形論理型言語のコンパイラ処理系のための抽象機械について, **コンピュータソフトウェア**, Vol. 18,No. 1(2001), pp. 39–60.
[14] 姜京順, 番原睦則, 田村直之: 線形論理型言語の効率的なリソース管理モデル, **コンピュータソフトウェア**, Vol. 18,No. 0(2001), pp. 138–154.

A SYN079-1 問題と変換結果

$$\{\ big\_f(f(a,b),f(b,c))\ \}$$
$$\{\ big\_f(f(b,c),f(a,c))\ \}$$
$$\{\ \neg big\_f(X,Y),\ \neg big\_f(Y,Z),\ big\_f(X,Z)\ \}$$
$$\{\ \neg big\_f(f(a,b),f(a,c))\ \}$$

```
main :-
  prolog_flag(unknown, _, fail),
  prolog_flag(occurs_check, _, on),
  (prove(0, Proof) ->
    write('PROVED'), nl
  ;
    write('FAILED'), nl
  ).

prove(Lim, Proof) :-
  p_BOT(Lim, Proof).
prove(Lim, Proof) :-
  propositional(no), Lim1 is Lim + 1,
  prove(Lim1, Proof).

propositional(no).

p_BOT(A,B) :- q_BOT(A,B).

p_big_f(A,B,C,D) :-
  r_not_big_f(E,F), A==E, B==F, !, fail.
p_big_f(A,B,C,D) :-
  r_big_f(A,B), D=reduction, top.
p_big_f(A,B,C,D) :-
  r_not_big_f(A,B)=>q_big_f(A,B,C,D).

p_not_big_f(A,B,C,D) :-
  r_big_f(E,F), A==E, B==F, !, fail.
p_not_big_f(A,B,C,D) :-
  r_not_big_f(A,B), D=reduction, top.
p_not_big_f(A,B,C,D) :-
  r_big_f(A,B)=>q_not_big_f(A,B,C,D).

% { big_f(f(a,b),f(b,c)) }
linear (
q_big_f(f(a,b),f(b,c),A,B) :-
  B=proof(clause(1,1,[])), top
).

% { big_f(f(b,c),f(a,c)) }
linear (
q_big_f(f(b,c),f(a,c),A,B) :-
  B=proof(clause(2,1,[])), top
).

% { ¬big_f(X,Y), ¬big_f(Y,Z), big_f(X,Z) }
(
q_not_big_f(A,B,D,E) :-
  D>=1, F is D-1,
  E=proof(clause(3,1,[2,3]),G,H),
  ( p_big_f(B,C,F,G) & p_not_big_f(A,C,F,H) )
) & (
q_not_big_f(B,C,I,J) :-
  I>=1, K is I-1,
  J=proof(clause(3,2,[1,3]),L,M),
  ( p_big_f(A,B,K,L) & p_not_big_f(A,C,K,M) )
) & (
q_big_f(A,C,N,O) :-
  N>=1, P is N-1,
  O=proof(clause(3,3,[1,2]),Q,R),
  ( p_big_f(A,B,P,Q) & p_big_f(B,C,P,R) )
).

% { ¬big_f(f(a,b),f(a,c)) }
linear (
q_BOT(A,B) :-
  B=proof(clause(4,0,[1]),C),
  p_big_f(f(a,b),f(a,c),A,C)
) & (
q_not_big_f(f(a,b),f(a,c),D,E) :-
  E=proof(clause(4,1,[])), top
).
```