TM-0976

メタライブラリ・マニュアル

越村　三幸（東芝）、藤田　博（三菱）、
長谷川　隆三

November, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

メタライブラリ・マニュアル

越村 三幸* 藤田 博† 長谷川 隆三
(財) 新世代コンピュータ技術開発機構

1 はじめに

メタライブラリは並列論理型言語 KL1 でメタプログラミングをする人のための共通ライブラリを提供する. 本マニュアルはメタライブラリの使い方を解説するものである.

KL1 でメタ機能を実現する際, オブジェクトレベルの変数をどのように表現するのかが問題となる. 逐次論理型言語 Prolog のメタ述語 var/1 のような述語は KL1 では健全な意味をもたない. したがってオブジェクトレベルの変数をメタレベルの変数を用いて表現し, var/1 により巧妙に識別する手段は利用できない. このような理由により, オブジェクトレベルの変数はメタレベルの特殊な基底項として表現するのが KL1 でメタプログラミングをする際の標準的な手段となる.

オブジェクトレベルの変数を基底項表現した場合, メタレベルでオブジェクトレベルの変数の管理をしなければならなくなる. この点で KL1 でのメタプログラミングは Prolog でのメタプログラミングに比べてプログラマの負担が著しく大きい. メタライブラリはこのようなメタプログラマの煩わしさを軽減するために作られたプログラム群である.

2 オブジェクトの表現法

オブジェクトは 表現 (**Exp**) と 環境 (**Env**) の対で表現される. 以下の表現と環境の対 (**Exp**,**Env**) は同じオブジェクト f(a) を表現している.

- **Exp** $= f(a)$, **Env** $= \{\}$
- **Exp** $= f(X)$, **Env** $= \{X \rightarrow a\}$
- **Exp** $= Y$, **Env** $= \{Y \rightarrow f(X), X \rightarrow a\}$

2.1 表現の形式

表現は KL1 のデータがそのまま使える. ただし, ベクタの第一要素はアトムでなければならない. オブジェクトレベルの変数は {N} (N は変数番号で環境のインデックス) と表現される.

2.2 環境の形式

環境は, ベクタで実現されている. 各要素が変数の値を表している. 第一要素は reserve されていてる.

3 メタライブラリの機能

メタライブラリは, 次の 3 つのグループに分けられる.

1. メタ述語 メタライブラリの主要部分. 述語として提供される.
2. データベース オブジェクトのデータベース. これは KL1 のプロセスとして実現されている.

*現在, 日本ビジネスオートメーション (株) 所属
†現在, 三菱電機 (株) 中央研究所 所属

3. オブジェクトの入出力をするためのライブラリ.

【注意】メタライブラリプログラムはエラー処理を行っていないため,不正な使い方をするとフェイルする場合がある.

【参考】メタライブラリ機能を用いたい場合,プログラム中に meta# 呼び出し名 と記述する. この記述はユーザ定義マクロ機能[1]によりコンパイルの前処理で展開される.

3.1 メタ述語

meta#match(Pattern,Target, Env,^NEnv)

meta#oneway_unify(Pattern,Target, Env,^NEnv)

Target が Pattern のインスタンスである時, Pattern 側の変数から Target 側のオブジェクトにポインタを張る. 新しくできた環境を NEnv とユニファイする. インスタンスでない時は,NEnv を fail とユニファイする.

【参考】oneway_unify は GHC のガードユニフィケーションに相当するものである. この時, Pattern はガード側 Target は呼びだしゴール側に相当する.

【注意 1】match は Target が基底であるとを仮定している. Target が基底でないときの健全性は保証されない.

【注意 2】oneway_unify では Pattern と Target がオブジェクトを共有していない (共有変数がない) ことを仮定している. そうでない時の健全性は保証されない.

meta#unify_oc(X,Y, Env,^NEnv)

meta#unify(X,Y, Env,^NEnv)

表現 X,Y は環境 NEnv の下で同じオブジェクトを表現する. X,Y がユニファイ可能でないとき,NEnv を fail とユニファイする.

【参考】unify_oc はオカーチェック付ユニファイである.

【注意 1】unify_oc,unify 共に循環構造のあるオブジェクトの場合には停止しないことがある.

【注意 2】unify は循環構造のオブジェクトを作る場合がある.

meta#copy_term(X,^NX, Env,^NEnv)

オブジェクト (X,Env) の variant オブジェクト (NX,NEnv) を作る.

【参考】これは環境 Env の GC にも使える.

meta#shallow(X, Env,^NEnv)

オブジェクト表現 X からたどれる Env 内のポインタを shallowing する.

meta#freeze(X,^FX, Env)

オブジェクト (X,Env) をフリーズし FX とユニファイする.

【参考】変数 {N}(N は変数番号) は {N,freeze} とフリーズされる.

[1] この機能は PIMOS 2.1 版より使用できる.

meta#melt(FX,^X)

freeze の逆変換.

meta#create_env(^Env, Size)

meta#create_env(^Env, Size,Top)

Size で指定した大きさの環境 Env を作る. Top は変数番号の初期値を示す. デフォルトは 1.

【注意】 $1 \leq \mathrm{Top} \leq \mathrm{Size}$ でなければならない.

【参考】変数番号は Top から昇順に割り振られていく.

meta#fresh_var(Env, VarNEnv)

新しい変数 X を作り環境 Env に登録する. 新しい環境を NEnv とし VarEnv と {X,NEnv} をユニファイする. 変数を登録できない時は {} をユニファイする.

meta#equal(X,Y, Env,^YN)

表現 X,Y が環境 Env の下で全く同じオブジェクトを表現している時は YN と yes , そうでない時は YN と no のユニフィケーションをする.

meta#unbound(X, Env,^YN)

表現 X が環境 Env の下で (オブジェクトレベルの) 変数の時は YN と yes , そうでない時は YN と no のユニフィケーションをする.

meta#is_type(X,Env, ^Type)

X のデータ型を検査する. X の型により Type には次のものがユニファイされる.

整数 `integer(Integer)`

アトム `atom(Atom)`

ストリング `string(String)`

リスト `list(List)`

ベクタ `vector(Vector)`

3.2 オブジェクトのデータベース

これは, オブジェクトのデータベースである. キーで検索・参照ができる.

3.2.1 生成

meta#database(S)

S はデータベースへのストリーム

3.2.2 プロトコル

do(S)

S は親のストリームと同じプロトコルのメッセージを要素とするストリームである. S 内のメッセージは連続して処理される. メッセージ間の連続性を保証したいときに do/1 を用いる必要がある.

【参考】 `S = [do(S1)|S2]` は `append(S1,S2, S)` と同じ意味である.

put(Key, Exp,Env)

キー Key でオブジェクト (Exp,Env) を登録する.

get(Key, ^ExpEnv)

get(Key, Env, ^ExpEnv)

キー Key で登録されているオブジェクトを環境 Env に追加し, ExpEnv と {Exp,NEnv} をユニファイする. ここで Exp はオブジェクトの表現, NEnv はオブジェクトを追加した環境である. オブジェクトが登録されていない場合は ExpEnv は {} とユニファイされる. Env の指定がないときは新たに環境を生成する.

remove(Key)

キーで登録されているオブジェクトを抹消する.

compile(ModuleID, ^CompiledCode)

データベースをコンパイルする. CompiledCode は KL1 コードの列である. CompiledCode をコンパイルすることにより述語としてデータベースを使用することができるようになる. 生成される述語は get/2 と get/3 である.

【注意】このメッセージを利用する時は, Key は基底項でなければならない.

3.3 入出力のための項変換

KL1 の Wrap された項をメタライブラリのオブジェクトに変換する述語である.

【参考】これらはオブジェクトの入出力に用いる.

meta#get_object(WrappedTerm,Env, ^ExpEnvVT)

meta#get_object(WrappedTerm, ^ExpNEnv)

WrappedTerm をオブジェクト (Exp,NEnv) に変換し, ExpEnvVT を {Exp,NEnv,VT} とユニファイする. ここで VT は変数の印字名に関する情報であり, { 変数名, 変数表現 } のリストである. Env が指定された場合, Env にオブジェクトを登録する.

meta#get_kl1_term(Exp,Env,VT, ^WrappedTerm)

meta#get_kl1_term(Exp,Env, ^WrappedTerm)

オブジェクト (Exp,Env) を変数情報 VT に基づいて KL1 の Wrap された項 WrappedTerm に変換する. VT を指定しない場合は [] を指定したものとみなされる.

A メタライブラリの運用

メタライブラリを使ったプログラムのコンパイル / 実行 / デバッグは PIMOS[3] の提供する機能をそのまま利用して行う. プログラム中に書かれた meta#XXX はコンパイルに先だってマクロ展開する必要がある. しかし, PIMOS 第2版では KL1 コンパイラはユーザ定義マクロ機能を提供していないので, meta#XXX 入りのソースはそのままではコンパイルできない. そこでマクロ展開とコンパイルを行う簡易コンパイラを用意した. ユーザ定義マクロ機能が提供されるまで暫定的に利用してもらいたい.

A.1 インストール

メタライブラリは KL1 プログラムの集まりとして提供される．それらはディレクトリに木状に格納されている．そのトップディレクトリが次のようなファイルとディレクトリからなる．このディレクトリを以下 MetaDir と呼ぶことにする．

1. ファイル：

Install	メタライブラリをインストールするためのリスナのコマンドファイル
Startup	メタライブラリを PIMOS 上にロードするためのリスナのコマンドファイル
Compile	簡易コンパイラを起動するためのリスナのコマンドファイル

2. ディレクトリ：

Copy	構造体コピー方式のメタライブラリプログラム
Share	構造体共有方式のメタライブラリプログラム
Common	その他のメタライブラリプログラム
Macro	マクロファイルとマクロ展開プログラム
Sample	サンプルプログラム

メタライブラリは以下の手順でインストールする．

1. PIMOS にログインする．
2. リスナを起動する．
3. リスナでカレントディレクトリを MetaDir にし，Install ファイルを take する．

```
?- cd MetaDir.
?- take("Install").
```

A.2 ロード

メタライブラリをロードするには以下のようにする．

1. PIMOS にログインする．
2. リスナを起動する．
3. リスナでカレントディレクトリを MetaDir にし，Startup ファイルを take する．

```
?- cd MetaDir.
?- take("Startup").
```

以降メタライブラリ機能を使用したプログラムがコンパイルできるようになる．

A.3 簡易コンパイラ

簡易コンパイラはコンパイルに先だって，マクロ展開をする[2]．マクロ展開後のソースは KL1 コンパイラに渡される．簡易コンパイラは以下のようにして起動される．

- リスナでカレントディレクトリを MetaDir にし，Compile ファイルを take する．

```
?- cd MetaDir.
?- take("Compile").
```

[2]PIMOS 2.1 版よりユーザ定義マクロが利用できるようになるので簡易コンパイラは不要になる．

Table 1: 簡易コンパイラコマンド

ディレクトリコマンド	
cd(DirNameStr)	カレントディレクトリの変更
pwd	カレントディレクトリの表示
ls	ファイルの一覧表示
ls(WildCardStr)	WildCardStr にマッチするファイルの一覧表示
コンパイラコマンド	
load_macro(MacroFileNameStr)	マクロファイルのロード
meta_compile(ListOfKL1FileNames)	KL1 プログラムのコンパイル
その他	
halt	簡易コンパイラの終了

B メタライブラリを使用したプログラム例

メタライブラリを使用した KL1 プログラムの例として, Prolog インタプリタと Model Elimination Theorm Prover についてプログラムのソースリストと起動例を載せる.

B.1 OR 並列 AND 逐次 Prolog インタプリタ

全解探索型 Prolog インタプリタを示す.

solve がインタプリタ本体である. solveAnd は and 関係にあるゴール列を逐次的に解いていく. solveOr は候補節が複数あった場合, その数の solveAnd プロセスを生成する. Prolog 節は, KL1 述語 ('$PrologDataBase':get/3) を呼び出すことにより参照することができる. したがって, consult することは KL1 プログラムを生成しそれをコンパイルすることに相当する. consult を実現しているのが consult である. consult はオブジェクトのデータベースを利用して, KL1 プログラムを生成している.

B.1.1 インタプリタの核部分

```
%
%     solveAnd(Goals,GoalsStack, Environment, ^Solution)
%
solveAnd([],[], Env, Sol) :- true | Sol = [Env].
solveAnd([],[Gs|Gss], Env, Sol) :- true | solveAnd(Gs,Gss, Env, Sol).
solveAnd([G|Gs],Gss, Env, Sol) :- true |
    clauses(G,Env, Clauses),
    solveOr(Clauses,G,Gs,Gss, Sol).

%
%     solveOr(Clausees, Goal,Goals.GoalsStack, ^Soution)
%
solveOr([{C,Env}|Cs], G,Gs,Gss, Sol) :- C = (H :- B) |
    Sol = {Sol1,Sol2},
    meta#unify(H,G, Env,Env1),
    expand(Env1, B,Gs,Gss, Sol1),
    solveOr(Cs, G,Gs,Gss, Sol2).
solveOr([], _,_,_, Sol) :- true | Sol = [].

%
```

```
%     expand(Environment, BodyGoals,Goals,GoalsStack, ^Solution)
%
expand(fail, _,_,_, Sol) :- true | Sol = [].
otherwise.
expand(Env, B,Gs,Gss, Sol) :- true | solveAnd(B,[Gs|Gss], Env, Sol).

%
%     clauses(Goal,Environment, ^Clauses)
%
clauses(G,Env, CLs) :- vector(G, Size), vector_element(G,0,F) |
    Arity := Size-1, clauses1({F,Arity},Env, CLs).
clauses(G,Env, CLs) :- atom(G) | clauses1({G,0},Env, CLs).

clauses1(FA,Env, CLs) :- true | clauses1(FA,0,Env, CLs).

clauses1(FA,M,Env, CLs) :- true |
    '$PrologDataBase':get({FA,M},Env, ExpEnv),
    clauses1Decide(ExpEnv, FA,M,Env, CLs).

clauses1Decide({}, _,_,_,CLs) :- true | CLs = [].
clauses1Decide(ExpEnv, FA,M,Env, CLs) :- ExpEnv = {_,_} |
    CLs = [ExpEnv|CLs1],
    clauses1(FA,^(M+1),Env, CLs1).
```

インタプリタの核部分は solveAnd/4 と solveOr/5 の二つである．Prolog 節の検索は clauses/3 が行う．

solveAnd/4 ゴールの列 Goals とゴール列のスタック GoalsStack とその環境 Environment を受け取り解 Solution を計算する．ゴール列を先頭から順に解いていく (第二, 三節)．ゴール列が空になったら解が見つかったことになるのでその時の環境を返す (第一節)．

solveOr/5 ゴール (G) と節のヘッド (H) をユニファイ (meta#unify(H,G, Env,Env1)) し，成功すればその時の環境でボディ部を解く (expand/3 第二節)．

clauses/3 ゴール G の候補節のリストを CLs に返す．

B.1.2 全ソースコード

インタプリタの核部分に算術演算の組込み述語を加え，トップレベルコマンドとしてディレクトリ関係コマンドと consult 機能を加えたプログラムを掲載する．

```
:- module prolog.
:- public go/0,go/1,loopDynamic/5.
:- with_macro pimos.

go :- true | go([putf(" Pure Prolog Interpreter ^n",[])]).

go(Do) :- true |
    shoen:raise(pimos_tag#shell,get_std_inter, Dev),
    Dev = [do(Do)|Dev1],
    loop(Dev1).

loop(Dev) :- true |
    loop(Dev,[], DB,[]),
%    meta#database(DB).
    meta::database:go(DB).
```

```
:- local_implicit io:oldnew, db:oldnew.
%
%   Top Loop
%
loop --> true |
    &io <<= [prompt(" ??- "), getwt(WT)],
    loop(WT).

loop(abnormal(ErrorInfo)) --> true |
    &io <<= [print_error(ErrorInfo)], loop.
loop(normal(empty)) --> true | loop.
loop(normal(wrap#halt)) --> true |
    &io <<= [putf( " End of Execution~n",[])].
loop(normal(wrap#cd(Dir))) --> true |
    {{cd(&io, Dir)}}, loop.
loop(normal(wrap#pwd)) --> true |
    {{pwd(&io)}}, loop.
loop(normal(wrap#ls)) --> true |
    {{ls(&io)}}, loop.
loop(normal(wrap#ls(WildCard))) --> true |
    {{ls(&io, WildCard)}}, loop.
loop(normal(wrap#meta_compile(Files))) --> true |
    {{metaCompile(&io,Files)}}, loop.
loop(normal(wrap#load_macro(Files))) --> true |
    {{loadMacro(&io,Files, Sig)}}, loopDynamic(Sig).
loop(normal(wrap#consult(Files))) --> true |
    consult(Files, Sig), loopDynamic(Sig).
otherwise.
loop(normal(Gs)) --> true | {{solve(&io,Gs)}}, loop.

loopDynamic(Dev,NDev, DB,NDB, normal) :- true |
    {{module_table:dynamic_link(meta::prolog,loop,{Dev,NDev, DB,NDB})}}.
loopDynamic(Dev,NDev, DB,NDB, abnormal) :- true | loop(Dev,NDev, DB,NDB).
%loopDynamic(Sig) --> wait(Sig) |
%    {{module_table:dynamic_link(meta::prolog,loop,{&io, &db})}}.

:- local_implicit io:oldnew.
%
%   Interpreter
%
solve(Gs) --> system_timer(_, T) |
    {{meta::utility:create_env(Env0,1000)}},
%    {{meta#create_env(Env0,1000)}},
    {{pairToList(Gs,Gs1)}},
    {{meta::trans:get_object(Gs1,Env0, ExpEnvVT)}},
%    {{meta#get_object(Gs1,Env0, ExpEnvVT)}},
    solve(ExpEnvVT, T).

solve({Gs,Env,VT}, T0) --> true |
    solveAnd(Gs,[], Env, Solution),
    solveCollect(Solution, N,T1, NSolution),
    solveDisplay(NSolution, N, T0,T1, VT).

solveCollect(S, N,T, NS) --> true |
    merge(S,S1), solveCollect(S1, 0,N, T,NS).
```

```
solveCollect([], M,N, T, NS) --> system_timer(_, T1) |
    N := M, T = T1, NS = [].
solveCollect([E|Es], M,N, T, NS) --> true |
    NS = [E|NS1],
    solveCollect(Es, ^(M+1),N, T,NS1).

solveDisplay(Envs, N, T0,T1, VT) --> true |
    solveDisplay(Envs,VT),
    {{display(&io," Number of Solution : ^t^n",[N])}},
    {{display(&io," ^t msec^n",[^((T1-T0)/16)])}}.

solveDisplay([], _) --> true | true.
solveDisplay([E|Es], VT) --> true |
    solveDisplayEach(VT,E),
    solveDisplay1(Es, VT).

solveDisplayEach(VT, Env) --> true | solveDisplayEach(VT,VT, Env).

solveDisplayEach([],_, _) --> true | true .
solveDisplayEach([{Name,Num}|VT],OVT, Env) --> true |
    {{meta::trans:get_kl1_term({Num},Env,OVT, WT)}},
%    {{meta#get_kl1_term({Num},Env,OVT, WT)}},
    {{display(&io, "^n ^s = ^w ",[Name,WT])}},
    solveDisplayEach(VT,OVT, Env).

solveDisplay1(Es, VT) --> true |
    &io <<= [get1(L)],
    solveDisplay1Decide(L, Es,VT).

solveDisplay1Decide(";", Es,VT) --> true | solveDisplay(Es, VT).
otherwise.
solveDisplay1Decide(_, _,_) --> true | true.

solveAnd([],[], Env, Sol) --> true | Sol = [Env].
solveAnd([],[Gs|Gss], Env, Sol) --> true | solveAnd(Gs,Gss, Env, Sol).
solveAnd([(X := Y)|Gs],Gss, Env, Sol) --> true |
    solveArth(X,Y, Gs,Gss, Env,Sol).
solveAnd([(X =:= Y)|Gs],Gss, Env, Sol) --> true |
    solveAComp(X,Y, (=:=),Gs,Gss, Env,Sol).
solveAnd([(X =\= Y)|Gs],Gss, Env, Sol) --> true |
    solveAComp(X,Y, (=\=),Gs,Gss, Env,Sol).
solveAnd([(X < Y)|Gs],Gss, Env, Sol) --> true |
    solveAComp(X,Y, (<),Gs,Gss, Env,Sol).
solveAnd([(X = Y)|Gs],Gss, Env, Sol) --> true |
    solveUnify(X,Y, Gs,Gss, Env,Sol).
otherwise.
solveAnd([G|Gs],Gss, Env, Sol) --> true |
    clauses(G,Env, Clauses),
    solveOr(Clauses,G,Gs,Gss, Sol).

solveOr([{C,Env}|Cs], G,Gs,Gss, Sol) --> C = (H :- B) |
    Sol = {Sol1,Sol2},
    {{meta::unify:unify(H,G, Env,Env1)}},
%    {{meta#unify(H,G, Env,Env1)}},
    expand(Env1, B,Gs,Gss, Sol1),
    solveOr(Cs, G,Gs,Gss, Sol2).
```

```
solveOr([], _,_,_, Sol) --> true | Sol = [].

expand(fail, _,_,_, Sol) --> true | Sol = [].
otherwise.
expand(Env, B,Gs,Gss, Sol) --> true | solveAnd(B,[Gs|Gss], Env, Sol).

clauses(G,Env, CLs) --> vector(G, Size), vector_element(G,0,F) |
    Arity := Size-1, clauses1({F,Arity},Env, CLs).
clauses(G,Env, CLs) --> atom(G) | clauses1({G,0},Env, CLs).

clauses1(FA,Env, CLs) --> true | clauses1(FA,0,Env, CLs).

clauses1(FA,M,Env, CLs) --> true |
    {{'$PrologDataBase':get({FA,M},Env, ExpEnv)}},
    clauses1Decide(ExpEnv, FA,M,Env, CLs).

clauses1Decide({}, _,_,_,CLs) --> true | CLs = [].
clauses1Decide(ExpEnv, FA,M,Env, CLs) --> ExpEnv = {_,_} |
    CLs = [ExpEnv|CLs1],
    clauses1(FA,~(M+1),Env, CLs1).

solveUnify(X,Y, Gs,Gss, Env,Sol) --> true |
    {{meta::unify:unify(X,Y, Env,NEnv)}},
%    {{meta#unify(X,Y, Env,NEnv)}},
    expand(NEnv, [],Gs,Gss, Sol).

solveArth(X,Y, Gs,Gss, Env,Sol) --> true |
    eval(Y,NY, Env),
    solveArthDecide(X,NY, Gs,Gss, Env,Sol).

solveArthDecide(X,Y, Gs,Gss, Env,Sol) --> Y = integer(YNum) |
    solveAnd([(X = YNum)|Gs],Gss, Env,Sol).
otherwise.
solveArthDecide(_,_, _,_, _,Sol) --> true | Sol = [].

eval(Y,NY, Env) --> true |
    {{meta::utility:is_type(Y,Env, Type)}},
%    {{meta#is_type(Y,Env, Type)}},
    evalDecide(Type,NY, Env).

evalDecide(vector({OP,Y1,Y2}),NY, Env) --> true |
    evalBinary(OP,Y1,Y2, NY, Env).
otherwise.
evalDecide(Y,NY, _) --> true | NY = Y.

evalBinary(OP,Y1,Y2, NY, Env) --> true |
    eval(Y1,NY1, Env), eval(Y2,NY2, Env),
    evalBinaryDecide(NY1,NY2, NY, OP).

evalBinaryDecide(integer(Y1),integer(Y2), NY, OP) --> true |
    evalBinary(OP, Y1,Y2, NY).
otherwise.
evalBinaryDecide({_,Y1},{_,Y2}, NY, OP) --> true | NY = {OP,Y1,Y2}.

evalBinary((+), Y1,Y2, NY) --> Y := Y1+Y2 | NY = integer(Y).
evalBinary((-), Y1,Y2, NY) --> Y := Y1-Y2 | NY = integer(Y).
```

```
evalBinary((*), Y1,Y2, NY) --> Y := Y1*Y2 | NY = integer(Y).
evalBinary((/), Y1,Y2, NY) --> Y := Y1/Y2 | NY = integer(Y).
evalBinary((mod), Y1,Y2, NY) --> Y := Y1 mod Y2 | NY = integer(Y).
otherwise.
evalBinary(OP, Y1,Y2, NY) --> true | NY = {OP,Y1,Y2}.

solveAComp(X,Y, OP,Gs,Gss, Env,Sol) --> true |
    eval(X,NX, Env), eval(Y,NY, Env),
    solveACompDecide(NX,NY, OP,Gs,Gss, Env,Sol).

solveACompDecide(X,Y, OP,Gs,Gss, Env,Sol) -->
    X = integer(XNum), Y = integer(YNum) |
    solveACompDecide1(XNum,YNum, OP,Gs,Gss, Env,Sol).
otherwise.
solveACompDecide(_,_, _,_,_, _,Sol) --> true | Sol = [].

solveACompDecide1(X,Y, (=:=),Gs,Gss, Env,Sol) --> X =:= Y |
    solveAnd(Gs,Gss, Env,Sol).
solveACompDecide1(X,Y, (=\=),Gs,Gss, Env,Sol) --> X =\= Y |
    solveAnd(Gs,Gss, Env,Sol).
solveACompDecide1(X,Y, (<),Gs,Gss, Env,Sol) --> X < Y |
    solveAnd(Gs,Gss, Env,Sol).
otherwise.
solveACompDecide1(_,_, _,_,_, _,Sol) --> true | Sol = [].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Several Commands
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- local_implicit io:oldnew.
%
%   cd(WrappedDirectoryName)
%
cd(string(Dir)) --> true | cd1(Dir).
otherwise.
cd(Dir) --> true |
    display(" Illegal directory specification ~w .~n",[Dir]), no.

cd1(Name) --> string_element(Name,0,#">") | cd2(Name).
otherwise.
cd1(Name) --> true | getEnvVar(task:directory, Val), cd1Decide(Val, Name).

cd1Decide({AName}, ZName) --> true |
    {{builtin#append_string([AName,">",ZName], Name)}}, cd2(Name).
cd1Decide({}, _) --> true | no.

cd2(Name) --> true | getDirectory(Name, Dir), cd3(Dir).

cd3({Dir}) --> true | Dir = [pathname(DName)], cd4(DName).
cd3({}) --> true | no.

cd4(normal(Name)) --> true |
    setEnvVar(task:directory,Name, St),
    cd4Decide(St, Name).
cd4(abnormal) --> true | no.
```

```
cd4Decide(normal, Name) --> true | display(" world = ~s~n",[Name]), yes.
cd4Decide(abnormal, _) --> true | no.

:- local_implicit io:oldnew.
%
%   pwd.
%
pwd --> true | getEnvVar(task:directory, Name), pwd(Name).

pwd({}) --> true | no.
pwd({Name}) --> true | display(" world = ~s~n",[Name]), yes.

:- local_implicit io:oldnew.
%
%   ls.
%
ls --> true | ls(wrap#"*.*").

ls(string(WildCard)) --> true |
    getDirectory(".",Dir), ls(Dir,WildCard).
otherwise.
ls(_) --> true | no.

ls({}, _) --> true | no.
ls({Dir}, WildCard) --> true |
    Dir = [pathname(PName),listing(WildCard,FNames)],
    ls1(PName,FNames).

ls1(normal(PName),normal(FNames)) --> true |
    display(" world = ~s~n",[PName]),
    ls1(FNames).

ls1([]) --> true | yes.
ls1([FName|FNames]) --> true |
    display(" ~s~n",[FName]), ls1(FNames).

:- local_implicit io:oldnew.
%
%   loadMacro(WrappedFileSpecification, -TerminationSignal)
%
loadMacro(string(File), St) --> true |
    {{fileOpen(&io, read,File, File0)}},
    loadMacroDecide(File0, St).
otherwise.
loadMacro(_, St) -->  true | St = abnormal, no.

loadMacroDecide({}, St) --> true | St = abnormal, {{no(&io)}}.
%loadMacroDecide({In}, St) --> true |
%    {{buffer:input_filter(In0,In)}},
%    {{meta::macroCompile:go(In0, Code)}},
%    Link = {(meta::prolog),[(meta::macroBank)]},
%    Option = [package(meta)],
%    {{compileCode(&io, Code, "AHO.KL1",Link,Option, Sig)}},
%    loadMacro1(Sig, St).
loadMacroDecide({In}, St) --> true |
```

```
    {{buffer:input_filter(In0,In)}},
    {{meta::macroCompile:go(In0, Code)}},
    {{getEnvVar(&io, task:package,Pac)}},
    loadMacroDecide1(Pac, Code, St).

loadMacroDecide1({Pac}, Code, St) --> true |
    setEnvVar(task:package,meta, St0),
    loadMacroDecide2(St0, Pac, Code, St).

loadMacroDecide2(normal, Pac, Code, St) --> true |
    Link = {prolog,[macroBank]},
    Option = [package(meta)],
    {{compileCode(&io, Code, "AHO.KL1",Link,Option, Sig)}},
    loadMacro1(Sig, Pac, St).

%loadMacro1(normal, St) --> true | St = normal, {{yes(&io)}}.
%loadMacro1(abnormal, St) --> true | St = abnormal, {{no(&io)}}.

loadMacro1(normal, Pac, St) --> true | St = normal,
    {{setEnvVar(&io, task:package,Pac,_)}}, {{yes(&io)}}.
loadMacro1(abnormal, Pac, St) --> true | St = normal,
    {{setEnvVar(&io, task:package,Pac,_)}}, {{no(&io)}}.

:- local_implicit io:oldnew.
%
%   metaCompile(WrappedFileSpecification)
%
metaCompile(Files) --> true |
    metaCompile(Code,[], Files),
    metaCompileComp(Code).

metaCompileComp([]) --> true | {{no(&io)}}.
metaCompileComp(Code) --> list(Code) |
    Link = {},
    Option = [no_indexing],
    {{compileCode(&io, Code, "IKA.KL1",Link,Option, Sig)}},
    metaCompileCompDecide(Sig).

metaCompileCompDecide(normal) --> true | {{yes(&io)}}.
metaCompileCompDecide(abnormal) --> true | {{no(&io)}}.

:- local_implicit io:oldnew, code:oldnew.
metaCompile(wrap#[]) --> true | true.
metaCompile(string(File)) --> true |
    {{fileOpen(&io, read,File, File0)}},
    metaCompileDecide(File0).
metaCompile(wrap#[F|Fs]) --> true |
    metaCompile(F), metaCompile(Fs).
otherwise.
metaCompile(Files) --> true |
    {{display(&io, " Illegal file specification ~w.~n",[Files])}}.

metaCompileDecide({}) --> true | true.
metaCompileDecide({In}) --> true |
    {{buffer:input_filter(In0,In)}},
    {{meta::macroExpand:go(In0, Out)}},
```

```
        &code <<= [do(Out)].

    :- local_implicit io:oldnew.
    compileCode(Code, FID,Link,Opt, Sig) --> true |
        {{windowOpen(&io, "Compiler Message",Window)}},
        {{fileOpen(&io, write, FID, File)}},
        compileCode(Window,File,FID,Link,Opt, Code,Sig).

    compileCode({Window},{File},FID,Link,Opt, Code,Sig) --> true |
        {{buffer:output_filter([do(Code),flush(St0)],File)}},
        {{buffer:interaction_filter(Window0,Window)}},
         Window0 = [activate(St1),reshape(at(718,75),char(40,12), St2)|Window1],
        compileCode(St0,St1,St2, Window1,FID,Link,Opt,Sig).

    compileCode(normal,normal,normal, Window,FID,Link,Opt,Sig) --> true |
        {{compileOption(Opt, Option,Pac)}},
        Window = [do(Window1),do(Window2)|Window3],
        {{compile:compile([FID], Option,Pac, Window1,Window2, St)}},
        compileCodeDone(St, Window3,Link,Sig).

    compileOption(Option, NOption,Package) :- true |
        compile:compile_option(Option0,Package0),
        compileOptionLoop(Option, Option0,Package0, NOption,Package).

    compileOptionLoop([], Opt,Pac, NOpt,NPac) :- true | NOpt = Opt, NPac = Pac.
    compileOptionLoop([package(Pac)|Opts], Opt,_, NOpt,NPac) :- true |
        compileOptionLoop(Opts, Opt,Pac, NOpt,NPac).
    compileOptionLoop([no_indexing|Opts], {C,I,D,S},Pac, NOpt,NPac) :- true |
        compileOptionLoop(Opts, {C,3,D,S},Pac, NOpt,NPac).
    compileOptionLoop([indexing|Opts], {C,I,D,S},Pac, NOpt,NPac) :- true |
        compileOptionLoop(Opts, {C,4,D,S},Pac, NOpt,NPac).

    compileCodeDone(normal, Window,Link,Sig) --> true |
        Window = [getl(L)],
        compileCodeDoneWait(L,Link,Sig).
    compileCodeDone(abnormal, Window,_,Sig) --> true |
        Window = [getl(_)], Sig = abnormal.

    compileCodeDoneWait(L,Link,Sig) --> wait(L) |
        compileCodeDoneLink(Link,Sig).

    compileCodeDoneLink({TModule,MNames},Sig) --> true |
        {{relink:go(TModule,MNames)}},
        compileCodeDoneSleep(Sig)@priority(*,0).
    compileCodeDoneLink({Modules},Sig) --> true |
        {{relink:go(Modules)}},
        compileCodeDoneSleep(Sig)@priority(*,0).
    compileCodeDoneLink({},Sig) --> true | Sig = normal.

    compileCodeDoneSleep(Sig) --> true | Sig = normal.

    :- local_implicit io:oldnew, db:oldnew.
    %
    %   consult(WrappedFileSpecification, -TerminationSignal)
    %
    consult(File, St) --> true |
```

```
    consult1(File, Cnt),
    consultCompile(Cnt, St).

consultCompile(0, St) --> true | St = abnormal, {{no(&io)}}.
%consultCompile(Cnt,St) --> Cnt > 0 |
%    &db <<= [compile('$PrologDataBase', Code)],
%    Link = {(meta::prolog),[(meta::'$PrologDataBase')]},
%    Option = [package(meta)],
%    {{compileCode(&io, Code, "TAKO.KL1",Link,Option, St)}},
%    {{yes(&io)}}.
consultCompile(Cnt, St) --> Cnt > 0 |
    &db <<= [compile('$PrologDataBase', Code)],
    {{getEnvVar(&io, task:package,Pac)}},
    consultCompileDecide1(Pac, Code, St).

consultCompileDecide1({Pac}, Code, St) --> true |
    {{setEnvVar(&io, task:package,meta,St0)}},
    consultCompileDecide1(St0, Pac, Code, St).

consultCompileDecide1(normal, Pac, Code, St) --> true |
    Link = {prolog,['$PrologDataBase']},
    Option = [package(meta),indexing],
    {{compileCode(&io, Code, "TAKO.KL1",Link,Option, Sig)}},
    consult1CompileDecide2(Sig, Pac, St).

consult1CompileDecide2(normal, Pac, St) --> true |
    {{setEnvVar(&io, task:package,Pac, St0)}},
    consult1CompileDecide3(St0,St).
consult1CompileDecide2(abnormal, Pac, St) --> true | St = abnormal,
    {{setEnvVar(&io, task:package,Pac,_)}}, {{no(&io)}}.

consult1CompileDecide3(normal, St) --> true |
    St = normal, {{yes(&io)}}.

consult1(wrap#[], Cnt) --> true | Cnt := 0.
consult1(string(File), Cnt) --> true |
    {{fileOpen(&io, read,File, File0)}},
    consult1Decide(File0, Cnt).
consult1(wrap#[F|Fs], Cnt) --> true |
    consult1(F, Cnt1), consult1(Fs, Cnt2), Cnt := Cnt1+Cnt2.
otherwise.
consult1(Files, Cnt) --> true | Cnt := 0,
    {{display(&io, " Illegal file specification ~w.~n",[Files])}}.

consult1Decide({}, Cnt) --> true | Cnt := 0.
consult1Decide({In}, Cnt) --> true |
    {{buffer:input_filter(In0,In)}},
    consultLoop(0,0, In0,Cnt).

consultLoop(FA,M, In,Cnt) --> true |
    In = [getwt(WT)|In1],
    consultLoop(WT, FA,M, In1,Cnt).

consultLoop(normal(end_of_file), _,_, In,Cnt) --> true |
    In = [], Cnt := 0.
consultLoop(normal(empty), FA,M, In,Cnt) --> true |
```

```
        consultLoop(FA,M, In,Cnt).
    consultLoop(abnormal(ErrorInfo), FA,M, In,Cnt) --> true |
        &io <<= [print_error(ErrorInfo)],
        consultLoop(FA,M, In,Cnt).
    otherwise.
    consultLoop(normal(WT), FA,M, In,Cnt) --> true |
        consultLoopP(WT, FA,M, In,Cnt).

    consultLoopP(wrap#(H :- B), FA,M, In,Cnt) --> true |
        {{pairToList(B, B1)}},
        consultLoopP(H,B1, FA,M, In,Cnt).
    otherwise.
    consultLoopP(H, FA,M, In,Cnt) --> true |
        consultLoopP(H,wrap#[], FA,M, In,Cnt).

    consultLoopP(H,B, FA,M, In,Cnt) --> true |
        {{getFA(H, NFA)}},
        consultLoopPDecide(NFA,FA, H,B, M, In,Cnt).

    getFA(H, FA) :- H = vector(V), vector(V,Size), vector_element(V,0,atom(F)) |
        Arity := Size-1, FA = {F,Arity}.
    getFA(H, FA) :- H = atom(F) | FA = {F,0}.

    consultLoopPDecide(FA,FA, H,B, M, In,Cnt) --> true |
        {{meta::trans:get_object(wrap#(H :- B), ExpEnvVT)}},
%       {{meta#get_object(wrap#(H :- B), ExpEnvVT)}},
        Cnt := Cnt1+1,
        consultLoopPEntry(ExpEnvVT, {FA,M}),
        consultLoop(FA,~(M+1), In,Cnt1).
    otherwise.
    consultLoopPDecide(FA,_, H,B, _, In,Cnt) --> true |
        &db <<= [get({FA,0}, ExpEnv)],
        consultLoopPDecide1(ExpEnv, FA, H,B, In,Cnt).

    consultLoopPEntry({Exp,Env,_}, ID) --> true | &db <<= [put(ID, Exp,Env)].

    consultLoopPDecide1({}, FA, H,B, In,Cnt) --> true |
        consultLoopPDecide(FA,FA, H,B, 0, In,Cnt).
    consultLoopPDecide1({_,_}, FA, H,B, In,Cnt) --> true |
        {{display(&io, " Warning Double Definition ~t .~n",[FA])}},
        consultLoopPRemove(FA),
        consultLoopPDecide(FA,FA, H,B, 0, In,Cnt).

    consultLoopPRemove(FA) --> true | consultLoopPRemove(FA,0).

    consultLoopPRemove(FA,M) --> true |
        &db <<= [get({FA,M}, ExpEnv)],
        consultLoopPRemoveDecide(ExpEnv, FA,M).

    consultLoopPRemoveDecide({}, _,_) --> true | true .
    consultLoopPRemoveDecide({_,_}, FA,M) --> true |
        &db <<= [remove({FA,M})],
        consultLoopPRemove(FA,~(M+1)).

    pairToList(wrap#(G,Gs), List) :- true |
        List = wrap#[G|List1], pairToList(Gs, List1).
```

```
otherwise.
pairToList(G, List) :- true | List = wrap#[G].

:- local_implicit io:oldnew.
%
%   Display facilities
%
display(FString,Arg) --> true | &io <<= [putf(FString,Arg)].

no --> true | &io <<= [putf(" no~n",[])].

yes --> true | &io <<= [putf(" yes~n",[])].

%
%   Manipulation for Environment Variable
%
getEnvVar(Name, Val) --> true |
    getGR(GR), getEnvVar1(GR, Name, Val).

getEnvVar1({}, _, Val) --> true | Val = {}.
getEnvVar1({GR}, Name, Val) --> true |
    GR = [getenv(Name, Val0)],
    getEnvVar2(Val0, Name,Val).

getEnvVar2(normal(Val), _,NVal) --> true | NVal = {Val}.
getEnvVar2(abnormal, Name,NVal) --> true | NVal = {},
    display(" ~s not found. ~n",[Name]).

setEnvVar(Name,Val, St) --> true |
    getGR(GR), setEnvVar1(GR, Name,Val, St).

setEnvVar1({}, _,_, St) --> true | St = abnormal.
setEnvVar1({GR}, Name,Val, St) --> true |
    GR = [setenv(Name,Val, St0)],
    setEnvVar2(St0, Name,Val, St).

setEnvVar2(normal, _,_, St) --> true | St = normal.
setEnvVar2(abnormal, Name,Val, St) --> true | St = abnormal,
    display(" ~t not set on ~s",[Val,Name]).

%
%   Manipulation for OS defining resource
%
getGR(GR) --> true |
    {{shoen:raise(pimos_tag#task,general_request,GR0)}},
    GR0 = [connection(St)|GR1],
    getGR(St, GR1,GR).

getGR(normal(_), GR,GR1) --> true | insertMerge(GR1,GR).
getGR(abnormal, _,GR) --> true | GR = {},
    display(" cannot make general request.~n",[]).

getFileSystem(FS) --> true | getRequestDevice(file, FS).

getRequestDevice(Kind, RD) --> true |
    getGR(GR), getRequestDevice(GR, Kind, RD).
```

```
getRequestDevice({}, _, RD) --> true | RD = {}.
getRequestDevice({GR}, Kind, RD) --> true |
    GR = [{Kind,RD0}], getRequestDeviceDecide(RD0,RD, Kind).

getRequestDeviceDecide(normal(RD0,_,_),RD, _) --> true | insertMerge(RD,RD0).
getRequestDeviceDecide(abnormal,RD, Kind) --> true | RD = {},
    display(" Cannot make ~t request device.~n",[Kind]).

getWindowSystem(WS) --> true | getRequestDevice(window, WS).

fileOpen(Kind,Name, File) --> true |
    getFileSystem(FS), fileOpen(FS, Kind,Name, File).

fileOpen({}, _,_, File) --> true | File = {}.
fileOpen({FS}, Kind,Name, File) --> true |
    FS = [open(Name,{Kind,File0})], fileOpenDecide(File0, Kind,Name, File).

fileOpenDecide(normal(File0,_,_), _,_, File) --> true | insertMerge(File,File0).
fileOpenDecide(abnormal, Kind,Name, File) --> true | File = {},
    display(" ~s not ~t open . ~n",[Name,Kind]).

windowOpen(Name, Window) --> true |
    getWindowSystem(WS),
    windowOpen(WS, Name, Window).

windowOpen({}, _, Window) --> true | Window = {}.
windowOpen({FS}, Name, Window) --> true |
    FS = [create(Name, Window0)],
    windowOpenDecide(Window0, Name, Window).

windowOpenDecide(normal(Window0,_,_), _, Window) --> true |
    insertMerge(Window,Window0).
windowOpenDecide(abnormal, Name, File) --> true | File = {},
    display(" ~s not open . ~n",[Name]).

getDirectory(Name, Dir) --> true |
    getFileSystem(FS), getDirectory(FS, Name,Dir).

getDirectory({}, _,Dir) --> true | Dir = {}.
getDirectory({FS}, Name,Dir) --> true |
    FS = [directory(Name, Dir0)], getDirectoryDecide(Dir0, Name,Dir).

getDirectoryDecide(normal(Dir0,_,_), _,Dir) --> true | insertMerge(Dir,Dir0).
getDirectoryDecide(abnormal, Name,Dir) --> true | Dir = {},
    display(" ~s not found.~n",[Name]).

insertMerge(Stream,Stream1) --> true |
    merge(Stream0,Stream1), Stream = {Stream0}.

:- module '$PrologDataBase'.
:- public get/2, get/3.

get(_, ExpEnv) :- true | ExpEnv = {}.

get(_,_, ExpEnv) :- true | ExpEnv = {}.
```

B.1.3 使用例

1. リスナを起動する.

2. Prolog インタプリタを起動する.

```
?- meta::prolog.
 Pure Prolog Interpreter
 ??-
```

【参考】ディレクトリコマンドとして A.3簡易コンパイラと同じものが使える.

3. プログラムをロードする.

```
??- consult("append.pl").
```

4. KL1 コンパイルが起動される. コンパイラ用のウインドウが現れコンパイラのメッセージが表示される. このウインドウより return を入力することによりリンカが起動される.

```
"meta:: prolog" Updated
 yes
 ??-
```

5. ゴールを入力する.

```
 ??- append(X,Y,[1,2,3]).

Relink Succeeded

 X = []
 Y = [1,2,3] ;

 X = [1]
 Y = [2,3] ;

 X = [1,2]
 Y = [3] ;

 X = [1,2,3]
 Y = [] ;
 Number of Solution : 4
 9 msec
 ??- cossult("queens.pl").
 Number of Solution : 0
 1 msec
 ??- consult("queens.pl").
"meta:: prolog" Updated
 yes
 ??- queens(4,X).

Relink Succeeded

 X = [3,1,4,2]
```

```
Number of Solution : 2
305 msec
??- queens(8,X).

X = [4,2,7,3,6,8,5,1] ;

X = [5,2,4,7,3,8,6,1]
Number of Solution : 92
79329 msec
??-
```

6. インタプリタを終了する.

```
??- halt.
End of Execution

?-
```

B.2 Model Elimination Theorem Prover

Model Elimination 法 (ME 法)[2, 5] に基づく証明プログラムを示す. ME 法は Ordered Linear(OL) 導出法 [1] と本質的に等価な証明法である. ME 法は factoring 不要な, 完全な input 手続きである.

問題は節形式で与え, top 節を指定しなければならない. Top 節と入力節は前処理され,Top 節は 'OLTop':clause/1 入力節は 'OLDabase':get/3 を呼び出すことにより参照できるようになる.

ME 法は 3 つの手続き extention, reduction contraction からなる. `extend/3` は extention, `reduce/5` は reduction, `reduceContract/3` は contraction を行う述語である.

B.2.1 ソースコード

```
:- module 'ME'.
:- public compile/0, go/0, go/2, go/3.

%
%   compile -- compiling input clauses --
%
compile :- true |
    shoen:raise(pimos_tag#shell,get_std_in,In),
    shoen:raise(pimos_tag#shell,get_std_out,Out),
    shoen:raise(pimos_tag#shell,get_std_mes,Mes),
    compile(In, Mes, Out).

%
%   compile(Input, Message, Output)
%
compile(In, Mes, Out) :- true |
    meta#database(DB), pool:keyed_set(Pool),
    In = [do(In1)|In2], Mes = [do(Mes1)|Mes2],
    DB = [do(DB1),do(DB2)|DB3], Pool = [do(Pool1)|Pool2],
    compileTop(In1, DB1,Pool1, Top, Mes1),
    compile(In2, DB2,Pool2, Mes2),
    compileOut(Top,DB3, Out).

%
%   compileTop(Input, MetaDatabase,Pool, -TopClause, Output)
%
```

```
compileTop(In, DB,Pool, Top, Out) :- true |
    nextTerm(In,In1, WT, Out,Out1),
    compileTopDecide(WT, In1, DB,Pool, Top, Out1).

compileTopDecide(normal(empty), In, DB,Pool, Top, Out) :- true |
    compileTop(In, DB,Pool, Top, Out).
compileTopDecide(normal(Clause), In, DB,Pool, Top, Out) :- Clause = list(_) |
    In = [], Out = [],
    clauseShuffle(Clause, Top0),
    entry(Top0, 0,DB,Pool),
    (Top0 = {CL,Env} -> reverse(CL, RCL), Top = {RCL,Env}).

%
%   compile(Input, MetaDatabase,Pool, Output)
%
compile(In, DB,Pool, Out) :- true | compile(In, -1, DB,Pool, Out).

compile(In, ID, DB,Pool, Out) :- true |
    nextTerm(In,In1, WT, Out,Out1),
    compileDecide(WT, In1, ID, DB,Pool, Out1).

compileDecide(normal(end_of_file), In, _, DB,Pool, Out) :- true |
    In = [], DB = [], Pool = [], Out = [].
compileDecide(normal(empty), In, ID, DB,Pool, Out) :- true |
    compile(In, ID, DB,Pool, Out).
compileDecide(normal(Clause), In, ID, DB,Pool, Out) :- Clause = list(_) |
    DB = [do(DB1)|DB2], Pool = [do(Pool1)|Pool2],
    clauseShuffle(Clause, SClause),
    entry(SClause, ID, DB1,Pool1),
    compile(In, ~(ID-1), DB2,Pool2, Out).

%
%   compileOut(+TopClause, MetaDatabase, Output)
%
compileOut(Top, DB, Out) :- true |
    Out = [do(Out1)|Out2],
    compileOutTop(Top, Out1),
    compileOutDB(DB, Out2).

compileOutTop({CL,Env}, Out) :- vector_element(Env,0,Top) |
    Out = [putf(":- module 'METop'.~n",[]),
           putf(":- public clause/1.~n~n",[]),
           putf("clause(Top) :- true |~n",[]),
           putf("    Top = {CL,Env},~n",[]),
           putf("    CL = ~t,",[CL]),
           putf("    meta#create_env(Env,~t,~t).~n~n",[~(Top-1),Top])].

compileOutDB(DB, Out) :- true | DB = [compile('MEDatabase',Out)].

%
%   entry(+Clause, +ID,MetaDatabase,Pool)
%
entry(CL, ID,DB,Pool) :- CL = {OCL,Env} | entry1(OCL,CL,[], ID,DB,Pool).

entry1([],_, _, _,DB,Pool) :- true | DB = [], Pool = [].
entry1([L|Ls],CL, H, ID,DB,Pool) :- true |
```

```
        getProp(L, PM0,FA), reverseSign(PM0,PM),
        Key = {PM,FA},
        member(Key, H, YN),
        entry1Decide(YN, Ls,CL, Key,H, ID,DB,Pool).

entry1Decide(yes, Ls,CL, _,H, ID,DB,Pool) :- true | entry1(Ls,CL, H, ID,DB,Pool).
entry1Decide(no, Ls,CL, Key,H, ID,DB,Pool) :- Key = {PM,FA} |
    DB = [do(DB1)|DB2],
    Pool = [put(Key, Num,ONum)|Pool1],
    entry11Decide(ONum,Num, {PM,FA,Num}, CL, ID,DB1),
    entry1(Ls,CL, [Key|H], ID,DB2,Pool1).

entry11Decide({},Num, Key, CL, ID,DB) :- true | Num := 0,
    entry12(DB, Key, CL, ID).
entry11Decide({ONum},Num, Key, CL, ID,DB) :- true | Num := ONum+1,
    entry12(DB, Key, CL, ID).

entry12(DB, Key, {CL,Env}, ID) :- true | DB = [put(Key, {ID,CL},Env)].

clauseShuffle(WrappedClause, OrderedClause) :- true |
    meta#get_object(WrappedClause, CLEnvVT),
    clauseShuffle1(CLEnvVT, OrderedClause).

clauseShuffle1({CL,Env,_}, OCL) :- true | OCL = {CL,Env}.

%
%   go -- main --
%
go :- true | go(50,10).

go(D,S) :- true | go(D,S, no_trace).

go(Depth, Step, TraceFlag) :- true |
    shoen:raise(pimos_tag#shell,get_std_out,Out),
    refute(Depth,Step, Trace),
    output(TraceFlag, Trace, Out).

%
%   refute(+SearchingDepth,+NextDepth, -TraceInformation)
%
refute(Depth,Step, Trace) :- true |
    'METop':clause(CL),
    refute(CL, Depth,Step, Trace).

refute({CL,Env}, Depth,Step, Trace) :- true |
    refute(CL,Env, Depth,Step, Trace, Mon, End),
    monitorInit([Mon], End).

refute(_,_, _,_, Trace, Mon, end) :- true | Trace = [], Mon = [].
alternatively.
refute(CL,Env, 0,OD, Trace, Mon, End) :- true |
    Mon = [{NMon,S}],
    refuteWait(S, CL,Env, OD,OD, Trace, NMon, End).
refute(_,_, D,_, Trace, Mon, End) :- D < 0 | Trace = [], Mon  = [], End = end.
refute(CL,Env, D,OD, Trace, Mon, End) :- D > 0 |
    extend(CL,Env, CLs),
```

```
    refuteExpand(CLs, ^(D-1),OD, Trace, Mon, End).

refuteExpand(_, _,_, Trace, Mon, end) :- true | Trace = [], Mon = [].
alternatively.
refuteExpand(CLs, D,OD, Trace, Mon, End) :- true |
    new_vector(Trace0,Size),
    refuteExpand(CLs, 0,Size, Trace0,Trace, D,OD, Mon, End).

refuteExpand(_, M,N, Tr,NTr, _,_, Mon, end) :- true |
    N := M, NTr = Tr, Mon = [].
alternatively.
refuteExpand([], M,N, Tr,NTr, _,_, Mon, _):- true |
    N := M, NTr = Tr, Mon = [].
refuteExpand([{CL,Env,P}|CLs], M,N, Tr,NTr, D,OD, Mon, End) :- true |
    reduceContract(CL,Env, CLEnvs),
    refuteExpandDecide(CLEnvs, P,CLs, M,N, Tr,NTr, D,OD, Mon, End).

reduceContract(Ls,Env, CLEnvs) :- true | reduceContract(Ls,Env, CLEnvs,[]).

reduceContract([],Env, A,Z) :- true | A = [refuted([],Env)|Z].
reduceContract['$'(_)|Ls],Env, A,Z) :- true | reduceContract(Ls,Env, A,Z).
otherwise.
reduceContract([L|Ls],Env, A,Z) :- true | reduce(L,Ls, Env, A,Z).

%
%   reduce(+Literal,+Literals, +Environment, CLEnvs,CLEnvs)
%
reduce(L,Ls, Env, A,Z) :- true |
    reducible(L,Ls, Env, Envs),
    reduceDecide(Envs, L,Ls,Env, A,Z).

reducible(_,[], Env, Envs) :- true | Envs = [].
reducible(L,['$'(L1)|Ls], Env, Envs) :- true |
    complement(L,L1, Env, YN),
    reducibleDecide(YN, L,Ls,Env, Envs).
otherwise.
reducible(L,[_|Ls], Env, Envs) :- true | reducible(L,Ls, Env, Envs).

reducibleDecide(yes(NEnv), L,Ls,Env, Envs) :- true |
    Envs = [NEnv|Envs1], reducible(L,Ls, Env, Envs1).
reducibleDecide(no(Env), L,Ls,_, Envs) :- true |
    reducible(L,Ls, Env, Envs).

reduceDecide([], L,Ls,Env, A,Z) :- true | A = [{[L|Ls],Env}|Z].
otherwise.
reduceDecide(Envs, _,Ls,_, A,Z) :- true | reduceGenerate(Envs, Ls, A,Z).

reduceGenerate([], _, A,Z) :- true | A = Z.
reduceGenerate([Env|Envs], Ls, A,Z) :- true |
    reduceContract(Ls,Env, A,B),
    reduceGenerate(Envs, Ls, B,Z).

refuteExpandDecide(_, _,_, M,N, Tr,NTr, _,_, Mon, end) :- true |
    N := M, NTr = Tr, Mon = [].
alternatively.
refuteExpandDecide([], _,CLs, M,N, Tr,NTr, D,OD, Mon, End) :- true |
```

```
    refuteExpand(CLs, M,N, Tr,NTr, D,OD, Mon, End).
refuteExpandDecide([CL|_], P,_, M,N, Tr,NTr, _,_, Mon, End) :-
    CL = refuted(CL1,Env) |
    set_vector_element(Tr,M, _,refuted(CL1,Env,P),NTr),
    N := M + 1, Mon = [], End = end.
refuteExpandDecide([CL|CLs], P,CCLs, M,N, Tr,NTr, D,OD, Mon, End) :-
    CL = {CL1,Env} |
    Mon = {Mon1,Mon2},
    set_vector_element(Tr,M, _,{CL1,Env,P,Trm}, Tr1),
    refute(CL1,Env, D,OD, Trm, Mon1, End),
    refuteExpandDecide(CLs, P,CCLs, ~(M+1),N, Tr1,NTr, D,OD, Mon2, End).

extend([L|Ls],Env, CLs) :- true |
    extendGetSides(L, Env, Sides,[]),
    extend1(Sides, L,Ls, CLs).

extendGetSides(L,Env, A,Z) :- true |
    getProp(L, PM,FA),
    extendGetSides(PM,FA,0, Env, A,Z).

extendGetSides(PM,FA,N, Env, A,Z) :- true |
    'MEDatabase':get({PM,FA,N}, Env, CLEnv),
    extendGetSidesDecide(CLEnv, PM,FA,N, Env, A,Z).

extendGetSidesDecide({}, _,_,_, _, A,Z) :- true | A = Z.
extendGetSidesDecide(CLEnv, PM,FA,N, Env, A,Z) :- CLEnv = {_,_} |
    A = [CLEnv|B],
    extendGetSides(PM,FA,~(N+1), Env, B,Z).

extend1([], _,_, CLs) :- true | CLs = [].
extend1([CLEnv|Ps], L,Ls, CLs) :- true |
    extend2(CLEnv, L,Ls, CLs,CLs1),
    extend1(Ps, L,Ls, CLs1).

extend2({{ID,CL},Env}, L,Ls, A,Z) :- true |
    extend2Unify(CL,L, Env, ID, ['$'(L)|Ls], A,Z).

extend2Unify([],_, _,  _, _, A,Z) :- true | A = Z.
extend2Unify([L0|Ls],L, Env,  ID, BLs, A,Z) :- true |
    complement(L0,L, Env, YN),
    (YN = no(_) -> A = B;
     YN = yes(NEnv) -> reverse(Ls,BLs, NLs),
        extend3(NLs,NEnv,ID, A,B)),
    extend2Unify(Ls,L, Env, ID, [L0|BLs], B,Z).

extend3(CL,Env, ID, A,Z) :- true |
    checkTautology(CL,Env, YN),
    (YN = yes   -> A = Z;
     YN = no    -> A = [{CL,Env,ID}|Z]).

checkTautology([],_, YN) :- true | YN = no.
checkTautology(['$'(_)|Ls],Env, YN) :- true | checkTautology(Ls,Env, YN).
checkTautology([-(L)|Ls],Env, YN) :- true |
    member(L,Ls, Env, equal, YN0),
    checkTautologyDecide(YN0, Ls, Env, YN).
otherwise.
```

```
checkTautology([L|Ls],Env, YN) :- true |
    member(-(L),Ls, Env, equal, YN0),
    checkTautologyDecide(YN0, Ls, Env, YN).

checkTautologyDecide(yes, _, _, YN) :- true | YN = yes.
checkTautologyDecide(no, Ls, Env, YN) :- true | checkTautology(Ls,Env, YN).

refuteWait(_, _,_, _,_, Trace, Mon, end) :- true | Trace = [], Mon = [].
alternatively.
refuteWait(go, CL,Env, D,OD, Trace, Mon, End) :- true |
    meta#copy_term(CL,NCL, Env,NEnv),
    refute(NCL,NEnv, D,OD, Trace, Mon, End).

monitorInit(NMon, End) :- true |
    monitorInit1(NMon, Mon0),
    merge(Mon0,Mon), monitor(Mon, [],_, End).

monitorInit1([], Mon) :- true | Mon = [].
monitorInit1([M|Ms], Mon) :- true | Mon = {M,Mon1}, monitorInit1(Ms, Mon1).

monitor(_, _,_, end) :- true | true.
alternatively.
monitor([], NMon,S, End) :- true | S = go, monitorInit(NMon, End).
monitor([{M,S0}|Mon], NMon,S, End) :- true | S0 = S,
    monitor(Mon, [M|NMon],S, End).

%
%   output(+TraceFlag, +TraceInformation, -OutStream)
%
output(Flg, Trace, Out) :- true | Out = [do(Out1)|Out2],
    (Flg = trace    -> Out1 = _Out1, Out2 = _Out2;
     Flg = no_trace -> Out1 = [], Out2 = _Out2;
     Flg = ignore   -> Out1 = [], Out2 = [];
     Flg = ignore(Tr) -> Out1 = [], Out2 = [], Tr = Trace),
    output(Trace, [0],0, _Out1, STrace),
    outputProof(STrace, _Out2).

outputProof(STrace, Out) :- true | merge(STrace,STrace1),
    outputProof(STrace1, 100000, [], Out).

outputProof([], _, Trace, Out) :- true | outputProof1(Trace, Out).
outputProof([{D,_}|STr], CD, Trace, Out) :- D >= CD |
    outputProof(STr, CD, Trace, Out).
outputProof([{D,Trace}|STr], CD, _, Out) :- D < CD |
    outputProof(STr, D, Trace, Out).

outputProof1(Trace, Out) :- true |
    reverse(Trace, RTrace),
    Out = [putf("~n~n----- Proof -----~n",[]),do(Out1),
           putf("----- End of Proof -----~n",[])],
    outputProof2(RTrace, Out1).

outputProof2([], Out) :- true | Out = [].
outputProof2([PrintArgs|PArgs], Out) :- true |
    Out = [putf("~t:~t, ~w~n",PrintArgs)|Out1],
    outputProof2(PArgs, Out1).
```

```
output(Trace, Width,Depth, Out, SuccessTrace) :- true |
    output(Trace, Width,Depth, Out, [],SuccessTrace).

output([], _,_, Out, _,STr) :- true | Out = [], STr = [].
output(Branch, [W|Width],Depth, Out, S,STr) :- vector(Branch,Size) |
    W1 := W + (Size - 1),
    outputArgs(0,Size, Branch, [W1|Width],~(Depth+1), Out, S,STr).

outputArgs(N,N, _, _,_, Out, _,STr) :- true | Out = [], STr = [].
outputArgs(M,N, B, W,D, Out, S,STr) :- vector_element(B,M,Bm) |
    Out = {Out1,Out2}, STr = {STr1,STr2},
    outputBranch(Bm, W,D, Out1, S,STr1),
    outputArgs(~(M+1),N, B, [0|W],D, Out2, S,STr2).

outputBranch(refuted(CL,Env,Parent), Width,Depth, Out, S,STr) :- true |
    reverse(Width, RWidth),
    meta#get_kl1_term(CL,Env, WT),
    PrintArgs = [{RWidth,Depth},Parent,WT],
    Out = [putf("~t:~t ~w     !! Refuted !!~n",PrintArgs)],
    STr = [{Depth,[PrintArgs|S]}].
otherwise.
outputBranch({CL,Env,Parent,Trace}, Width,Depth, Out, S,STr) :- true |
    reverse(Width,RWidth),
    reverse(CL, RCL),
    meta#get_kl1_term(RCL,Env, WT),
    PrintArgs = [{RWidth,Depth},Parent,WT],
    Out = [putf("~t:~t ~w~n",PrintArgs)|Out1],
    output(Trace, Width,Depth, Out1, [PrintArgs|S],STr).

%
%   library
%
append([],Y, Z) :- true | Z = Y.
append([A|X],Y, Z) :- true | Z = [A|Z1], append(X,Y, Z1).

reverse(X, Y) :- true | reverse(X,[], Y).

reverse([],S, Y) :- true | Y = S.
reverse([A|X],S, Y) :- true | reverse(X,[A|S], Y).

member(_,[], YN) :- true | YN = no.
member(L,[L|_], YN) :- true | YN = yes.
otherwise.
member(L,[_|Ls], YN) :- true | member(L,Ls, YN).

member(_,[], _, Type, YN) :- Type = equal | YN = no.
%member(_,[], _, Type, YN) :- Type = one_way(_) | YN = {}.
member(L,[L1|Ls], Env, Type, YN) :- Type = equal |
    meta#equal(L,L1, Env, YN0),
    memberDecide(YN0, L,Ls, Env, Type, YN).
%member(L,[L1|Ls], Env, Type, YN) :- Type = one_way(Tmp) |
%    meta#oneway_unify_tmp(L,L1, Tmp,Env, NTmpEnv),
%    memberDecide(NTmpEnv, L,Ls, Env, Type, YN).

memberDecide(yes, _,_, _, _, YN) :- true | YN = yes.
```

```
memberDecide(no, L,Ls, Env, Type, YN) :- true | member(L,Ls, Env, Type, YN).
memberDecide(TmpEnv, _,_, _, _, YN) :- TmpEnv = {_,_} | YN = TmpEnv.
memberDecide(fail, L,Ls, Env, Type, YN) :- true | member(L,Ls, Env, Type, YN).

complement(-(L),L1, Env, YN) :- true | complement1(L1,L, Env, YN).
complement(L1,-(L), Env, YN) :- true | complement1(L1,L, Env, YN).
otherwise.
complement(_,_, Env, YN) :- true | YN = no(Env).

complement1(-(_),_, Env, YN) :- true | YN = no(Env).
otherwise.
complement1(L,L1, Env, YN) :- true | unifiable(L,L1, Env, YN).

unifiable(L,L1, Env, YN) :- true |
    meta#unify_oc(L,L1, Env,NEnv),
    (NEnv = fail    -> YN = no(Env);
    otherwise;
     true   -> YN = yes(NEnv)).

nextTerm(InA,InZ, WT, OutA,OutZ) :- true |
    InA = [getwt(WT0)|InB],
    nextTermDecide(WT0,WT, InB,InZ, OutA,OutZ).

nextTermDecide(WT,NWT, InA,InZ, OutA,OutZ) :- WT = abnormal(ErrorInfo) |
    OutA = [print_error(ErrorInfo)|OutB],
    nextTerm(InA,InZ, NWT, OutB,OutZ).
otherwise.
nextTermDecide(WT,NWT, InA,InZ, OutA,OutZ) :- true |
    NWT = WT, InA = InZ, OutA = OutZ.

getProp(-(L), PM,FA) :- true | PM = '-', getProp(L, FA).
otherwise.
getProp(L, PM,FA) :- true | PM = '+', getProp(L, FA).

getProp(L, FA) :- atom(L) | FA = {L,0}.
getProp(L, FA) :- vector(L,Size), vector_element(L,0,F) | FA = {F,~(Size-1)}.

reverseSign('+', PM) :- true | PM = '-'.
reverseSign('-', PM) :- true | PM = '+'.


:- module 'METop'.
:- public clause/1.
clause(CL) :- true | CL = {}.

:- module 'MEDatabase'.
:- public get/3.
get(_,_, ExpEnv) :- true | ExpEnv = {}.
```

プログラムは前処理部分 compile と主要部分 refute からなる.

1. 前処理部分. 標準入力より節を読み込んで標準出力より KL1 プログラムを書き出す. 生成される KL1 プログラムは 'OLTop':clause/1,'OLDabase':get/3 である. 最初の入力節が top 節となる.

   【例】節 $P(x) \vee Q(f(x)) \vee \neg R(a)$ は [p(X),q(f(X)),-r(a)] と表現する.

2. 主要部分. 与えられた節を center 節として手続きを進めていく. refute/7 が最上位ループである. 枠つきリテラルは '$'(Literal) と '$' で包まれている.

refute/7 他の or 分岐した refute/7 が Model 消去成功したら手続きを止める (第一節). 予め指定された深さまで探索が進んだ場合一時探索を中断する (第二節). モニタ monitor/4 から探索続行指令が届いた場合は探索を続行する (refuteWait/8 の第二節). 以上以外の場合, extention を試み extend/3 全ての可能性について refute/7 をフォークする (第四節).

reduceContract/3 Extention で得られた節に対して contraction と reduction するのがこの述語である. Contraction の仕方は一意なのに対し, reduction の仕方は色々あることに注意されたい.

reduce/5 Center 節の右端リテラル L で節が reduction 可能か調べ可能であれば reduction し, 結果を差分リスト A,Z で返す. Reduction した場合, さらに reduction または contraction できる場合もあるので再帰する必要がある (reduceGenerate/4).

B.2.2 使用例

1. リスナより前処理プログラムを起動する.

```
?- 'ME':compile <= file("sample.in") => file(""sample.kl1").
 54590 reductions
 8464 msec

?-
```

前処理プログラムは節の集合を入力とし KL1 プログラムを生成する. 得られたプログラムは節の検索に利用される.

【例】上記例で次のような "sample.in" を入力すると

```
:::::::::::::::
sample.in
:::::::::::::::
% Top Clause
[-d(a,b)].

% Other Clauses
[m(a,s(c),s(b))].
[p(a)].
[m(X,X,s(X))].
[-m(X,Y,Z),m(Y,X,Z)].
[-m(X,Y,Z),d(X,Z)].
[-p(X),-m(Y,Z,U),-d(X,U),d(X,Y),d(X,Z)].
```

"sample.kl1" として次の KL1 プログラムが得られる.

```
:::::::::::::::
sample.kl1
:::::::::::::::
:- module 'METop'.
:- public clause/1.

clause(Top) :- true |
     Top = {CL,Env},
     CL = [- d(a,b)],     meta#create_env(Env,0,1).
```

```
:- module('MEDatabase') .
:- public get/2,get/3 .
get(Key, ExpEnv) :- Key = m(3)+ 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,3,4),
     Exp = ({-4,[- m({1},{2},{3}),m({2},{1},{3})]}).
get(Key, ExpEnv) :- Key = p(1)+ 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,4,5),
     Exp = ({-6,[- p({1}),- m({2},{3},{4}),- d({1},{4}),d({1},{2}),d({1},{3})]}).
get(Key, ExpEnv) :- Key = m(3)- 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,0,1),
     Exp = ({-1,[m(a,s(c),s(b))]}).
get(Key, ExpEnv) :- Key = p(1)- 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,0,1),
     Exp = ({-2,[p(a)]}).
get(Key, ExpEnv) :- Key = m(3)+ 2 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,4,5),
     Exp = ({-6,[- p({1}),- m({2},{3},{4}),- d({1},{4}),d({1},{2}),d({1},{3})]}).
get(Key, ExpEnv) :- Key = d(2)+ 1 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,4,5),
     Exp = ({-6,[- p({1}),- m({2},{3},{4}),- d({1},{4}),d({1},{2}),d({1},{3})]}).
get(Key, ExpEnv) :- Key = m(3)- 2 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,3,4),
     Exp = ({-4,[- m({1},{2},{3}),m({2},{1},{3})]}).
get(Key, ExpEnv) :- Key = d(2)- 1 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,4,5),
     Exp = ({-6,[- p({1}),- m({2},{3},{4}),- d({1},{4}),d({1},{2}),d({1},{3})]}).
get(Key, ExpEnv) :- Key = d(2)+ 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,0,1),
     Exp = ({0,[- d(a,b)]}).
get(Key, ExpEnv) :- Key = m(3)+ 1 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,3,4),
     Exp = ({-5,[- m({1},{2},{3}),d({1},{3})]}).
get(Key, ExpEnv) :- Key = m(3)- 1 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,1,2),
     Exp = ({-3,[m({1},{1},s({1}))]}).
get(Key, ExpEnv) :- Key = d(2)- 0 |
     ExpEnv = {Exp,Env},
     meta#create_env(Env,3,4),
     Exp = ({-5,[- m({1},{2},{3}),d({1},{3})]}).
otherwise.
get(_, ExpEnv) :- true | ExpEnv = {}.

get(Key,Env, ExpEnv) :- Key = m(3)+ 0, vector_element(Env,0,X1)
,    X2 := X1+1
,    X3 := X2+1
```

```
  ,    X4 := X3+1
   |
       getExtendEnv(X4, Env,Env1),
       Exp = ({-4,[- m({X1},{X2},{X3}),m({X2},{X1},{X3})]}),
       ExpEnv = {Exp,NEnv},
       set_vector_element(Env1,0, _,X4, NEnv).
 get(Key,Env, ExpEnv) :- Key = p(1)+ 0, vector_element(Env,0,X1)
  ,    X2 := X1+1
  ,    X3 := X2+1
  ,    X4 := X3+1
  ,    X5 := X4+1
   |
       getExtendEnv(X5, Env,Env1),
       Exp = ({-6,[- p({X1}),- m({X2},{X3},{X4}),- d({X1},{X4}),d({X1},{X2}),d({X1},{X3})]}),
       ExpEnv = {Exp,NEnv},
       set_vector_element(Env1,0, _,X5, NEnv).
 get(Key,Env, ExpEnv) :- Key = m(3)- 0 |
       Exp = ({-1,[m(a,s(c),s(b))]}),
       ExpEnv = {Exp,Env}.
 get(Key,Env, ExpEnv) :- Key = p(1)- 0 |
       Exp = ({-2,[p(a)]}),
       ExpEnv = {Exp,Env}.
 get(Key,Env, ExpEnv) :- Key = m(3)+ 2, vector_element(Env,0,X1)
  ,    X2 := X1+1
  ,    X3 := X2+1
  ,    X4 := X3+1
  ,    X5 := X4+1
   |
       getExtendEnv(X5, Env,Env1),
       Exp = ({-6,[- p({X1}),- m({X2},{X3},{X4}),- d({X1},{X4}),d({X1},{X2}),d({X1},{X3})]}),
       ExpEnv = {Exp,NEnv},
       set_vector_element(Env1,0, _,X5, NEnv).
 get(Key,Env, ExpEnv) :- Key = d(2)+ 1, vector_element(Env,0,X1)
  ,    X2 := X1+1
  ,    X3 := X2+1
  ,    X4 := X3+1
  ,    X5 := X4+1
   |
       getExtendEnv(X5, Env,Env1),
       Exp = ({-6,[- p({X1}),- m({X2},{X3},{X4}),- d({X1},{X4}),d({X1},{X2}),d({X1},{X3})]}),
       ExpEnv = {Exp,NEnv},
       set_vector_element(Env1,0, _,X5, NEnv).
 get(Key,Env, ExpEnv) :- Key = m(3)- 2, vector_element(Env,0,X1)
  ,    X2 := X1+1
  ,    X3 := X2+1
  ,    X4 := X3+1
   |
       getExtendEnv(X4, Env,Env1),
       Exp = ({-4,[- m({X1},{X2},{X3}),m({X2},{X1},{X3})]}),
       ExpEnv = {Exp,NEnv},
       set_vector_element(Env1,0, _,X4, NEnv).
 get(Key,Env, ExpEnv) :- Key = d(2)- 1, vector_element(Env,0,X1)
  ,    X2 := X1+1
  ,    X3 := X2+1
  ,    X4 := X3+1
  ,    X5 := X4+1
```

```
 |
      getExtendEnv(X5, Env,Env1),
      Exp = ({-6,[- p({X1}),- m({X2},{X3},{X4}),- d({X1},{X4}),d({X1},{X2}),d({X1},{X3})]}),
      ExpEnv = {Exp,NEnv},
      set_vector_element(Env1,0, _,X5, NEnv).
get(Key,Env, ExpEnv) :- Key = d(2)+ 0 |
      Exp = ({0,[- d(a,b)]}),
      ExpEnv = {Exp,Env}.
get(Key,Env, ExpEnv) :- Key = m(3)+ 1, vector_element(Env,0,X1)
,     X2 := X1+1
,     X3 := X2+1
,     X4 := X3+1
 |
      getExtendEnv(X4, Env,Env1),
      Exp = ({-5,[- m({X1},{X2},{X3}),d({X1},{X3})]}),
      ExpEnv = {Exp,NEnv},
      set_vector_element(Env1,0, _,X4, NEnv).
get(Key,Env, ExpEnv) :- Key = m(3)- 1, vector_element(Env,0,X1)
,     X2 := X1+1
 |
      getExtendEnv(X2, Env,Env1),
      Exp = ({-3,[m({X1},{X1},s({X1}))]}),
      ExpEnv = {Exp,NEnv},
      set_vector_element(Env1,0, _,X2, NEnv).
get(Key,Env, ExpEnv) :- Key = d(2)- 0, vector_element(Env,0,X1)
,     X2 := X1+1
,     X3 := X2+1
,     X4 := X3+1
 |
      getExtendEnv(X4, Env,Env1),
      Exp = ({-5,[- m({X1},{X2},{X3}),d({X1},{X3})]}),
      ExpEnv = {Exp,NEnv},
      set_vector_element(Env1,0, _,X4, NEnv).
otherwise.
get(_,_, ExpEnv) :- true | ExpEnv = {}.

getExtendEnv(Top, Env,NEnv) :- vector(Env,Size), Top =< Size |
      NEnv = Env.
getExtendEnv(Top, Env,NEnv) :- vector(Env,Size), Top > Size |
      meta#create_env(Env0,Top,~(Top+1)),
      meta#set_subenv(Env0,1,Env, NEnv).
```

2. メタライブラリ用マクロファイルを指定する[3].

```
?- setenv(macro: initial_file,">sys>user>koshimura>kl1>Metalib>Macro>meta.mac",X).
 X = normal
 25 reductions
 124 msec

?-
```

3. 出力ファイルをコンパイルし,再リンクする.

```
?- compile("sample.kl1").
```

[3] この機能は PIMOS 2.1 版より使用化. 使用できない場合は A.3 簡易コンパイラを使用すること

```
** KL1 Compiler **

Compile File : icpsi167::>sys>user>koshimura>kl1>Prover>ME>sample.kl1.1

Compile Module : METop
clause/ 1
Compile Succeeded : METop

Compile Module : MEDatabase
get/ 2
get/ 3
getExtendEnv/ 3
Compile Succeeded : MEDatabase

!WARNING! 'with_macro' declaration not found
"miyuki::MEDatabase" Updated
"miyuki::METop" Updated
Compilation Time = 42542 [MSEC]

Total Number of Warning : 1
 811495 reductions
 45129 msec

?- relink(['ME']).
"miyuki::ME" Updated

Relink Succeeded
 10293 reductions
 1713 msec

?-
```

4. Prover を起動する.

```
?- 'ME':go.


----- Proof -----
{[1,0],1}:-6, [$(- d(a,b)),- p(a),- m(b,b,_4),- d(a,_4)]
{[1,0],2}:-5, [$(- d(a,b)),- p(a),- m(b,b,_4),$(- d(a,_4)),- m(a,_6,_4)]
{[1,0],3}:-1, [$(- d(a,b)),- p(a),- m(b,b,s(b))]
{[1,0],4}:-3, [$(- d(a,b)),- p(a)]
{[1,0],5}:-2, []
----- End of Proof -----
 187915 reductions
 5936 msec

?-
```

5. 探索の深さを変えて起動してみる[1].

[1] PE では探索は depth-first, left-to-right に行われるため, 探索空間の刈り込みは実際上行われない. 浅い深さの探索で解が見つかる場合は, 深さの指定を浅くして探索すると速く解が見つかる場合がある.

```
?- 'ME':go(10,10).


----- Proof -----
{[1,0],1}:-6, [$(- d(a,b)),- p(a),- m(b,b,_4),- d(a,_4)]
{[1,0],2}:-5, [$(- d(a,b)),- p(a),- m(b,b,_4),$(- d(a,_4)),- m(a,_6,_4)]
{[1,0],3}:-1, [$(- d(a,b)),- p(a),- m(b,b,s(b))]
{[1,0],4}:-3, [$(- d(a,b)),- p(a)]
{[1,0],5}:-2, []
----- End of Proof -----
 16814 reductions
 1451 msec

?-
```

C メタライブラリのソースコード

メタライブラリはデータの処理方式によって構造体共有方式と構造体コピー方式の 2 通りの方式がある. 構造体共有方式ではユニファイされたオブジェクトは必ず共有されるのに対し, 構造体コピー方式では必ずしも共有されるとは限らない.

【例】環境が $\{X \to a, Y \to a\}$ の時, $X = Y$ を実行すると, 構造体共有方式では環境が $\{X \to Y, Y \to a\}$ となるのに対し, 構造体コピー方式では環境は変化しない.

【参考】メタライブラリは, 構造体コピー方式が標準である.

C.1 構造体コピー方式のメタライブラリプログラム

```
::::::::::::::
copyTerm.kl1
::::::::::::::
:- module copy_term.
:- public copy_term/4,copy/7.

copy_term(X,NX, Env,NEnv) :- vector(Env,Size) |
    new_vector(GEnv0, Size),
    copy(X,NX, [], 1, Env, GEnv0,NEnv).

copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- atom(X) |
    NX = X, copyCont(Cont, Max, Env, GEnv,NGEnv).
copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- integer(X) |
    NX = X, copyCont(Cont, Max, Env, GEnv,NGEnv).
copy({Num},NVar, Cont, Max, Env, GEnv,NGEnv) :- integer(Num) |
    copyVar(Num,NVar, Cont, Max, Env, GEnv,NGEnv).
copy([H|T],NList, Cont, Max, Env, GEnv,NGEnv) :- true |
    NList = [NH|NT],
    copy(H,NH, [{T,NT}|Cont], Max, Env, GEnv,NGEnv).
copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- vector(X, Size), Size > 1 |
    copyCont([{1,Size, X,NX}|Cont], Max, Env, GEnv,NGEnv).

copyVar(Num,NVar, Cont, Max, Env, GEnv,NGEnv) :- true |
    copyVar(Num,NVar, _, Cont, Max, Env, GEnv,NGEnv).

copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :- vector_element(Env,Num,0) |
```

```
        NVar = {Max},
        S = markPoint(Max),
        set_vector_element(GEnv,Max, _,0, GEnv1),
        set_vector_element(Env,Num, _,S, Env1),
        copyCont(Cont, ~(Max+1), Env1, GEnv1,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,point(Ref)) |
        set_vector_element(Env,Num, _,S, Env1),
        copyVar(Ref,NVar, S, Cont, Max, Env1, GEnv,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,data(Data)) |
        NVar = NData,
        S = markData(NData),
        set_vector_element(Env,Num, _,S, Env1),
        copy(Data,NData, Cont, Max, Env1, GEnv,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,markPoint(Ref)) |
        NVar = {Ref},
        S = markPoint(Ref),
        copyCont(Cont, Max, Env, GEnv,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,markData(Data)) |
        NVar = Data,
        S = markData(Data),
        copyCont(Cont, Max, Env, GEnv,NGEnv).

    copyCont([], Max, _, GEnv,NGEnv) :- true |
        set_vector_element(GEnv,0, _,Max, NGEnv).
    copyCont([{X,NX}|Cont], Max, Env, GEnv,NGEnv) :- true |
        copy(X,NX, Cont, Max, Env, GEnv,NGEnv).
    copyCont([{N,N,X,NX}|Cont], Max, Env, GEnv,NGEnv) :- true |
        NX = X,
        copyCont(Cont, Max, Env, GEnv,NGEnv).
    copyCont([{M,N,X,NX}|Cont], Max, Env, GEnv,NGEnv) :- M < N |
        set_vector_element(X,M, Xm,NXm, X1),
        copy(Xm,NXm, [{~(M+1),N, X1,NX}|Cont], Max, Env, GEnv,NGEnv).
    ::::::::::::::
    match.kl1
    ::::::::::::::
    :- module match.
    :- public match/4,match/5.

    %
    %   match(+Pattern,+Target, +Env,-Env)
    %
    match(X,Y, Env,NEnv) :- true | match(X,Y, [], Env,NEnv).

    %
    %   match(+Pattern,+Target, +Continuation, +Env,-Env)
    %
    match({M},Y, Cont, Env,NEnv) :- integer(M) | matchVarL(M,Y, Cont, Env,NEnv).
    match(X,{M}, Cont, Env,NEnv) :- integer(M) | matchVarR(X,M, Cont, Env,NEnv).
    match([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
        match(XH,YH, [{XT,YT}|Cont], Env,NEnv).
    match(X,Y, Cont, Env,NEnv) :- vector(X,Size), vector(Y,Size), Size > 1,
        vector_element(X,0,F), vector_element(Y,0,F) |
```

```
    matchCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
match(X,X, Cont, Env,NEnv) :- true | matchCont(Cont, Env,NEnv).
otherwise.
match(X,Y, _, _,NEnv) :- true | NEnv = fail.

matchCont([], Env,NEnv) :- true | NEnv = Env.
matchCont([{X,Y}|Cont], Env,NEnv) :- true | match(X,Y, Cont, Env,NEnv).
matchCont([{M,M,X,Y}|Cont], Env,NEnv) :- true | matchCont(Cont, Env,NEnv).
matchCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M,Xm), vector_element(Y,M,Ym) |
    match(Xm,Ym, [{M1,N,X,Y}|Cont], Env,NEnv).

matchVarL(XNum,{YNum}, Cont, Env,NEnv) :- integer(YNum) |
    matchVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
matchVarL(XNum,Y, Cont, Env,NEnv) :- true | matchVarStr(XNum,Y, Cont, Env,NEnv).

matchVarR({XNum},YNum, Cont, Env,NEnv) :- integer(XNum) |
    matchVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
matchVarR(X,YNum, Cont, Env,NEnv) :- true | matchStrVar(X,YNum, Cont, Env,NEnv).

matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarVar(XRef,YNum, Cont, Env,NEnv).
matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    matchStrVar(XVal,YNum, Cont, Env,NEnv).
matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    matchVarVar(XNum,YRef, Cont, Env,NEnv).
matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,data(YVal)) |
    matchVarStr(XNum,YVal, Cont, Env,NEnv).
matchVarVar(XNum,YNum, _, Env,NEnv) :-  % User's Error
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    NEnv = fail.

matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarStr(XRef,Y, Cont, Env,NEnv).
matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    match(XVal,Y, Cont, Env,NEnv).
matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    set_vector_element(Env, XNum, _,data(Y), Env1),
    matchCont(Cont, Env1,NEnv).

matchStrVar(X,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    matchStrVar(X,YRef, Cont, Env,NEnv).
matchStrVar(X,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,data(YVal)) |
    match(X,YVal, Cont, Env,NEnv).
matchStrVar(_,YNum, _, Env,NEnv) :- vector_element(Env,YNum,0) | % User's Error
    NEnv = fail.
:::::::::::::::
matchOne.kl1
:::::::::::::::
:- module oneway_unify.
:- public match/4,match/7,pointReal/3.

%
```

```
%   match(+Pattern,+Target, +Env,-Env)
%
match(X,Y, Env,NEnv) :- true | match(X,Y, normal,[],[], Env,NEnv).


%
%   match(+Pattern,+Target, +Position,+ListOfTmpPoint,+Continuation, +Env,-Env)
%
match({M},Y, Pos,Tmp,Cont, Env,NEnv) :- integer(M) |
    matchVarL(M,Y, Pos,Tmp,Cont, Env,NEnv).
match(X,{M}, Pos,Tmp,Cont, Env,NEnv) :- integer(M) |
    matchVarR(X,M, Pos,Tmp,Cont, Env,NEnv).
match([XH|XT],[YH|YT], Pos,Tmp,Cont, Env,NEnv) :- true |
    match(XH,YH, Pos,Tmp,[{XT,YT,Pos}|Cont], Env,NEnv).
match(X,Y, Pos,Tmp,Cont, Env,NEnv) :- vector(X,Size), vector(Y,Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    matchCont([{1,Size,X,Y,Pos}|Cont], Tmp, Env,NEnv).
otherwise.
match(X,X, _,Tmp,Cont, Env,NEnv) :- true | matchCont(Cont, Tmp, Env,NEnv).
otherwise.
match(X,Y, _,_,_, _,NEnv) :- true | NEnv = fail.

matchCont([tmp], Tmp, Env,NTmpEnv) :- true | NTmpEnv = {Tmp,Env}.
matchCont([], Tmp, Env,NEnv) :- true | pointReal(Tmp, Env,NEnv).
matchCont([{X,Y,Pos}|Cont], Tmp, Env,NEnv) :- true |
    match(X,Y, Pos,Tmp,Cont, Env,NEnv).
matchCont([{M,M,X,Y,Pos}|Cont], Tmp, Env,NEnv) :- true |
    matchCont(Cont, Tmp, Env,NEnv).
matchCont([{M,N,X,Y,Pos}|Cont], Tmp, Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M,Xm), vector_element(Y,M,Ym) |
    match(Xm,Ym, Pos,Tmp,[{M1,N,X,Y,Pos}|Cont], Env,NEnv).

pointReal([], Env,NEnv) :- true | NEnv = Env.
pointReal([Num|NumS], Env,NEnv) :- true |
    set_vector_element(Env,Num, Old,New, Env1),
    pointRealDecide(Old,New, NumS, Env1,NEnv).

pointRealDecide(pseudoPoint(Ref), New, NumS, Env,NEnv) :- true |
    New = point(Ref), pointReal(NumS, Env,NEnv).
pointRealDecide(pseudoData(Ref), New, NumS, Env,NEnv) :- true |
    New = data(Ref), pointReal(NumS, Env,NEnv).

matchVarL(XNum,{YNum}, Pos,Tmp,Cont, Env,NEnv) :- integer(YNum) |
    matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv).
otherwise.
matchVarL(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- true |
    matchVarStr(XNum,Y, Pos,Tmp,Cont, Env,NEnv).

matchVarR({XNum},YNum, Pos,Tmp,Cont, Env,NEnv) :- integer(XNum) |
    matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv).
otherwise.
matchVarR(X,YNum, Pos,Tmp,Cont, Env,NEnv) :- true |
    matchStrVar(X,YNum, Pos,Tmp,Cont, Env,NEnv).

matchVarVar(XNum,XNum, _,Tmp,Cont, Env,NEnv) :- true |
    matchCont(Cont, Tmp, Env,NEnv).
```

```
otherwise.
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,point(XRef)) |
    matchVarVar(XRef,YNum, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,data(XVal)) |
    matchStrVar(XVal,YNum, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,point(YRef)) |
    matchVarVar(XNum,YRef, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,data(YVal)) |
    matchVarStr(XNum,YVal, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoPoint(XRef)) |
    matchVarVar(XRef,YNum, abnormal,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoData(XVal)) |
    matchStrVar(XVal,YNum, abnormal,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    set_vector_element(Env, XNum, _,pseudoPoint(YNum), Env1),
    matchCont(Cont, [XNum|Tmp], Env1,NEnv).
matchVarVar(XNum,YNum, abnormal,_,_, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    NEnv = fail.

matchVarStr(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarStr(XRef,Y, Pos,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    match(XVal,Y, Pos,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoPoint(XRef)) |
    matchVarStr(XRef,Y, abnormal,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv):-
    vector_element(Env,XNum,pseudoData(XVal)) |
    match(XVal,Y, abnormal,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    set_vector_element(Env, XNum, _,pseudoData(Y), Env1),
    matchCont(Cont, [XNum|Tmp], Env1,NEnv).
matchVarStr(XNum,_, abnormal,_,_, Env,NEnv) :- vector_element(Env,XNum,0) |
    NEnv = fail.

matchStrVar(X,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,point(YRef)) |
    matchStrVar(X,YRef, Pos,Tmp,Cont, Env,NEnv).
matchStrVar(X,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,data(YVal)) |
    match(X,YVal, Pos,Tmp,Cont, Env,NEnv).
matchStrVar(_,YNum, _,_,_, Env,NEnv) :- vector_element(Env,YNum,0) |
    NEnv = fail.
:::::::::::::::
shallow.kl1
:::::::::::::::
:- module shallow.
:- public shallow/3.
```

```
shallow(X, Env,NEnv) :- vector(Env,Size) |
    new_string(Mark, Size, 1), shallow(X, [], Mark, Env,NEnv).

shallow(X, Cont, Mark, Env,NEnv) :- atom(X) |
    shallowCont(Cont, Mark, Env,NEnv).
shallow(X, Cont, Mark, Env,NEnv) :- integer(X) |
    shallowCont(Cont, Mark, Env,NEnv).
shallow({Num}, Cont, Mark, Env,NEnv) :- integer(Num) |
    shallowVar(Num, Cont, Mark, Env,NEnv).
shallow([H|T], Cont, Mark, Env,NEnv) :- true |
    shallow(H, [{T}|Cont], Mark, Env,NEnv).
shallow(X, Cont, Mark, Env,NEnv) :- vector(X, Size), Size > 1 |
    shallowCont([{1,Size,X}|Cont], Mark, Env,NEnv).

shallowVar(Num, Cont, Mark, Env,NEnv) :- true |
    shallowVar(Num, _, Cont, Mark, Env,NEnv).

shallowVar(Num, S, Cont, Mark, Env,NEnv) :- string_element(Mark,Num,2#"1") |
    shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv).
shallowVar(Num, S, Cont, Mark, Env,NEnv) :- string_element(Mark,Num,2#"0") |
    shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv).

shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :- vector_element(Env,Num,0) |
    S = point(Num),
    shallowCont(Cont, Mark, Env,NEnv).
shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,data(Data)) |
    S = data(Data),
    shallowCont(Cont, Mark, Env,NEnv).
shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,point(Ref)) |
    S = point(Ref),
    shallowCont(Cont, Mark, Env,NEnv).

shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :- vector_element(Env,Num,0) |
    S = point(Num),
    set_string_element(Mark,Num,2#"1",NMark),
    shallowCont(Cont, NMark, Env,NEnv).
shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,data(Data)) |
    S = data(Data),
    set_string_element(Mark,Num,2#"1",NMark),
    shallow(Data, Cont, NMark, Env,NEnv).
shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,point(Ref)) |
    set_string_element(Mark,Num,2#"1",NMark),
    set_vector_element(Env,Num, _,S, Env1),
    shallowVar(Ref, S, Cont, NMark, Env1,NEnv).

shallowCont([], _, Env,NEnv) :- true | NEnv = Env.
shallowCont([{X}|Cont], Mark, Env,NEnv) :- true |
    shallow(X, Cont, Mark, Env,NEnv).
shallowCont([{M,M,X}|Cont], Mark, Env,NEnv) :- true |
    shallowCont(Cont, Mark, Env,NEnv).
shallowCont([{M,N,X}|Cont], Mark, Env,NEnv) :- M < N, vector_element(X,M, Xn) |
```

```
    shallow(Xn, [{^(M+1),N, X}|Cont], Mark, Env,NEnv).
:::::::::::::::
unify.kl1
:::::::::::::::
:- module unify.
:- public unify/4, unify/5.

%
%   unify(+X,+Y, +Env,-Env)
%
unify(X,Y, Env,NEnv) :- true | unify(X,Y, [], Env,NEnv).

unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unify(X,Y, _, _,NEnv) :- true | NEnv = fail.

unifyCont([], Env,NEnv) :- true | NEnv = Env.
unifyCont([{X,Y}|Cont], Env,NEnv) :- true | unify(X,Y, Cont, Env,NEnv).
unifyCont([{N,N,X,Y}|Cont], Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
unifyCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M, Xm), vector_element(Y,M, Ym) |
    unify(Xm,Ym, [{M1,N, X,Y}|Cont], Env,NEnv).

unifyVar(XNum, {YNum}, Cont, Env,NEnv) :- integer(YNum) |
    unifyVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
unifyVar(XNum, Y, Cont, Env,NEnv) :- true |
    unifyVarStr(XNum,Y, Cont, Env,NEnv).

unifyVarVar(XNum,XNum, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarVar(XRef,YNum, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    unifyVarStr(YNum,XVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    unifyVarVar(XNum,YRef, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,data(YVal)) |
    unifyVarStr(XNum,YVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont,Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    set_vector_element(Env,XNum, _,point(YNum), Env1),
    unifyCont(Cont, Env1,NEnv).

unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarStr(XRef,Y, Cont, Env,NEnv).
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    unify(XVal,Y, Cont, Env,NEnv).
```

```
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    set_vector_element(Env,XNum, _,data(Y),Env1),
    unifyCont(Cont, Env1,NEnv).
:::::::::::::::
unifyOc.kl1
:::::::::::::::
:- module unify_oc.
:- public unify/4, unify/5.

%
%   unify(+X,+Y, +Env,-Env)
%
unify(X,Y, Env,NEnv) :- true | unify(X,Y, [], Env,NEnv).

unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unify(X,Y, _, _,NEnv) :- true | NEnv = fail.

unifyCont([], Env,NEnv) :- true | NEnv = Env.
unifyCont([{X,Y}|Cont], Env,NEnv) :- true | unify(X,Y, Cont, Env,NEnv).
unifyCont([{N,N,X,Y}|Cont], Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
unifyCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M, Xm), vector_element(Y,M, Ym) |
    unify(Xm,Ym, [{M1,N, X,Y}|Cont], Env,NEnv).

unifyVar(XNum, {YNum}, Cont, Env,NEnv) :- integer(YNum) |
    unifyVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
unifyVar(XNum, Y, Cont, Env,NEnv) :- true |
    unifyVarStr(XNum,Y, Cont, Env,NEnv).

unifyVarVar(XNum,XNum, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarVar(XRef,YNum, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    unifyVarStr(YNum,XVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    unifyVarVar(XNum,YRef, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,data(YVal)) |
    unifyVarStr(XNum,YVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont,Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    set_vector_element(Env,XNum, _,point(YNum), Env1),
    unifyCont(Cont, Env1,NEnv).

unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarStr(XRef,Y, Cont, Env,NEnv).
```

```
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    unify(XVal,Y, Cont, Env,NEnv).
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    checkOccur(XNum,Y, [{}|Y],Cont, Env,NEnv).

checkOccurCont([{}|Y], XNum,Cont, Env,NEnv) :- true |
    set_vector_element(Env,XNum, _,data(Y), Env1),
    unifyCont(Cont, Env1,NEnv).
checkOccurCont([{Y}|OCont], XNum,Cont, Env,NEnv) :- true |
    checkOccur(XNum,Y, OCont,Cont, Env,NEnv).
checkOccurCont([{N,N,Y}|OCont], XNum,Cont, Env,NEnv) :- true |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).
checkOccurCont([{M,N,Y}|OCont], XNum,Cont, Env,NEnv) :-
    M < N, M1 := M + 1, vector_element(Y,M, Ym) |
    checkOccur(XNum,Ym, [{M1,N,Y}|OCont],Cont, Env,NEnv).

checkOccur(XNum,{YNum}, OCont,Cont, Env,NEnv) :- integer(YNum) |
    checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv).
checkOccur(XNum,[YH|YT], OCont,Cont, Env,NEnv) :- true |
    checkOccur(XNum,YH, [{YT}|OCont],Cont, Env,NEnv).
checkOccur(XNum,Y, OCont,Cont, Env,NEnv) :- vector(Y, Size), Size > 1 |
    checkOccurCont([{1,Size,Y}|OCont], XNum,Cont, Env,NEnv).
otherwise.
checkOccur(XNum,Y, OCont,Cont, Env,NEnv) :- true |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).

checkOccurVar(XNum,XNum, _,_, _,NEnv) :- true | NEnv = fail.
otherwise.
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :- vector_element(Env,YNum, 0) |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :-
    vector_element(Env,YNum, point(YRef)) |
    checkOccurVar(XNum,YRef, OCont,Cont, Env,NEnv).
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :-
    vector_element(Env,YNum, data(YData)) |
    checkOccur(XNum,YData, OCont,Cont, Env,NEnv).
```

C.2 構造体共有方式のメタライブラリプログラム

```
:::::::::::::::
copyTerm.kl1
:::::::::::::::
:- module copy_termShare.
:- public copy_term/4, copy/7.

copy_term(X,NX, Env,NEnv) :- vector(Env,Size) |
    new_vector(Env0, Size),
    copy(X,NX, [], 1, Env, Env0,NEnv).

copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- atom(X) |
    NX = X, copyCont(Cont, Max, Env, GEnv,NGEnv).
copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- integer(X) |
    NX = X, copyCont(Cont, Max, Env, GEnv,NGEnv).
copy({Num},NVar, Cont, Max, Env, GEnv,NGEnv) :- integer(Num) |
    copyVar(Num,NVar, Cont, Max, Env, GEnv,NGEnv).
copy([H|T],NList, Cont, Max, Env, GEnv,NGEnv) :- true |
```

```
        NList = [NH|NT],
        copy(H,NH, [{T,NT}|Cont], Max, Env, GEnv,NGEnv).
    copy(X,NX, Cont, Max, Env, GEnv,NGEnv) :- vector(X, Size), Size > 1 |
        copyCont([{1,Size, X,NX}|Cont], Max, Env, GEnv,NGEnv).

    copyVar(Num,NVar, Cont, Max, Env, GEnv,NGEnv) :- true |
        copyVar(Num,NVar, _, Cont, Max, Env, GEnv,NGEnv).

    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,0) |
        NVar = {Max},
        S = markPoint(Max),
        set_vector_element(GEnv,Max, _,0, GEnv1),
        set_vector_element(Env,Num, _,S, Env1),
        copyCont(Cont, ~(Max+1), Env1, GEnv1,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,point(Ref)) |
        set_vector_element(Env,Num, _,S, Env1),
        copyVar(Ref,NVar, S, Cont, Max, Env1, GEnv,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,data(Data)) |
        NVar = {Max},
        S = markPoint(Max),
        set_vector_element(GEnv,Max, _,data(NData), GEnv1),
        set_vector_element(Env,Num, _,S, Env1),
        copy(Data,NData, Cont, ~(Max+1), Env1, GEnv1,NGEnv).
    copyVar(Num,NVar, S, Cont, Max, Env, GEnv,NGEnv) :-
        vector_element(Env,Num,markPoint(Ref)) |
        NVar = {Ref},
        S = markPoint(Ref),
        copyCont(Cont, Max, Env, GEnv,NGEnv).

    copyCont([], Max, _, GEnv,NGEnv) :- true |
        set_vector_element(GEnv,0, _,Max, NGEnv).
    copyCont([{X,NX}|Cont], Max, Env, GEnv,NGEnv) :- true |
        copy(X,NX, Cont, Max, Env, GEnv,NGEnv).
    copyCont([{N,N,X,NX}|Cont], Max, Env, GEnv,NGEnv) :- true |
        NX = X,
        copyCont(Cont, Max, Env, GEnv,NGEnv).
    copyCont([{M,N,X,NX}|Cont], Max, Env, GEnv,NGEnv) :- M < N |
        set_vector_element(X,M, Xm,NXm, X1),
        copy(Xm,NXm, [{~(M+1),N, X1,NX}|Cont], Max, Env, GEnv,NGEnv).
::::::::::::::
match.kl1
::::::::::::::
:- module matchShare.
:- public match/4,match/5.

%
%   match(+Pattern,+Target, +Env,-Env)
%
match(X,Y, Env,NEnv) :- true | match(X,Y, [], Env,NEnv).

%
%   match(+Pattern,+Target, +Continuation, +Env,-Env)
%
```

```
match({M},Y, Cont, Env,NEnv) :- integer(M) | matchVarL(M,Y, Cont, Env,NEnv).
match(X,{M}, Cont, Env,NEnv) :- integer(M) | matchVarR(X,M, Cont, Env,NEnv).
match([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    match(XH,YH, [{XT,YT}|Cont], Env,NEnv).
match(X,Y, Cont, Env,NEnv) :- vector(X,Size), vector(Y,Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    matchCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
match(X,X, Cont, Env,NEnv) :- true | matchCont(Cont, Env,NEnv).
otherwise.
match(X,Y, _, _,NEnv) :- true | NEnv = fail.

matchCont([], Env,NEnv) :- true | NEnv = Env.
matchCont([{X,Y}|Cont], Env,NEnv) :- true | match(X,Y, Cont, Env,NEnv).
matchCont([{M,M,X,Y}|Cont], Env,NEnv) :- true | matchCont(Cont, Env,NEnv).
matchCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M,Xm), vector_element(Y,M,Ym) |
    match(Xm,Ym, [{M1,N,X,Y}|Cont], Env,NEnv).

matchVarL(XNum,{YNum}, Cont, Env,NEnv) :- integer(YNum) |
    matchVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
matchVarL(XNum,Y, Cont, Env,NEnv) :- true | matchVarStr(XNum,Y, Cont, Env,NEnv).

matchVarR({XNum},YNum, Cont, Env,NEnv) :- integer(XNum) |
    matchVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
matchVarR(X,YNum, Cont, Env,NEnv) :- true | matchStrVar(X,YNum, Cont, Env,NEnv).

matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarVar(XRef,YNum, Cont, Env,NEnv).
matchVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    matchVarVar(XNum,YRef, Cont, Env,NEnv).
matchVarVar(XNum,YNum, Cont, Env,NEnv) :-
    vector_element(Env,XNum,data(XVal)), vector_element(Env,YNum,0) |
    NEnv = fail.
matchVarVar(XNum,YNum, Cont, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,data(YVal)) |
    set_vector_element(Env,XNum, _,point(YNum), Env1),
    matchCont(Cont, Env1,NEnv).
matchVarVar(XNum,YNum, _, Env,NEnv) :-  % User's Error
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    NEnv = fail.

matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarStr(XRef,Y, Cont, Env,NEnv).
matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    match(XVal,Y, Cont, Env,NEnv).
matchVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    set_vector_element(Env, XNum, _,data(Y), Env1),
    matchCont(Cont, Env1,NEnv).

matchStrVar(X,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    matchStrVar(X,YRef, Cont, Env,NEnv).
matchStrVar(X,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,data(YVal)) |
    match(X,YVal, Cont, Env,NEnv).
```

```
matchStrVar(_,YNum, _, Env,NEnv) :- vector_element(Env,YNum,0) | % User's Error
    NEnv = fail.
:::::::::::::::
matchOne.kl1
:::::::::::::::
:- module oneway_unifyShare.
:- public match/4,match/7,pointReal/3.

%
%   match(+Pattern,+Target, +Env,-Env)
%
match(X,Y, Env,NEnv) :- true | match(X,Y, normal,[],[], Env,NEnv).

%
%   match(+Pattern,+Target, +Position,+ListOfTmpPoint,+Continuation, +Env,-Env)
%
match({M},Y, Pos,Tmp,Cont, Env,NEnv) :- integer(M) |
    matchVarL(M,Y, Pos,Tmp,Cont, Env,NEnv).
match(X,{M}, Pos,Tmp,Cont, Env,NEnv) :- integer(M) |
    matchVarR(X,M, Pos,Tmp,Cont, Env,NEnv).
match([XH|XT],[YH|YT], Pos,Tmp,Cont, Env,NEnv) :- true |
    match(XH,YH, Pos,Tmp,[{XT,YT,Pos}|Cont], Env,NEnv).
match(X,Y, Pos,Tmp,Cont, Env,NEnv) :- vector(X,Size), vector(Y,Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    matchCont([{1,Size,X,Y,Pos}|Cont], Tmp, Env,NEnv).
otherwise.
match(X,X, _,Tmp,Cont, Env,NEnv) :- true | matchCont(Cont, Tmp, Env,NEnv).
otherwise.
match(X,Y, _,_,_, _,NEnv) :- true | NEnv = fail.

matchCont([tmp], Tmp, Env,NTmpEnv) :- true | NTmpEnv = {Tmp,Env}.
matchCont([{X,Y,Pos}|Cont], Tmp, Env,NEnv) :- true |
    match(X,Y, Pos,Tmp,Cont, Env,NEnv).
matchCont([{M,M,X,Y,Pos}|Cont], Tmp, Env,NEnv) :- true |
    matchCont(Cont, Tmp, Env,NEnv).
matchCont([{M,N,X,Y,Pos}|Cont], Tmp, Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M,Xm), vector_element(Y,M,Ym) |
    match(Xm,Ym, Pos,Tmp,[{M1,N,X,Y,Pos}|Cont], Env,NEnv).

pointReal([], Env,NEnv) :- true | NEnv = Env.
pointReal([Num|NumS], Env,NEnv) :- true |
    set_vector_element(Env,Num, Old,New, Env1),
    pointRealDecide(Old,New, NumS, Env1,NEnv).

pointRealDecide(pseudoPoint(Ref), New, NumS, Env,NEnv) :- true |
    New = point(Ref), pointReal(NumS, Env,NEnv).
pointRealDecide(pseudoData(Ref), New, NumS, Env,NEnv) :- true |
    New = data(Ref), pointReal(NumS, Env,NEnv).
otherwise.
pointRealDecide(Old, New, NumS, Env,NEnv) :- true |
    New = Old, pointReal(NumS, Env,NEnv).

matchVarL(XNum,{YNum}, Pos,Tmp,Cont, Env,NEnv) :- integer(YNum) |
    matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv).
otherwise.
matchVarL(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- true |
```

```
    matchVarStr(XNum,{Y}, Pos,Tmp,Cont, Env,NEnv).

matchVarR({XNum},YNum, Pos,Tmp,Cont, Env,NEnv) :- integer(XNum) |
    matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv).
otherwise.
matchVarR(X,YNum, Pos,Tmp,Cont, Env,NEnv) :- true |
    matchStrVar({X},YNum, Pos,Tmp,Cont, Env,NEnv).

matchVarVar(XNum,XNum, _,Tmp,Cont, Env,NEnv) :- true |
    matchCont(Cont, Tmp, Env,NEnv).
otherwise.
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,point(XRef)) |
    matchVarVar(XRef,YNum, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,data(XVal)) |
    matchStrVar({XNum,XVal},YNum, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,point(YRef)) |
    matchVarVar(XNum,YRef, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,data(YVal)) |
    matchVarStr(XNum,{YNum,YVal}, Pos,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoPoint(XRef)) |
    matchVarVar(XRef,YNum, abnormal,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoData(XVal)) |
    matchStrVar({XNum,XVal},YNum, abnormal,Tmp,Cont, Env,NEnv).
matchVarVar(XNum,YNum, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    set_vector_element(Env, XNum, _,pseudoPoint(YNum), Env1),
    matchCont(Cont, [XNum|Tmp], Env1,NEnv).
matchVarVar(XNum,YNum, abnormal,_,_, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    NEnv = fail.

matchVarStr(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    matchVarStr(XRef,Y, Pos,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, Pos,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    (Y = {YVal}  -> match(XVal,YVal, Pos,Tmp,Cont, Env,NEnv);
     Y = {YNum,YVal}    ->  set_vector_element(Env,XNum, _,pseudoPoint(YNum), Env1),
        match(XVal,YVal, Pos,[XNum|Tmp],Cont, Env1,NEnv)).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,XNum,pseudoPoint(XRef)) |
    matchVarStr(XRef,Y, abnormal,Tmp,Cont, Env,NEnv).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv):-
    vector_element(Env,XNum,pseudoData(XVal)) |
    (Y = {YVal} -> match(XVal,YVal, abnormal,Tmp,Cont, Env,NEnv);
     Y = {YNum,YVal}    ->  set_vector_element(Env,XNum, _,pseudoPoint(YNum), Env1),
        match(XVal,YVal, abnormal,[XNum|Tmp],Cont, Env1,NEnv)).
matchVarStr(XNum,Y, normal,Tmp,Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    (Y = {YVal} -> set_vector_element(Env, XNum, _,pseudoData(YVal), Env1),
        matchCont(Cont, [XNum|Tmp], Env1,NEnv);
     Y = {YNum,YVal} -> set_vector_element(Env, XNum, _,pseudoPoint(YNum), Env1),
        matchCont(Cont, [XNum|Tmp], Env1,NEnv)).
```

```
matchVarStr(XNum,_, abnormal,_,_, Env,NEnv) :- vector_element(Env,XNum,0) |
    NEnv = fail.

matchStrVar(X,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,point(YRef)) |
    matchStrVar(X,YRef, Pos,Tmp,Cont, Env,NEnv).
matchStrVar(X,YNum, Pos,Tmp,Cont, Env,NEnv) :-
    vector_element(Env,YNum,data(YVal)) |
    (X = {XVal} -> match(XVal,YVal, Pos,Tmp,Cont, Env,NEnv);
     X = {XNum,XVal} -> set_vector_element(Env,XNum, _,pseudoPoint(YNum), Env1),
        match(XVal,YVal, Pos,[XNum|Tmp],Cont, Env1,NEnv)).
matchStrVar(_,YNum, _,_,_, Env,NEnv) :- vector_element(Env,YNum,0) |
    NEnv = fail.
:::::::::::::::
shallow.kl1
:::::::::::::::
:- module shallowShare.
:- public shallow/3.

shallow(X, Env,NEnv) :- vector(Env,Size) |
    new_string(Mark, Size, 1), shallow(X, [], Mark, Env,NEnv).

shallow(X, Cont, Mark, Env,NEnv) :- atom(X) |
    shallowCont(Cont, Mark, Env,NEnv).
shallow(X, Cont, Mark, Env,NEnv) :- integer(X) |
    shallowCont(Cont, Mark, Env,NEnv).
shallow({Num}, Cont, Mark, Env,NEnv) :- integer(Num) |
    shallowVar(Num, Cont, Mark, Env,NEnv).
shallow([H|T], Cont, Mark, Env,NEnv) :- true |
    shallow(H, [{T}|Cont], Mark, Env,NEnv).
shallow(X, Cont, Mark, Env,NEnv) :- vector(X,Size), Size > 1 |
    shallowCont([{1,Size,X}|Cont], Mark, Env,NEnv).

shallowVar(Num, Cont, Mark, Env,NEnv) :- true |
    shallowVar(Num, _, Cont, Mark, Env,NEnv).

shallowVar(Num, S, Cont, Mark, Env,NEnv) :- string_element(Mark,Num,2#"1") |
    shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv).
shallowVar(Num, S, Cont, Mark, Env,NEnv) :- string_element(Mark,Num,2#"0") |
    shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv).

shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :- vector_element(Env,Num,0) |
    S = point(Num),
    shallowCont(Cont, Mark, Env,NEnv).
shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,data(Data)) |
    S = point(Num),
    shallowCont(Cont, Mark, Env,NEnv).
shallowVarShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,point(Ref)) |
    S = point(Ref),
    shallowCont(Cont, Mark, Env,NEnv).

shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :- vector_element(Env,Num,0) |
    S = point(Num),
    set_string_element(Mark,Num,2#"1",NMark),
```

```
    shallowCont(Cont, NMark, Env,NEnv).
shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,data(Data)) |
    S = point(Num),
    set_string_element(Mark,Num,2#"1",NMark),
    shallow(Data, Cont, NMark, Env,NEnv).
shallowVarNoShallowed(Num, S, Cont, Mark, Env,NEnv) :-
    vector_element(Env,Num,point(Ref)) |
    set_string_element(Mark,Num,2#"1",NMark),
    set_vector_element(Env,Num, _,S, Env1),
    shallowVar(Ref, S, Cont, NMark, Env1,NEnv).

shallowCont([], _, Env,NEnv) :- true | NEnv = Env.
shallowCont([{X}|Cont], Mark, Env,NEnv) :- true |
    shallow(X, Cont, Mark, Env,NEnv).
shallowCont([{M,M,X}|Cont], Mark, Env,NEnv) :- true |
    shallowCont(Cont, Mark, Env,NEnv).
shallowCont([{M,N,X}|Cont], Mark, Env,NEnv) :- M < N, vector_element(X,M, Xn) |
    shallow(Xn, [{~(M+1),N, X}|Cont], Mark, Env,NEnv).
:::::::::::::::
unify.kl1
:::::::::::::::
:- module unifyShare.
:- public unify/4, unify/5.

%
%   unify(+X,+Y, +Env,-Env)
%
unify(X,Y, Env,NEnv) :- true | unify(X,Y, [], Env,NEnv).

unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unify(X,Y, _, _,NEnv) :- true | NEnv = fail.

unifyCont([], Env,NEnv) :- true | NEnv = Env.
unifyCont([{X,Y}|Cont], Env,NEnv) :- true | unify(X,Y, Cont, Env,NEnv).
unifyCont([{N,N,X,Y}|Cont], Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
unifyCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M, Xm), vector_element(Y,M, Ym) |
    unify(Xm,Ym, [{M1,N, X,Y}|Cont], Env,NEnv).

unifyVar(XNum, {YNum}, Cont, Env,NEnv) :- integer(YNum) |
    unifyVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
unifyVar(XNum, Y, Cont, Env,NEnv) :- true |
    unifyVarStr(XNum,Y, Cont, Env,NEnv).

unifyVarVar(XNum,XNum, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
```

```
  otherwise.
  unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
      unifyVarVar(XRef,YNum, Cont, Env,NEnv).
  unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
      unifyVarVar(XNum,YRef, Cont, Env,NEnv).
  unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
      vector_element(Env,YNum,0), vector_element(Env,XNum,data(XVal)) |
      unifyVarVarStr(YNum,XNum, Cont, Env,NEnv).
  unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
      vector_element(Env,XNum,0), vector_element(Env,YNum,data(YVal)) |
      unifyVarVarStr(XNum,YNum, Cont, Env,NEnv).
  unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
      vector_element(Env,XNum,data(XVal)), vector_element(Env,YNum,data(YVal)) |
      set_vector_element(Env,XNum, _,point(YNum), Env1),
      unify(XVal,YVal, Cont, Env1,NEnv).
  unifyVarVar(XNum,YNum, Cont,Env,NEnv) :-
      vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
      set_vector_element(Env,XNum, _,point(YNum), Env1),
      unifyCont(Cont, Env1,NEnv).

  unifyVarVarStr(XNum,YNum, Cont, Env,NEnv) :- true |
      set_vector_element(Env,XNum, _,point(YNum), Env1),
      unifyCont(Cont, Env1,NEnv).

  unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
      unifyVarStr(XRef,Y, Cont, Env,NEnv).
  unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
      unify(XVal,Y, Cont, Env,NEnv).
  unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
      set_vector_element(Env,XNum, _,data(Y),Env1),
      unifyCont(Cont, Env1,NEnv).
  ::::::::::::::
  unifyOc.kl1
  ::::::::::::::
  :- module unify_ocShare.
  :- public unify/4, unify/5.

  %
  %   unify(+X,+Y, +Env,-Env)
  %
  unify(X,Y, Env,NEnv) :- true | unify(X,Y, [], Env,NEnv).

  unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
  unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
  unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
      unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
  unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 1,
      vector_element(X,0,F), vector_element(Y,0,F) |
      unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
  otherwise.
  unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
  otherwise.
  unify(X,Y, _, _,NEnv) :- true | NEnv = fail.

  unifyCont([], Env,NEnv) :- true | NEnv = Env.
  unifyCont([{X,Y}|Cont], Env,NEnv) :- true | unify(X,Y, Cont, Env,NEnv).
```

```
unifyCont([{N,N,X,Y}|Cont], Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
unifyCont([{M,N,X,Y}|Cont], Env,NEnv) :- M < N, M1 := M + 1,
    vector_element(X,M, Xm), vector_element(Y,M, Ym) |
    unify(Xm,Ym, [{M1,N, X,Y}|Cont], Env,NEnv).

unifyVar(XNum, {YNum}, Cont, Env,NEnv) :- integer(YNum) |
    unifyVarVar(XNum,YNum, Cont, Env,NEnv).
otherwise.
unifyVar(XNum, Y, Cont, Env,NEnv) :- true |
    unifyVarStr(XNum,Y, Cont, Env,NEnv).

unifyVarVar(XNum,XNum, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarVar(XRef,YNum, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :- vector_element(Env,YNum,point(YRef)) |
    unifyVarVar(XNum,YRef, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
    vector_element(Env,XNum,data(XVal)), vector_element(Env,YNum,0) |
    unifyVarVarStr(YNum,XNum,XVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,data(YVal)) |
    unifyVarVarStr(XNum,YNum,YVal, Cont, Env,NEnv).
unifyVarVar(XNum,YNum, Cont, Env,NEnv) :-
    vector_element(Env,XNum,data(XVal)), vector_element(Env,YNum,data(YVal)) |
    checkOccur(XNum,YVal, [point|YNum],[{XVal,YVal}|Cont], Env,NEnv).
unifyVarVar(XNum,YNum, Cont,Env,NEnv) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    set_vector_element(Env,XNum, _,point(YNum), Env1),
    unifyCont(Cont, Env1,NEnv).

unifyVarVarStr(XNum,YNum,YVal, Cont, Env,NEnv) :- true |
    checkOccur(XNum,YVal, [point|YNum],Cont, Env,NEnv).

unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,point(XRef)) |
    unifyVarStr(XRef,Y, Cont, Env,NEnv).
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,data(XVal)) |
    unify(XVal,Y, Cont, Env,NEnv).
unifyVarStr(XNum,Y, Cont, Env,NEnv) :- vector_element(Env,XNum,0) |
    checkOccur(XNum,Y, [data|Y],Cont, Env,NEnv).

checkOccurCont([F|Y], XNum,Cont, Env,NEnv) :- atom(F) |
    set_vector_element(Env,XNum, _,{F,Y}, Env1),
    unifyCont(Cont, Env1,NEnv).
checkOccurCont([{Y}|OCont], XNum,Cont, Env,NEnv) :- true |
    checkOccur(XNum,Y, OCont,Cont, Env,NEnv).
checkOccurCont([{N,N,Y}|OCont], XNum,Cont, Env,NEnv) :- true |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).
checkOccurCont([{M,N,Y}|OCont], XNum,Cont, Env,NEnv) :-
    M < N, M1 := M + 1, vector_element(Y,M, Ym) |
    checkOccur(XNum,Ym, [{M1,N,Y}|OCont],Cont, Env,NEnv).

checkOccur(XNum,{YNum}, OCont,Cont, Env,NEnv) :- integer(YNum) |
    checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv).
checkOccur(XNum,[YH|YT], OCont,Cont, Env,NEnv) :- true |
    checkOccur(XNum,YH, [{YT}|OCont],Cont, Env,NEnv).
```

```
checkOccur(XNum,Y, OCont,Cont, Env,NEnv) :- vector(Y, Size), Size > 1 |
    checkOccurCont([{1,Size,Y}|OCont], XNum,Cont, Env,NEnv).
otherwise.
checkOccur(XNum,Y, OCont,Cont, Env,NEnv) :- true |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).

checkOccurVar(XNum,XNum, _,_, _,NEnv) :- true | NEnv = fail.
otherwise.
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :- vector_element(Env,YNum, 0) |
    checkOccurCont(OCont, XNum,Cont, Env,NEnv).
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :-
    vector_element(Env,YNum, point(YRef)) |
    checkOccurVar(XNum,YRef, OCont,Cont, Env,NEnv).
checkOccurVar(XNum,YNum, OCont,Cont, Env,NEnv) :-
    vector_element(Env,YNum, data(YData)) |
    checkOccur(XNum,YData, OCont,Cont, Env,NEnv).
```

C.3 その他のメタライブラリプログラム

```
:::::::::::::::
database.kl1
:::::::::::::::
:- module(database).
:- public go/1.

go(S) :- true | merge(S,S1), pool:keyed_set(Pool),
    go(S1, [],Pool).

go([], [],Pool) :- true | Pool = [].
go([], [S|SS],Pool) :- true | go(S, SS,Pool).
go([do(S)|Reqs], SS,Pool) :- true |
    merge(S,S1), go(S1, [Reqs|SS],Pool).
go([count(N)|Reqs], SS,Pool) :- true |
    Pool = [count(N)|Pool1], go(Reqs, SS,Pool1).
go([put(Key, Exp,Env)|Reqs], SS,Pool) :- true |
    copy_term:copy_term(Exp,NExp, Env,NEnv),
    Pool = [put(Key, {NExp,NEnv},_)|Pool1],
    go(Reqs, SS,Pool1).
go([get(Key, ExpEnv)|Reqs], SS,Pool) :- true |
    get(Key, ExpEnv, Pool,Pool1),
    go(Reqs, SS,Pool1).
go([get(Key,Env, ExpEnv)|Reqs], SS,Pool) :- true |
    get(Key,Env, ExpEnv, Pool,Pool1),
    go(Reqs, SS,Pool1).
go([getAll(KDs)|Reqs], SS,Pool) :- true |
    Pool = [carbon_copy(KDs)|Pool1],
    go(Reqs, SS,Pool1).
go([remove(Key)|Reqs], SS,Pool) :- true |
    Pool = [get_if_any(Key,_)|Pool1],
    go(Reqs, SS,Pool1).
go([compile(MID,Code)|Reqs], SS,Pool) :- true |
    Pool = [carbon_copy(KDs)|Pool1],
    compile(MID, KDs, Code),
    go(Reqs, SS,Pool1).

get(Key, ExpEnv, A,Z) :- true |
```

```
    utility:create_env(ZEnv,0), get(Key,ZEnv, ExpEnv, A,Z).

get(Key,Env, ExpEnv, A,Z) :- true |
    A = [get_and_put_if_any(Key, Old,New, _)|Z],
    getDecide(Old,New, Env, ExpEnv).

getDecide({},_, _, ExpEnv) :- true | ExpEnv = {}.
getDecide({{Exp0,Env0}},New, Env, ExpEnv) :-
    vector_element(Env0,0,Top0), vector_element(Env,0,Top), vector(Env,Size) |
    New ={Exp0,Env0},
    getDecideDecide(Top0,Top,Size, Exp0,Env0, Env, ExpEnv).

getDecideDecide(Top0,Top,Size, Exp0,Env0, Env, ExpEnv) :-
    Top0+Top-1 =< Size |    % Top0+Top-2 : Number of variables
                            % Size-1 : Limit of number of variable in Env
    ExpEnv = {Exp,NEnv},
    copy_term:copy(Exp0,Exp, [], Top, Env0, Env,NEnv).
otherwise.
getDecideDecide(Top0,Top,Size, Exp0,Env0, Env, ExpEnv) :- true |
    ExpEnv = {Exp,NEnv},
    utility:create_env(Env1,^(Top0+Top-2)),
    utility:set_subenv(Env1,1,Env, Env2),
    copy_term:copy(Exp0,Exp, [], Top, Env0, Env2,NEnv).

compile(MID,KDs, Code) :- true |
    Code = [print_depth(PDepth,400),print_length(PLength,1000),
            print_string_length(PSLength,1000),
            print_atom_length(PALength,1000)|Code1],
    makeDec(MID, Code1,Code2),
    compile(KDs, Code2,[nl|Code3], Code3,[nl|Code4]),
    makeUtil(Code4,Code5),
    Code5 = [print_depth(PDepth),print_length(PLength),
            print_string_length(PSLength),print_atom_length(PALength)].

makeDec(MID, A,Z) :- true |
    A = [putf(string#":- module(~T) .~n",[MID]),
         putf(string#":- public get/2,get/3 .~n",[])|Z].

compile([], A,G, X,Z) :- true |
    A = [putf(string#"otherwise.~n",[]),
         putf(string#"get(_, ExpEnv) :- true | ExpEnv = {}.~n",[])|G],
    X = [putf(string#"otherwise.~n",[]),
         putf(string#"get(_,_, ExpEnv) :- true | ExpEnv = {}.~n",[])|Z].
compile([{Key,{Exp0,Env0}}|KDs], A,G, X,Z) :- vector_element(Env0,0,Top0) |
    Top01 := Top0-1,
    A = [putf(string#"get(Key, ExpEnv) :- Key = ~T |~n",[Key]),
         putf(string#"      ExpEnv = {Exp,Env},~n",[]),
         putf(string#"      meta#create_env(Env,~T,~T),~n",[Top01,Top0]),
         putf(string#"      Exp = (~T).~n",[Exp0])|B],
    X = [putf(string#"get(Key,Env, ExpEnv) :- Key = ~T",[Key])|Y],
    compileGuard(Top01, Y,Y1),
    compileBody(Exp0,Exp),
    Y1 = [putf(string#"      Exp = (~W),~n",[Exp])|Y2],
    (Top01 =:= 0    -> Y2 = [
      putf(string#"      ExpEnv = {Exp,Env}.~n",[])|Y3];
     Top01 > 0  -> Y2 = [
```

```
        putf(string#"     ExpEnv = {Exp,NEnv},~n",[]),
        putf(string#"     set_vector_element(Env1,0, _,X~t, NEnv).~n",[Top0])|Y3]),
      compile(KDs, B,G, Y3,Z).

compileGuard(0, A,Z) :- true | A = [putf(string#" |~n",[])|Z].
compileGuard(N, A,Z) :- N > 0 |
    N1 := N + 1,
    A = [putf(string#", vector_element(Env,0,X1)~n",[])|B],
    compileGuard1(1,N1, B,[putf(string#" |~n",[])|C]),
    C = [putf(string#"     getExtendEnv(X~t, Env,Env1),~n",[N1])|Z].

compileGuard1(N,N, A,Z) :- true | A = Z.
compileGuard1(M,N, A,Z) :- M < N |
    M1 := M + 1,
    A = [putf(string#",     X~t := X~t+1~n",[M1,M])|B],
    compileGuard1(M1,N, B,Z).

compileBody(X,NX) :- integer(X) | NX = integer(X).
compileBody(X,NX) :- atom(X) | NX = atom(X).
compileBody(X,NX) :- string(X,_,_) | NX = string(X).
compileBody({Num},NX) :- integer(Num)|
    mklib:integer_to_string(Num,10, C),
    (C = normal(Str) -> builtin#append_string("X",Str, Name),
        NX = vector({variable(Name)})).
compileBody([H|T],NList) :- true |
    NList = list([NH|NT]),
    compileBody(H,NH), compileBody(T,NT).
compileBody(X,NX) :- vector(X,Size) |
    NX = vector(V),
    compileBodyArgs(0,Size, X,V).

compileBodyArgs(N,N, V,NV) :- true | NV = V.
compileBodyArgs(M,N, V,NV) :- M < N|
    set_vector_element(V,M, Vm,NVm, V1),
    compileBody(Vm,NVm),
    compileBodyArgs(~(M+1),N, V1,NV).

makeUtil(A,Z) :- true | A = [
    putf(string#"getExtendEnv(Top, Env,NEnv) :- vector(Env,Size), Top =< Size |~n",[]),
    putf(string#"     NEnv = Env.~n",[]),
    putf(string#"getExtendEnv(Top, Env,NEnv) :- vector(Env,Size), Top > Size |~n",[]),
    putf(string#"     meta#create_env(Env0,Top,~~(Top+1)),~n",[]),
    putf(string#"     meta#set_subenv(Env0,1,Env, NEnv).~n",[])|Z].
:::::::::::::::
equal.kl1
:::::::::::::::
:- module equal.
:- public equal/4, equal/5.

%
%   equal(+Pattern,+Target, +Env,-YesOrNo)
%
equal(X,Y, Env,YN) :- true | equal(X,Y, [], Env,YN).

%
%   equal(+Pattern,+Target, +Continuation, +Env,-Env)
```

```
%
equal({M},Y, Cont, Env,YN) :- integer(M) | equalVar(M,Y, Cont, Env,YN).
equal(X,{M}, Cont, Env,YN) :- integer(M) | equalVar(M,X, Cont, Env,YN).
equal([XH|XT],[YH|YT], Cont, Env,YN) :- true |
    equal(XH,YH, [{XT,YT}|Cont], Env,YN).
equal(X,Y, Cont, Env,YN) :- vector(X,Size), vector(Y,Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    equalCont([{1,Size,X,Y}|Cont], Env,YN).
otherwise.
equal(X,X, Cont, Env,YN) :- true | equalCont(Cont, Env,YN).
otherwise.
equal(X,Y, _, _,YN) :- true | YN = no.

equalCont([], _,YN) :- true | YN = yes.
equalCont([{X,Y}|Cont], Env,YN) :- true | equal(X,Y, Cont, Env,YN).
equalCont([{M,M,X,Y}|Cont], Env,YN) :- true | equalCont(Cont, Env,YN).
equalCont([{M,N,X,Y}|Cont], Env,YN) :- M < N, M1 := M + 1,
    vector_element(X,M,Xm), vector_element(Y,M,Ym) |
    equal(Xm,Ym, [{M1,N,X,Y}|Cont], Env,YN).

equalVar(XNum,{YNum}, Cont, Env,YN) :- integer(YNum) |
    equalVarVar(XNum,YNum, Cont, Env,YN).
otherwise.
equalVar(XNum,Y, Cont, Env,YN) :- true | equalVarStr(XNum,Y, Cont, Env,YN).

equalVarVar(XNum,XNum, Cont, Env,YN) :- true | equalCont(Cont, Env,YN).
otherwise.
equalVarVar(XNum,YNum, Cont, Env,YN) :- vector_element(Env,XNum,point(XRef)) |
    equalVarVar(XRef,YNum, Cont, Env,YN).
equalVarVar(XNum,YNum, Cont, Env,YN) :- vector_element(Env,YNum,point(YRef)) |
    equalVarVar(XNum,YRef, Cont, Env,YN).
equalVarVar(XNum,YNum, Cont, Env,YN) :- vector_element(Env,XNum,data(XVal)) |
    equalVarStr(YNum,XVal, Cont, Env,YN).
equalVarVar(XNum,YNum, Cont, Env,YN) :- vector_element(Env,YNum,data(YVal)) |
    equalVarStr(XNum,YVal, Cont, Env,YN).
equalVarVar(XNum,YNum, _, Env,YN) :-
    vector_element(Env,XNum,0), vector_element(Env,YNum,0) |
    YN = no.

equalVarStr(XNum,Y, Cont, Env,YN) :- vector_element(Env,XNum,point(XRef)) |
    equalVarStr(XRef,Y, Cont, Env,YN).
equalVarStr(XNum,Y, Cont, Env,YN) :- vector_element(Env,XNum,data(XVal)) |
    equal(XVal,Y, Cont, Env,YN).
equalVarStr(XNum,_, _, Env,YN) :- vector_element(Env,XNum,0) | YN = no.
:::::::::::::::
io.kl1
:::::::::::::::
:- module(trans).
:- public get_object/2, get_object/3.
:- public get_kl1_term/3, get_kl1_term/4.
:- with_macro pimos.

%
%   get_object(+WrappedTerm, -ExpressionEnvironmentVariabletable)
%
get_object(WT, ExpEnvVT) :- true |
```

```
        ExpEnvVT = {Exp,Env,VT},
        getObject0(WT,Exp, 1,Size, VT),
        utility:create_env(Env,~(Size-1),Size).

    %
    %   get_object(+WrappedTerm,+Environment, -ExpressionEnvironmentVariableTable)
    %
    get_object(WT,Env, ExpEnvVT) :- vector(Env,Size), vector_element(Env,0,Top) |
        getObject0(WT,Exp, Top,NTop,VT),
        (NTop > Size     -> ExpEnv = {};
         NTop =< Size    -> ExpEnvVT = {Exp,NEnv,VT},
            set_vector_element(Env,0, _,NTop, NEnv)).

    getObject0(WT,Exp, Top,NTop, VT) :- true |
        getObject(WT,Exp, Top,NTop, Pool1),
        pool:keyed_set([do(Pool1)|Pool2]),
        variableTable(Pool2, VT).

    getObject(atom(Atom),Exp, M,N, Pool) :- true |
        Exp = Atom, N := M, Pool = [].
    getObject(string(String),Exp, M,N, Pool) :- true |
        Exp = String, N := M, Pool = [].
    getObject(integer(Integer),Exp, M,N, Pool) :- true |
        Exp = Integer, N := M, Pool = [].
    getObject(wrap#[H|T],Exp, M,N, Pool) :- true |
        Pool = {Pool1,Pool2}, Exp = [ExpH|ExpT],
        getObject(H,ExpH, M,M1, Pool1),
        getObject(T,ExpT, M1,N, Pool2).
    getObject(vector(V),Exp, M,N, Pool) :-
        vector(V,Size), vector_element(V,0, atom(_)) |
        getObjectArgs(0,Size, V,Exp, M,N, Pool).
    getObject(variable(Name),Exp, M,N, Pool) :- true |
        getObjectVar(Name,Exp, M,N, Pool).

    getObjectVar(string#"_",Exp, M,N, Pool) :- true |
        Exp = {M}, N := M+1, Pool = [].
    otherwise.
    getObjectVar(Name,Exp, M,N, Pool) :- true |
        Pool = [put(Name, New,Old)],
        (Old = {}    -> Exp = {M}, New = M, N := M + 1;
         Old = {Num}    -> Exp = {Num}, New = Num, N := M).

    getObjectArgs(To,To, V,Exp, M,N, Pool) :- true |
        Exp = V, N := M, Pool = [].
    getObjectArgs(From,To, V,Exp, M,N, Pool) :- From < To |
        Pool = {Pool1,Pool2},
        set_vector_element(V,From, Vf,NVf, V1),
        getObject(Vf,NVf, M,M1, Pool1),
        getObjectArgs(~(From+1),To, V1,Exp, M1,N, Pool2).

    variableTable(Pool, VT) :- true | Pool = [get_all(VT)].

    %
    %   get_kl1_term(+Expression,+Environment, -WrappedTerm)
    %
    get_kl1_term(Exp,Env, WT) :- true |
```

```
    getKL1Term(Exp,Env,[], WT).

%
%   get_kl1_term(+Expression,+Environment,+VariableTable,
%                        -WrappedTerm)
%
get_kl1_term(Exp,Env,VT, WT) :- true | getKL1Term(Exp,Env,VT, WT).

getKL1Term(Atom,_,_, WT) :- atom(Atom) | WT = atom(Atom).
getKL1Term(Int,_,_, WT) :- integer(Int) | WT = integer(Int).
getKL1Term(Str,_,_, WT) :- string(Str,_,_) | WT = string(Str).
getKL1Term({Num},Env,VT, WT) :- integer(Num) |
    getKL1TermVar(Num,Env,VT, WT).
getKL1Term([Car|Cdr],Env,VT, WT) :- true |
    WT = list([WCar|WCdr]),
    getKL1Term(Car,Env,VT, WCar),
    getKL1Term(Cdr,Env,VT, WCdr).
getKL1Term(V,Env,VT, WT) :- vector(V,Size), vector_element(V,0,F), atom(F) |
    WT = vector(NV),
    getKL1TermArgs(0,Size, V,Env,VT, NV).

getKL1TermArgs(N,N, V,_,_, NV) :- true | NV = V.
getKL1TermArgs(M,N, V,Env,VT, NV) :- M < N |
    set_vector_element(V,M, Vm,NVm, V1),
    getKL1Term(Vm,Env,VT, NVm),
    getKL1TermArgs(M+1),N, V1,Env,VT, NV).

getKL1TermVar(Num,Env,VT, WT) :- vector_element(Env,Num,0) |
    getKL1TermVar1(VT, Num, WT).
getKL1TermVar(Num,Env,VT, WT) :- vector_element(Env,Num,point(Ref)) |
    getKL1TermVar(Ref,Env,VT, WT).
getKL1TermVar(Num,Env,VT, WT) :- vector_element(Env,Num,data(Data)) |
    getKL1Term(Data,Env,VT, WT).

getKL1TermVar1([], Num, WT) :- true |
    Init = [size(10),atom_table(_),operator_pool(_)],
    buffer:character_pool(Init, [putt(Num),getb(5,NStr)]),
    builtin#append_string(string#"_",NStr, Name),
    WT = variable(Name).
getKL1TermVar1([{Name,Num}|_], Num, WT) :- true | WT = variable(Name).
otherwise.
getKL1TermVar1([_|VT], Num, WT) :- true | getKL1TermVar1(VT, Num, WT).
:::::::::::::::
util.kl1
:::::::::::::::
:- module utility.
:- public unbound/3, create_env/2, create_env/3, set_subenv/4,
        fresh_var/2, is_type/3, entry_object/5.

unbound({Num}, Env,YN) :- integer(Num) | unbound1(Num, Env,YN).
otherwise.
unbound(X, _,YN) :- true | YN = no.

unbound1(Num, Env,YN) :- vector_element(Env, Num,0) | YN = yes.
unbound1(Num, Env,YN) :- vector_element(Env, Num,point(Ref)) |
    unbound1(Ref, Env,YN).
```

```
unbound1(Num, Env,YN) :- vector_element(Env, Num,data(_)) | YN = no.
otherwise.
unbound!(_, _,YN) :- true | YN = abnormal.

create_env(Env,Size) :- true | create_env(Env,Size,1).
create_env(Env,Size,Top) :- Size >= 0 |
    new_vector(Env0,^(Size+1)),
    set_vector_element(Env0,0, _,Top, Env).

set_subenv(Env,Pos,SubEnv, NEnv) :-
    vector_element(Env,0,Top), vector(SubEnv,Size) |
    M := Pos+Size-1,
    set_subenvDecide(M,Top, Env,Pos,SubEnv, NEnv).

set_subenvDecide(M,Top, Env,Pos,SubEnv, NEnv) :- M =< Top |
    setSubEnvArgs(Env,Pos, SubEnv,1, NEnv).
set_subenvDecide(M,Top, Env,Pos,SubEnv, NEnv) :- M > Top |
    set_vector_element(Env,0, _,M, Env1),
    setSubEnvArgs(Env1,Pos, SubEnv,1, NEnv).

setSubEnvArgs(Env,Pos, SubEnv,M, NEnv) :- vector_element(SubEnv,M, E) |
    set_vector_element(Env,Pos, _,E, Env1),
    setSubEnvArgs(Env1,^(Pos+1), SubEnv,^(M+1), NEnv).
otherwise.
setSubEnvArgs(Env,_, _,_, NEnv) :- true | NEnv = Env.

fresh_var(Env, New) :- vector_element(Env,0,Top), vector(Env,Size), Top =< Size |
    New = {{Top},NEnv},
    set_vector_element(Env,0, _,^(Top+1), NEnv).
otherwise.
fresh_var(_, New) :- true | New = fail.

entry_object(Exp,Env, Env0, NExp,NEnv) :-
    vector_element(Env,0,Top), vector(Env0,Size0), vector_element(Env0,0,Top0) |
    M := Size0-Top0,    % variable entry capacity in Env0
    N := Top - 1,       % The number of variables which Exp includes at most
    entry_objectDecide(^(M-N), Top0,N, Exp,Env, Env0, NExp,NEnv).

entry_objectDecide(Num, Top0,_, Exp,Env, Env0, NExp,NEnv) :- Num >= 0 |
    copy_term:copy(Exp,NExp, [], Top0, Env, Env0,NEnv).
entry_objectDecide(Num, Top0,N, Exp,Env, Env0, NExp,NEnv) :- Num < 0 |
    utility:create_env(Env_, ^(Top0+N-1), Top0),
    set_subenv(Env_,1,Env0, Env_1),
    copy_term:copy(Exp,NExp, [], Top0, Env, Env_1,NEnv).

is_type(X,Env, Type) :- true |
    eval(X,NX, Env), isType(NX, Type).

isType(X, Type) :- integer(X) | Type = integer(X).
isType(X, Type) :- atom(X) | Type = atom(X).
isType(X, Type) :- string(X,_,_) | Type = string(X).
isType(X, Type) :- X = {Num}, integer(Num) | Type = variable(X).
isType(X, Type) :- list(X) | Type = list(X).
isType(X, Type) :- vector(X,Size),Size > 1 | Type = vector(X).

eval(X,NX, _) :- integer(X) | NX = X.
```

```
eval(X,NX, _) :- atom(X) | NX = X.
eval(X,NX, _) :- string(X,_,_) | NX = X.
eval({M},NX, Env) :- integer(M) | evalVar(M,NX, Env).
eval([H|T],List, _) :- true | List = [H|T].
eval(V,NV, _) :- vector(V,Size), Size > 1 | NV = V.

evalVar(Num,NX, Env) :- vector_element(Env,Num, point(Ref)) |
    evalVar(Ref,NX, Env).
evalVar(Num,NX, Env) :- vector_element(Env,Num, data(Val)) | NX = Val.
evalVar(Num,NX, Env) :- vector_element(Env,Num, 0) | NX = {Num}.
:::::::::::::::
variable.kl1
:::::::::::::::
:- module variable.
:- public freeze/3, freeze/4, melt/2, melt/3.

freeze(X,NX, Env) :- true | freeze(X,NX, [], Env).

freeze(X,NX, Cont, Env) :- atom(X) |
    NX = X, freezeCont(Cont, Env).
freeze(X,NX, Cont, Env) :- integer(X) |
    NX = X, freezeCont(Cont, Env).
freeze({Num},NVar, Cont, Env) :- integer(Num) | freezeVar(Num,NVar, Cont, Env).
freeze([H|T],NList, Cont, Env) :- true |
    NList = [NH|NT],
    freeze(H,NH, [{T,NT}|Cont], Env).
freeze(X,NX, Cont, Env) :- vector(X, Size), Size > 1 |
    freezeCont([{1,Size, X,NX}|Cont], Env).

freezeVar(Num,NVar, Cont, Env) :- true |
    freezeVar(Num,NVar, _, Cont, Env).

freezeVar(Num,NVar, S, Cont, Env) :- vector_element(Env,Num,0) |
    NVar = {Num,freezed},
    S = markPoint(Num),
    set_vector_element(Env,Num, _,S, Env1),
    freezeCont(Cont, Env1).
freezeVar(Num,NVar, S, Cont, Env) :-  vector_element(Env,Num,point(Ref)) |
    set_vector_element(Env,Num, _,S, Env1),
    freezeVar(Ref,NVar, S, Cont, Env1).
freezeVar(Num,NVar, S, Cont, Env) :- vector_element(Env,Num,data(Data)) |
    NVar = NData,
    S = markData(NData),
    set_vector_element(Env,Num, _,S, Env1),
    freeze(Data,NData, Cont, Env1).
freezeVar(Num,NVar, S, Cont, Env) :- vector_element(Env,Num,markPoint(Ref)) |
    NVar = {Ref,freezed},
    S = markPoint(Ref),
    freezeCont(Cont, Env).
freezeVar(Num,NVar, S, Cont, Env) :- vector_element(Env,Num,markData(Data)) |
    NVar = Data,
    S = markData(Data),
    freezeCont(Cont, Env).

freezeCont([], _) :- true | true.
freezeCont([{X,NX}|Cont], Env) :- true | freeze(X,NX, Cont, Env).
```

```
freezeCont([{N,N,X,NX}|Cont], Env) :- true | NX = X, freezeCont(Cont, Env).
freezeCont([{M,N,X,NX}|Cont], Env) :- M < N |
    set_vector_element(X,M, Xm,NXm, X1),
    freeze(Xm,NXm, [{~(M+1),N, X1,NX}|Cont], Env).

%
%   melt(+FrozenTerm, -MeltedTerm)
%
melt(X, FX) :- true | melt(X,FX, []).

melt(X,FX, Cont) :- atom(X) |  FX = X, meltCont(Cont).
melt(X,FX, Cont) :- integer(X) | FX = X, meltCont(Cont).
melt({Num,freezed},NVar, Cont) :- integer(Num) | NVar = {Num}, meltCont(Cont).
melt([H|T],NList, Cont) :- true |
    NList = [NH|NT],
    melt(H,NH, [{T,NT}|Cont]).
melt(X,NX, Cont) :- vector_element(X,0,F), atom(F) |
    meltCont([{1,Size, X,NX}|Cont]).

meltCont([]) :- true | true.
meltCont([{X,NX}|Cont]) :- true | melt(X,NX, Cont).
meltCont([{N,N,X,NX}|Cont]) :- true | NX = X, meltCont(Cont).
meltCont([{M,N,X,NX}|Cont]) :- M < N |
    set_vector_element(X,M, Xm,NXm, X1),
    melt(Xm,NXm, [{~(M+1),N, X1,NX}|Cont]).
```

C.4 マクロ展開プログラムと標準マクロ定義

C.4.1 マクロ展開プログラム

```
:::::::::::::::
macroCompile.kl1
:::::::::::::::
:- module macroCompile.
:- public go/0, go/2.

go :- true |
    shoen:raise(pimos_tag#shell,get_std_in, In),
    shoen:raise(pimos_tag#shell,get_std_out, Out),
    go(In, Out).

makeDec(A,Z) :- true |
    A = [putf(":- module macroBank.~n",[]),
         putf(":- public expand/2.~2n",[])|Z].

go(In, Out) :- true |
    Out = [print_length(OPLeng,40),print_depth(OPDepth,100),
           print_atom_length(OALeng,200),
           print_string_length(OSLeng,300)|Out1],
    makeDec(Out1,Out2),
    go(In, Out2,Out3),
    Out3 = [print_length(OPLeng),print_depth(OPDepth),
           print_atom_length(OALeng),print_string_length(OSLeng)].

go(In, Out,NOut) :- true | In = [getwt(WT)|In1],
    goDecide(WT, In1,Out,NOut).
```

```
goDecide(abnormal(ErrorInfo), In,Out,NOut) :- true |
    Out = [print_error(ErrorInfo)|Out1],
    go(In, Out1,NOut).
goDecide(normal(WT), In,Out,NOut) :- true |
    goNormal(WT, In,Out,NOut).

goNormal(end_of_file, In,Out,NOut) :- true | In = [], Out = [
    putf("otherwise.~n",[]),
    putf("expand(WT, Expand) :- true | Expand = nochange(WT).~n",[])|NOut].
goNormal(empty, In,Out,NOut) :- true | go(In, Out,NOut).
goNormal(WT, In,Out,NOut) :- WT = wrap#(X => Y) |
    Out = [putf("expand(_1, _2) :-~n",[]),
           putf("      _1 = wrap#(~W) |~n",[X]),
           putf("      _2 = expanded(wrap#(~W)).~n",[Y])|Out1],
    go(In, Out1,NOut).
otherwise.
goNormal(_, In,Out,NOut) :- true | go(In, Out,NOut).
:::::::::::::::
macroExpand.kl1
:::::::::::::::
:- module macroExpand.
:- public go/0,go/2.

go :- true |
    shoen:raise(pimos_tag#shell,get_std_in, In),
    shoen:raise(pimos_tag#shell,get_std_out, Out),
    go(In, Out).

go(In,Out) :- true |
    Out = [print_length(OPLeng,40),print_depth(OPDepth,100),
           print_atom_length(OALeng,200),
           print_string_length(OSLeng,300)|Out1],
    go(In, Out1,Out2),
    Out2 = [print_length(OPLeng),print_depth(OPDepth),
           print_atom_length(OALeng),print_string_length(OSLeng)].

go(In, Out,NOut) :- true | In = [getwt(WT)|In1],
    goDecide(WT, In1,Out,NOut).

goDecide(abnormal(ErrorInfo), In,Out,NOut) :- true |
    Out = [print_error(ErrorInfo)|Out1],
    go(In, Out1,NOut).
goDecide(normal(WT), In,Out,NOut) :- true | goNormal(WT, In,Out,NOut).

goNormal(end_of_file, In,Out,NOut) :- true | In = [], Out = NOut.
goNormal(empty, In,Out,NOut) :- true | go(In, Out,NOut).
otherwise.
goNormal(WT, In,Out,NOut) :- true |
    expand(WT, NWT),
    Out = [putwtq(NWT),putl(" .")|Out1],
    go(In, Out1,NOut).

expand(WT, NWT) :- true |
    macroBank:expand(WT, Expanded),
    expandDecide(Expanded, NWT).
```

```
expandDecide(expanded(WT), NWT) :- true | NWT = WT.
expandDecide(nochange(WT), NWT) :- true | expand1(WT, NWT).

expand1(WT, NWT) :- WT = atom(_) | NWT = WT.
expand1(WT, NWT) :- WT = integer(_) | NWT = WT.
expand1(WT, NWT) :- WT = string(_) | NWT = WT.
expand1(WT, NWT) :- WT = variable(_) | NWT = WT.
expand1(wrap#[H|T], NWT) :- true | NWT = wrap#[NH|NT],
    expand(H, NH), expand(T, NT).
expand1(vector(V), NWT) :- vector(V, Size) |
    NWT = vector(NV),
    expandArgs(0,Size, V,NV).

expandArgs(N,N, V,NV) :- true | NV = V.
expandArgs(M,N, V,NV) :- M < N |
    set_vector_element(V,M, Vm,NVm, V1),
    expand(Vm,NVm),
    expandArgs(M+1,N, V1,NV).

:- module macroBank.
:- public expand/2.

expand(_1, _2) :-
    _1 = wrap#(meta# match(Pattern,Target,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: match: match(Pattern,Target,normal,[],[],Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# oneway_unify(Pattern,Target,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: oneway_unify: match(Pattern,Target,normal,[],[],Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# oneway_unify_tmp(Pattern,Target,Tmp,Env,NTmpEnv)) |
    _2 = expanded(wrap#(meta:: oneway_unify: match(Pattern,Target,normal,Tmp,[tmp],Env,NTmpEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# oneway_unify_point_real(Tmp,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: oneway_unify: pointReal(Tmp,Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# unify(X,Y,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: unify: unify(X,Y,[],Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# unify_oc(X,Y,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: unify_oc: unify(X,Y,[],Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# copy_term(X,NX,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: copy_term: copy_term(X,NX,Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# shallow(X,Env,NEnv)) |
    _2 = expanded(wrap#(meta:: shallow: shallow(X,Env,NEnv))).
expand(_1, _2) :-
    _1 = wrap#(meta# equal(X,Y,Env,YN)) |
    _2 = expanded(wrap#(meta:: equal: equal(X,Y,[],Env,YN))).
expand(_1, _2) :-
    _1 = wrap#(meta# freeze(X,FX,Env)) |
    _2 = expanded(wrap#(meta:: variable: freeze(X,FX,[],Env))).
expand(_1, _2) :-
    _1 = wrap#(meta# melt(FX,X)) |
    _2 = expanded(wrap#(meta:: variable: freeze(FX,X,[]))).
expand(_1, _2) :-
```

```
        _1 = wrap#(meta# entry_object(Exp,Env,Env0,NExp,NEnv)) |
        _2 = expanded(wrap#(meta:: utility: entry_object(Exp,Env,Env0,NExp,NEnv))).
    expand(_1, _2) :-
        _1 = wrap#(meta# create_env(Env,Size)) |
        _2 = expanded(wrap#(meta:: utility: create_env(Env,Size,1))).
    expand(_1, _2) :-
        _1 = wrap#(meta# create_env(Env,Size,Top)) |
        _2 = expanded(wrap#(meta:: utility: create_env(Env,Size,Top))).
    expand(_1, _2) :-
        _1 = wrap#(meta# set_subenv(Env,Pos,SubEnv,NEnv)) |
        _2 = expanded(wrap#(meta:: utility: set_subenv(Env,Pos,SubEnv,NEnv))).
    expand(_1, _2) :-
        _1 = wrap#(meta# fresh_var(Env,VarNEnv)) |
        _2 = expanded(wrap#(meta:: utility: fresh_var(Env,VarNEnv))).
    expand(_1, _2) :-
        _1 = wrap#(meta# unbound(X,Env,YN)) |
        _2 = expanded(wrap#(meta:: utility: unbound(X,Env,YN))).
    expand(_1, _2) :-
        _1 = wrap#(meta# is_type(X,Env,Type)) |
        _2 = expanded(wrap#(meta:: utility: is_type(X,Env,Type))).
    expand(_1, _2) :-
        _1 = wrap#(meta# database(DB)) |
        _2 = expanded(wrap#(meta:: database: go(DB))).
    expand(_1, _2) :-
        _1 = wrap#(meta# get_object(WT,Env,ExpEnvVT)) |
        _2 = expanded(wrap#(meta:: trans: get_object(WT,Env,ExpEnvVT))).
    expand(_1, _2) :-
        _1 = wrap#(meta# get_object(WT,ExpEnvVT)) |
        _2 = expanded(wrap#(meta:: trans: get_object(WT,ExpEnvVT))).
    expand(_1, _2) :-
        _1 = wrap#(meta# get_kl1_term(Exp,Env,VT,WT)) |
        _2 = expanded(wrap#(meta:: trans: get_kl1_term(Exp,Env,VT,WT))).
    expand(_1, _2) :-
        _1 = wrap#(meta# get_kl1_term(Exp,Env,WT)) |
        _2 = expanded(wrap#(meta:: trans: get_kl1_term(Exp,Env,[],WT))).
    otherwise.
    expand(WT, Expand) :- true | Expand = nochange(WT).
```

C.1.2 標準マクロ定義

```
meta#match(Pattern,Target, Env,NEnv) =>
    meta::match:match(Pattern,Target, normal,[],[], Env,NEnv).
%    meta::matchShare:match(Pattern,Target, normal,[],[], Env,NEnv).
meta#oneway_unify(Pattern,Target, Env,NEnv) =>
    meta::oneway_unify:match(Pattern,Target, normal,[],[], Env,NEnv).
%    meta::oneway_unifyShare:match(Pattern,Target, normal,[],[], Env,NEnv).

meta#oneway_unify_tmp(Pattern,Target, Tmp,Env, NTmpEnv) =>
    meta::oneway_unify:match(Pattern,Target, normal,Tmp,[tmp], Env,NTmpEnv).
%    meta::oneway_unifyShare:match(Pattern,Target, normal,Tmp,[tmp], Env,NTmpEnv).

meta#oneway_unify_point_real(Tmp,Env, NEnv) =>
    meta::oneway_unify:pointReal(Tmp,Env, NEnv).
%    meta::oneway_unifyShare:pointReal(Tmp,Env, NEnv).
```

```
meta#unify(X,Y, Env,NEnv) =>
    meta::unify:unify(X,Y, [], Env,NEnv).
%    meta::unifyShare:unify(X,Y, [], Env,NEnv).
meta#unify_oc(X,Y, Env,NEnv) =>
    meta::unify_oc:unify(X,Y, [], Env,NEnv).
%    meta::unify_ocShare:unify(X,Y, [], Env,NEnv).

meta#copy_term(X,NX, Env,NEnv) =>
    meta::copy_term:copy_term(X,NX, Env,NEnv).
%    meta::copy_termShare:copy_term(X,NX, Env,NEnv).
meta#shallow(X, Env,NEnv) =>
    meta::shallow:shallow(X, Env,NEnv).
%    meta::shallowShare:shallow(X, Env,NEnv).

meta#equal(X,Y, Env,YN) =>
    meta::equal:equal(X,Y, [], Env,YN).

meta#freeze(X,FX, Env) =>
    meta::variable:freeze(X,FX, [], Env).
meta#melt(FX, X) =>
    meta::variable:freeze(FX,X, []).

meta#entry_object(Exp,Env, Env0, NExp,NEnv) =>
    meta::utility:entry_object(Exp,Env, Env0, NExp,NEnv).

meta#create_env(Env,Size) =>
    meta::utility:create_env(Env,Size,1).
meta#create_env(Env,Size,Top) =>
    meta::utility:create_env(Env,Size,Top).
meta#set_subenv(Env,Pos,SubEnv, NEnv) =>
    meta::utility:set_subenv(Env,Pos,SubEnv, NEnv).
meta#fresh_var(Env, VarNEnv) =>
    meta::utility:fresh_var(Env, VarNEnv).
meta#unbound(X,Env, YN) =>
    meta::utility:unbound(X,Env, YN).
meta#is_type(X,Env, Type) =>
    meta::utility:is_type(X,Env, Type).

meta#database(DB) =>
    meta::database:go(DB).

meta#get_object(WT, Env,ExpEnvVT) =>
    meta::trans:get_object(WT, Env,ExpEnvVT).
meta#get_object(WT, ExpEnvVT) =>
    meta::trans:get_object(WT, ExpEnvVT).
meta#get_kl1_term(Exp,Env,VT, WT) =>
    meta::trans:get_kl1_term(Exp,Env,VT, WT).
meta#get_kl1_term(Exp,Env, WT) =>
    meta::trans:get_kl1_term(Exp,Env,[], WT).
```

D　メタライブラリの高速化について

プログラムを展開することにより場合により 2 倍近く高速化を計ることができる [4].

【例】ユニファイプログラム

- オリジナルプログラム

```
unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 1,
    vector_element(X,0,F), vector_element(Y,0,F) |
    unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unify(X,Y, _, _,NEnv) :- true | NEnv = fail.
```

- 展開したプログラム

```
unify({N},Y, Cont, Env,NEnv) :- integer(N) | unifyVar(N,Y, Cont, Env,NEnv).
unify(X,{M}, Cont, Env,NEnv) :- integer(M) | unifyVar(M,X, Cont, Env,NEnv).
unify([XH|XT],[YH|YT], Cont, Env,NEnv) :- true |
    unify(XH,YH, [{XT,YT}|Cont], Env,NEnv).
unify({F,X},{F,Y}, Cont, Env,NEnv) :- true | unify(X,Y, Cont, Env,NEnv).
unify({F,X,X1},{F,Y,Y1}, Cont, Env,NEnv) :- true |
    unify(X,Y, [{X1,Y1}|Cont], Env,NEnv).
unify({F,X,X1,X2},{F,Y,Y1,Y2}, Cont, Env,NEnv) :- true |
    unify(X,Y, [{X1,Y1},{X2,Y2}|Cont], Env,NEnv).
unify({F,X,X1,X2,X3},{F,Y,Y1,Y2,Y3}, Cont, Env,NEnv) :- true |
    unify(X,Y, [{X1,Y1},{X2,Y2},{X3,Y3}|Cont], Env,NEnv).
unify(X,Y, Cont, Env,NEnv) :- vector(X, Size), vector(Y, Size), Size > 4,
    vector_element(X,0,F), vector_element(Y,0,F) |
    unifyCont([{1,Size,X,Y}|Cont], Env,NEnv).
otherwise.
unify(X,X, Cont, Env,NEnv) :- true | unifyCont(Cont, Env,NEnv).
otherwise.
unify(X,Y, _, _,NEnv) :- true | NEnv = fail.
```

References

[1] C.-L. Chang, R.C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Computer Science and Applied Mathematics Series, Academic Press, London, 1973.

[2] D.W.Loveland, *Automated Theorem Proving:A Logical Basis*, Fundamental Studies in Computer Science, Vol. 6, North-Holland, New York, 1978.

[3] ICOT PIMOS 開発グループ. PIMOS マニュアル (第 2 版) 1990.

[4] 越村 三幸, 藤田 博, 長谷川 隆三. KL1 上のユニフィケーションプログラムとその評価, ICOT TM 準備中.

[5] 長谷川 隆三, 藤田 博, 藤田 正幸. 定理証明技法 / システムのサーベイ, ICOT TM 準備中.