TR-566

KL1による定理証明プログラム

藤田　博，長谷川隆三

May, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191－5
Telex ICOT J32964

Institute for New Generation Computer Technology

KL1による定理証明プログラム

藤田 博, 長谷川 隆三
ICOT・第5研究室

1. はじめに

我々はKL1をベースにした一階述語論理の高速な定理証明システムの実現を目指している。一階述語論理の定理証明の方式としては、タブロー法、レゾリューション法[Wos 88]、コネクション法[Bibel 86]など幾つかあるが、いずれが我々の目的に最も適しているか現在検討中である。これまでに、タブロー法に属すると考えられる、Prologをベースとした二つの既存の定理証明プログラムの検討を行い、KL1による試作を進めている。一つはStickelのPTTP(Prolog Technology Theorem Prover)[Stickel 88]、もう一つはManthey&BryのSATCHMO[Bry 88]である。いずれもPrologをベースに、その長所を活かしつつ最小限の拡張を行うことによってフルファーストオーダーの定理証明を行えるようにしたものである。

ここで主要な問題となるのは、証明対象となる論理式に現れる変数の扱い方である。これをKL1変数で表現することができるか否か、あるいはこれをKL1の基底項で表現した場合に、変数束縛の伝搬、ユニフィケーション、節のコピーや環境の管理はどうするか、といった問題が生じる。PTTPではオカーチェック付きユニフィケーションが不可欠であるが、SATCHMOの方は前向き推論に限定すれば照合で済む。この点、変数の扱いが簡単になるSATCHMOの方式がKL1インプリメンテーションに適していると思われる。実際、証明対象となる論理式に現れる変数をKL1変数で表現できたために、KL1版SATCHMOはインタプリタ方式でありながら、PSI-II上の擬似並列実行においてすらProlog版に優る速度が得られた。本稿では、このSATCHMOのKL1インプリメンテーションを中心に報告する。

2. SATCHMO

SATCHMOは、任意の一階述語論理式(の否定)をrange-restrictedな節集合に変換したものを対象とする。各節は、前件 → 後件 の形をしており、前件はtrueか原子論理式の連言、後件はfalseか原子論理式の選言である。以下では前件がtrueの節をtrue節、後件がfalseの節をfalse節と呼び、単に節と言うときは前件がtrueでもなく後件がfalseでもない節のことを指す。range-restrictedの条件とは、「後件中に変数が現れていれば、それらはすべて前件中にも現れていなければならない」というものである。SATCHMOの証明方式(逐次)を図1に示す。

ここで、次の例題を考えてみよう。

問題S1:

```
(C1)    p(X), s(X) ---> false.
(C2)    q(X), s(Y) ---> false.
(C3)    q(X) ---> s(g(X)).
(C4)    r(X) ---> s(X).
(C5)    p(X) ---> q(X) ; r(X).
(C6)    true ---> p(a) ; q(b).
```

(1) モデル候補の生成
　(a) true 節:
　　true → F_1 ; F_2 ; ⋯ ; F_n.
　から、今のモデル候補に F_i を追加(拡張)する。
　(F_1, F_2, ⋯, F_n はグランド)
　(b) ある節:
　　A_1, A_2, ⋯ ,A_m → C_1 ; C_2 ; ⋯ ; C_n.
　においてある代入のもとに、今のモデル候補が A_1, A_2, ⋯, A_m の全てを含み、C_1, C_2, ⋯, C_n のいずれも含まなければ、C_j を今のモデル候補に追加(拡張)する。
(2) モデル生成の成功(節集合は充足可能)
　全ての節:
　　A_1, A_2, ⋯ A_m → C_1 ; C_2 ; ⋯ C_n.
　においてある代入のもとに、今のモデル候補が A_1, A_2, ⋯, A_m の全てを含み、C_1, C_2, ⋯, C_n のいずかを含んでいれば、それはこの節集合の一つのモデルである。
(3) モデル生成の失敗
　ある false 節:
　　A_1, A_2, ⋯ A_m → false
　においてある代入のもとに、今のモデル候補が A_1, A_2, ⋯, A_m の全てを含むとき、それはこの節集合のモデルではない。
(4) モデル生成の再試行(backtrack)
　1) でモデル生成に失敗したとき、
　1) の F_i ／ C_j を除く残りの F_{i+1}, F_{i+2}, ⋯, F_n ／ C_{j+1}, C_{j+2}, ⋯, C_n の中からモデル拡張の候補を選びなおす。
(5) モデル生成の全失敗(節集合は充足不能)
　1) で F_1, F_2, ⋯, F_n ／ C_1, C_2, ⋯, C_n の全てのモデル拡張の候補が 3) のモデル生成の失敗を導くとき、この節集合にモデルはない。

図 1　SATCHMO の証明方式

C1 から C6 まで全て range-restricted な節である。何故ならば、C1,C2,C6 では後件に変数は現れないし、C3,C4,C5 で後件に現れる変数 X は前件にも現れている。

図 1 に従えば、まず C6 によって p(a) がモデル候補に加えられる。次に C5 によって q(a) が追加される。さらに C3 によって s(g(a)) が追加される。しかし C2 によってこのモデル候補 {p(a),q(a),s(g(a))} は失敗であることがわかる。そこで C5 で r(a) を選び直す。すると C4 によって s(a) が追加されるが、C1 によって今度のモデル候補 {p(a),r(a),s(a)} も失敗であることがわかる。

そこで C6 に戻って q(b) を選び直す。すると C3 によって s(g(b)) が追加される。しかし、またしても C2 によってモデル候補 {q(b),s(g(b))} が失敗であることがわかる。C5,C6 での選択は全て尽くしたから、モデル生成は全て失敗したことになる。即ち、問題 S1 は充足不能であることがわかる。

以下、特に断らない限り、単にモデルと言う場合は、生成過程の途上にあるモデル候補のことを指すことにする。

3. KL1 インプリメンテーション

SATCHMO を並列化しようとするとき、次の二つの可能性が考えられる。

(1) モデル生成において、モデルに加えるアトムの候補が複数ある場合に並列化 (OR 並列) できる。この場合、モデル生成の再試行 (backtrack) が並列実行で置き換えられる。

(2) 前件の充足を調べるステップにおいて、複数のアトムからなる前件に対するチェックを並列化 (AND 並列) できる。

しかし、簡単のためここでは図1の逐次的アルゴリズムを忠実に KL1 コーディングすることを考える。

3.1 問題点

並列性を如何に引き出すかをさておいたとしても、SATCHMO を KL1 でインプリメントしようとすると大きな困難に直面する (この問題は、Prolog や GHC のメタインタプリタ、Knuth-Bendix の完備化手続きなどを GHC や KL1 でインプリメントしたときの問題と全く同じである)。即ち、問題に現れる変数の扱いである。

Prolog でインプリメントするときには、問題に現れる変数を Prolog 変数で都合よく表現することができた。そこでは、var 述語、組み込みのユニフィケーションが便利に利用できた。厳密にいえばこのような方法はメタとオブジェクトの相違を曖昧にする好ましからぬ用法かもしれないが、プログラムの規模、効率の良さなど実利は確かに大きい。

しかし、KL1 ではこの手は通用しない。言語仕様上、原理的に不可能である。というのは、var 述語のように変数の非束縛を判定することは並列実行下では意味をなさないし、ヘッド側で行われるユニフィケーションは呼び側の変数を具体化しない一方向 (照合) に制限され、ボディ側で行われるユニフィケーションは失敗が許されないからである。

3.2 解決策

上述の問題点に関して、考えられる解決策には二通りある。

(1) 問題に現れる変数は KL1 の基底項で表現する。

(2) 問題に現れる変数をそのまま KL1 変数で表現する。

(1) はメタプログラミングの正道かもしれない。しかし、ユニフィケーション、サブスティテューション、リネーミングといった変数にまつわるあらゆる操作を全て詳細にプログラムする必要がある。そのプログラムの規模、複雑さはメインアルゴリズムの規模、複雑さに比べて無視できないものとなる。当然、そのままではオーダーを数桁も劣化させるようなオーバヘッドのために実行には耐えられない。これを改善する一つの手段は部分計算であろうが、残念なことに現在のところ KL1 において自己適用可能な部分計算が実現されていないため、メタプログラムの一つである部分計算が上述の場合と同種のオーバヘッドを伴い実用的でない。

(2) は (1) とは逆に、ユニフィケーション、サブスティテューション、リネーミングといった変数にまつわる操作をなるべくプログラムせずに、インプリメント言語の処理系に肩代わりさせようというアプローチである。ただし、KL1 では Prolog の場合のようにはうまくいかないので工夫が要る。

```
:- module satchmo_problem.
:- public model/1, nc/1, c/4.

model(M) :- true | M=[].
nc(NC) :- true | NC=6.

c(1,p(X),[],    R) :- true | R=cont.
c(1,s(X),[p(X)],R) :- true | R=false.        %  (C1)  p(X), s(X) ---> false.
c(2,q(X),[],    R) :- true | R=cont.
c(2,s(Y),[q(X)],R) :- true | R=false.        %  (C2)  q(X), s(Y) ---> false.
c(3,q(X),[],    R) :- true | R=[s(g(X))].    %  (C3)  q(X) ---> s(g(X)).
c(4,r(X),[],    R) :- true | R=[s(X)].       %  (C4)  r(X) ---> s(X).
c(5,p(X),[],    R) :- true | R=[q(X),r(X)].  %  (C5)  p(X) ---> q(X) ; r(X).
c(6,true,[],    R) :- true | R=[p(a),q(b)].  %  (C6)  true ---> p(a) ; q(b).
otherwise.
c(_,_,_,R) :- true | R=fail.
```

図2　KL1の節形式に変換された問題S1

SATCHMOの場合、好都合なことにフルユニフィケーションは必要ない。何故ならば、節をモデルにつきあわせて充足性を判定する際、変数は一方の節のみに現れ、他方のモデル側には現れないからである(range-restrictedな節集合から生成されるモデルは常にグランドアトムのみから成る)。即ち、変数の束縛は照合だけで行われるから、この操作はKL1のヘッドユニフィケーションで実現できることがわかる。残る問題は、一つの節の異なるアトムに出現する同名の変数への値の伝搬の方法であるが、一旦照合が済んだアトムはもはや変数を含まない(グランドである)ことを考え合わせると、これもKL1のヘッドユニフィケーションで実現できることがわかるのである。

具体的な例でこれを見てみよう。図2は、先の問題S1をKL1の節形式に変換したものである。c(N,P,GS,R)のNは問題の節の番号、Pは前件中のアトム、GSは前件中でPより左に現れるアトムの列、Rはこの節の呼び出しが成功した場合に返される値である。ここで、C1は二本のKL1節で表現されていることに注意しよう。1番めの節では、C1の最初のアトムp(X)がモデル中の要素(グランドアトム)と照合をとられる。この照合が成功すれば、呼び手(SATCHMOインタプリタ)は対応するp(X)のインスタンスを保持しておき、二つめのアトムs(X)の照合に進む。このとき、呼び手はp(X)のインスタンスを第2引き数としてcの2番めの節を呼ぶことになり、ここでs(X)のXが先のp(X)のXと同じ値をもつことになる。

3.3 インタプリタ

図3にKL1コーディングされたSATCHMOインタプリタを示す。do(A)は、問題に指定された初期モデルMでfalseプロセスを起動し、結果Aを得る。

false(M,A)は、satisfy_clausesを起動して、問題として与えられたNC本の節をモデルMの下で充足させることを試み、その結果をAに与える。Aの値は2通りで、節集合が充足可能な場合は充足した一つのモデルをsat(M1)の形で返し、充足不能の場合は調べた全てのモデルのリストをunsat(Ms)の形で返す。

satisfy_clauses(Cn,NC,M,A2,A)は、逐次的に1番めからNC番めまでの節の充足可能性を順に調べ、その結果をAに返す。Cnは現在調べている節の番号で、A2はそれ以前の節の結果である。各節からの結果はcl_sat、sat(M1)、unsat(Ms)の3通りある。

cl_satは、現在調べているCn節がMで充足されたことを示し、この場合には次の節が調べられる。現在調べているCn節からsat(M1)かunsat(Ms)が返されたときは、以後の節を調べることなくその結果をsatisfy_clauses全体の結果としてAに返す。

```
:- module satchmo.
:- public do/1, false/2.

do(A) :- true | satchmo_problem:model(M), false(M,A).

false(M,A) :- true |
    satchmo_problem:nc(NC),
    satisfy_clauses(0,NC,M,cl_sat,A).

satisfy_clauses(Cn,NC,M,A2,A) :- Cn < NC |
    Cn1 := Cn + 1,
    satisfy_ante(Cn1,[],[true|M],M,A2,A1),
    satisfy_clauses(Cn1,NC,M,A1,A).
satisfy_clauses(NC,NC,_,sat(M1),  A) :- true | A = sat(M1).
satisfy_clauses(NC,NC,M,cl_sat,   A) :- true | A = sat(M).
satisfy_clauses(NC,NC,M,unsat(Ms),A) :- true | A = unsat(Ms).

satisfy_ante(Cn,GS,[P|M2],M,cl_sat,A) :- true |
    satchmo_problem:c(Cn,P,GS,R),
    satisfy_ante1(Cn,R,P,GS,M2,M,A).
satisfy_ante(Cn,_,[],_,cl_sat,A) :- true | A=cl_sat.
otherwise.
satisfy_ante(_,_,_,_,A1,A) :- true | A=A1.

satisfy_ante1(Cn,fail,P,GS,M2,M,A) :- true |
    satisfy_ante(Cn,GS,M2,M,cl_sat,A).
satisfy_ante1(Cn,cont,P,GS,M2,M,A) :- true |
    satisfy_ante(Cn,[P|GS],M,M,cl_sat,A1),
    satisfy_ante(Cn,GS,M2,M,A1,A).
satisfy_ante1(Cn,false,P,GS,M2,M,A) :- true | A=unsat(M).
satisfy_ante1(Cn,R,P,GS,M2,M,A) :- R=[_|_] |
    satisfy_cnsq(R,R,M,A1),
    satisfy_ante(Cn,GS,M2,M,A1,A).

satisfy_cnsq([Fact|R2],R,M,A) :- true | check_cnsq(Fact,R2,R,M,M,A).
satisfy_cnsq([],R,M,A) :- true | extend_model(R,M,A).

check_cnsq(Fact,R2,R,[Fact|M2],M,A) :- true | A=cl_sat.
check_cnsq(Fact,R2,R,[],M,A) :- true | satisfy_cnsq(R2,R,M,A).
otherwise.
check_cnsq(Fact,R2,R,[_|M2],M,A) :- true | check_cnsq(Fact,R2,R,M2,M,A).

extend_model([Fact|R2],M,A) :- true |
    false([Fact|M],A1),
    extend_model(R2,M,A2),
    both(A1,A2,A).
extend_model([],M,A) :- true | A=unsat([]).

both(unsat(M1),unsat(M2),A) :- true | A=unsat([M1|M2]).
both(A1,_,A) :- A1=sat(_) | A=A1.
both(_,A2,A) :- A2=sat(_) | A=A2.
```

図3　KL1によるSATCHMOインタプリタ

1番めからNC番めまでの全ての節がcl_satを返した場合、問題の節集合はMで充足可能とわかるので、sat(M)を返す。

Cn節からsat(M1)が返されるのは、その節がモデルMの下で充足されていなくて、そのモデルMに可能な拡張を施して得られたある一つのモデルM1について節集合の充足可能がわかった場合(M1に対して起動された最下段のあるfalseプロセスがsat(M1)を返す場合)である。

unsat(Ms)が返されるのは2通りあって、Cn節がfalse節で、前件がMで充足された場合(Ms=M)、およびCn節がfalse節でなく、モデルMに可能な拡張を施して得ら

れた全てのモデルMsについて節集合の充足不能がわかった場合(Msに対して起動された最下段の各falseプロセスがunsat(Mi)を返す場合)である。

satisfy_ante(Cn,GS,[P|M2],M,A2,A)は、Cn節の前件中に出現するアトムを左から右へ順に調べ、Cn節の充足可能性の判定結果をAに返す。GSは、現時点までに既にMのいずれかの要素と照合に成功したアトムのリストである。satisfy_anteでは、今調べようとしているアトムとモデルMとの照合を行うため、ワーキング(探索中の)モデルリスト[P|M2]から先頭の要素Pを取り出し、Cn、PおよびGSに照合するKL1節cを呼び出す。cの呼び出し側のPおよびGSには変数は含まれないことに注意。また、このときcの定義節側に含まれる変数に対して自動的に新変数が確保される。

次にsatisfy_ante1(Cn,R,P,GS,M2,M,A)は、cの呼び出しの結果であるRの値に応じて以下の処理を進める。

(1) R=failの場合(現在のアトムが今のモデル要素Pで充足不能): satisfy_anteを再帰的に起動して残りのモデルM2と照合を試みる。

(2) R=contの場合:現在のアトムとモデル要素Pとの照合によって生じた変数束縛情報を以後の照合に伝搬させるため、GSに今のモデル要素Pを追加して、前件中の次のアトムに対するsatisfy_anteを起動する。それと同時に、残りのモデルM2に対して新たなsatisfy_anteを起動する。

(3) R=falseの場合(前件の最後のアトムが今のモデル要素Pで充足可能): unsat(M)を返す。

(4) R=[_|_](後件を表すアトムのリスト)の場合(前件の最後のアトムが今のモデル要素Pで充足可能): satisfy_cnsqを起動して後件の充足性を調べると同時に、残りのモデルM2に対して新たなsatisfy_anteを起動する。

satisfy_cnsqは、後件がfalseでない節でsatisfy_anteで前件がMの下で充足されたものについて、Mで後件が充足されているかどうかをcheck_cnsqによって調べる。充足されている場合は、その節全体がMで充足されているので結果cl_satを返す。そうでない場合は、extend_modelによって後件部の選言に現れる各アトムに対してそのアトムを一つだけMに追加したモデルを作り、拡張されたそれぞれのモデルについてfalseの再帰プロセスを起動する。これらのfalseプロセスの結果はbothによって合成されて親のfalseプロセスの結果を決定する。即ち、拡張されたそれぞれのモデルの下で、子供のfalseプロセスの結果が全て充足不能ならば、親のfalseプロセスはunsat(Ms)を返す。他方、拡張されたあるモデルの下で、子供のfalseプロセスの結果が充足可能ならば、親のfalseプロセスはsat(M1)を返す。

4. 性能評価

表1にPTTP、SATCHMO (Prolog版)、SATCHMO (KL1版)の性能比較を示す。S1～S3はいずれも有限領域を扱う問題であるが、これらについてはPTTPに比べてSATCHMO (Prolog版)がはるかに高速であることがわかる(問題S3についてはPTTPでは30分を経過しても解が得られなかった)。一方、SATCHMOのKL1版はProlog版に比べて3～4倍高速であることがわかる。ただし、問題S2についてはProlog版の方が2倍弱速い。これは、Prolog版においては図1に示したボトムアップなモデル生成手続きに加えて、falseゴールから逆向きに推論を行うことによって不要なモデル生成を回避するトップダウンの評価機構が組み込まれており、この効果が現れているためであろうと考えられる。

表1 性能評価

問題	S1	S2*	S3*
PTTP†	86msec	24sec	?(>30min)
SATCHMO in Prolog†	16msec	68msec	6.3sec
SATCHMO in KL1‡	5.4msec	107msec	1.5sec
(リダクション数)	(390)	(8,495)	(118,237)

† (Sicstus Prolog V0.6 on SUN3/260)
‡ (pseudo-Multi-PSI single-PE on PSI-II)
* 問題S2, S3は付録を参照

ここで示したKL1版SATCHMOはインタプリタ方式によるものであったが、問題の節集合とその解法をKL1プログラムに直接ハンドコンパイルする方式[Fuchi 90]も可能である。我々のインタプリタ方式をこれと比較した場合、約3倍ほどの定数オーバヘッドに収まっていることがわかった。インタプリタ方式のオーバヘッドとしては驚くほど小さいものと思えるが、両方式ともコントロールは本質的に同じである他、我々の方式でも既に問題の節をKL1節に変換していることを考え合わせれば、妥当な数字であるといえよう。また、この程度のオーバヘッドであれば部分計算技法を適用することによって直接ハンドコンパイル方式に匹敵する性能を得ることが可能であろう。

5. 議論

PTTPや、レゾリューションに基づくフルファーストオーダーの定理証明システムにはオカーチェック付きユニフィケーションが必要である。

オカーチェック付きユニフィケーションをKL1でプログラムするとき、論理式中の変数を例えばvar(x)のような基底項を用いて表す。また、ユニフィケーションゴールunify(X,Y,S)はXとYをユニファイした結果をSに返すようなものとする。Sの値はユニファイに失敗した場合はfailで、成功した場合は変数の束縛のリストである。このようにして書いたunify(X,Y,S)の実行時間はKL1の組み込みユニフィケーションX=Yの場合の150～200倍ほどであった。このようにオーバヘッドが大きいとやはり実用には耐え難いといわざるを得ない。従って、現在のKL1の機能ではPTTPやレゾリューション方式よりもSATCHMOの方が実現が容易な証明方式であると思われる。

図3のSATCHMOインタプリタを基本型として、幾つかの拡張が考えられる。

(1) OR並列

まず、選言を後件にもつ節によって拡張されるモデルを同時に処理するOR並列処理を考えることができる。この場合、資源はあくまで有限であること、分岐の数が大きいような問題ではたちまち資源が食い潰されることなどを考慮して、資源数に見合うだけOR並列展開し、それ以後はAND並列処理を行う(restricted-OR)など、何らかの制限戦略が必要になるであろう。

(2) クローズインデクシング

問題の節集合が大きくなるにつれ、節の選択に要するコストが増大する。図3に示したインタプリタでは、falseの再帰プロセスが起動される度にいつもNC本の節全部を調べるが、モデルを拡張した節の後件と他の節の前件のコネクションを考えれ

ば、この再帰プロセスで調べるべき節の数を絞ることができる。このような修正は構文的解析に基づいて比較的容易に行うことができる。また、PrologやKL1処理系で行っているようなクローズインデクシングも自動的に利用できる。

(3) モデル拡張ステップの縮約

また、今のモデルに対して充足されておらずモデルの拡張を要する節が複数ある場合、その全ての拡張の組み合わせを1ステップで計算することにより、ステップ数を縮めることが考えられる。例えば、次の問題を考えよう。

```
(C0)   true ---> p(a).
(C1)   p(X) ---> r1(X).
   ...
(Cn)   p(X) ---> rn(X).
(Cx)   r1(X), r2(X), ..., rn(X) ---> false.
```

この問題に図1の手続きを適用すると、n+2ステップ要する。しかし、C0でモデル{p(a)}が生成された後、C1からCnまでがモデル拡張の候補となっていることがわかる。そこで、r1(a)からrn(a)までを一度にモデルに加えることができて、この場合3ステップで終了する。このようにして、一つのモデル生成の失敗が即座にわかる可能性がある。

しかし、一般には後件が幾つかのアトムの選言であるから、モデル拡張の一つの候補はそれぞれの節の選言から一つずつ選んだものとなり、この組み合わせのための計算量の増大がステップ数の短縮に見合わない程大きい場合もあり得る。

(4) モデルの共有

他に、SATCHMOのモデル拡張のステップにおいて、選ばれた節が実はホーン節で、生成されるモデル要素はすべてのモデルに共通である場合がある。この場合、共通に参照できるモデル部分は、選言を後件にもつ節によって拡張される他のモデルとは別の領域に置くのがよい。こうして領域を節約することが可能である。

(5) 消去戦略その他

さらに、レゾリューションにおけるファクタリングやサブサンプションに相当する消去戦略を導入して計算の重複を減らすことが考えられる。その際、余分な記憶領域と照合の手間が導入されるが、これが計算の重複の削減に見合うかどうかは問題に大きく依存している。実際的なプルーバーはこれらの戦略をパラメタ化されたオプションとして装備し、これらのパラメタの選択は結局ユーザの試行錯誤の上決定されるものと考えることになるであろう。(適正パラメタの抽出にニューロネットの手法に基づく学習を適用するという研究[Ertel 90]がある。)

6. 結論

SATCHMOがユニフィケーション(+オカーチェック)を必要としないことはKL1でインプリメントするのに極めて好都合であった。問題の中の変数はKL1変数で表現し、その束縛はKL1のヘッドユニフィケーション(即ち照合)で行うことができるからである。また、節のコピー(異なる節インスタンスを作ること)とそれに付随する新変数の導入(名前替え)は、KL1節の呼びだしの機構によって自動的に行われる。この利点を活用するためには問題の節を適切なKL1の節集合に変換する必要があるが、この変換は極めて機械的にかつ安価に行えるので障害にはならない。

SATCHMOは、有限領域については極めて効率のよい証明システムであり、ここで開発したKL1プログラミング技法は、データベース、TMS等の有限領域を扱う様々

な応用の実現技術として有用である。一方、無限領域についてはdom述語を導入して対処することになるが、これはそのままでは全くのブルートフォース的解決策である。ある種のヒューリスティックスを入れて探索空間を適正に抑えない限り、たちまち組み合わせ的爆発のために資源を消費し尽くしてしまい、証明結果を得るまでには至らない。一つの解決策としてノングランドのモデルを許すことにし、レゾリューション法におけるサブサンプションに相当する消去戦略を導入することを現在試みているところである。

今後、レゾリューション法、コネクション法などについても検討を進めて行く予定であるが、そこでもやはり証明対象となる論理式に現れる変数の扱い方がキーになるであろう。即ち、高速な定理証明システムの実現可能性はSATCHMOのようにフルユニフィケーションからうまく逃れ得るか、あるいは効率のよいフルユニフィケーションがKL1コーディングで実現され得るかにかかっていると思われる。しかし、もしKL1処理系側で基底項表現された変数とそれを含む項の間の効率のよいフルユニフィケーションが組み込みないしユーティリティとしてサポートされれば、この問題は一挙に解決するに違いない。

謝辞

本研究は、PTTPならびにSATCHMOのPrologにおける成果に端を発している。古川次長にはこれらのシステムの紹介とKL1版インタプリタ作成の機会を与えて頂き、様々な御助言を頂いた。渕所長には本研究の当初から数々の貴重な示唆を頂いた。実際、ここで報告したKL1版SATCHMOインタプリタの作成技法は、渕所長の開発されたSATCHMO問題および解法をESPやKL1プログラムへハンドコンパイルする技法に負う所が多い。また、藤田正幸氏には他の定理証明技法との比較検討等に関する有意義な議論を頂いた。最後に御指導、御討論頂いた上記の方々に謝意を表します。

参考文献

[Bibel 86] Wolfgang Bibel, Automated Theorem Proving, Vieweg, 1986.

[Wos 88] Larry Wos, Automated Reasoning -- 33 Basic Research Problems -, Prentice-Hall, 1988.

[Stickel 88] Mark E. Stickel, A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, in *Journal of Automated Reasoning* **4** pp.353-380, 1988.

[Bry 88] Rainer manthey and François Bry, SATCHMO: a theorem prover implemented in Prolog, in *Proc. of CADE 88, Argonne, illinois,* 1988.

[Fuchi 90] Kazuhiro Fuchi, experimental programs in ESP/KL1 for a SATCHMO prover, 1990.

[Ertel 90] Christian Suttner and Wolfgang Ertel, Using Connectionist Networks for Guiding the Search of a Theorem, TUM-19011 SFB-Bericht Nr.342/8/90 A, Institute fur Informatik, Technische Universitat Munchen, 1990.

付録: SATCHMO(KL1 版) による問題記述 (S2, S3)

```
%       S2 : Shubert's Steamroller problem

:- module satchmo_problem.  :- public model/1, nc/1, c/4.
model(M) :- true | M =
    [likes(s,i(s)),likes(c,h(c)),likes(b,c),                    %  likes(s,i(s)).
                                                                %  likes(c,h(c)).
                                                                %  likes(b,c).
     smaller(f,w),smaller(b,f),smaller(s,b),smaller(c,b),       %  smaller(f,w).
                                                                %  smaller(b,f).
                                                                %  smaller(s,b).
                                                                %  smaller(c,b).
     plant(g),plant(i(s)),plant(h(c)),                          %  plant(g).
                                                                %  plant(i(s)).
                                                                %  plant(h(c)).
     animal(w),animal(f),animal(b),animal(s),animal(c)].        %  animal(w).
                                                                %  animal(f).
                                                                %  animal(b).
                                                                %  animal(s).
                                                                %  animal(c).
nc(NC) :- true | NC=6.
c(1,likes(w,f),[],R) :- true | R=false.     %  likes(w,f) ---> false.
c(2,likes(w,g),[],R) :- true | R=false.     %  likes(w,g) ---> false.
c(3,likes(b,s),[],R) :- true | R=false.     %  likes(b,s) ---> false.
c(4,animal(X), [],R) :- true | R=cont.                          %  animal(X),
c(4,animal(Y), [animal(X)],R) :- true | R=cont.                 %  animal(Y),
c(4,likes(X,Y),[animal(Y),animal(X)],R) :- true | R=cont.       %  likes(X,Y),
c(4,likes(Y,g),[likes(X,Y),animal(Y),animal(X)],R) :-           %  likes(Y,g)
        true | R=false.                                         %  ---> false.
c(5,animal(X),    [],R) :- true | R=cont.                       %  animal(X),
c(5,animal(Y),    [animal(X)],R) :- true | R=cont.              %  animal(Y),
c(5,smaller(Y,X),[animal(Y),animal(X)],R) :- true | R=cont.     %  smaller(Y,X)
c(5,plant(W),     [smaller(Y,X),animal(Y),animal(X)],R) :-      %  plant(W),
        true | R=cont.
c(5,likes(Y,W),   [plant(W),smaller(Y,X),animal(Y),animal(X)],R) :-
        true | R=cont.                                          %  likes(Y,W),
c(5,plant(Z),     [likes(Y,W),plant(W),smaller(Y,X),animal(Y),animal(X)],R) :-
        true | R=[likes(X,Y),likes(X,Z)].                       %  plant(Z)
                                        %  ---> likes(X,Y) ; likes(X,Z).
otherwise.
c(_,_,_,R) :- true | R=fail.
```

```
%       S3 : Pelletier & Rudnicki's problem

:- module satchmo_problem.  :- public model/1, nc/1, c/4.
model(M) :- true | M = [dom(a),dom(b),dom(c),dom(d)].   %  dom(a).
                                                        %  dom(b).
                                                        %  dom(c).
                                                        %  dom(d).
nc(NC) :- true | NC = 6.
c(1,p(a,b),[],R) :- true | R=false.           %  p(a,b) ---> false.
c(2,q(c,d),[],R) :- true | R=false.           %  q(c,d) ---> false.
c(3,p(X,Y),[],R) :- true | R=cont.            %  p(X,Y),
c(3,p(Y,Z),[p(X,Y)],R) :- true | R=[p(X,Z)]. %          p(Y,Z) ---> p(X,Z).
c(4,q(X,Y),[],R) :- true | R=cont.            %  q(X,Y),
c(4,q(Y,Z),[q(X,Y)],R) :- true | R=[q(X,Z)]. %          q(Y,Z) ---> q(X,Z).
c(5,p(X,Y),[],R) :- true | R=[p(Y,X)].        %  p(X,Y) ---> p(Y,X).
c(6,dom(X),[],R) :- true | R=cont.            %  dom(X),
c(6,dom(Y),[dom(X)],R) :- true |              %          dom(Y)
        R=[p(X,Y),q(X,Y)].                    %           ---> p(X,Y) ; q(X,Y).
otherwise.
c(_,_,_,R) :- true | R=fail.
```