制約論理型プログラミング言語
ｃｕ－ｐｒｏｌｏｇ

<< 内容 >>

ｃｕ－ｐｒｏｌｏｇユーザーズマニュアル

ｃｕ－ｐｒｏｌｏｇソースファイル概要
ｉｎｃｌｕｄｅ．ｈ ドキュメンテーション
ｖａｒｓｅｔ．ｈ、ｇｌｏｂａｌｖ．ｈ ドキュメンテーション
ｍａｉｎ．ｃ ドキュメンテーション
ｍａｉｎｓｕｂ．ｃ ドキュメンテーション
ｎｅｗ．ｃ ドキュメンテーション
ｒｅａｄ．ｃ ドキュメンテーション
ｐｒｉｎｔ．ｃ ドキュメンテーション
ｒｅｆｕｔｅ．ｃ ドキュメンテーション
ｓｙｓｐｒｅｄ．ｃ ドキュメンテーション
ｕｎｉｆｙ．ｃ ドキュメンテーション
ｃｕｎｉｆｙ．ｃ ドキュメンテーション
ｇｅｎｆｕｎｃ．ｃ ドキュメンテーション
ｍｏｄｕｌａｒ．ｃ ドキュメンテーション
ｓｐｌｉｔ．ｃ ドキュメンテーション
ｒｅｄｕｃｅ．ｃ ドキュメンテーション
ｉｎｔｅｇ．ｃ ドキュメンテーション

ｃｕ－ｐｒｏｌｏｇ全ソースファイル

日本語句構造文法(JPSG)による日本語パーザソースプログラム

〔著者〕

津田 宏
(財)新世代コンピュータ技術開発機構 第二研究室
住所 ： 東京都港区三田1-4-28 三田国際ビル21F
(郵便番号 108)
TEL ： (03) 456-3194
FAX ： (03) 456-1618
E-mail: tsuda@icot.jp

<< c u − P r o l o g　ユーザーズマニュアル >>

89. 6.28 for Ver. 2.2
89. 7.27 for Ver. 2.3

これは,制約論理型プログラミング言語cu-Prologのユーザーズマニュアルである。cu-Prologは,prologの素式列を制約にできるので、記号的、組み合わせ的な制約を自然に記述する事ができるという特徴がある。自然言語処理ならびにAI全般への応用に適するプログラミング言語である。

0．コンパイルの方法

cu-Prologのソースファイルは、C言語で書かれ以下のモジュールからなる。

include.h, funclist.h, varset.h, globalv.h, sysp.h, main.c, mainsub.c, new.c, read.c, print.c, refute.c, syspred.c, unify.c, cunify.c, genfunc.c, modular.c, split.c, reduce.c, inte.c

cu-Prologの実行コードを得るには、全モジュールを分割コンパイルしリンクで合わせるだけである。ただし、使用するＯＳ、コンパイラによりinclude.hの最初の、

```
#define MSDOS ????
```

の値を書き換える必要がある。

ＭＳ－ＤＯＳのＣコンパイラでスモールモデルの場合。
（ｉｎｔが１６ビット、ポインタサイズが１６ビット）
例：MS-C ver.4, Turbo-C ver.1.5

```
#ｄｅｆｉｎｅ　ＭＳＤＯＳ　　１
```

ＭＳ－ＤＯＳのＣコンパイラでラージ（ヒュージ）モデルの場合。
例：MS-C ver.4 (large), Turbo-C ver.1.5(huge)
（ｉｎｔが１６ビット、ポインタサイズが３２ビット）

```
#ｄｅｆｉｎｅ　ＭＳＤＯＳ　　２
```

ＵＮＩＸのＣコンパイラの場合。
例：Sun-3, Symmetry
（ｉｎｔ、ポインタサイズ共に３２ビット）

```
#ｄｅｆｉｎｅ　ＭＳＤＯＳ　　０
```

デフォルトはUNIXなので、単にmakeすればよい。

また、include.h のCPUTIMEの値もコンパイラにより書き変える必要がある。
UNIX4.2/4.3 BSDでtimes()関数があり、1/n単位でCPU-timeを計測する場合には、

```
#define CPUTIME n　　（デフォルトは60）
```

とする。
それ以外には 0 とする。この時は、実行時のcpuタイムが表示されない。

なお、実行していて、
user heap overflow がでるなら　ＨＥＡＰ＿ＳＩＺＥ
user stack overflow がでるなら　ＵＳＴＡＣＫ＿ＳＩＺＥ

system heap overflowがでるなら　ＳＨＥＡＰ＿ＳＩＺＥの値をそれぞれ大きくするとよい。

1．起動法

　　ｃｕ　＜リターン＞

とする。また、起動後すぐファイルを読み込むときには、

　　ｃｕ　ファイル名　＜リターン＞

とする。

2．終了法

トップレベルから、%Ω　＜リターン＞と入力する。

3．プログラムの記述

項：

　　アトム、変数、複合項から成る。

アトム：

　　定数。英小文字ではじまる任意の文字列。現バージョンでは、漢字は扱えません。

変数：

　　英大文字またはアンダーラインから始まる任意の文字列。アンダーラインのみは無名変数で、任意の２つの無名変数は異なった変数として扱う。

複合項：

　　pを文字列、t1,t2,...,tnを項としたとき、p(t1,t2,...,tn)を複合項という。n=0のときは、pのみとなる（例えば、カット,fail,定数）。

他の項の引数となりうる複合項を関数と呼び、そのときｐをファンクタと呼ぶ。
例えば、リスト。
他の引数とならない複合項を述語と呼び、そのときｐを述語名と呼ぶ。

プログラム節の記述：

cu-Prologのプログラム節は,Constraint Added Horn Clause(CAHC: 制約付ホーン節)と呼ばれ,通常のホーン節に制約を加えたものになっている．CAHCは以下の３種類の節からなる。

1．<事実>

```
A.
A ; C1,...,Cn.
```

（例）

```
fly(bird).
member(X,[X|Y]).
f(a,X,Y);c0(X),c1(Y).
```

2．<規則>

```
A :- B1,...,Bm.
A :- B1,...,Bm;C1,...,Cn.
```

（m = 0 の時は事実節と同値である。）
（例）

```
bird(X):-fly(X).
member(X,[Y|Z]):-member(X,Z).
f(a,b,X,Y):-g(u,X),h(Y,X);c0(X,Y).
```

3．<質問>

```
        :- B1,...,Bm.
        :- B1,...,Bm;C1,...,Cn.
```

（例）

```
:-fly(chiken).
:-member(X,[a,b,c]).
:-f(X,Y,a);c0(X,Y).
```

Aをヘッド（頭部）、B1,...,Bmをボディ（本体），C1,...,Cnを制約と呼ぶ。

ここで,制約は以下のモジュラーという標準形でなければならない．

[定義]
次の3条件を満たす時,C1,...,Cn はモジュラーである(nilもモジュラーとする)．
1．Ciはモジュラー定義述語
2．Ciの引数は全て変数
3．C1,...,Cnで,同じ変数は2個所に現れない．

また，
[定義]
fがモジュラー定義述語とは,fの定義節の本体が全てモジュラーである事をいう。
（cu-Prologの現バージョンでは、このチェックは行っていない）

4．リスト
関数の代表的なものがリストである。引数は２つとる。
Lispの cons(A,B) を [A|B] と記述し、nil を [] と記述する。
また、以下のような略記もできる。

```
[A|[]]      は  [A]
[A|[B]]     は  [A,B]
[A|[B,C]]   は  [A,B,C]
[A,[B|C]]   は  [A,B|C]
```

5．組み込み述語、関数

現在、ｃｕ－ｐｒｏｌｏｇでは以下のような組み込み述語が用意されている。

！：
　　カット述語。引数なし。ゴールの中でカットを越えたら、それ以前にはバックトラックしない。

fail：
　　引数なし。無条件で失敗する述語。節の最後に置いて、全解探索などに用いる。

write(X) :
　　Xは変数または文字列をとる。変数の値などをプリントする。

nl :
　　改行

tab :
　　タブをプリント

pcon :
　　その時点での制約をプリントする。

eq(X,Y) :
　　XとYが単一化可能の時成功する述語

var(X) :
　　Xが具体化していない変数の場合に成功する述語

unify(C,NC) :
　　制約をモジュラーに変換する。Cはリストの形の制約、NCは変数を入れる。成功すると、Cと等価でモジュラーな制約がNCに入る。

tree(H) :
　　パーザーで用い、履歴を木構造でプリントする。Hにはヒストリーの項（後述のｔ（））を入れる。

以下は、JPSGパーザ専用の組み込み関数である。

ｃａｔ（Pos,Form,Adjacent,Adjunct,Subcat,Sem）：
　　句構造のカテゴリーを表す関数。

t() :
　　ヒストリーを表す関数。パーザーで木構造を表示するときに履歴をとっておくのに使う。

ヒストリーの定義は以下である。

・カテゴリーはヒストリー
・C，Wがカテゴリーの時、ｔ（C，W，［］）はヒストリー
・L，Rがヒストリーで、Mがカテゴリーかｔ（C，W，［］）の形　（C，Wはカテゴリー）の時、ｔ（M，L，R）はヒストリー

なお、ｔｒｅｅ（）により、ｔ（C，W，［］）の形のヒストリーは、

　　C－－W

ｔ（M，L，R）の形のヒストリーは

　　－－－M

```
        |
        |--L
        |
        |--R
```

と表示される。

6．ファイル入出力

ｃｕ－Ｐｒｏｌｏｇでは、プログラムファイルの読み書きや、実行時のログファイルをとることができる。

◎プログラムの読み込み

システム実行時に読み込むには、
　　ｃｕ　ファイル名　<リターン>
とする。ファイルの内容は、画面には表示されない。
また、トップレベルから内容を画面に表示せずに読み込むには、
　　"ファイル名"　<リターン>
ファイルの内容を画面に表示しながら読み込むには、
　　"ファイル名？　<リターン>
とする。後者はデバッグ時等に使用する。

◎プログラムの保存

トップレベルから、
　　%w　ファイル名　<リターン>
とする。

◎ログファイルの設定、解除

ログファイルを設定するには、
　　%l　ログファイル名　<リターン>
とする。以後の全画面がそのままファイルに格納される。
ログを中止するには、
　　%l　ｎｏ　<リターン>
とする。

7．制約単一化機構の使い方

制約単一化機構のうち，モジュラー変換ルーチンのみを使うこともできる。

◎@モード：トップレベルから変換ルーチンを単独で使う。
　　@　制約の並び　<リターン>
とする。すると、制約に等価でモジュラーな制約とそれの定義を返す。

◎ｕｎｉｆｙ（）
　　Prologの中で変換ルーチンを使いたいときは、このｕｎｉｆｙを使う。

なお、変換ルーチンに関する各種コマンドには、以下のものが用意されている。
いずれもトップレベルから入力する。

%r
　　新しく定義された節の簡約化を行うかどうかのフラグを設定する。%
rでオン・オフできる。デフォルトはオン。

%n　述語名
　　変換で新たに作られる述語名の最初の部分を設定する。デフォルトは
cなので、c0, c1,……という述語が次々定義される。

%a
　　全モジュラーモード。新しく定義される述語の定義はすべてモジュラー
にする。デフォルトはこちらのモード。

%o
　　m-solvableモード。新しく定義される述語の定義のうち、一つの節だ
けモジュラーにする。

%L
　　integrate()の履歴を表示する。

8. トレース機能

cu-prologには、デバッグ用にトレース機能がある。トレースには2
種類あり、

・ノーマルトレース……途中でストップしない

・ステップトレース……スパイフラグのついた述語が展開されるときに実行を
一時停止し、入力待ちになる。cで中止、リターンで継続。

◎ スパイフラグの設定
ある述語だけ振舞いを注目したい場合は、
　　%p　述語名　<リターン>
とする。フラグは複数の述語に設定できる。
%pコマンドはスイッチになっているので、もう一度実行すると、スパイオフとなる。
また、全述語のスパイフラグをオンするには、
　　%p　*<リターン>
全述語のスパイフラグをオフするには、
　　%p　.　<リターン>
とする。
なお、スパイした述語名を知るには、
　　%p　?<リターン>
である。
制約変換ルーチン(fold/unfold変換)のスパイスイッチは、
　　%p >
である。

◎ ステップトレースの設定
スパイした述語の呼び出しをステップトレースするには、

%s　<リターン>

とする。これは、スイッチになっているので、もう一度行うとステップトレースオフになる。
ステップトレース時には、カーソルの形が　>　になる。

◎ ノーマルトレースの設定
上と同様に、

%t　<リターン>

で、スパイした述語のノーマルトレースをオン・オフする。
ノーマルトレース時には、カーソルの形が　$　になる。

9. その他のコマンド

トップレベルから実行するコマンドは、そのほか以下のようなものがある。いずれも%から始まる。

%h　　ヘルプの表示

%d <述語名>
　　述語の定義の表示。
%d.
　　簡約で消された述語を除いた全ての述語の定義を表示する。
%d/
　　簡約で消された述語を含め全ての述語の定義を表示する。
%d ?
述語の名前と各種フラグの状況を表示する。

組み込み述語	*
スパイフラグの立っている述語	+
簡約化されている述語	−
再帰を含む述語	/

のマークがそれぞれつく。

%c　数字
　　Prologの推論段数の最大値を設定する。無限ループを解消できる。

%f
　　システムヒープの残りを表示する。

%q
　　cu-Prologを終了する。

%s
　　現状態を記憶する。これ以後付け加わった制約や述語は、%$ですべて消すことができる。

%$
　　%s で設定したシステムの状態に戻す。

%3
　　ガーベジコレクションを行う。%&で設定した状態以降の述語や制約を、TEMPF.###ファイルに書き出し、状態復帰した後、再度読み込む。

１０．参考：ＢＮＦ記法によるcu-Prologの構文

```
<alpha numerical char>::=<capital> | <small> | <digit>
<capital>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<small>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit>::=0|1|2|3|4|5|6|7|8|9
<name>::=<alpha numerical char> | <alpha numerical char><name>
<string>::=<empty> | <name>
<var>::=<capital><string> | _<string>
<constant>::=<small><string> | '<name>'
<functor>::=<name>
<predicate>::=<name>
<term>::=<var> | <constant> | <functor>(<term list>)
<term list>::=<term> | <term>,<term list>
<atomic formula>::=<predicate>(<term list>) | <predicate>
<atomic formula list>::=<atomic formula> |
                        <atomic formula>,<atomic formula list>
<constraint>::=<atomic formula list>
<clause>::=
        <atomic formula>. |
        <atomic formula>;<constraint>. |
        <atomic formula>:-<atomic formula list>. |
        <atomic formula>:-<atomic formula list>;<constraint>. |
        ?-<atomic formula list>.|
        ?-<atomic formula list>;<constraint>.
```

１１．cu-Prolog速修マニュアル

　　とにかくcu-Prologを使ってみたいという人のための、モデルセッションです。

＊セッション１：制約変換ルーチンを動かす。

```
tsuda#icot21[1] cup              %cu-Prologを立ち上げる

 *****  cu - Prolog   Ver. 2.36  *****
                  (c) ICOT, H.Tsuda.
          help ->  %h

_member(X,[X|Y]).                %member()述語の定義
_member(X,[Y|Z]):-member(X,Z).
_append([],X,X).                 %append()述語の定義
_append([A|X],Y,[A|Z]):-append(X,Y,Z).

_? member(X,[a,b,c]).            %制約member(X,[a,b,c])を変換する
```

```
solution = c0(X)                    %等価な制約c0(X)を返す
c1(b).                              %新述語 c0(),c1() の定義
c1(c).
c0(a).
c0(X0):-c1(X0).
CPU time = 0.017 sec                %変換時間

_© member(X,[a,b,c,d,e]),member(X,[c,d,e,f,g]). %元の制約

solution = c4(X)                    %等価な制約
c7(d).                              %新述語の定義
c7(e).
c6(c).
c6(X0):-c7(X0).
c4(X0):-c6(X0).
CPU time = 0.017 sec

_© member(A,X),append(X,Y,Z).       %元の制約

solution = c10(A, X, Y, Z)          %等価な制約
c12(X2, X2, Y3, Y0, Z1):-append(Y3, Y0, Z1).
c12(X0, Y3, [A1|X2], Y3, [A1|Z4]):-c12(X0, A1, X2, Y3, Z4).
c10(A0, [A1|X2], Y3, [A1|Z4]):-c12(A0, A1, X2, Y3, Z4). %fold変換により再帰的な定↓
↑義になる
CPU time = 0.017 sec
_%L
 +— List integrate() traces ⟶ %新述語のリスト
<4,2> c10(A, X, Y, Z) <=> append(X, Y, Z), member(A, X).
<4,2> c13(Y0, Z1, X2, Z4) <=> append(Z4, Y0, Z1), member(X2, Z4).
<5,2> c12(A0, A1, X2, Y3, Z4) <=> append(X2, Y3, Z4), member(A0, [A1|X2]).
<1,2> c4(X) <=> member(X, [c,d,e,f,g]), member(X, [a,b,c,d,e]).
<1,2> c5(X0) <=> member(X0, [c,d,e,f,g]), member(X0, [b,c,d,e]).
<1,2> c8(X0) <=> member(X0, [e]), member(X0, [c,d,e,f,g]).
<1,2> c7(X0) <=> member(X0, [d,e]), member(X0, [c,d,e,f,g]).
<1,2> c6(X0) <=> member(X0, [c,d,e]), member(X0, [c,d,e,f,g]).
<1,1> c2(X0) <=> member(X0, [c]).
<1,1> c1(X0) <=> member(X0, [b,c]).
<1,1> c0(X) <=> member(X, [a,b,c]).

_%p member                  %member述語をスパイする
 +++ spy member  +++
_%t                         %ノーマルトレースモードにする（カーソルが$になる）

 +++ normal trace on +++
$:-member(X,[b,c,d]).       %member(X,[b,c,d])のトレース
 [1>>member(X_0, [b,c,d])
      <=1=member(X, [X|Y]).  %一つのプログラム節を適用
Success.
CPU time = 0.000 sec
```

```
member(b, [b,c,d]) ;
  X = b;                          %1つの解を出力。;を入力し、別解を探索
       <=1=member(X, [Y|Z]):-member(X, Z).
 [2>>member(X_22, [c,d])          %次のゴール
       <=2=member(X, [X|Y]).
Success.
CPU time = 0.000 sec
member(c, [b,c,d]) ;
  X = c;                          %2番目の解。別解を探索
       <=2=member(X, [Y|Z]):-member(X, Z).
 [3>>member(X_38, [d])
       <=3=member(X, [X|Y]).
Success.
CPU time = 0.000 sec
member(d, [b,c,d]) ;
  X = d;                          %3番目の解。別解を探索
       <=3=member(X, [Y|Z]):-member(X, Z).
 [4>>member(X_54, [])
  <<4] fail
  <<3] fail
  <<2] fail
  <<1] fail
  <<0] fail
no.                               %他に解はない
CPU time = 0.000 sec

$%p >                             %制約変換ルーチンにスパイをかける
 +++ spy fold/unfold transformation
$%n new                           %新述語名を new??() にする
$@ member(A,X),append(X,Y,Z).

1 transfer  member(A_6, X_8), append(X_8, Y_10, Z_12)  : A_6, X_8, Y_10, Z_12
    <1> new predicate new3 is defined.
1=1(new3)  <<- append([], X, X).        %unfold変換
2 transfer  member(A_6, [])  : A_6, X_164
2 => Fail
1=2(new3)  <<- append([A|X], Y, [A|Z]):-append(X, Y, Z).  %unfold変換
2 transfer  append(X_166, Y_168, Z_170), member(A_6, [A_164|X_166])  : A_6, A_164↓
↑, X_166, Y_168, Z_170
    <2> new predicate new4 is defined.
2=1(new4)  <<- member(X, [X|Y]).        %unfold変換
3 transfer  append(Y_323, Y_168, Z_170)  : Y_168, Z_170, X_321, Y_323
3 => append(Y3, Y0, Z1)
2=2(new4)  <<- member(X, [Y|Z]):-member(X, Z).  %unfold変換
3 transfer  member(X_321, Z_325), append(Z_325, Y_168, Z_170)  : Y_168, Z_170, X_↓
↑321, Y_323, Z_325
3===fold new3(A, X, Y, Z) <=> append(X, Y, Z), member(A, X)     %fold変換
3 => new3(X2, Z4, Y0, Z1)
2 => new4(A0, A1, X2, Y3, Z4)
1 => new3(A, X, Y, Z)
```

```
*** reduce  new4(X2, Y3, Z4, Y0, Z1):-new3(X2, Z4, Y0, Z1). <- new3(A0, [A1|X2], ↓
↑Y3, [A1|Z4]):-new4(A0, A1, X2, Y3, Z4).
          =>  new4(A0, Y3, [A1|X2], Y3, [A1|Z4]):-new4(A0, A1, X2, Y3, Z4).
        %new3()の定義節は一つしかないので、reductionしてnew4()の定義に吸収する

solution = new3(A, X, Y, Z)      %等価な制約
new4(X2, X2, Y3, Y0, Z1):-append(Y3, Y0, Z1).  %新述語の定義
new4(A0, Y3, [A1|X2], Y3, [A1|Z4]):-new4(A0, A1, X2, Y3, Z4).
CPU time = 0.050 sec
```

＊セッション２ ： 制約付ホーン節によるプログラム

```
_"t2.p?                          %エコーバック付きでファイル(t2.p)から読み込む
open 't2.p'
member(X,[X|Y]).                 %member()述語定義
member(X,[Y|Z]):-member(X,Z).
append([],X,X).                  %append()述語定義
append([A|X],Y,[A|Z]):-append(X,Y,Z).
inter(A,X,Y):-member(A,X);member(A,Y). %制約付ホーン節によるintersection

:-inter(X,[p,o,i,p,o,i,p,o,i,p,o,i],[x,c,v,a,s,d,a,s,d,a,s,da]).
Failure.
CPU time = 0.067 sec
:-member(X,[p,o,i,p,o,i,p,o,i,p,o,i]),member(X,[x,c,v,a,s,d,a,s,d,a,s,da]).
Failure.
CPU time = 0.083 sec             %通常のprologによる実行が一番遅い
@member(X,[p,o,i,p,o,i,p,o,i,p,o,i]),member(X,[x,c,v,a,s,d,a,s,d,a,s,da]).
solution = fail.
CPU time = 0.067 sec
```

※※※※［ 注意事項 ］※※※※※※※※※※※※※※※※※※※※※※※※

cu-Prologは実験的、無保証なソフトウェアであり、本説明書の仕様は変更することがあります。cu-Prologの著作権は、ICOTにあるので、研究用にのみ使用して下さい。もちろん、このソースプログラムそのもの又はこれに改良を加えたものを売り物としてはいけません。ソフトウェアにバグを見つけられた方、及び御意見のある方等は、下記まで御連絡下さい。バージョンアップの参考にさせていただきます。

津田 宏

(財)新世代コンピュータ技術開発機構 第二研究室

住所　：東京都港区三田1-4-28 三田国際ビル21F
　　(郵便番号 108)
TEL　：(03) 456-3194
FAX　：(03) 456-1618
E-mail: tsuda@icot.jp

ｃｕ－Ｐｒｏｌｏｇ　ソースファイル概要

津田 宏

ICOT第二研究室

第1版　８８．７．２
第2版　89.6.28

1．ヘッダーファイル（＊．ｈ）

ｃｕ－ｐｒｏｌｏｇのヘッダファイルは以下の５つである。

ｉｎｃｌｕｄｅ．ｈ
データ構造体の定義、各種マクロの定義、ヒープ及びスタックサイズの設定を行う。全ソースファイルから呼ばれている。

ｆｕｎｃｌｉｓｔ．ｈ
＊．ｃファイルに含まれる関数の一覧。これを読み込むことにより、どのモジュールに入っている関数も自由に使うことができる。ｉｎｃｌｕｄｅ．ｈから呼ばれている。

ｖａｒｓｅｔ．ｈ
全モジュールで共通して使われるグローバル変数の定義を行う。ｍａｉｎ．ｃで呼ばれる。

ｇｌｏｂａｌｖ．ｈ
グローバル変数の外部参照用のヘッダ。ｍａｉｎ．ｃ以外のモジュールで使用する。

ｓｙｓｐ．ｈ
ｃｕ－Ｐｒｏｌｏｇのシステム組み込み述語を表す変数の外部参照用ヘッダ。変数の定義自体はｓｙｓｐｒｅｄ．ｃで行っている。ｓｙｓｐｒｅｄ．ｃ以外のモジュールで、組み込み述語に対する操作が必要なものに使用する。

2．Ｃソースファイル（＊．ｃ）

現在、ｃｕ－Ｐｒｏｌｏｇの Ｃソースファイルは以下のモジュールから成り立っている。

ｍａｉｎ．ｃ
ｐｒｏｌｏｇのトップレベル、エラーの処理を行う。

ｍａｉｎｓｕｂ．ｃ
ｍａｉｎ．ｃのサブプログラム集。ヘルプ、オプション（％）コマンド、ファイル操作等

ｎｅｗ．ｃ
ユーザーヒープ、システムヒープ、ユーザースタックの定義。ヒープ、

スタック関係の関数がすべて定義されている。ヒープ上に構造体をallocする関数は　N＊＊＊　という名前で作られている。

read.c

ホーン節の読み込みについての関数が定義されている。他モジュールとのやりとりは主に　cbuf（1文字）、nbuf（文字列）の2つのグローバル変数に基づいて行う。

print.c

節、項などの表示関係の関数が定義されている。ほとんどの関数は、項や節を表す変数と、その環境（主にeという変数で表す）とをペアにして呼ばれる。

refute.c

いわゆるprologの反駁本体の処理を行う。refute（c, e）という関数がほとんどを占める。組み込み述語に対しては、syspred.cファイルに含まれるsystempred（）という関数により処理する。

syspred.c

cu－Prologのprologの組み込み述語の処理を行う。systempred（）関数は、項を引数にとり、それが
・組み込み述語でtrueのときはSYSTRUE
・組み込み述語でfailのときはSYSFAIL
・組み込み述語でないときは　0を返す。
（近いうちに、遅延状態を示すSYSSUSPENDも組み込む予定）現在の組み込み述語は、Unify、Write、NewLine、Equal（=）がある。

unify.c

ユニフィケーションの処理を行う。オッカーチェックも行っている。

cunify.c

制約単一化（依存伝播）の処理に必要なツール関数集。up＊＊（）関数群は、ユーザーヒープに格納されている一時的な項や節を、システムヒープにコピーするのに使う。greater（）は2つの項の順序を計算する。integrateのトレースを格納するときにこれを用いて節をソートすることで能率を上げる。goodterm（）は基礎項からなる節をprologとして実行し、true／failを返す。

genfunc.c

制約単一化で新たに作られる節の名前を作り出す。通常はc0, c1...という名前のものができるが、gennameというグローバル変数を書き換えることにより変えることができる。

modular.c

制約単一化の呼び出し関数、startmodular（）をはじめ、modularizeの処理を行う。

split.c

制約単一化の処理中、項を変数の同値類により分割する処理を行う。現在は与えられた項を静的に分割するだけだが、将来は増進的に分割が行えるように改良して行きたい。

ｒｅｄｕｃｅ．ｃ
制約単一化でできたホーン節を簡約化する処理を行う。定義がただ一つの述語を本体に含む節を強制的に書き換えている。

ｉｎｔｅｇ．ｃ
制約単一化の主要部、ｉｎｔｅｇｒａｔｅの処理を行う。ｉｎｔｅｇｒａｔｅ（）という関数がそれ。少し長すぎるので、分割した方がよかろう。

3．コンパイルの方法

ｃｕ－Ｐｒｏｌｏｇの実行コードを得るには、以上のモジュールを分割コンパイルしリンクで合わせるだけである。ただし、使用するＯＳ、コンパイラによりinclude.hの最初の、

```
#define MSDOS ????
```

の値を書き換える必要がある。

ＭＳ－ＤＯＳのＣコンパイラでスモールモデルの場合。
（ｉｎｔが１６ビット、ポインタサイズが１６ビット）
例：ＭＳ－Ｃ　ｖｅｒ．４、Ｔｕｒｂｏ－Ｃ　ｖｅｒ．１．５
　　＃ｄｅｆｉｎｅ　ＭＳＤＯＳ　　１

ＭＳ－ＤＯＳのC-コンパイラでラージ（ヒュージ）モデルの場合。
例：ＭＳ－Ｃ　ｖｅｒ．４（ｌａｒｇｅ）
Ｔｕｒｂｏ－Ｃ　ｖｅｒ．１．５（ｈｕｇｅ）
（ｉｎｔが１６ビット、ポインタサイズが３２ビット）
　　＃ｄｅｆｉｎｅ　ＭＳＤＯＳ　　２

ＵＮＩＸのＣコンパイラの場合。
例：Ｕｌｔｌｉｘ
（ｉｎｔ、ポインタサイズ共に３２ビット）
　　＃ｄｅｆｉｎｅ　ＭＳＤＯＳ　　０
　　デフォルトはUNNIXなので,単にmakeすればよい.

なお、実行していて
　user heap overflow　がでるなら　ＨＥＡＰ＿ＳＩＺＥ
　user stack overflow がでるなら　ＵＳＴＡＣＫ＿ＳＩＺＥ
　system heap overflowがでるなら　ＳＨＥＡＰ＿ＳＩＺＥの値をそれぞれ大きくするとよい。

また、include.h のCPUTIMEの値もコンパイラにより書き変える必要がある。
UNIX4.2/4.3 BSDでtimes()関数があり,1/n単位でCPU-timeを計測する場合には,
　　#define CPUTIME n　　（普通は60）
とする.

それ以外には 0 とする.この時は,実行時のcpuタイムが表示されない.

2.　ｉｎｃｌｕｄｅ．ｈ　ドキュメンテーション

第1版　　８８．７．２
　　８８．１０．２８
第2版　　８８．１１．２９　(ver.2.00)
　　89.6.28

1．概要

　このヘッダでは、全ての構造体が定義され、重要なマクロや変数も定義される。全てのモジュールにも取り込まれる。従ってきわめて大切なヘッダファイルである。

2．＃ｄｅｆｉｎｅ　文　説明

MSDOS

　使用する機種、コンパイラを設定する。変数領域が大きすぎるなどのコンパイルエラーが起こったら、この値が正しいかどうかまず調べることが必要である。
１：ＭＳ－ＤＯＳスモールモデル（ｉｎｔ、ポインタサイズ共に１６ビット）
２：ＭＳ－ＤＯＳラージ（ヒュージ）モデル（ｉｎｔ１６ビット、ポインタサイズ３２ビット）
０：Ｕｎｉｘ（ｉｎｔ、ポインタサイズ共に３２ビット）
の値に設定する。

CPUTIME

　ＵＮＩＸ4.2/3 BSDのＣコンパイラでサポートされている、times()関数（各プロセスのｃｐｕタイムをｌｏｎｇ整数で返す）がない場合は０にする。times()関数が1/n秒単位で計測する時は(普通は1/60単位),

　　#define CPUTIME n

と設定する.この値を指定した場合,トップレベルで実行時間を表示する。

ＨＥＡＰ_ＳＩＺＥ

　ユーザーヒープの大きさを設定する。ユーザーヒープにはｐｒｏｌｏｇ反駁実行時のノード等、一時的データ構造が格納される。user heap overflowのエラーが起こったらこの値を大きくする。

ＳＨＥＡＰ_ＳＩＺＥ

　システムヒープの大きさを設定する。システムヒープには、プログラムホーン節、制約単一化により新たに生成された述語の定義、ｉｎｔｅｇｒａｔｅの履歴、などが格納される。制約単一化機構を使うためには、この大きさが十分なければならない。system heap overflowのエラーが起こったらこの値を大きくする。

ＵＳＴＡＣＫ_ＳＩＺＥ

　ユーザースタックの大きさを設定する。ユーザースタックは、ｐｒｏｌｏｇまたは制約単一化で、主にポインタの付け替えの際に用いる。utack overflowエラーが起きたらこの値を大きくする。ｐｒｏｌｏｇでかなり深い推論をしない限りはこのエラーは滅多に起こらない。

ENV_SIZE

pair構造体のサイズ．MS-DOS huge model の時のみ4．他は2．

tputc ()
tprint? (X, V1, V2……)
　　ログファイルへの出力も兼ねたprint関数マクロ。ソースファイルのprint文にはprintfを使わずにすべてtprintを使う。Xにはフォーマット、V?は引数を入れて用いる。引数の数により異なったマクロを使わなければならないことに注意。lfpはログファイルポインタを表す変数で、通常はNULL、ログをとるときはファイルへのポインタが入る。

readword (S)
　　ログファイルへの出力を兼ねた、入力用マクロ。Sには文字列（配列）を表す変数をとる。

#define skipline
　　CRが来るまで行を読み飛ばす。

NL
　　改行。ログファイルにも対応している。

alpha
white
　　アルファベット、空白文字を検査する。if文の条件に使用。

advance
　　空白文字列を読み飛ばし、次の文字をcbufに入れる。

KEYIN
　　キーボード入力状態かチェックする。fp（入力ファイルポインタ）がstdinか調べている。if文の条件に使用する。

以下の4つのマクロはvar構造体に関するものである。varは変数型を表すデータ構造であるが、プログラム中ではキャスト演算子によりterm型として使われることが多い。ファンクタ、述語の引数には、項、変数どちらもとるためである。従って、プログラムの変数が変数型であるかどうかを調べるには、以下のようなマクロを使う必要がある。tはいずれもterm型の変数を表す。

isvar (t)
　　tが変数であるかどうか？

vnumber (t)
　　tの変数番号を得る

vname (t)
　　tの変数名を得る

vlink (t)
　　tとリンクしている次の、変数型データ構造をterm型として返す。

vconsraint(t)
　　tに関する制約を返す.

isconst(t)
　　tが定数複合項であるかどうか?

notconst(t)
　　tが定数複合項でないかどうか?

down (p, t, e)
struct pair *p, *e
struct term *t
　　pは何も代入していない変数、(t, e)で調べたい項を代入して呼ぶ。(t, e)が最終的に変数にバインドしているときは、(t, p)によりその変数を表して返す。(t, e)が最終的に無変数(項、定数)にバインドしているときは、pにはNULLが入る。

基本的な使い方としては、ある項(t, e)がフリー変数かどうかを検査するときに用いる。down (p, t, e)を行い、pがNULLの時がフリー変数.

downを使う際の注意としては、
```
down (e[i], t->t_arg[k], p[j])
```
のように、変数以外のものを引数に指定してはならないことである。この場合は、
```
e1=e[i]
t1=t->t_arg[k]
p1=p[j]
down (e1, t1, p1)
```
とする。downの実行後、e1, t1, p1いずれもが書き換えられる可能性があるためである。

SYSFUN
　　システム組み込み述語であるかどうかのフラグ

systemfun (F)
　　func型Fを組み込み述語とする。

issystem (F)
　　Fが組み込み述語かどうか?

isuser (F)
　　Fが組み込み述語でない(ユーザー述語である)かどうか?

SPYFUN
　　述語がスパイ状態にあるかどうかのフラグ

spyfun (F)
　　Fのスパイフラグをonにする。

nospyfun (F)
　　Fのスパイフラグをoffにする。

isspy (F)
　　Fがスパイされているかどうか？

isnospy (F)
　　Fがスパイされていないかどうか？

spyswitch (F)
　　Fのスパイフラグを変える（on<->offのスイッチである）

REDUCEDFUN
　　述語がmodularize () で簡約化の対象になったかどうかの
フラグ

reducedfun (F)
　　Fのreduceフラグをonにする。

isreduced (F)
　　Fがreduceされているかどうか？

isnoreduced (F)
　　Fがreduceされていないかどうか？

FINITEFUN
　　述語が有限（展開木が無限にならない）かどうかのフラグ。

finitefun (F)
　　Fのフラグをonにする。

isfinite (F)
　　Fが有限述語かどうか？

isinfinite (F)
　　Fが有限でない述語（再帰を含む述語）かどうか？

TB
　　トレース時のプリントルーチンの最初に置く

STB (F)
　　Fに述語をとる。Fがスパイ状態にあるときトレースするルーチンの
最初に置く。

TE
STE
　　トレースエンド。TB，STBではじめたルーチンの最後におく。

snew（s）
　　sには構造体名をとる。sの示す構造体をｈｅａｐ（システムヒープ）上にａｌｌｏｃする。

ｎｅｗ（s）
　　sの示す構造体をｈｅａｐ（ユーザーヒープ）上にａｌｌｏｃする。

Termalloc(a)
　　aには数値（引数の個数）をとる。引数をa個持つｔｅｒｍ構造体をｓｈｅａｐ上にａｌｌｏｃする。

ｔｅｍｐｔｅｒｍ（a）
　　引数をa個持つｔｅｒｍ構造体をｈｅａｐ上にａｌｌｏｃする。

ＭＦＡＩＬ
　　ｍｏｄｕｌａｒｉｚｅがｆａｉｌしたことを示す。

ＳＹＳＴＲＵＥ
ＳＹＳＦＡＩＬ
　　ｓｙｓｐｒｅｄ．ｃ、ｒｅｆｕｔｅ．ｃ参照。システム組み込み述語が成功したか失敗したかを示す。

最後にｆｕｎｃｌｉｓｔ．ｈを読み込んでいる。

３．構造体定義　説明
凡例：　Ｓ　：システムヒープのみに格納される。
　　Ｈ　：ユーザーヒープのみに格納される。
　　無印：両ヒープに格納される。

変数型　Ｓ
```
struct var {
        int v_type;      変数の場合は１、それ以外は０
        int v_number;    各節ごとに変数番号がつく。
        char v_name[16];  変数名
        struct var *v_link;      次の変数へのリンク。最後はＮＵＬＬ。
        struct clause *v_constraint;  CAHCの制約
};
```

関数（ファンクタ、述語）型　Ｓ
```
struct func {
        int f_arity,     引数の数、０の場合は定数。
            f_number,    関数番号（項の大小関係に使う）
            f_mark;      述語の状態（システム、スパイ等）
        char f_name[16];          関数（述語）名
        struct set *f_set;        述語の定義へのポインタ
        struct func *f_link;      次の関数へのポインタ
        struct itrace *f_integ;   ｉｎｔｅｇｒａｔｅトレースへのポインタ
```

```
};
```

項型

```
struct term {
        struct func *t_func;    述語（ファンクタ）へのポインタ
        int t_constant;         この項が定数のみから成る時,1
        struct term *t_arg[1];  引数項へのポインタ。
};
```

節型

```
struct clause {
        struct term *c_form;    項へのポインタ
        struct clause *c_link;  次の節へのポインタ
};
```

定義型（ホーン節の定義をまとめる）　S

```
struct set {
        struct clause *s_clause;  定義ホーン節本体
        int s_vnumber;            定義に含まれる変数の個数
        struct term *s_vlist;     定義に含まれる変数リストへのポインタ
        strcut clause *s_constraint;    CAHCの制約部分
        struct set *s_link;       次の定義へのポインタ
};
```

環境型（ストラクチャ・シェアリング用の、変数に対する環境）　H

```
struct pair {
        struct term *p_body;    対応する項
        struct pair *p_env;     対応する環境
};
```

スタック型

```
struct ustack {
        int *u_addr;    アドレス
        int u_val;      値
};
```

ノード型　H　（導出木のノード。ｒｅｆｕｔｅ．ｃで主に使う）

```
struct node {
        struct clause *n_clause;  ゴール節
        struct pair *n_env;       ノードにおける変数環境
        struct set *n_set;        定義ホーン節へのポインタ
        struct node *n_link,      親ノード
                    *n_last;      バックトラック先のノード
        struct constraint *n_constraifnt;       制約
```

```
        int *n_hp,              ノード定義時のヒープポインタ
            n_count,            ノード番号（トレース時に使う）
            n_spy;              ノードがスパイの対象かのフラグ
        struct ustack *n_usp;   ノード定義時のスタックポインタ
};

制約型 H（CAHCの制約変換処理で使う）
struct constraint{
        struct clase *co_clause;        制約(リテラルの並び) S
        struct pair *co_env;            制約と変数を結ぶ環境
        struct term *co_var;            制約に含まれる変数のリスト
        int     co_vnumber;             制約に含まれる変数の個数
}

環境付き節型 H（ｃｕで使う）
struct eclause {        /*  environment + clause(copy) */
        struct term     *c_form;  項へのポインタ
        struct pair     *c_env;   項の環境
        struct eclause  *c_link;  次の節へのポインタ
};


全変数型 H （ｃｕでｃｎｏｄｅ毎に変数をかき集める）
struct allvar{          /* var structure for CU */
        struct term     *v_var,   古い変数へのポインタ
                        *v_svar;  新しい変数へのポインタ
        struct pair     *v_env;   環境
        struct eclause  *v_eclause;  この変数を含む節（ｓｐｌｉｔ．ｃ）
        struct allvar   *v_link;  次の変数へのポインタ
        int             v_flag;   分割用ワークフラグ
};


制約単一化ノード型 H （integrateとmodularizeの変数の受渡しにつかう）
struct cnode {
        struct eclause  *n_eclause;  節
        struct pair     *n_env;      対応する環境
        struct set      *n_set;      取り上げている定義
        struct cnode    *n_link;     親ノード（分割時）
        int             n_vnumber;   変数の数
        struct allvar   *n_avlist;   全変数リストへのポインタ
};


ｉｎｔｅｇｒａｔｅトレース型 S
struct itrace{          /* integrate trace */
        struct clause   *it_clause;  定義節
        int             it_vnumber,   含まれる変数の個数
                        it_cnumber;    含まれる項の個数
        struct itrace   *it_link;    次のトレースへのポインタ（％Ｌ）
```

};

3．ｖａｒｓｅｔ．ｈ　ｇｌｏｂａｌｖ．ｈ　ドキュメンテーション

第1版　８８．７．２
第2版　８８．１１．２９（ver.2.00）

1．概要

複数のモジュールで共通に使うグローバル変数は、１箇所で管理しておいた方が見通しがよくなる。ｖａｒｓｅｔ．ｈはグローバル変数を初期定義するヘッダでｍａｉｎ．ｃで読み込まれている。ｇｌｏｂａｌｖ．ｈはｍａｉｎ．ｃ以外のモジュールで、外部変数としてグローバル変数を参照する際のヘッダである。

2．変数名の説明

ｆｐ
入力ファイルポインタ。キーボード入力時は、ｓｔｄｉｎ外部ファイルからの入力時は、そのファイルへのポインタ、が入る

ｗｆｐ
出力ファイルポインタ。
通常出力時は、ｓｔｄｏｕｔ
無出力時（外部プログラムの読み込みの時）は、ＮＵＬＬ
％ｗで定義節を外部ファイルに書き出すときは、ファイルへのポインタ、が入る。

ｌｆｐ
ログファイルポインタ。通常時は、ＮＵＬＬログをとるときは、ログファイルへのポインタ、が入る

ｔｔｙ
テレタイプかどうか？

ｃｂｕｆ
キャラクタバッファ。入力された１文字をしまう。

ｎｂｕｆ
文字列バッファ。入力文字列（１６文字）をしまう。

ｔｆｌｕｇ
トレースフラグ。
０：トレースなし
１：ノーマルトレース（停止なし）
２：ステップトレース（ゴール毎に停止する）

ｓｆｌｕｇ
モジュラー変換モードフラグ
０：Ｍ－Ｓｏｌｖａｂｌｅモード
１：全モジュラーモード

cutf
　　カットフラグ。refuteでカットを読み込むと1になる。
　　cf. Nnode() (in new.c)

v_number
　　変数の数。Nvar () 関数のなかで1ずつ増える。

v_list
　　変数リストのトップ。Nvar () 関数で変わる。

f_list
　　述語リストのトップ。const_list定数（arity＝0の述語）リストのトップ。

n_last
　　prologで、バックトラックで戻るノードを指定するときに使用。Nnode () で書き換えられる。

newf_list
　　integrateのトレースのリストのトップ。

Refcount
　　refuteのカウンタ、これ以上深いレベルの探索は強制的にフェイルあつかいにする。%cコマンドで書き換えることができる。

LIST
　　リストファンクタ

CUNIFY
　　prologの組み込み述語としての制約単一化ファンクタ

CUT
　　cut（!）項

NIL
　　リストの最後、つまり []。

FAIL
　　fail項

gennameN
　　制約単一化でできる変数の名前。genfunc. c参照のこと。

logfile□
　　ログファイル名をしまう。

4.　ｍａｉｎ．ｃドキュメンテーション

第1版　８８．７．３、１１．８
第2版　８８．１１．２９　(ver.2.00)
　　89.6.28

1．概要

　　このモジュールはｃｕ－Ｐｒｏｌｏｇのトップレベルの処理を行う。

2．変数、マクロ説明

ｒｅｓｅｔ
　　ｊｍｐ＿ｂｕｆ変数。エラーからトップレベルに大域脱出（ｌｏｎｇ
ｊｍｐ）をするときに使う。

ｈ，ｓｈ，ｕ，shp_save,shp_init
　　それぞれ、ユーザーヒープ、システムヒープ、ユーザースタック、
定数リストの値をセーブするのに用いる。
f_list_save, f_list_init
　　f_listの値をセーブするのに用いる。
const_list_save, const_list_init
　　const_listの値をセーブするのに用いる。

3．関数の説明

ｍａｉｎ ()
　　ｃｕ－ｐｒｏｌｏｇのトップレベルの処理を行う。
％ｉコマンドを追加するときは、ｃａｓｅ文の並びに付け加えればよい。
ラベルＴＯＰＬＥＶＥＬ以下がトップレベルのループ。
入力カーソルの形状はトレースモードにより異なる。通常は　_
ノーマルトレース時（ｔｆｌａｇ＝１）　は　$
ステップトレース時（ｔｆｌａｇ＝２）　は　>である。

ｅｒｒｏｒ（ｓ）
　　ｓにはエラーメッセージが入る。トップレベルの処理に大域脱出する。

ｐｒｅｐａｒｅ ()
　　グローバル変数、組み込み述語などを初期化する。UNIXの場合は
signal(SIGINT,SIG_IGN)で^C割り込みをカットしている。

ｓｙｓｔｅｍｃｏｍｍａｎｄ（ｃ）
　　％コマンドを実行する。ｃには％の次の１文字が入る。

ｄｅｆｃｌａｕｓｅ ()
　　α：－β1,β2,...,βm;C1,C2,...,Cn　の形の節（定義節）の読み
込みと、システムヒープへの定義のセットを行う。

open_title ()
　　cu−Prologのオープニングタイトル表示

push_status ()
　　%&コマンド（現状態記憶）処理を行う。主な大域変数をセーブする。pop_status()で、システムをこの状態に戻すことができる。

pop_status ()
　　%$コマンド（元の状態復帰）処理を行う。push_status()で設定した状態にシステムを戻す。

init_status ()
　　システムの状態を立ち上げ時に戻す。

garbagecollect ()
　　ガーベジコレクションを行う。push_status()以降に作られた、ユーザー述語をTEMP.###という一時ファイルに書き出し、pop_status()で状態復帰した後、TEMP.###を再度読み込む。

trans_routine ()
　　モジュラー変換ルーチン、@c1,c2,...,cn.　の処理を行う。

questionclause ()
　　質問節、:-g1,...,gm,c1,c2,...,cn.　の処理を行う。

5. Mainsub.c ドキュメンテーション
第1版　88.7.3, 11.8, 11.10
第2版　88.11.29(ver.2.00) 89.6.28(ver.2.2)

1．概要

　　このモジュールはmain.cでPrologのトップレベルから呼ばれるサブプログラムから成っている。

2．関数の説明

putcursor ()
　　prologのトップレベルのカーソルを表示する。
　　通常時は、tflag＝0で、 _
　　ノーマルトレース時は、tflag＝1で、 $
　　ステップトレース時は、tflag＝2で、 >である。

stepswitch ()
　　%sコマンドによる、ステップトレースon/offのスイッチを行う。
tflagは、
　　0：トレースなし
　　1：通常トレース
　　2：ステップトレース
を表している。

traceswitch ()
　　%tコマンドによる、ノーマルトレースon/offのスイッチを行う。

reduceswitch ()
　　%rコマンドによる、モジュラー変換の簡約化モードon/offのスイッチを行う。オンの場合は、変換で新たにできた述語のうち、定義節が一つしかないものは簡約化される。

spyswitch (fname)
　　%pコマンドに対応。fnameで表される述語のスパイフラグをon/offする。spychange()はinclude. h参照のこと。
　　fname＝"*" のときは、全述語スパイオン.
　　fname＝"." のときは、全述語スパイオフ
　　fname＝"?" のときは、スパイ述語の表示
それ以外の時は、fnameという述語名の述語のスパイをオン・オフする。

showdef (fname)
　　%d (display) コマンドに対応。fnameには%dに続いて入力された文字列が入って呼ばれる。fnameの値により4通りがある。

(1)fname = "/"
　　reduceされた述語も含めて、全ての定義を表示する。

(2)fname = "."
　　reduceされた述語を除いた述語の全定義を表示する。

(3)fname = "?"

述語の名前、各種フラグ状況を表示する。

組み込み述語には ＊

スパイされている場合は ＋,

ｒｅｄｕｃｅされている述語には －

再帰を含む述語には ／

のマークがそれぞれつく。

(4)fnameが上のいずれでもないとき

ｆｎａｍｅで示される述語の定義のみを表示する。その名前の述語が存在しない場合は、その旨表示する。

ｌｏｇｈａｎｄｌｅ（ｆｎａｍｅ）

％ｌコマンドに対応している。fnameには引数の文字列が入っている。fnameが"no"の場合は、ログファイルをcloseする。それ以外の場合はfnameで示される名前のログファイルをオープンする。同名のファイルが存在する場合にはオープンできない等のエラーチェックも行う。

ａｌｌｓｐｙ（ｎ）

％ｔ，％ｕコマンドで用いる。ｎ＝１のときは、全述語のスパイフラグをonする。ｎが他の値の時には全述語のスパイフラグをoffにする。

ｈｅｌｐｍｅｎｕ ()

ヘルプメッセージを出力する。

ｆｒｅｅｈｅａｐ ()

システムヒープの残りを表示する。

ｉｎｉｔ＿ｓｙｓｐｒｅｄ ()

システム組み込み述語を初期化する。main.cのprepare()から呼ばれる。

ｆｉｌｅｗｒｉｔｅ（ｎ）

％ｗコマンドに対応。ｎにはファイル名が入って呼ばれる。簡単なエラーチェックを行った後、ｗｆｐを出力ファイルに変えて、ユーザー述語でreduceされていないものの全定義を表示するだけである。

ｄｉｓｐ＿ｆｕｎｃ＿ｄｅｆ（ｆ＿ｆｒｏｍ，ｆ＿ｔｏ）

f_list上で、f_fromからf_toまでのユーザー定義で、簡約されていない述語の定義を表示する。

ｓｅｔ＿ｉｎｐｕｔｆｉｌｅ（ｎ）

ｎにはファイル名が入る。読み込みファイルポインタｆｐをｎに設定する。

ｒｅａｄｆｉｌｅ ()

"file name"コマンド（ファイル読み込み）処理を行う。

set_eof()

ファイル読み込みのEOF処理を行う。

finiteflug (fn)

述語fnの、finiteフラグをチェックする。fnのfiniteフラグが立った場合にのみ1を返し、それ以外の場合（fnがあらかじめfinite、またはfnがinfiniteの場合）は0を返す。ちなみに有限な述語の定義は次である。

定義節が、ヘッドのみからなる述語は有限
定義節のボディが有限な述語から成る述語は有限

setfinite (funclist, funclast)

funclistからfunclastまでの述語のfiniteフラグを設定する。

oscommand ()

OSのコマンドインタープリターを起動する。

以下の2関数は、計算時間表示ルーチンである。times()がサポートされているCコンパイラ(UNIX 4.2/3 BSD)でないと動かない。include.hの

#define TIMEPRINT ???

文で、コンパイラの種類をセットする。使い方は、時刻を計りたいルーチンのすぐ前で、settimer()し、計算の直後にprinttime()する。

printtime ()

タイマーの値を表示する。

settime ()

タイマーをゼロにセットする。

6.　new.c　ドキュメンテーション

第1版　88. 7. 3
第2版　89.6.28

1．概要

　　このモジュールにはヒープ、スタック等のメモリー管理をおこなう関数が入っている。

2．変数の説明

sheap []

　　システムヒープ。プログラムホーン節、制約単一化により作られた新述語の定義、integrateのトレースなどが入る。ガーベジコレクションは現在行っていないのでかなり大きいサイズが必要である。

*shp

　　システムヒープポインタ。

heap []

　　ユーザーヒープ。prologのrefuteで一時的に作られるデータ構造（例：ノード）などが入る。

*hp

　　ユーザーヒープポインタ

ustack []

　　ユーザースタック。リンクを付け換えたりするとき、古いリンクアドレスをスタックに入れておくと、簡単に元に戻すことができる。

*usp

　　ユーザースタックポインタ

FNUMBER

　　関数番号。初期値は0で、新述語を定義するたびに1ずつ増える。節をソートする時に使う。

3．関数説明

salloc (n)

　　nワード（int型のビット数）システムヒープ上にとる。

alloc (n)

　　nワード、ユーザーヒープ上にとる。

Nvar ()

　　新たなvarを定義する。nbufに変数名、v_numberに変数番号、v_listに変数リストのトップ、を入れて呼ぶ。
Nvar () の中では、v_number, v_listが書き換えられるの

で、連続してｖａｒを定義するときは、一番最初にｖ_ ｎｕｍｂｅｒ＝０，ｖ_ ｌｉｓｔ＝ＮＵＬＬとしてｎｂｕｆだけを変えて呼んでいけばよい。新たにできたｖａｒ変数のアドレスを返す。

ｆｕｎｃｓｅａｒｃｈ（ｆｎａｍｅ）

ｆｎａｍｅには述語名を入れて呼ぶ。ｆ_ ｌｉｓｔを順に調べて、対応する述語のｆｕｎｃ構造体へのポインタを返す。もしｆｎａｍｅで示される述語がないときには、ＮＵＬＬを返す。

Ｎｆｕｎｃ（ｎ，ａ）

ｎにはユーザー定義述語名、ａには引数の個数を入れて呼ぶ。あらたに新述語を作り、そのｆｕｎｃ構造体のポインタを返す。新述語はｆ_ ｌｉｓｔに登録される。定数はａ＝０の場合で、ｃｏｎｓｔ_ ｌｉｓｔに登録される。Ｎｓｙｓｆｕｎｃ（ｎ，ａ）システム組み込み述語の定義に使うという違いをのぞけばＮｆｕｎｃ（）と同じ。

Nterm(n)

引数がn個のterm構造体をsheap上にallocする．

Ｎｅｎｖ（ｎ）

ｎには整数（変数の個数）を入れて呼ぶ。変数の数に対応する環境をユーザーヒープ上にとる。ｐａｉｒ構造体は２つのポインタのペアなので、ＭＳ－ＤＯＳラージモデルの場合１つにつき４ワード（＝８バイト）、そのほかの場合は２ワード必要となる。

Ｎｎｏｄｅ（ｌ）

ｒｅｆｕｔｅで用いる。ｌにはノードへのポインタを入れて呼ぶ。ｎ－＞ｎ_ ｌｉｎｋ＝ｌとなるようなノードｎをユーザーヒープ上に作り、そのアドレスを返す。ｎ－＞ｎ_ ｌａｓｔはｃｕｔｆが定義されているときはｌ，それ以外はグローバル変数のｎ_ ｌａｓｔになる。ｎ_ ｌａｓｔ，ｃｕｔｆが書き換えられて処理が終わる。

Ｎｃｎｏｄｅ（ｌ）

制約単一化で用いる。ｌにはｃｎｏｄｅへのポインタを入れて呼ぶ。ｎ－＞ｎ_ ｌａｓｔ＝ｌとなるようなノードｎをユーザーヒープ上に作り、そのアドレスを返す。

Ｎｅｃｌａｕｓｅ（ｃ，ｅ，ｅｃ）

ｃを節、ｅをその環境、ｅｃを任意のｅｃｌａｕｓｅとして呼ぶ。（ｃ，ｅ）と等価なｅｃｌａｕｓｅをつくりその末尾にｅｃをつけて返す。つまり、

ｃ＝ｃ１，ｃ２，ｃ３

の場合、

（ｃ１，ｅ）－（ｃ２，ｅ）－（ｃ３，ｅ）－ｅｃ

なるｅｃｌａｕｓｅリストを返す。

Ｎａｌｌｖａｒ（ｖｌ，ｅ）

ｖｌを変数リスト、ｅをその環境として呼ぶ。（ｖｌ，ｅ）の中の具体化していない変数に対応するａｌｌｖａｒ構造リストを作り、返す。

t r a c e a p p e n d (t, c)

cは元の制約節、tにはi n t e g r a t e () で変換されたcと等価な項、を入れて呼ぶ。cの中の項の個数、変数の個数を計算し、n e w f _ l i s tに付け加える。例えば、

t = c 0 (X, Y, Z)

c = a p p e n d (X, Y, Z), m e m b (X, Z)

だと、i t _ c n u m b e r = 2、i t _ v n u m b e r = 3になる。

u p u s h (p)

pにはアドレスを入れて呼ぶ。pのアドレスとその中身をペアにしてスタックに格納する。MS－DOSラージモデルでは、アドレスが32ビットになるので2回に分けて格納している。

u n d o (u)

uには古いスタックポインタを入れる。u s pがuになるまでスタックをポップしスタックに積まれていたアドレスを元に戻す。

7.　ｒｅａｄ．ｃ　ドキュメンテーション

第1版　８８．７．３
第2版　89.6.28

1．概要

　　このモジュールはホーン節の読み込みに関する関数が集まっている。

2．マクロの説明

#ｄｅｆｉｎｅ　ｕｐｐｅｒ（ｘ）
　　ｘ（文字）が大文字または_のときにｔｒｕｅになる。読み込んだ文字列が変数かどうかのチェックに使う。

pp
3．関数の説明

ｎｅｘｔ（）
　　ｆｐ（入力ファイルポインタ）から１文字取り出してｃｂｕｆにしまう。ＥＯＦの場合はfpのエラーをクリアーした後に，ｃｂｕｆ＝EOFとする。

ａｄｖ（）
　　空白文字をとばして、次の文字をｃｂｕｆにいれて返す。

ｃｈｅｃｋ（ｃ）
　　ｃｂｕｆがｃのときはｃｂｕｆに次の文字を入れて１を返す。そうでなければ０を返す。

ｓｋｉｐ（ｃ）
　　ｃｂｕｆがｃでないときはエラー。そうでないときはｃｂｕｆに次の文字を入れて返す。

ｐｅｒｉｏｄ（）
　　ｃｂｕｆが"."でないときはエラー。そうでないときは何もしない。

ｋｅｙｒｅａｄ（ａ）
　　ユーザー（ｓｔｄｉｎ）からの入力（文字＋ＣＲ）を待つ。それがａで表される文字から始まれば１を、そうでなければ０を返す。ステップや別解探索のときに'ｃ'を入力したかどうか調べるのに使う。

Ｒｎａｍｅ（）
　　ｆｐからアルファベット列を１５文字まで読み込んでｎｂｕｆにしまう。（この１５という数は変える必要がある！！）

Ｒｌｉｓｔ（）
　　ｆｐからリストを読み込み、そのｔｅｒｍ構造体のアドレスを返す。リストはＤＥＣ－１０ｐｒｏｌｏｇの記法に従い、

[a, b, c] は [a, [b, [c, []]]]
[a, b, X] は [a, [b, [X, []]]]
[a, b | X] は [a, [b, X]]
の略記とする。

#define ARGMAX 63
　　ひとつのファンクタまたは述語の引数の個数の最大値

Ｒｔｅｒｍ ()
　　ｆｐから項を読み込み、そのアドレスを返す。
　　　　項：：＝変数｜定数｜複合項
　　　　複合項：：＝述語名（項の列）
なお,定数複合項(変数を含まない複合項)を導入したために,Rterm()およびRlist()は,書き換えた．定数複合項は t->t_constant を1とする．制約変換ルーチンでホーン節のコピーを大量に作る時に,定数複合項はコピーしなくて済むのでメモリの節約になる．

Ｒｆｏｒｍ ()
　　ｆｐから式を読み込む。式とは変数でない項である。

Ｒｃｌａｕｓｅ ()
　　ｆｐから節を読み込む。カットだけ別に扱っている。

Rconstraint()
　　制約読み込みルーチン

Rliteral()
　　Rconstraint()から呼ばれ,制約の1つのリテラルを読み込む

Rvar()
　　Rliteral()から呼ばれ,制約の引数(変数)を読み込む

8.　ｐｒｉｎｔ．ｃ　ドキュメンテーション

第1版　８８．７．３
第2版　89.6.28

1．概要

　　このモジュールは、節や項などをプリントアウトするのに必要な関数の集まりである。

2．変数の説明

Ｌｉｓｔｌｅｖｅｌ
　　リストの深さ。［ａ，ｂ，［］］を［ａ，ｂ］と表示したいときに使っている。

3．関数の説明tabp(n)
　　n個のタブをプリントする

Ｐｖａｒ（ｔ，ｎ）
　　ｔには変数（を表すｔｅｒｍ）、ｎには数を入れて呼ぶ。”変数名_ｎ”のようにプリントされる。ただしｔが無名変数（_）の場合は”_ｎ”とプリントされる。

Ｐｔｅｒｍ（ｔ，ｅ）
　　ｔを項、ｅをその環境として呼び、ｅのもとでのｔを表示する。ｔそのものをプリントしたいときはｅ＝ＮＵＬＬとして呼ぶ。リストだけ、特別なプリントルーチンがいる。

Ｐｌｉｓｔ（ｔ，ｅ）
　　ｔをリスト項、ｅをその環境として呼ぶ。現在、次のような記法を許している。
　　［ａ，［ｂ，［ｃ，［］］］］－>［ａ，ｂ，ｃ］
　　［ａ，［ｂ，Ｘ］］－>［ａ，ｂ｜Ｘ］

Ｐｃｌａｕｓｅ（ｃ，ｅ）
　　環境ｅ付き節ｃの表示を行う。Ｐｔｅｒｍ（）を使っている。

Ｐｅｃｌａｕｓｅ（ｃ）
　　ｅｃｌａｕｓｅのｃの表示を行う。

Ｐａｌｌｖａｒ（ａ）
　　ａｌｌｖａｒ構造ａの表示を行う。

Ｓｈｏｗｈｏｒｎ（ｃ，ｃｓｔ，ｅ）
　　ｃを定義節、cstを制約，ｅをそれらの環境とすると、（ｃ，ｅ）をＡ：－Ｂ；Ｃ　の形に表示する。ｃ＝ｃ１，ｃ２，ｃ３……ならば、ｃ１：－ｃ２，ｃ３……，となる

Pgoal(n)

nはノード。refute()でトレース時に、制約付ゴールを表示する。

Pconstraint(co)

coはconstraint。制約を表示する。

Ｓｈｏｗｆｕｎｃ（ｆ）

述語ｆの定義を、すべてプリントする。

Ｓｈｏｗｔｒａｃｅ（ｃ）

ｃをintegrate()のトレースの定義節として、定義をＡ＜＝＞Ｂ.
の形で表示する。
ｃ＝ｃ１，ｃ２，ｃ３……ならば、

ｃ１＜＝＞ｃ２，ｃ３…….

となる

9.　　r e f u t e . c　ドキュメンテーション

第1版　８８．７．３
第2版　89.6.28

1．概要
　　このモジュールはｐｒｏｌｏｇにおける主処理（反駁）を行う。

2．マクロ説明

#ｄｅｆｉｎｅ　ＮＴＢ
　　ｒｅｆｕｔｅにおいて、トレース結果を出す場合は、トレースフラグ（ｔｆｌｕｇ）が立っていてかつ、そのノードのスパイフラグ（ｎ－＞ｎ＿ｓｐｙ）が立っている時である。

#define NTE
　　end of NTB

3．関数説明

ｔｅｒｍ　＊ｖｒｅｖｅｒｓｅ（Ｖａｒｌｉｓｔ）
　　ｖａｒｌｉｓｔで示された変数列を逆転する。これは、ｐｒｏｌｏｇの実行結果のプリントアウトの時に、変数を出現順に表示するのに用いる。例えば、
　　：－ｆ（Ｘ，Ｙ，Ｚ）．
という節を読み込んだとき、変数リストは
　　Ｚ－＞Ｙ－＞Ｘ－＞０
となっているので、普通にプリントするとＺ，Ｙ，Ｘの順になってしまうのである。

ｖｏｉｄ　ｒｅｆｕｔｅ（ｃ，ｅ）
　　反駁処理本体。現在の所はごく普通に、最左原子式選択、深さ優先探索を使っている。ｃにはトップレベルのゴール節、ｅには初期環境（Ｎｅｎｖでつくったばかりのもの）を入れて呼ぶ。
ｇｏｔｏ文を多用しているので読みにくいかもしれない。４つのラベルから成る。

ＲＥＦＵＴＥ、ＵＮＩＦＹ：
ゴールの一番左の原子式（ｎ－＞ｎ＿ｃｌａｕｓｃ）を選び、それと単一化するプログラムホーン節の本体をゴールとする一段低いノードを作り出す。深さ優先のため、ＵＮＩＦＹの一番最後でＲＥＦＵＴＥにジャンプしており、見かけは無限ループになっている。ゴールが空節（成功時）の場合はＰＲＯＣＥＥＤに、ｆａｉｌ（失敗時）の場合はｎをバックトラックポイントのノードにしてＢＡＣＫＴＲＡＣＫに飛ぶことにより、ループから抜けることになる。

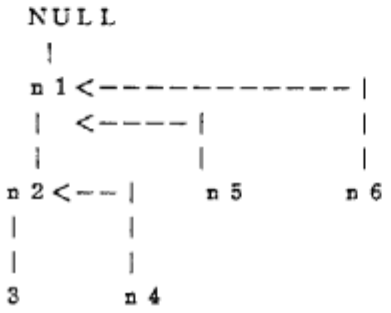
ＰＲＯＣＥＥＤ
あるゴールの証明が成功して、空節になった場合に呼ばれる。一つ前のゴールに移り、次の原子式を展開の対象として選び、ＲＥＦＵＴＥに飛ぶ。ゴールがトップレベルの時は、ｔｒｕｅまたは変数の具体化情報、制約をプリントアウ

トして、次の解を探すかどうかユーザーに聞く。次の解を探索する場合には、
ｎをｎ＿ｌａｓｔ（最も新しいバックトラックポイント）にしてＢＡＣＫＴＲ
ＡＣＫへ飛ぶ。次の解を探索しない場合はｒｅｆｕｔｅ（）関数から抜ける。

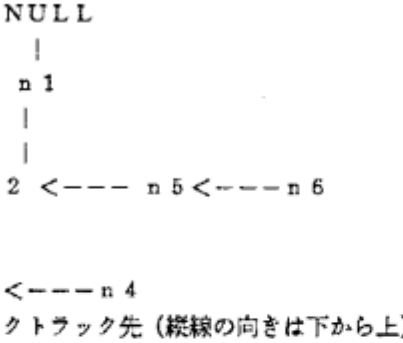
ＢＡＣＫＴＲＡＣＫ
あるゴールの証明が失敗した場合に呼ばれる。ｎにはバックトラックで戻るノー
ドがしまわれている。もしｎがＮＵＬＬのときはトップレベルのゴールの証明
が失敗したことになり、ｆａｉｌ．をプリントアウトしてｒｅｆｕｔｅ（）関
数から抜ける。そうでない場合は、ヒープポインタ、スタックを最初にｎが呼
ばれた状態に戻し、単一化の対象となるプログラムホーン節に次のものを選び、
ＲＥＦＵＴＥに返る。

4．データ構造説明。

ｎ－＞ｎ＿ｌｉｎｋ、ｎ－＞ｎ＿ｌａｓｔ
　一つのノードに対し、他のノードへのリンクポインタが２つある。ｎ＿ｌｉｎｋの方は親ノード、ｎ＿ｌａｓｔの方はバックトラックポイントを示すノードが入る。
例えばｎ－＞ｎ＿ｌｉｎｋは



と、親子関係（向きは子から親へ）を表し、
ｎ－＞ｎ＿ｌａｓｔは



のようにバックトラック先（縦線の向きは下から上）を表す。

カットが入るとｎ－＞ｎ＿ｌａｓｔは次のようになる。

（ｎ４を評価した時点で、ｎ１－＞ｎ＿ｓｅｔもＮＵＬＬにしてＯＲ方向への枝刈りも行う。）

Pbinding(vlist,e)

vlistを変数リスト、eをその環境として、変数の束縛を表示する。

１０．　ｓｙｓｐｒｅｄ．ｃ　ドキュメンテーション

第1版　８８．１０．３０，１１．７
第2版　89.7.26

１．概要

　　このモジュールはｃｕ－Ｐｒｏｌｏｇの組み込み述語を定義する。おもにｒｅｆｕｔｅ．ｃから呼ばれる。

２．変数の説明

組み込み関数の名前は、ｆｕｎｃ構造体へのポインタで、ＭＯＤＵＬＡＲ＿Ｐ，ＩＮＴＥＧ＿Ｐのように最後に＿Ｐをつける。なお、組込み関数を他のモジュールから参照する場合には、sysp.hにも同じ名前で定義をする。

現在サポートされている(か、その作業中の）関数は、
ｗｒｉｔｅ:write()　コンソールに表示する
ｒｅａｄ:read()　標準入力
ｍａｋｅｌｉｓｔ:　=..　関数。複合項をリストにする。　(未)
ｎａｍｅ:name(X,L) 項Ｘの文字列のキャラクタがリストＬ　(未)
ｃｏｕｎｔ:count()　いわゆるＬｉｓｐのｇｅｎｓｙｍ　(未)
ｎｌ:nl　改行
ｔａｂ:tab　タブ
ｅｑｕａｌ:equal(X,Y)　でｘとｙをユニファイする。
ｉｓ:is(X,Y)　is(N,N+1)のように使う。　(未)
ｔｒｅｅ:tree(Hist)　ヒストリを木の形で表示する。

他に定義されているｆｕｎｃ変数
ｍｏｄｕｌａｒ＿ｐ:modularize()　内部関数（プログラム中では使えない）
ｉｎｔｅｇｒａｔｅ＿ｐ:integrate()　内部関数（同上）
ｃａｔ＿ｐ：カテゴリー（単一化文法のノード）
ｃｎａｍｅ＿ｐ：制約リスト

他の変数
ｉｎｔ　ＣＯＵＮＴＮＵＭＢＥＲ
　　Ｇｅｎｓｙｍ () 関数の種。

＃ｄｅｆｉｎｅ　ＴＲＥＥＭＡＸ　２０
　　ｔｒｅｅ () 述語でプリントできる木の深さの最大値。

ｔｒｅｅｈｉｓｔ［ＴＲＥＥＭＡＸ］
　　ｔｒｅｅｐｒｉｎｔ () 用ワーク配列。

３．関数の説明

ｄｅｆｓｙｓｐｒｅｄ ()
　　組み込み述語定義。ｍａｉｎ．ｃのｐｒｅｐａｒｅ () 関数でシステム立ち上げ時に１度だけ呼ばれる。述語名を定義している。

ｓｙｓｔｅｍｐｒｅｄ（ｎ）
　　refute()でのノードnを引数に取る。
t = n->n_clause->c_form,　e = n->n_env とすると、返値は
（ｔ，ｅ）が組み込み述語で、ｔｒｕｅのとき　ＳＹＳＴＲＵＥ
（ｔ，ｅ）が組み込み述語で、ｆａｉｌのとき　ＳＹＳＦＡＩＬ
（ｔ，ｅ）が組み込み述語でないとき、　０
である。ＳＹＳＴＲＵＥの場合は、この関数の副作用として組み込み述語が実行される。

ｅｑｕａｌｐｒｅｄ（ｔ１，ｔ２，ｅ）
　　ｅｑｕａｌ（Ｘ，Ｙ）述語処理部分。（ｔ１，ｅ）が（ｔ２，ｅ）と単一化するかどうかを調べる。
単一化する場合はＳＹＳＴＲＵＥ
単一化失敗の場合はＳＹＳＦＡＩＬ
を返す。

以下、Ｐｔｒｅｅ（）からＰｓｃｃｏｎｔｓｕｂ（）までは、現在サポートしている、履歴表示関数ｔｒｅｅ（）の定義である。

Ｐｔｒｅｅ（ｔ，ｅ）
　　ｔｒｅｅ（Ｈ）述語処理部分。ヒストリーの定義は、次である。

(1)カテゴリーはヒストリー

(2)C,Wがカテゴリーのとき、t(C,W,□)はヒストリー

(3)L,Rがヒストリー、Mがカテゴリーまたはt(C,W,□)の形(C,Wはカテゴリー)のとき、t(M,L,R)はヒストリー

ｔ（Ｃ，Ｗ，［］）の形のヒストリーは、

　　Ｃ－－Ｗ

ｔ（Ｍ，Ｌ，Ｒ）の形のヒストリーは

　　－－－Ｍ
　　　｜
　　　｜－－Ｌ
　　　｜
　　　｜－－Ｒと表示される。

Ｐｓｅｍ（ｔ，ｅ）
（ｔ，ｅ）にはｓｅｍ　featureの値が入っており、それを表示する。

ＰＣａｔ（ｔ，ｅ，ｆ）
　　（ｔ，ｅ）にはカテゴリーを表す項、ｆには１かそれ以外の値が入る。カテゴリーの内容を表示するルーチン。ｆが１のときはｓｅｍ素性の値を表示せず、それ以外の時は表示する。

Ｐｓｕｂｃａｔ（ｔ，ｅ，ｓ）

（ｔ，ｅ）にはｓｕｂｃａｔ素性のようにリストをとる素性の値、ｓには素性名が入る（ＳＣ，ＡＪＡ等）が入る。ｓｕｂｃａｔ素性のプリントルーチン。素性名に続き、素性の値は｛｝で囲んで表示する。

Ｐｓｕｂｃａｔｃｏｎｔ（ｔ，ｅ）

（ｔ，ｅ）にはカテゴリのリスト構造が入る。ｓｕｂｃａｔ素性の中身を表示するルーチン。Ｐｓｕｂｃａｔ（）から呼ばれる。

Ｐｓｃｃｏｎｔｓｕｂ（ｔ，ｅ，ｉ）

（ｔ，ｅ）にはカテゴリのリスト構造、ｉは０か１が入る。ｓｕｂｃａｔ素性の中身を表示するルーチン。Ｐｓｕｂｃａｔｃｏｎｔ（）から呼ばれる。つまり｛ａ，ｂ，ｃ｝のうち、　ａ，ｂ，ｃ　を表示する。この際、　ａ　／　，ｂ　／　，ｃ　と３回この関数が呼ばれる。ｉ＝０のときは　ａ　のように最初にカンマを打たず、ｉ＝１のときは、，ｂ　／　，ｃ　のようにカンマの次に第１要素を表示する。

以下、ｔｅｒｍｎａｍｅ（）からｃｎａｍｅ（）までは、ｔｒｅｅ（）に関する関数だが、現在は使われていない。これでやろうとしていたことは、［ｃ１２（Ｘ１，Ｙ１，Ｚ１２），ｃ１３（Ｕ２２，Ｖ２３）］のような長い制約を［ｃ１２，ｃ１３］と簡略化して表示することである。

ｃｈａｒ　＊ｔｅｒｍｎａｍｅ（ｔ，ｅ）

（ｔ，ｅ）は複合項。その述語名を返す。

ｐｉｃｋｎａｍｅ（ｔ，ｅ）

（ｔ，ｅ）はリスト形式の制約。例えば、［ｆ（Ｘ），ｇ（Ｙ，Ｚ）］。制約の項の個数（つまりリストの長さ）を返す。（ｔ，ｅ）がリスト形式でないときはエラーで、０を返す。

副作用として、ｃｎａｍｅｔｍｐ［］というスタティック文字列配列に制約の述語名（上の例だと、ｆ，ｇ）を入れる。

ｔｅｒｍ　＊ｃｎｌｉｓｔｍａｋｅ（ｎ）

ｎは整数で制約の個数が入る。

ｃｎａｍｅｔｍｐ［］配列の要素（制約の述語名）からなるリストを返す。

ｃｎａｍｅ（ｃｎｓｔ，ｃｎ，ｅ）

ｃｎｓｔにはリスト形式の制約。（ｃｎ，ｅ）はインスタンシェイトされていない変数とその環境をとる。変数ｃｎをｃｎｓｔの述語名からなるリストにインスタンシェイトする。うまくインスタンシェイトしたとき、ＳＹＳＴＲＵＥ，失敗したとき、ＳＹＳＦＡＩＬを返す。

11.　unify.c　ドキュメンテーション

第1版　88. 7. 3

1．概要

このモジュールは、prologにおける通常の単一化を行う。

2．関数説明

いずれの関数も、単一化成功時にはそのままreturn、失敗時にはlongjmpで、unifyを呼んだモジュール（例えばrefute.c）のsetjmp文に大域脱出する。失敗時にはsetjmp関数の値は1になる。jmp_buf変数名はfailである。

```
void  ocheck (p, t, e)
pair  *p, *e
term  *t
```

オッカーチェックを行う。pという環境が表す変数が、（t, e）という項に出現する場合は単一化失敗、出現しない場合は何もせずに返る。

オッカーチェックとは、例えばf（X, X）とf（X, g（X））とを単一化して、Xがg（g（……　と無限になってしまうのを防ぐことである。

```
void  unify (t, e, u, f)
pair  *e, *f
term  *t, *u
```

単一化本体の処理を行う。つまり（t, e）＝（u, f）となるように、環境e, fを書き換える。環境を書き換えるときには、upush ()関数で前の値を保存している。これをやっておけば、unifyを呼んだ方のモジュールで、単一化環境を清算したいときにはundo ()を行えばよい。

ちなみにprologにおける単一化とは次のように定義されている。

・変数はどれとも単一化する。
・アトム（定数）は同じアトムとのみ単一化する。
・複合項どうしは、ファンクタ（述語名）が等しく、対応する各引数がそれぞれ単一化するときのみ、単一化する。

１２．　ｃｕｎｉｆｙ．ｃドキュメンテーション

第1版　８８．１１．７
第2版　89.6.28, 7.26

１．概要

ｃｕｎｉｆｙ．ｃは制約単一化関係の各種サブルーチンの寄せ集めである。ｍｏｄｕｌａｒ．ｃ、ｉｎｔｅｇ．ｃ、ｓｐｌｉｔ．ｃ、ｒｅｄｕｃｅ．ｃ等からおもに呼ばれる。

２．変数説明

```
#ｄｅｆｉｎｅ　VMAX　５０

ｉｎｔ　ｖａｒｏｃｃｕｒ［VMAX］
```

制約に現れる変数のうち出現頻度最大のものを選ぶルーチンで使う。VMAXは変数の個数の最大値、ｖａｒｏｃｃｕｒ［］は、各変数が何回出現しているかを要素にとる。ｖａｒｃｈｅｃｋ（）でセットされる。

３．関数説明

関数の間の関係は次のようになっている。

```
clauseapp()
up()
upclause() -> varcheck()
              insert() -> greater()
unchanged()
upvar()
tabp()
goodterm()
occured() -> eq2() -> unify2()

transform() -> onestep_reduce() -> onestep_termreduce() -> compress_clase()
```

ｃｌａｕｓｅａｐｐ（ｃ１，ｃ２）

ｃ１，ｃ２をｅｃｌａｕｓｅへのポインタにとる。ｃ１の最後にｃ２をつなげる。ただし、スタックにｃ１の最後のセルのポインタをｐｕｓｈしているので、ｕｎｄｏすればリストは切れて元に戻る。

ｔｅｒｍ　ｕｐ（ｔ，ｅ）

（ｔ，ｅ）はユーザーヒープ（ｈｐ［］）に格納されている項。これを、システムヒープ（ｓｈｐ［］）上に吸い上げ、新しく作られた項を返す。

この関数を呼ぶ前にはあらかじめ、（ｔ，ｅ）のうち、具体化されていない変数ｈｖとその環境ｈｖｅについては、ｈｖｅ－＞ｐ＿ｅｎｖが、あるａｌｌｖａｒ構造体ａｖを差し、ａｖ－＞ｖ＿ｓｖａｒがシステムヒープ上の変数を指しているとする。新たな項はシステムヒープ上に、それらの変数について構成される。なお,定数複合項(変数を含まない複合項: t->t_constant = 1)を導入する事で,無駄なコピーを作るのを防いでいる.

varcheck (t, e)

(t, e) は制約をあらわす基礎項。この中の変数の頻度をvaroccur [] 配列にセットする。なお、変数の順番は、(t, e) のなかの変数に対応するshp [] 上の変数の変数番号によっている。

clause *insert (c, cl)

cが1つのclause、clがclauseの列を指すとする。cl の列は、項の評価値の大きい順にソートされているとする。cの評価値を計算し、clの中の正しい位置に挿入する。cと、clの項をはじめから比較していき、cが最初に大きくなったところに挿入している。

clause *upclause (ec)

ecはhp [] 内の、環境付き項(eclause型)を指す。ecで表されるclause列をshp [] 上に吸い上げるのだが、その際に項の評価値の大きい順にソートした節をつくる。また、副作用として、integ.c中で使われるECMAX(最大項)というグローバル変数に、ec中の最大項を入れる。

graater (t1, t2)

t1, t2を項として、両者の評価値の比較を行う。返す値は

t1=t2 のとき 0
t1>t2 のとき 1
t1<t2 のとき 2

である。比較方法は、基本的には 複合項 > 変数 > [] である。
・[] = [] それ以外は [] は何よりも小さい。
・変数同士は、出現頻度が多いものほど大きい。出現頻度が同じ変数は=
・述語名、ファンクタ名のf_numberの値が大きい項の方が大。
同じ述語名、ファンクタ名の項に対しては、引数を最初から比較して決める.

unchanged (a)

aは1つのallvar構造を指す。aの指す変数が何か(変数でもいい)に具体化されたときは0そうでないときは1を返す。

term *upvar (a)

allvar構造体aの指す変数(t, e)が最終的に指している変数(t. p)から、shp [] 上に新変数を作る。新変数の名前は、変数(t, p)の名前にv_number(この関数を呼ぶ前にあらかじめ値を入れておく)を続けたものになる。

tabp (n)

n桁のタブを表示する。

int goodterm (t, e)

(t, e)は、変数を全く含んでいない項。モジュラー変換の対象にはならないが、この項に矛盾があっては変換が失敗してしまう。この関数は、(t, e)を通常のprologで実行して、trueなら1、failなら0を返す。

term ＊occurd（c，n）

cをclause型の節、nをcが含んでいる変数の個数とする。integrateの履歴に、integrate（cc）＝tがあり、ccが変数の順序を除いて同一の場合は、項t（の変数をcにより並べ変えたもの）を返す。履歴になかった場合はＮＵＬＬを返す。

int eq2（c1，c2，e）

節c1，c2と、環境eをとる。＊occured（）関数で、履歴節と候補節が変数の付け替えをのぞいて同一か調べる。成功時には1、失敗時には0がかえる。副作用として、c1＝（c2，e）となるように環境eを作る。

void unify2（t1，t2，e）

項t1，t2が変数の付け替えを除いて同一のものか、テストする。成功時は、t1＝（t2，e）となるようにeが書き換えられ、失敗時は、jmp_buf変数eqfailを通じて、eq2（）に大域脱出する。

struct constraint *transform(precond, newc, newenv)

CAHCの制約変換処理を行う.precondは古い制約,(newc,newnev)は新しい制約で,両者の共通制約を返す.2つの制約の和を一旦sheap上に吸い上げ,startmodular()を呼んでいる.cu()でやっているのと同じ方法.制約変換失敗時は,trans_failでrefute()に大域脱出する.

struct clause *onestep_reduce(cl,env)

制約節(cl,env)を簡約する。制約に含まれる素式のうち、
定義節が一つのみからなり、かつ頭部のみの節からなる
ものを手続きとして実行する。

void onestep_termreduce(c,e)
struct clause *compress_clause(cl)

これらは、onestep_reduce()のサブルーチンである。

１３．　ｇｅｎｆｕｎｃ．ｃ　ドキュメンテーション

第1版　８８．７．３

１．概要

このモジュールは制約単一化で自動的に作られる関数名を設定する。

２．変数説明

ｇｅｎｎａｍｅ []

関数名の最初の文字列。ｍａｉｎ．ｃで定義される。初期値は”ｃ”であるので、ｃ０，ｃ１……という新たな関数が作成されることになる。ｃｕ－Ｐｒｏｌｏｇのトップレベルから、%n　コマンドで書き換えることができる。

３．関数説明

ｃｈａｒ　＊ｉｔｏａ（ｎ，ｓ）

整数ｎを文字列ｓに変換する。

ｖｏｉｄ　ｇｅｎｆｕｎｃ（）

新関数名（ｇｅｎｎａｍｅ [] ＋番号）をｎｂｕｆに入れて返す。

１４．　ｍｏｄｕｌａｒ．ｃドキュメンテーション

ｍｏｄｕｌａｒ．ｃドキュメンテーション

第1版　８８．１１．７第2版　　89.6.29

１．概説

このモジュールは、制約単一化アルゴリズムの、modularlize()の処理を行う。ｉｎｔｅｇ．ｃのｉｎｔｅｇｒａｔｅ()と互いを呼びあいながら、変換を進めていく。

２．変数の説明

#ｄｅｆｉｎｅ　MODULARMAX

ある変換において、modularize()の呼び出し回数の上限を決める。これ以上の回数を呼び出すとｆａｉｌになる。無限ループの解消法。

ｊｍｐ＿ｂｕｆ　ｍｏｄｅｒｒｏｒ

modularize()で、例えばステップモードで強制終了するときに、大域脱出するときの変数。startmodular()でセットされ、modularize()から脱出する際に使う。

ｊｍｐ＿ｂｕｆ　ｓｐｌｉｔｆａｉｌ

制約を変数による同値類に分類するルーチンで、失敗したときの大域脱出変数。

３．関数の説明

関数呼び出しの概要としては、
ｕｎｉｆｙ()述語で実行

```
cu()->temset(),tolist()
|
startmodular()
|
modularize() <-> integrate()
```

@モードでの実行

```
modular()
|
startmodular()
|
modularize() <-> integrate()
```

eclause *clausereverse(eclist)

eclistは、eclauseを指す。eclistのリストを逆順にし、逆順リストのトップを返す。変数による同値類分類で、節の順番が逆転してしまうのを、直すのに使う。

modular(c)

cは制約を節の形でとる。cu-prologのトップレベルでの@モードの処理を行う。cは@の次に入力される節である。main.cから直接呼ばれる。制約を変換し、その結果を表示する。

struct clause *startmodualr(clist,varlist,varnumber)

モジュラー変換ルーチンのエントリー。clistは制約の節(shp [] 内)varlistはclistの中の変数の列(shp [] 内)varnumberは変数の個数。変換成功時は、新たにできた制約の節(shp [] 内)を返す。失敗時は、MFAIL(include.h参照)を返す。やっていることは、

1. 制約の変換,
2. 新述語の定義の簡約化,
3. 述語の有限フラグのセットである。

clause *modularize(n,modc)

変換アルゴリズムのmodularize()の処理本体。integ.cのintegrate()と互いを呼び合ながら処理を進める。nはノード。制約や変数リストなどをまとめて渡している。modcはこの関数を呼んだ回数。これがMODULARMAXを越えると強制的にfailする。変換が成功すれば、変換された節(新たにshp [] に作られる)を返し、失敗の時はMFAILを返す。

この関数でやっていることは、

1. modcとMODULARMAXとの比較。MODULARMAXを越えていたら、MFAILを返して終了。

2. split()を呼び、制約を変数による同値類に分ける。splitの失敗は、splitfailなる変数を通じて大域脱出でわかる。split失敗のときは、MFAILを返して終了。

3. 各同値類についてintegrate()を呼ぶ。一つでも失敗になる(FAILが返ってくる)の場合は、MFAILを返して終了。

4. integrate()の返値(項)を集めて、新たに節を作り、それを返して終了。

int cu(t,e)

組み込み述語のunify(旧制約リスト、新制約リスト)の処理を行う。(t, e)はhp [] 上の項であり、unify(引数1、引数2)の形をしている。引数1はリストの形の制約、引数2は変数でなければならない。組み込み述語unify()が成功するときは、1、failのときは0を返す。処

理手順は次の通りである。

1．（t，e）の第1引数のリスト形式の制約を、ｓｈｐ［］上に節の形で吸い上げる。そのときに変数も新たに定義する。なお、第1引数がリスト形式でないとき、また、第2引数が変数でないときは、０を返して終了する。

2．ｓｈｐ［］上の制約、変数、変数の個数を、startmodular()の変換ルーチンに渡す。

3．変換失敗(ＭＦＡＩＬが返ってきた)の時は、０を返して終了。

4．変換された制約(ｓｈｐ［］上)を、リストの形に直して、引数２とｕｎｉｆｙして、１を返す。

ｔｅｒｍ　＊ｔｏｌｉｓｔ(ｃ)
　　ｓｈｐ［］上の節ｃの各項を要素に持つリストを作り、返す。

ｔｅｒｍ　＊ｔｅｒｍｓｅｔ(ｔ，ｅ)
　　ｃｕ()の処理１(制約をｓｈｐ［］上に吸い上げる)を行う。（ｔ，ｅ）は制約を表すリテラル(項)。これをｓｈｐ［］上にコピーした項を返す。途中で、新変数も定義される。従って、この(項)。これをｓｈｐ［］上にコピーした項を返す。途中で、新変数も定義される。したがって、この関数を呼ぶ前に、ｖ＿ｎｕｍｂｅｒ＝０，ｖ＿ｌｉｓｔ＝ＮＵＬＬとしておかなければならない。ワークエリアとして、変数に対する環境ｐのｐ－＞ｐ＿ｅｎｖ(フリーな変数は、ｐ－＞ｐ＿ｂｏｄｙ＝ＮＵＬＬで、ｐ＿ｅｎｖの値は未定)を使っている。一度ｕｐした変数を再びｕｐしないようにチェックしている。
なお,定数複合項(変数を含まない複合項: 定数リストなど)を導入したために，無駄なコピーを作らないようになった．

15. split.c ドキュメンテーション

第1版 88. 11. 8

1. 概要

モジュラー変換ルーチンの、modularize () で、変数の同値類により制約を分類する処理を行っている。

2. 変数の説明

```
extern jmp_buf splitfail
```

同値類分類が失敗したときは、大域脱出（longjmp）で、modular.cのmodularize () 関数に戻る。divide () で使われる。分類が失敗するときとは、変数を一つも含んでいない項をprologとして実行したときfailになった場合である。

3. 関数の説明

関数呼び出しの概要としては（上が下を呼んでいる）、

```
modularize()
|
split ()
|
divide () ->modularize()に大域脱出
|
avsearch ()
|
contained ()
|
eq ()
```

```
eclause *ecreverse (eclist)
```

eclistは、環境付き節のリスト。eclistを逆順にして戻す。

```
cnode *split (n)
```

nは節の制約を含んだcnode。
制約を変数による同値類に分類し、cnodeのリストで返す。
例えば、制約：f(X,Y,Z),g(Z),h(U,V)
変数：X,Y,Z,U,Vが、

```
X,Y,Z .... f(X,Y,Z),g(Z)
U,V ..... h(U,V)
```

の2つのノードの列に分割される。

手順は、

1．まず、ｄｉｖｉｄｅ（）で、制約の節を変数のａｌｌｖａｒ構造のｖ_ｅｃｌｕａｓｅにくっつける。

X……g(Z),f(X,Y,Z)
Y……g(Z),f(X,Y,Z)
Z……g(Z),f(X,Y,Z)
U……h(U,V)
V……h(U,V)

のようになる。これは、一つ一つの項を見て、どの変数が含まれているか調べながら、逐進的にクラスを作っていけばできる。上の例では、f(X,Y,Z)の変数を見て、

X...f(X,Y,Z)
Y...f(X,Y,Z)
Z...f(X,Y,Z)

ができ、次にf(Z)をみると、Zはすでに、f(X,Y,Z)がついているので、g(Z)の次に、f(X,Y,Z)をくっつける。また、Z以外の変数でf(X,Y,Z)がついているものも、g(Z),f(X,Y,Z)を指すようにする。その結果、

X...g(Z),f(X,Y,Z)
Y...g(Z),f(X,Y,Z)
Z...g(Z),f(X,Y,Z)

となる。

2．クラス毎に、新ｃｎｏｄｅを作り、変数、制約を分割して格納する。

ｄｉｖｉｄｅ（ｃ，ａ）
　　ｃには節の形式の制約。ａはその中に含まれる変数に対応した、ａｌｌｖａｒ構造リスト。ｓｐｌｉｔ（）で述べたような手順で、ｃの節の項を一つずつクラスを作りながら、変数の後にくっつけていく。ｓｐｌｉｔ（）から呼ばれる。スタティック変数ＳＥＲＣＨＥＤは、ａｖｓｅａｒｃｈ（）で項がどれかの変数の後に付くと１になる。従って変数を一つも含まない項を検出できる。そのような項は同値類の対象にはならないが、ｇｏｏｄｔｅｒｍ（）関数（ｃｕｎｉｆｙ．ｃ参照）で、ｐｒｏｌｏｇとして実行して、矛盾がないか確認しなければならない。もし、矛盾があった場合は、ｓｐｌｉｔｆａｉｌにより、ｍｏｄｕｌｒｉｚｅ（）に大域脱出する。

ａｖｓｅａｒｃｈ（ｔ，ｅ，ｃ，ａｖｌｉｓｔ）
　　項を変数の後にくっつける処理をおこなう本体。ｄｉｖｉｄｅ（）から呼ばれる。（ｔ，ｅ）は、検索中の項（ｃもしくはｃの引数など）をあらわす。再帰的に呼ばれるたび変わる。ｃは対象となる項。ａｖｌｉｓｔは変数に対応するａｌｌｖａｒリスト。

ｉｎｔ　ｃｏｎｔａｉｎｅｄ（ｃ１，ｃ２）

c 1は項、c 2は項のリスト（節）を表す。c 1がc 2の中に含まれている場合は1、含まれていない場合は0を返す。a v s e a r c h () から呼ばれる。

i n t　e q (t 1, e 1, t 2, e 2)

c o n t a i n e d () から呼ばれる。(t 1, e 1) と (t 2, e 2) とが同一の項を指している場合は1、そうでないときは0を返す。

１６．　ｒｅｄｕｃｅ．ｃ　ドキュメンテーション

第1版　８８．７．１９
第2版　89.6.29

１．概要

　　このモジュールはｉｎｔｅｇｒａｔｅにより作られた定義節の簡約を行う。現在の所は、定義ホーン節が一つしかない述語を強制的に書き換えているだけである。

２．関数説明

Ｓｖａｒ（ｖ）
　　補助関数。変数リストｖの表示をする。

ｃｏｐｙｖａｒ（ｖａｒｌｉｓｔ）
　　変数リストｖａｒｌｉｓｔのコピーを返す。

ｎｔｈｖａｒ（ｖａｒｌｉｓｔ，ｎ）
　　変数リストｖａｒｌｉｓｔのｎ番目の変数（ｖｎｕｍｂｅｒ（ｖ）＝ｎとなる変数）を返す。

ｃｏｐｙｔｅｒｍ（ｏｌｄｔｅｒｍ，ｎｅｗｖａｒ）
　　ｏｌｄｔｅｒｍのヴァリアントをつくる。つまり、ｏｌｄｔｅｒｍの中の変数のみを変数リストｎｅｗｖａｒで置き換えたコピーを返す。
定数複合項(変数を含まない複合項)を導入する事で,無駄なコピーを作らない.

ｃｏｐｙｃｌａｕｓｅ（ｃ，ｖ）
　　節ｃの新変数ｖによるヴァリアントを返す。

ｒｅｍａｉｎｖａｒ（ｖａｒｌｉｓｔ，ｐ）
　　ｖａｒｌｉｓｔを変数リスト、ｐをその環境とする。ｖａｒｌｉｓｔのうちｉｎｓｔａｎｔｉａｔｅされていない変数のみをリストにして返す。

ｖａｒａｐｐｅｎｄ（ｖ１，ｖ２）
　　変数リストｖ１，ｖ２をつなげた変数リストを返す。

ｉｎｓｅｒｔｃｌ（ｃ１，ｃｌｉｓｔ，ｃ２）
　　引数はいずれも節。ｃ１－＞ｃｌｉｓｔ－＞ｃ２という構造を作り、ｃ２の一つ前の節（ｃ－＞ｃ＿ｌｉｎｋ＝ｃ２となるｃ）を返す。

ｖａｒｒｅｒｎｕｍ（ｖ）
　　変数リストｖのｖｎｕｍｂｅｒを（０から順に）書き換える。

ｈｅａｄｕｐｄａｔｅ（ｔ，ｐ）
　　項ｔをその環境ｐによりアップデートする。

ｒｅｄｕｃｅ（ｓ）
　　定義ホーン節を簡約する。ｓに定義（ｓｅｔ）をいれる。
例えば

1. f(a,V):-u(V).
2. g(X,Y,Z):-h(X),f(Y,Z).

とすると、節の簡約の手順は次の通り。

・1節に含まれる変数をコピーする。（V'とする）
・2.のf（Y，Z）と、1のヴァリアト f（a，V'）とを単一化する。
・f（Y，Z）の代わりにu（V'）を置き換える。
・2の変数のうち具体化していないもの（X，Y）と、V'とをつなげてあらたな変数リストとする。このときvnumberも新しくする。

以上により2.は

g（X，a，V'）：－h（X），u（V'）.

となる。

17. integ.cドキュメンテーション

第1版 88.11.8

1. 概説

モジュラー変換ルーチンの、integrate () の処理を行う。

2. 変数の説明

eclause *ECMAX

評価値の最も高い項が入る。cunify.cのupclause () でセットされる。

3. 関数の説明

メインとなるのはintegrate () で、残りの関数は、integrate () から呼ばれる。

printvar (vl)

変数リストvlを表示する。デバッグ用。

clause *msolve (n)

m可解モードでは、新述語の定義節のうち一つだけを右辺をモジュラーにし、あとは右辺に依存関係があってもよいとしている。この関数は、後者の定義節の右辺を作り出す。

set *setconcat (slist, s)

slistをある述語の定義セット、sを新たな定義とする。slistの一番最後にsを付け加える。

eclause *targetclasue (eclist)

eclist節の中で、展開すべき項を返す。選択の順位は、1. 有限な述語の項2. リスト ([] を除く) など変数以外を引数に持つ項3. [] (NIL) を引数に持つ項である。eclistの項全てにおいて引数が変数の場合は、NULLを返す。

eclause *removeclause (ec, eclist)

ecは項、eclistは節。eclistの中からecを取り除いた節を返す。

term *integrate (n, modc, intc)

integrate () 変換ルーチン本体。
nは制約、変数を含むcnodc,
modcはmodularizeの回数,

ｉｎｔｃはｉｎｔｅｇｒａｔｅの回数である。
　　変換が成功したときは、変換した項を返し、失敗したときは、ＦＡＩＬを返す。

主な流れとしては、次のようになる。
1．制約が［］のときは、［］を返す。
2．制約が一つのモジュラーな項から成っているときは、その項をｓｈｐ［］にコピーしたものを返す。
Ｎｏｒｍａｌ：3．制約のうち、展開すべき項を選ぶ。ｔａｒｇｅｔｃｌａｕｓｅ（）　がＮＵＬＬの場合は、ＥＣＭＡＸ項を選ぶ。
4．新述語名を決定する。変換がうまくいったときに、返す項（ｔｒ　ｅｔｕｒｎ）も定義する。
ＵＮＩＦＹ：5．制約をｕｎｆｏｌｄ（展開）し、ｍｏｄｕｌａｒｉｚｅ（）に渡す。
6．新述語の定義節を付け加える。